



POLITECNICO
DI TORINO

POLITECNICO DI TORINO

Master of science-level of the Bologna process in
Mechatronic Engineering

Final project work

Implementation of the IEEE 802.11p protocol on embedded systems

Advisors

prof. Claudio Ettore Casetti
dott. Marco Malinverno

Francesco RAVIGLIONE
Student ID: 236040

DECEMBER 2018

Summary

In the following master degree thesis, an implementation and integration of the IEEE 802.11p protocol in two different models of *PC Engines* boards [1] is investigated. The focus is put on the physical and MAC layers.

PC Engines is a Swiss producer that makes and sells embedded boards, typically based on AMD x86 processors. These boards are specifically targeted to networking applications and always include at least one mini PCI or mini PCIe slot to accommodate a proper WNIC (Wireless Network Interface Controller), providing a quite customizable architecture, as investigated in literature [2].

In both cases, an embedded Linux operating system (OpenWrt) is used, providing all the proper tools needed to develop the work and giving a high degree of flexibility, being completely open source.

The integration of other WAVE important features will be investigated too, such as the different traffic classes defined in the MAC layer of 802.11.

The final goal is to test the implementation, getting data to be analyzed in order to study the system performance, on both boards, highlighting possible problems that could arise.

Part of this work is also related to analyzing the ITS frequency spectrum used by the boards, by means of a proper analyzer hardware, eventually finding out few PHY layer problems affecting the older boards.

Systems like the one which is investigated in this work actually represent an important part of the ITS applications which are becoming more and more of interest both in the

industrial and academic world.

We tried in particular to concentrate on open source solutions, since they are ideal for researching with V2X protocols, allowing implementing new features and patching existing ones with much more ease than commercial products which start to appear nowadays on the market.

The chapters are organized as follows:

1. The first chapter is an introduction to the world of VANETs and related protocols, that are one of the main enabling technologies for the future ITS (Intelligent transportation systems). It also lists existing embedded solutions for vehicular networks.
2. The second chapter introduces the WAVE (Vehicular Access in Vehicular Environment) stack, introduced by IEEE and object of this work.
3. The third chapter is related to the *PC Engines ALIX* boards, presenting the work I did on them and the obtained results in terms of the boards communicating using WAVE. This chapter also introduces the OpenWrt Linux distribution, highlighting its advantages and how the Linux wireless subsystem works.
4. The fourth chapter is related to how the ITS frequency spectrum can be analyzed by means of a spectrum analyzer and to the problems that were found with the ALIX boards
5. The fifth chapter is related to the *PC Engines APU* boards and to the integration of the 802.11p functionalities on them.
6. The sixth chapter is related to the measurements performed on the APU boards, which resulted to be more stable than the older ALIX boards, and to the obtained results.
7. The seventh chapter contains the conclusion of this work, highlighting some possible future improvements.

Contents

1	Introduction	7
1.1	Intelligent Transport Systems	7
1.2	VANETs	9
1.2.1	Basics	9
1.2.2	Components and types of communication	10
1.2.3	Routing	12
1.2.4	Standards for physical and MAC layers	14
1.3	Existing embedded solutions for VANETs	18
1.3.1	Used boards and objectives	22
2	IEEE WAVE	23
2.1	WAVE physical layer	26
2.2	WAVE MAC layer	28
2.2.1	EDCA	28
2.2.2	IEEE 1609.4 for multi-channel operations	31
3	ALIX boards, UNEX DCMA 86P2, OpenWrt and Linux wireless subsystem	37
3.1	ALIX boards and development host	37
3.2	Desk configuration and connections	42
3.3	UNEX DCMA 86P2	44
3.4	Used antennas	45
3.5	The OpenWrt Linux distribution	45
3.5.1	The Linux wireless subsystem	50
3.6	Connecting with the boards	58
3.6.1	PuTTY	58
3.6.2	WinSCP	59
3.7	OpenWrt GCDC 2011 and network configuration	61
3.7.1	Configuration of the boards	62
3.7.2	iPerf	72
3.7.3	Testing the VTL Java code	79
3.7.4	Moving to newer versions of OpenWrt	83
3.8	LEDE 17.01 and OpenC2X	85

3.8.1	LEDE toolchain	89
3.8.2	LEDE images, SquashFS and ext4	91
3.8.3	Patch creation and management: quilt	96
3.8.4	Patching ath5k and iperf	103
3.8.5	C programs for broadcast transmissions	111
3.8.6	Packet sniffing: methodologies	116
3.9	Packet sniffing: RadioTap	125
3.10	OpenWrt 18.06.1	126
4	Analyzing the DSRC spectrum	129
4.1	Introduction	129
4.2	MetaGeek Wi-Spy DBx and Chanalyzer	130
4.3	Kismet Spectools	133
4.3.1	Patching Kismet Spectools	134
4.3.2	Results of the patching work	158
4.4	Looking for possible interferences	160
4.5	ALIX spectrum usage and transmission problems	161
5	APU boards and UNEX DHXA-222	167
5.1	The APU boards	167
5.2	UNEX DHXA-222	171
5.3	Installing a Linux based operating system on the SSD	172
5.3.1	Serial connection with the APU board	175
5.4	Compiling OpenWrt for the APU boards	178
5.5	Configuring the APU boards	179
5.6	NTP synchronization with chrony	181
5.7	Sniffing with an APU board	185
5.8	Bitrates on the APU boards and correct way of measuring them	186
6	Measurements on the APU boards and results	195
6.1	Throughput and packet loss measurements	195
6.1.1	Network configuration and conditions	196
6.1.2	Scripts and commands	197
6.1.3	Plots and results	202
6.1.4	Additional considerations	210
6.2	Characterization of the buffered transmission	210
6.2.1	Network configuration and conditions	211
6.2.2	Scripts and commands, iPerf UDP send loop	213
6.2.3	Plots and results: first set	228
6.2.4	Plots and results: second set	231
6.3	More systematic throughput and packet loss measurements	238
6.3.1	Network configuration and conditions	238

6.3.2	Scripts and commands	239
6.3.3	Plots and results	240
6.4	Traffic classes	247
6.4.1	Network configuration and conditions	248
6.4.2	Scripts and commands	249
6.4.3	Plots and results: first set	260
6.4.4	Plots and results: second set	269
6.5	Indoor received power and connectivity measurements	271
6.5.1	Network configuration and conditions	272
6.5.2	Scripts and commands, meaning of the power values	273
6.5.3	Plots and results	286
6.5.4	Additional considerations	290
6.6	Broadcast communication issue with Linux kernel version 4	290
7	Conclusions	295
7.1	APU boards startup	295
7.2	Channel switching	296
7.3	Improvements	296
7.4	Conclusions	297
7.5	Acknowledgements	298
	Appendices	301
	Appendix A	302
	C programs for broadcast transmissions	302
	rawsock library	302
	Sender program	335
	Receiver program	344
	Receiver program (using AF_INET socket)	348
	Compiler commands	352
	Appendix B	353
	OpenWrt 18.06.1 patches	353
	001-iperf-MAC_AC-patch.patch	353
	202-restore_ocb.patch	358
	600-DE-openC2X-regdb.patch	359
	601-IT-regdb.patch	360
	998-ath5k_ocb.patch	361
	998-ath9k_allow_11p.patch	368
	999-Enable-queueing-in-all-4-ACs-BE-BK-VI-VO.patch	371
	999-Get-hw-queue-pending-stats-from-ath9k-via-netlink.patch	372
	999-ITS-G5D-channels-fix.patch	386
	.bashrc (on the development PC) - modified lines	387

Appendix C	388
OpenWrt 18.06.1 configuration file for the APU boards	388
Appendix D	427
APU boards network configuration files and iw_startup	427
APU_102	427
APU_103	431
Appendix E	434
chrony and system configuration files	434
Appendix F	438
millisleep utility and iPerf debug modifications	438
millisleep	438
iPerf debug modifications	441
Appendix G	454
Scripts for systematic measurements of throughput and packet loss	454
3 Mbit/s	454
Development PC data log data extraction script - 3 Mbit/s	459
6 Mbit/s	462
Development PC data log data extraction script - 6 Mbit/s	466
12 Mbit/s	469
Development PC data log data extraction script - 12 Mbit/s	473

Abbreviations

Automotive

- **ADAS:** Advanced Driver-Assistance Systems, that are implemented to assist the driver in the driving process

Hardware and embedded systems

- **AC97:** Audio Codec 97
- **CAN:** Controller Area Network, protocol used for internal vehicle communications, based on a 2-wire asynchronous serial bus with multi-master transmissions
- **CF:** Compact Flash
- **DMA:** Direct Memory Access
- **DRAM:** Dynamic Random Access Memory
- **FPGA:** Field Programmable Gate Array
- **FTL:** Flash Translation Layer
- **GNSS:** Global Navigation Satellite System
- **GPP:** General Purpose Processor

-
- **IDE:** Integrated Drive Electronics
 - **LEDE:** Linux Embedded Development Enviroment
 - **NEMA:** National Electrical Manufacturers Association, defining standard for electrical materials and embedded systems enclosures
 - **NIC:** Network Interface Controller
 - **OS:** Operating System
 - **PCIe:** Peripheral Component Interconnect (PCI) Express
 - **RTC:** Real-Time Clock, a computer clock (often implemented in an integrated circuit) keeping track of the current time, usually getting its power from a separate battery (or, possibly, from supercapacitors)
 - **SCP:** Secure Copy Protocol
 - **SDK:** Software Development Toolkit
 - **SMA:** SubMiniature version A
 - **SSH:** Secure SHell
 - **VIF:** Virtual network InterFace

IEEE WAVE and ETSI standards

- **AC:** Access Category, corresponding to one of the 4 levels of priority used in EDCA
- **BSM:** Basic Safety Message
- **CAM:** Cooperative Awareness Message
- **CCH:** Control Channel

-
- **DCC:** Decentralized Congestion Control
 - **DCF:** Distributed Coordination Function
 - **DENM:** Decentralized Environment Notification Message
 - **DIFS:** Arbitration InterFrame Space
 - **DIFS:** Distributed InterFrame Space
 - **EDCA:** Enhanced Distributed Channel Access
 - **EDCAF:** Enhanced Distributed Channel Access Function
 - **MLME:** MAC subLayer Management Entity
 - **QoS:** Quality of Service
 - **SAM:** Service Announcement Message
 - **SCH:** Service Channel
 - **UP:** User Priority
 - **WAVE:** Wireless Access in Vehicular Environments
 - **WSA:** WAVE Short Advertisement
 - **WSM:** WAVE Short Message
 - **WSMP:** WAVE Short Message Protocol

Organizations

- **ARIB:** Association of Radio Industries and Businesses
- **CEN:** Comité Européen de Normalisation

-
- **ETSI:** European Telecommunications Standards Institute
 - **FCC:** Federal Communications Commission
 - **IEEE:** Institute of Electrical and Electronic Engineers
 - **INRiM:** Istituto Nazionale di Ricerca Metrologica
 - **ITSA:** Intelligent Transportation Society of America
 - **SAE:** Society of Automotive Engineers

Telecommunications and WiFi

- **BPSK:** Binary Phase Shift Keying, equivalent to 2-PSK
- **BRAN:** Broadband Radio Access Network
- **BSS:** Basic Service Set
- **BSSID:** Basic Service Set Identifier
- **CRDA:** Central Regulatory Domain Agent
- **CSMA/CA:** Carrier Sense Multiple Access with Collision Avoidance
- **DFS:** Dynamic Frequency Selection
- **ESSID:** Extended Service Set Identifier
- **IBSS:** Independent Basic Service Set
- **ICI:** Inter Carrier Interference
- **IP:** Internet Protocol
- **IPG:** InterPacket Gap

-
- **ISI:** Inter Symbol Interference
 - **MAC:** Medium Access Control (layer)
 - **NTP:** Network Time Protocol
 - **OCB:** Outside Context of a BSS
 - **OFDM:** Orthogonal Frequency-Division Multiplexing
 - **PHY:** Physical (layer)
 - **PSK:** Phase Shift Keying
 - **QAM:** Quadrature Amplitude Modulation
 - **RF:** Radio Frequency
 - **RLAN:** Radio Local Area Network
 - **SDR:** Software Defined Radio
 - **SDU:** Service Data Unit, unit of data passed down from an upper OSI layer to a lower OSI layer
 - **TCP:** Transmission Control Protocol, connection oriented protocol over IP
 - **TDMA:** Time Division Multiple Access
 - **TIM:** Traffic Indication Map
 - **ToS:** Type of Service
 - **UDP:** User Datagram Protocol, connectionless protocol used over IP
 - **USRP:** Universal Software Radio Peripheral
 - **WLAN:** Wireless Local Area Network

-
- **WM:** Wireless Medium
 - **WME:** Wireless Multimedia Extension, a Wi-Fi Alliance interoperability certification for the 802.11e features, including the 4 AC used in 802.11p EDCA mechanism
 - **WNIC:** Wireless Network Interface Controller

Vehicular communications

- **C-V2X:** Cellular Vehicle-to-everything
- **DSRC:** Dedicated Short-Range Communications
- **GCDC:** Grand Cooperative Driving Challenge
- **ITS:** Intelligent Transport Systems
- **NLOS:** Non-line-of-sight
- **OBU:** On Board Unit
- **OEM:** Original Equipment Manufacturer
- **RSU:** Road Side Unit
- **VANET:** Vehicular Ad-hoc NETWORK
- **VTL:** Virtual Traffic Lights

Chapter 1

Introduction

1.1 Intelligent Transport Systems

The world of Intelligent Transport Systems (ITS) is gaining more and more importance in the modern world. It is possible to define "ITS" any system that uses ICT technologies to improve safety, performance and productivity of any transportation system, as defined by the 2010/40/EU directive of the European Union [3].

The goal of ITS systems is to enable comfort and safety applications such as:

- Platooning, in which cars or heavier vehicles can group together to form a platoon, all traveling together like a train, composed by different wagons, would do. This allows to increase road safety (for example by making all heavy vehicles travel in a line in a single lane) and to have an optimal usage of the road infrastructure, increasing its overall capacity without going into congestion. Of course, continuous communication between the platoon members must occur. This application has also its importance in saving fuel for the involved vehicles.
- Virtual traffic lights (VTL), in which the vehicles exchange their positions, obtained from accurate GPS receivers (or from other positioning systems), and use them to autonomously coordinate the precedence at the intersection in an efficient manner.

Each driver could then see, through a display, a virtual traffic light to understand whether he can proceed or not. A VTL could also involve a fixed road side unit, that will be in this case responsible for the traffic regulation, leaving less duty to the single cars.

This application has been already investigated by different research groups, with very good results [4]. In [4] they present a work by the IEIIT Wireless Communication Systems Group and they describe what a VTL system is.

- Non-line-of-sight (NLOS), more informally called See-through, allowing vehicles to see through blind spots or event through, for instance, heavy vehicles that are following or preceding, limiting the normal view. This is actually investigated by important OEMs such as Ford, NXP and Cohda [2]).
- Collision avoidance, with hazard warning and/or using technologies such as see-through. Collision avoidance is one of the most important safety related ITS applications. Statistics show that the death rate of road accidents is actually very worrying, with 3400 people killed every day.

This application involves applying algorithms, based on vehicle exchanging data such as their positions, speeds and heading, that are able to detected potential dangerous situations and act accordingly, from simply warning the driver to start an emergency brake maneuver and eventually warn also other incoming vehicles through wireless or mobile communications. Using this kind of technology, the goal is to limit as much as possible the risk of collision, drastically reducing the road accidents and with the final target of ending road fatalities (in the US the goal is to have zero road fatalities by 2050). Among other applications, as studied by a research group in Politecnico di Torino, this one actually requires most vehicle to be ITS equipped in order to have good performances, needing a so called high “penetration rate” (a percentage indicating how fast a certain technologies is adopted and how much it is actually widespread) [5].

Most of these applications, if not all, require cooperative driving, with vehicles exchanging information between them and with the road infrastructure. This implies creating a network of connected cars, able to regulate themselves depending on the received information and to enable technologies such as the ones presented before.

1.2 VANETs

1.2.1 Basics

Vehicular networks, in general, can be based on mobile *ad hoc* networks, and this is the case in which it is possible to speak about *VANETs*, or even on cellular networks (in this case it is more common to talk about *C-V2X*). While cellular networks seems to be better in terms of packet reception ratio and for longer distance transmissions, mobile *ad hoc* networks seems to be better in relation to lower latency and shorter distances [5].

VANET stands for Vehicular Ad-hoc NETworks: they are in fact self-organized networks, composed by vehicles and fixed road units, that are created only when necessary (*ad hoc* in Latin means *for this*, as they are network created on the fly for a specific reason), just like any other mobile *ad hoc* network. They are characterized by wireless communication, typically based on "WiFi" technologies¹, highly mobile nodes, very dynamic network topology, often difficult to predict, and typically require GPS localization.

One of the requirements for VANETs is the ability to establish a connection in a fast way, since the vehicles may be moving even at high speed. Two vehicles may be even visible, for very high speeds, for less than one second, and this can be critical in safety related applications. [6, p. 24]

Due to this, VANETs rely on a variant of the IBSS (Independent BSS) architecture, that do not need any Access Point to pass through. This allows to establish a connection

¹In Europe and U.S. the MAC and physical layers are mostly based on the IEEE 802.11p protocol, an approved amendment of the 802.11 standard that includes enhancements for the vehicular case, described later on

in a shorter time with respect to BSS with infrastructure, as time represents a critical requirement in this case, and takes into consideration the highly dynamical topology of these networks.

In fact, VANETs do not have any fixed topology, and, if needed, the nodes have to exchange data in order to try to understand which other nodes are near them. This kind of data exchange is required in some proposed vehicular network algorithms, such as DV-CAST (figure 1.1), which is presented as an example of algorithm involving VANETs and which is proposed as an algorithm to solve the problem of network fragmentation when a vehicle wants to transmit data to other ones, for example to notify them of an incoming hazard.

In this solution, a vehicle that receives some broadcasted information from other vehicles will rebroadcast it depending on a "map" of the vehicles around it, that is built thanks to *Hello* beacons that are periodically sent by all the vehicles participating in the VANET. For instance, if there are vehicles traveling in the opposite direction and if there are no neighbors further away with respect to the frame source (i.e. the current vehicle is the last of its "line" and then the network is fragmented, as shown in figure 1.1), the information is actually re-broadcasted and received by the opposite direction neighbors, that will store and forward it when other vehicles are encountered. More details about this algorithm are found in [7].

As written before, vehicles are moving and this generates frequent network topology changes and fragmentations. This actually represents one of the major challenges for VANETs, together with the fact that the nodes are moving, generating fast fading channels. As reported in literature, such as in [8], this is a quite relevant problem for which numerous studies and practical experiments have been carried on.

1.2.2 Components and types of communication

It is possible to distinguish the following main components of a VANET:

- OBU (On Board Unit): it is the device embedded in each vehicle, composed by an

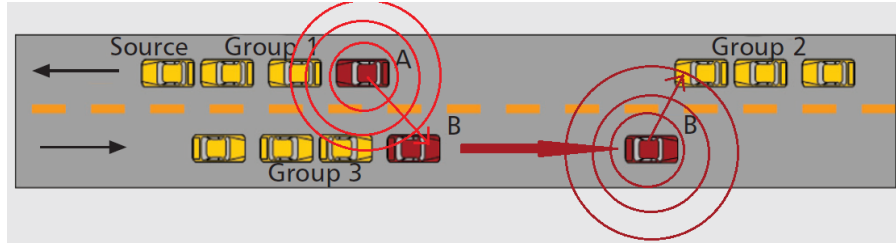


Figure 1.1: DV-CAST example: vehicle A, which is supposed to have received a message about an hazard that it has to re-broadcast, thanks to the *Hello* beacons is able to understand that vehicle B is passing through in the opposite direction. So, it will re-broadcast the message that will be received by B. B will proceed further and re-broadcast again when it encounters vehicles in the opposite direction (i.e. in the same direction as A). Thanks to this mechanism, group 2 cars will be able to receive the hazard notification even though the network is fragmented. Taken from [7] with custom modifications.

embedded system able to communicate through wireless communication.

- **RSU (Road Side Unit):** it is a fixed device positioned along the road, working as a fixed access point. It can provide additional services and support to connected vehicles and connection to the Internet. They can be connected in a mesh and they can be an important element since they can help regulate the traffic flow (for example by interfacing with a traffic light controller), improve safety, improve emergency response in case of hazard, and so on. NXP is for example actually providing solutions suitable for integration inside RSUs, such as the automotive vision processor S32V234 [9].
- **GPS or other positioning systems:** it is a very important component for every ITS application: without any localization system it would be impossible, for instance, to locate an hazard and send a proper notification to surrounding vehicles. The position (and speed) of the cars is used in VTL, in the DV-CAST algorithm to properly send the *Hello* beacons, for any kind of cooperative driving, in most variants of routing algorithms (mainly for georouting based ones, but also for flooding algorithms

and broadcast suppression schemes²). Some routing algorithms that try to use position estimation instead of GPS readings have been proposed [10], but localization systems such as GPS still remain a fundamental component.

In general, it is possible to talk about:

- V2V (Vehicle to vehicle) communication, between OBUs
- V2I or I2V (Vehicle to infrastructure or vice versa) communication, between RSU and OBU [6]
- V2D (Vehicle to Device) communication, involving the vehicle and any device connected to the vehicle itself
- V2P (Vehicle to Pedestrian) communication, which has strong safety implications due to pedestrians being the most vulnerable road users; V2P is an important keyword for example in pedestrian detection ITS applications [11]: for instance, a research group in Politecnico di Torino actually proposed and simulated a collision avoidance algorithm with very good results, analyzing both the vehicle-to-vehicle and vehicle-to-pedestrian detection, in which information about pedestrians is collected through their smartphones in a V2V/V2P framework [12]
- V2N (Vehicle to Network), in which the vehicle directly communicates with the network, instead of having communications with the RSU and then to the network

To talk about vehicular communications in general, the term *V2X* (Vehicle to *everything*) has been coined.

1.2.3 Routing

Another important problem related to VANETs is actually routing. While it is possible to use standard mobile network routing algorithms, they may not be completely suitable

²Described later on

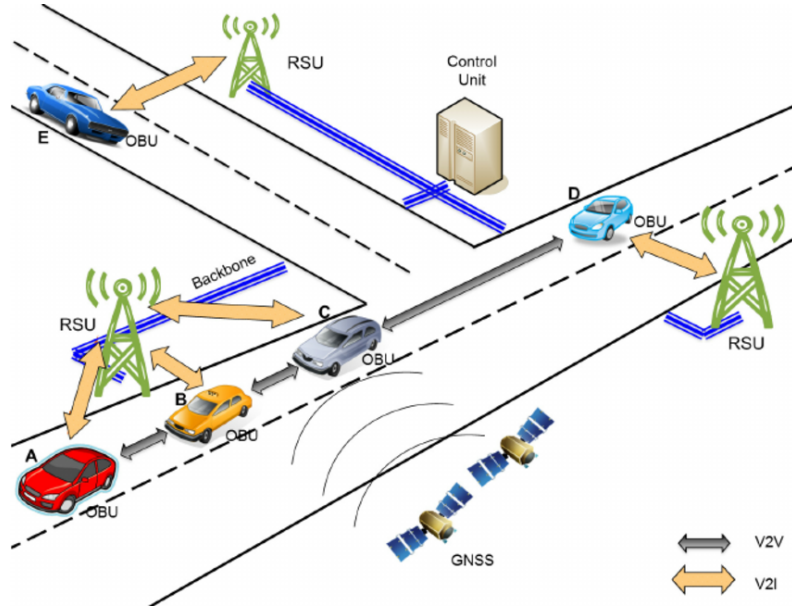


Figure 1.2: V2V and V2I example. Taken from [13].

for vehicular applications. They use, in fact, periodic update procedures to maintain an up to date map of the network, with increased complexity and, possibly, more time lost and inefficiency in running the algorithm in the case of a highly dynamic vehicular environment [10] [14]. Without going into more complex algorithm that have been proposed in these years, such as the one described by [10], two standard solutions for VANETs are:

1. *Georouting*: vehicle position and its distance from the destination are used to route a frame, in a multi-hop approach. It is important to have both a “greedy” operation mode to properly choose the next hop (possibly choosing it as the nearest to the destination) and a “recovery” operation mode to avoid and “circumnavigate” obstacles when they tend to block the communication (for instance, a node may be the nearest to the destination, but it may not be able to route the information towards the next best hop due to obstacles). While georouting seems to be very effective, in practice and with a standard implementation it may require quite a large overhead to find next hops, estimate the distances, understand when obstacles are blocking the message

routing and so on. Some efficient and effective georouting schemes have been proposed though, trying to overcome the problems of inaccuracies in the vehicles and destination positions and routing non-optimality due to outdated topology knowledge, depending on the frequency at which the nodes exchange information about their positions and on their speeds. Another problem is in fact the following one: if the vehicles move faster than the *beaconing* frequency, routing may take the wrong nodes (i.e. vehicles) into account and generate routing inefficiencies and higher latency [14].

2. *Flooding*: one simpler solution, with respect to georouting, is called *flooding*: the solution is actually to broadcast to all nodes the messages when some information has to be sent. Then, these nodes will be responsible for rebroadcasting the information to other vehicles. In fact, in many safety applications there's a little time to set up a topology or use complex routing algorithms. However, performing an uncontrolled flooding can lead, mainly in the case of high vehicle density, to the risk of jamming the channel with too many vehicles all transmitting together. Solutions are presented in the form of *broadcast suppression* schemes, in which the rebroadcast happens only depending on certain conditions. Some scheme, as written before, require GPS information to estimate, for example, the distance to the message source and act accordingly [15].

1.2.4 Standards for physical and MAC layers

Vehicular networks are nowadays based on two main families of standards, for what concerns the MAC and physical layers: DSRC (*Dedicated Short-Range Communications*) standards and cellular based standards. The DSRC standards have been established mainly by ETSI, IEEE and ARIB while cellular based standards are developed by 3GPP and will include 5G technologies, built upon on the current LTE framework.

In particular, concentrating on DSRC standards and VANET networks, that are the

main objective of this work, they are the result of the efforts of ETSI, in Europe, proposing the ITS-G5 standard, IEEE, with 802.11p and 1609.x, in the U.S., and ARIB, with T109, in Japan. Since ARIB T109 takes quite a different approach with respect to ETSI ITS-G5 and IEEE protocols, using a 700 MHz lower frequency band and a mixed CSMA/CA-TDMA medium access control, it will not be considered in details in this work. There are, however, articles presenting more in depth the Japanese standard and comparing it with the ETSI and IEEE ones [16].

Both the ETSI and IEEE standards use the IEEE 802.11p physical and MAC layers. IEEE 802.11p is an approved amendment of the 802.11 family of standards, related to wireless local area networks [17], that introduces enhancements to the known 802.11 protocols to make them suited for the automotive and vehicular use case.

In 1997 FCC (the *Federal Communications Commission*) assigned a 75 MHz bandwidth between 5.850 and 5.925 GHz for vehicular communications, after a petition from ITSA (*Intelligent Transportation Society of America*³), including seven 10 MHz channels and a 5 MHz guard band, needed to avoid invading other technologies' frequencies due to, for instance, Doppler effect.

In 2004, the IEEE 802.11p and IEEE 1609.x (WAVE) working groups were formed and finally in 2010 the new 802.11p amendment was released, using exactly the frequencies reserved by FCC. These frequency bands are called DSRC (*Dedicated Short Range Communications*) frequency bands and are used for all the ITS applications.

In such a way, seven channels are defined: one central *Control Channel (CCH)* and six *Service Channels (SCH)*. Services are advertised on the control channel and ITS stations can tune to specific SCH in order to use the desired ones. The IEEE channel structure will be described more in detail in chapter 2.

³Home page: <https://www.itsa.org/>

In Europe the same frequency bands⁴ have been reserved for vehicular communications, and ETSI, the European standardization organization (*ETSI* stands for *European Telecommunications Standards Institute*), started working on a standard for vehicular communications, ending up releasing ITS-G5, in the *ETSI EN 302 663* document [18].

ITS-G5 is using most of the physical and MAC layer characteristics defined by IEEE 802.11-2016 (802.11p amendment), focusing more on application requirements, foreseeing a multi radio/multi antenna system that avoids channel switching⁵, adding a congestion-based selection of some access parameters (introducing the so called *Decentralized Congestion Control* which tries to adapt the transmission parameters to avoid overloading the channel in case of high load) and defining a new *Facilities* layer, between *Application* and *Networking* [19] [20].

In this additional layer, an *Application Support* sublayer is defined, in which three application support messages are defined:

- CAM (*Cooperative Awareness Message*): broadcasted every 1-10 Hz depending on the vehicle speed and behavior, they contain safety and period speed/position information.
- DENM (*Decentralized Environment Notification Message*): event-based messages that can be used to carry hazard information when needed, containing position information and having typically a local scope. They are sent and reported as long as the ITS station detecting the event is perceiving it.
- SAM (*Service Announcement Message*): they advertise other services available in specific *Service Channels*.

A more fine-grained *SCH* assignment is defined too, together with a more detailed

⁴According to [17, p. 3289] the official frequency allocation for Europe is actually 5.855-5.925 GHz. The first 5 MHz of the US allocation are in any case unused, representing a guard band. An ITS-G5C range at 5.470-5.725 GHz is defined too.

⁵Channel switching for IEEE protocols is described later on

intended usage of the frequency ranges, as reported in tables 1.1 and 1.2.

	Frequency range [GHz]	Usage
ITS-G5D	5.905-5.925	Reserved (future ITS applications)
ITS-G5A	5.875-5.905	ITS safety related applications
ITS-G5B	5.855-5.875	ITS non-safety related applications
<i>ITS-G5C</i>	<i>5.470-5.725</i>	<i>Broadband radio access networks band - RLAN/WLAN/BRAN band - transmit power control, dynamic frequency selection and uniform spreading are mandatory to avoid co-channel interference and with radar systems - cannot be used when outside of a basic service set - more details in [18, appendix C]</i>

Table 1.1: Frequency allocation for ITS in Europe

	Frequency range [GHz]	Channel type	IEEE channel number
ITS-G5A	5.895-5.905	G5-CCH	180
	5.885-5.895	G5-SCH2	178
	5.875-5.885	G5-SCH1	176
ITS-G5B	5.865-5.875	G5-SCH3	174
	5.855-5.865	G5-SCH4	172
<i>ITS-G5C</i>	<i>5.470-5.725</i>	<i>G5-SCH7</i>	<i>94 to 145</i>
ITS-G5D	5.905-5.915	G5-SCH5	182
	5.915-5.925	G5-SCH6	184

Table 1.2: Channel allocation prescribed by the standard for ITS in Europe

For what concerns the IEEE protocol stack the upper layers are instead defined by the IEEE 1609.x standards and the SAE J2735 standard, which defines the so called *Basic Safety Message(s) (BSM)*, similar to the European *CAM*, that comprise 16 different sub-types of messages (including a *reserved* message) and that are transmitted periodically every 100 *ms* (i.e. with a 10 *Hz* frequency). They contain periodic information such as vehicle speeds and positions and they can also be used to report safety information when

needed [21].

To conclude, DSRC frequencies around 5.8/5.9 GHz are actually used for electronic tolling applications, such as *Telepass*®, that, in Europe, is using the so called *CEN DSRC* band, around 5.8 GHz and separated from the other V2X bands [18, Figure 3, p.10]. In fact, as reported by the ETSI standards, ITS-G5 and CEN DSRC should co-exist without interfering. In any case, since the two systems use adjacent frequencies (5.8 GHz and 5.9 GHz), some research has been done in order to avoid any potential interference [22].

1.3 Existing embedded solutions for VANETs

This work will concentrate on DSRC standard and on the IEEE 802.11p amendment (part of the IEEE 802.11-2016 specifications).

All standard compliant OBUs and RSUs will surely contain embedded systems able to perform wireless communications inside the 5.8-5.9 GHz frequency band prescribed by the standard. In this work embedded systems composed by *PC Engines* boards and running Linux operating systems are considered.

These boards must be able to communicate in the DSRC frequency range and to implement all the main features prescribed by standard and related to the physical and MAC layers.

There are, however, other emerging technologies, both open and commercial, that are able to support vehicular communication and the 802.11p protocol. In the awareness that this will not be a complete and exhaustive list (some commercial technologies are not even released to the public and they are probably covered by commercial secrets [2]), some available technologies, as of today, are listed, taking as reference the list in [2]:

- **ARADA Systems**, a company that focuses on solutions for connected vehicles, has produced the *LocoMate*™ series of products, providing hardware for both RSUs and OBUs. Every hardware board is integrated with bluetooth, GPS with 1 m accuracy and high power 802.11p radios, including a full WAVE stack implemented inside (as

described in chapter 2, WAVE stands for *Wireless Access in Vehicular Environments* and it comprises the IEEE 802.11p and 1609.x stacks). The RSU units sold by ARADA are using a NEMA enclosure with water resistance and IP67 certification. SAE J2735 is supported too. This is a commercial solution, as opposed to some open source solutions that are presented below.

- Another solution and prototype is the one produced by **Cohda Wireless**: their latest product is the *MK5*, a complete system for OBUs and RSUs (both the OBU and RSU hardware boards are produced). This system implements both the IEEE and ETSI stacks. According to Cohda their hardware is rugged, small and relatively low cost, with dual 802.11p radio. These systems are based on the automotive chip *RoadLINK* produced by NXP (for which the *RoadLINK SAF5400 Single Chip Modem* is actually in preproduction by NXP, advertised to support IEEE, ETSI, ARIB standards and channel switching) and on a firmware written by Cohda. They offer connection for two 5.9 GHz antennas, GNSS receiver, micro SD card, 12 V DC power supply, serial and Ethernet connections (the Ethernet connection can be used to interface with the vehicle CAN network). Their RSU module comprises also a rugged NEMA enclosure to resist to water and harsh environmental conditions. They also provide an *MK5 XBU*, a DSRC radio device with integrated security and studied for the harsh conditions of underground mining. They provide also an advanced SDK (*CohdaMobility MKx SDK*), a software development kit that can be used in conjunction to Cohda hardware, allowing the running and compiling of applications for Cohda boards and including software, as they report on their website [23], "*necessary to modify and rebuild the MKx firmware to your specifications*".
- The Austrian **Kapsch TrafficCom** group is also developing solutions for connected vehicles, targeted at DSRC 5.9 GHz devices, such as the *Onboard unit KVE-3320 V2X ECU*, supporting ETSI and IEEE standards and with a maximum output power of 23 dBm; as an option, an internal antenna can be inserted. They are based on

the ARM® Cortex® A7 core (based on the ARM instruction set, which is different than the x86 one used by the processors embedded in the boards used for this work), operating up to 528 MHz, and they provide a CAN 2.0B interface to connect to the internal vehicle network. They also have a group developing V2X software, part of the *Connected Vehicle Software Suite*, which is advertised as "*a rapid development framework for V2X applications with a small footprint and high reliability*". A variety of V2X applications are currently, at the date of writing, being developed by Kapsch and an SDK will be part of the software suite, for developers that want to build V2X applications.

- **Marvell** is instead developing a family of automotive wireless transceivers (*Marvell® 88W8987xA*), integrating IEEE 802.11ac and 802.11p for V2X and ADAS applications.
- Moving to more open solutions, that have the advantage of being more accessible and customizable for research and experiments, **UNEX** produced the *DCMA 86P2* miniPCI wireless cards, compatible with DSCR and V2X applications. It can be used in any board as a separate component. This is actually the wireless card used with the ALIX boards considered in this work. It is based on the Atheros *AR5414* chipset, thus supporting the *ath5k* Linux driver. The advantage of this card is that it should be designed specifically for VANETs. It has been also used in various research and practical works, such as [2] [6] [8] [24]. Other newer cards can be used, with proper patching to the operating system, to support and research with IEEE 802.11p, as UNEX *DHXA-222*, based on the *AR9462* chipset, supported by the *ath9k* Linux driver [25].
- The Swiss producer **PC Engines** makes and sells embedded boards, typically based on the *AMD Geode* x86 processor or on the *AMD Embedded G-Series* x86 family of processors. These boards are specifically targeted to networking applications and always include at least one miniPCI slot to accommodate a proper WNIC (Wireless

Network Interface Controller), such as Unex cards, providing a quite customizable architecture, as investigated in literature [2]. All the boards used in this work are from PC Engines.

- Bastian Bloessl, Michele Segata, Christoph Sommer and Falko Dressler demonstrated in their work [8] that it is also possible to build a GPP (General Purpose Processor - as opposed to FPGAs, that have better performance but higher cost and less flexibility) SDR (Software Defined Radio) able to communicate using a standard compliant implementation of both an 802.11p received and transmitter, using the GNU Radio framework, a software development toolkit complete with a graphical user interface (*GNU Radio Companion*) that provides signal processing blocks and libraries. The advantage of this approach, as stated by the authors, is both the ability to access details of the physical layer that would be otherwise difficult to analyze and to be able to use the same code for both simulation and implementation. The disadvantage of this method is, however, the delay and the jitter, introduced by the use of software signal processing and of a non real time operating system. They were, however, able to implement also some time-critical functionalities on an FPGA, keeping the main physical layer software implementation and showing very good results for broadcasted communications, as they happen most of the time in VANETs. As hardware they used an **Ettus Research** USRP (Universal Software Radio Peripheral) N210, an hardware platform used for implementing SDRs, connected to a host computer and an Ettus Research *XCVR 2450 daughterboard*, i.e. a transceiver able to operate with both 10 MHz and 20 MHz channels; this transceiver supports half-duplex operations in the 2.4 and 5 GHz bands, up to 6 GHz [26]. This seems again a quite customizable architecture, since the same SDR algorithms can be implemented on different URSP and using different transceivers. Their work is now available here, under an Open Source license: <https://www.wime-project.net/>.

1.3.1 Used boards and objectives

The final objective of this work is to implement and integrate the IEEE 802.11p protocol inside two different models of *PC Engines* boards [1]. The focus will be put on the lower layers, particularly on the physical and MAC layers.

The two boards that are used are the following ones:

- ALIX 3d2, with UNEX DCMA-86P2 wireless cards, connected through miniPCI and AMD Geode 500 MHz MX800 CPU.
- APU 1d, with UNEX HXA-222 dual-band 2x2 wireless cards, connected through miniPCI express and AMD G series T40E 1GHz dual core CPU, with 64 bit support.

A Linux operating system is used, providing all the proper tools needed to develop the work. In particular, for the ALIX boards the *OpenWrt* distribution is used, being well suitable for embedded devices and networking applications.

The integration of other important vehicular network features will be investigated too, such as the different traffic classes defined in the MAC layer of 802.11p.

Another important objective is to test the implementation, getting data to be analyzed in order to study the system performance, on both boards, highlighting possible problems that could arise.

Part of this work is also related to analyzing the ITS frequency spectrum used by the boards, by means of a proper analyzer hardware.

Chapter 2

IEEE WAVE

A WAVE system is a radio communications system intended to provide seamless, interoperable services to transportation. These services include those recognized by the U.S. National Intelligent Transportation Systems Architecture and many others contemplated by the automotive and transportation infrastructure industries. [27, p. 7]

This chapter will serve as a short introduction to IEEE WAVE protocols.

When talking about VANETs and related protocols, it is common to hear about *IEEE WAVE*.

WAVE means *Wireless Access in Vehicular Environments*. In order to enable such a technology, IEEE defined a family of standards, already introduced in chapter 1, as the amendment 802.11p to the 802.11 protocol (introduced in 2010 and updated in 2016 [17]) and as the 1609.x family of standards. Overall, WAVE covers both the physical (PHY) and MAC layers, using as physical layer 802.11p. The WAVE protocol architecture is shown in figure 2.1.

As it is possible to see from the figure, the WAVE stack supports, as higher layers, both IP and non-IP protocols. In particular, non-IP frames are transmitted using another protocol called *WAVE Short Message Protocol (WSMP)*, which is specified inside the IEEE

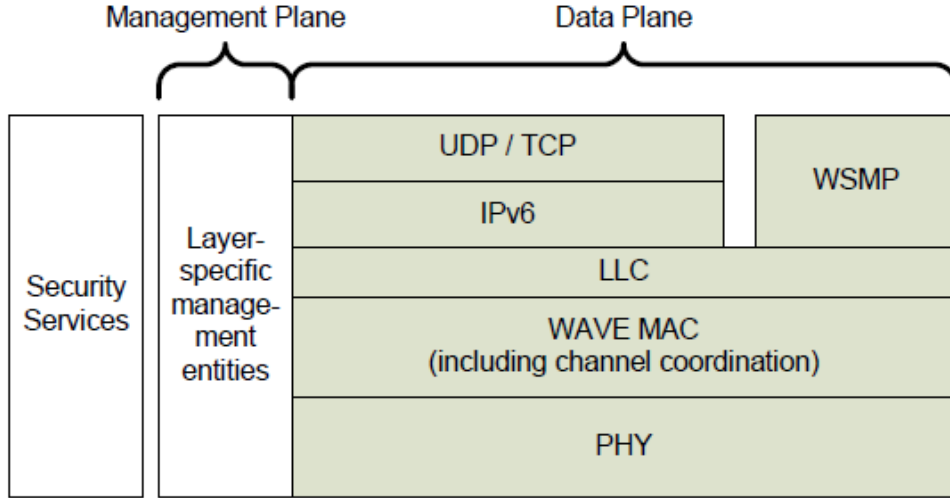


Figure 2.1: WAVE protocol architecture, optimized for vehicular environments, from [28, p. 12].

1609.3 document. This protocol serves as an optimized protocol which minimized communications overhead, thus being able to satisfy in a better way the criticality demands of vehicular networks; it also provides both networking and transport functions, respectively through two headers called *WSMP-N-Header* and *WSMP-T-Header*. The messages that are using WSMP are called *WAVE Short Messages* or, in short, *WSMs* [27].

Behind all of this, security services, as an important part of the WAVE stack, are defined in IEEE 1609.2.

The protocols that are part of the WAVE stack are:

- *IEEE 802.11-2016, IEEE Std 802.11p: Wireless Access in Vehicular Environments (Amendment 6).*
- *IEEE 1609.0: IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture* - it contains an overview about WAVE, providing a context under which all the other WAVE standard documents shall be put [29].

- *IEEE 1609.2: IEEE Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages* - describing all the security aspects [28].
- *IEEE 1609.3: IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services* - providing a description of the WAVE Networking services from the Network layer upwards, such as WAVE management and service advertising, including the WSMP protocol definition too [27].
- *IEEE 1609.4: IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-Channel Operation* - defining "MAC sublayer functions and services" and channel related aspects, such as channel routing, channel coordination and common time reference estimation [30].
- *IEEE 1609.11: IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Over-the-Air Electronic Payment Data, Exchange Protocol for Intelligent Transportation Systems (ITS)* - defining a standard for electronic payments in WAVE based applications, using OBU and RSU as EPS¹ equipment.
- *IEEE 1609.12: IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Identifier Allocations* - specifying the allocation of WAVE identifiers (i.e. how they should be assigned), such as PSID² identifiers, used to identify the application services offered by a certain provider (for example an RSU that provides some services to passing by vehicles), and admitted EtherType³ values (in the LLC sublayer) for IEEE 1609: 0x86DD for IPv6 (no IPv4 is foreseen by the standard, but, as written in [30], the transmission of other frame types is not prohibited) and 0x88DC for WSMP [31].

¹Electronic Payment Service

²Provider Service Identifier

³Indicating the higher layer protocol that is being used.

Moreover, since vehicular communications require immediate data exchange, a new mode, based on the IBSS architecture, is introduced. This mode is called *Outside Context of a BSS* (in short *OCB*).

When a wireless interface is set in OCB mode, it is able to directly send data frames to either broadcast or multicast destinations.

As this work will concentrate on physical and MAC layers, they are briefly described below.

2.1 WAVE physical layer

As mentioned in chapter 1, the physical layer is based on 802.11p.

802.11p is (mainly) based on two other 802.11 amendments: 802.11a, which was based on the 5 GHz band and on an OFDM modulation, and 802.11e, introducing some QoS⁴ enhancements.

OFDM is based here on 64 orthogonal subcarriers, including 48 data subcarriers and 4 pilot ones. Depending on the used modulation, IEEE 802.11p actually supports data rates from 3 Mb/s (with BPSK) to 27 Mb/s (with 64QAM).

Smaller channels are used, having a 10 MHz width instead of 20 MHz, in order to try to counteract fading, multi-path and Doppler effects which may cause the ISI phenomenon (*Inter Symbol Interference*). Another countermeasure against ISI is the introduction of a *cyclic prefix* for any OFDM symbol, which is essentially a repetition of the end of the OFDM symbol that is repeated at the beginning of the symbol itself, instead of introducing a silent guard band. This is not described here, but has some good advantages against ISI and ICI (*Inter Carrier Interference*) too [32].

This has the effect of doubling all the timing parameters: 802.11a in fact supported data rates up to $27 \times 2 = 54$ Mb/s.

⁴Quality of Service

The physical layer main characteristics are well resumed in table 2.1, adapted from [6, p. 23], which was already an adaptation of [33, Table 2].

Parameter	IEEE 802.11p (WAVE)	IEEE 802.11a/g
Frequency band (around)	5.9 GHz	5 GHz
Channel bandwidth	10 MHz	20 MHz
Supported data rates (Mb/s)	3, 4.5, 6, 9, 12, 18, 24, 27	6, 9, 12, 18, 24, 36, 48, 54
Modulations	BPSK, QPSK, 16QAM, 64QAM	BPSK, QPSK, 16QAM, 64QAM
Number of data subcarriers	48	48
Number of OFDM subcarriers	64	64
ODFM symbol interval	$8\mu s$	$4\mu s$

Table 2.1: IEEE 802.11p PHY main characteristics

The available 70 MHz (excluding the guard band), from 5.855 to 5.925 GHz are divided into 7 10 MHz channels, representing, as mentioned in chapter 1, one *Control Channel* (CCH) and six *Service Channels* (SCH) [33]. The channels are represented in figure 2.2.

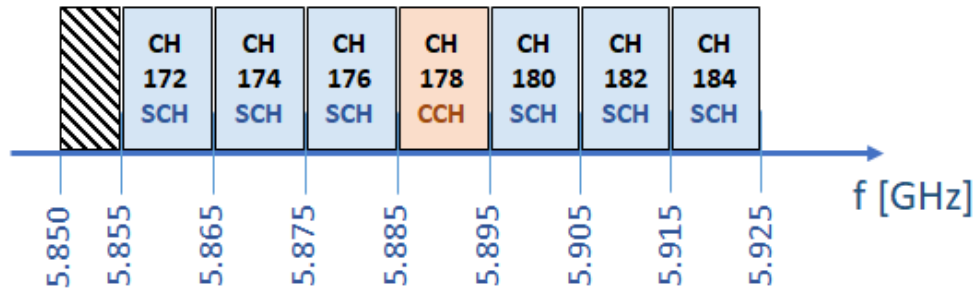


Figure 2.2: WAVE channels. The IEEE channel number (CH) is computed in such a way that $f(CH) = 5000 + 5 \times CH$ [MHz]

The CCH is mapped to channel 178 (corresponding instead to an SCH - G5-SCH2 - in the ETSI standard) and it is used for high priority messages or control data, carrying

only non-IP frames (i.e. using WSMP).

2.2 WAVE MAC layer

WAVE MAC layer is substantially based on the standard 802.11 MAC layer, with some major modifications and added functions related to multi-channel operations⁵ such as:

- It operates outside the context of a BSS (*Basic Service Set*), as described in the previous chapter.
- It uses a variation of the CSMA/CA DCF (*Distributed Coordination Function*), called EDCA (*Enhanced Distributed Channel Access*), that defines the *DIFS* times (now called *AIFS*) in a different way according to traffic classes, based on user priorities (*UP*) and access categories (*AC*) to support QoS. EDCA is defined in the IEEE 802.11-2016 document.

Some peculiar WAVE MAC layer characteristics are described below.

2.2.1 EDCA

WAVE MAC layer is actually using a channel access based on *CSMA/CA*, called *EDCA*.

All the data, with respect to a standard *DCF*, is divided into 4 *Access Categories*, each having a different priority, starting from 8 *User Priorities*.

The four access categories are the following ones (lowest priority-shortest *AIFS[N]* to highest priority-shortest *AIFS[N]*):

1. AC_BK: Background, to which user priorities 1 and 2 are mapped.
2. AC_BE: Best Effort, corresponding to UP 0 and 3.
3. AC_VI: Video, corresponding to UP 4 and 5.

⁵Defined in IEEE 1609.4

4. AC_VO: Voice, corresponding to UP 6 and 7.

Note that the user priorities written here are the same as the ones defined in another IEEE standard (for a different purpose), i.e. in IEEE 802.1D, constituting a base of the spanning tree protocol for bridges and switches. The priorities from the lowest to the highest are, according to [17, Table 10.1], 1-2-0-3-4-5-6-7.

Each access category corresponds to a transmit queue inside the MAC layer implementation; each of these queues will then have to virtually contend the channel access as in DCF with backoff (like if the contention happened inside the MAC layer), after it has been idle for an Arbitration InterFrame Space (*AIFS*), which assumes the meaning of the standard Distributed InterFrame Space (*DIFS*). The queue winning the contention will be able to access the physical channel.

Each queue has its own (*AIFS[N]*), which is shorter for higher priorities queues and longer for lower priority ones.

The backoff procedure, in EDCA, is performed using AC-dependent parameters such as minimum and maximum contention window size, that are reported in table 2.2 [17].

AC	CW _{min}	CW _{max}	AIFSN	TXOP limit
AC_BK	aCW _{min}	aCW _{max}	9	0
AC_BE	aCW _{min}	aCW _{max}	6	0
AC_VI	(aCW _{min} +1)/2-1	aCW _{min}	3	0
AC_VO	(aCW _{min} +1)/4-1	(aCW _{min} +1)/2-1	2	0

Table 2.2: Default EDCA parameters when operating in ad-hoc mode. From [17, p. 899]

In this way, in case of contention, the high priority queue will transmit, *likely*, before the low priority queue. The fact some randomness is still remaining due to the backoff mechanism ensures that the highest priority queue does not monopolize the channel.

The standard EDCA architecture is shown in figure 2.3.

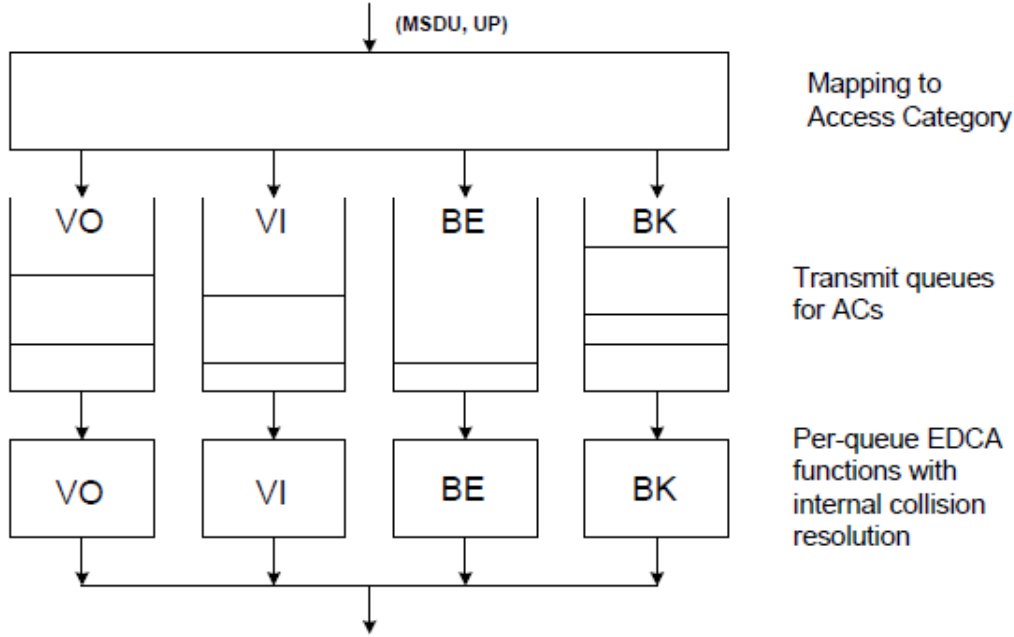


Figure 2.3: Reference standard EDCA architecture. From [17, p. 1378]

The standard foresees also an *Alternate EDCA* mode: when activated, six queues are used instead of four. This will result in two added queues (A_VI - Video (alternate), A_VO - Voice (Alternate)) and a different UP-queue mapping:

- UP=4 → A_VI queue - Video (alternate)
- UP=5 → VI queue - Video (primary)
- UP=6 → VO queue - Voice (primary)
- UP=7 → A_VO queue - Voice (alternate)

The alternate queues share the same EDCA functions as the VI and VO queues, as shown in figure 2.4, and the proper queue, between primary and alternate, is selected according to a scheduling function working above the EDCA mechanism. This scheduling function selects which SDU should be passed to the EDCA function below [17].

An EDCA Function (*EDCAF*) can be defined as follows (taken from IEEE 802.11-2016):

Definition 1. *Enhanced distributed channel access function (EDCAF): a logical function in quality-of-service (QoS) station (STA) that determines, using enhanced distributed channel access (EDCA), when a frame in the transmit queue with the associated access category (AC) is permitted to be transmitted via the wireless medium (WM). There is one EDCAF per AC.*

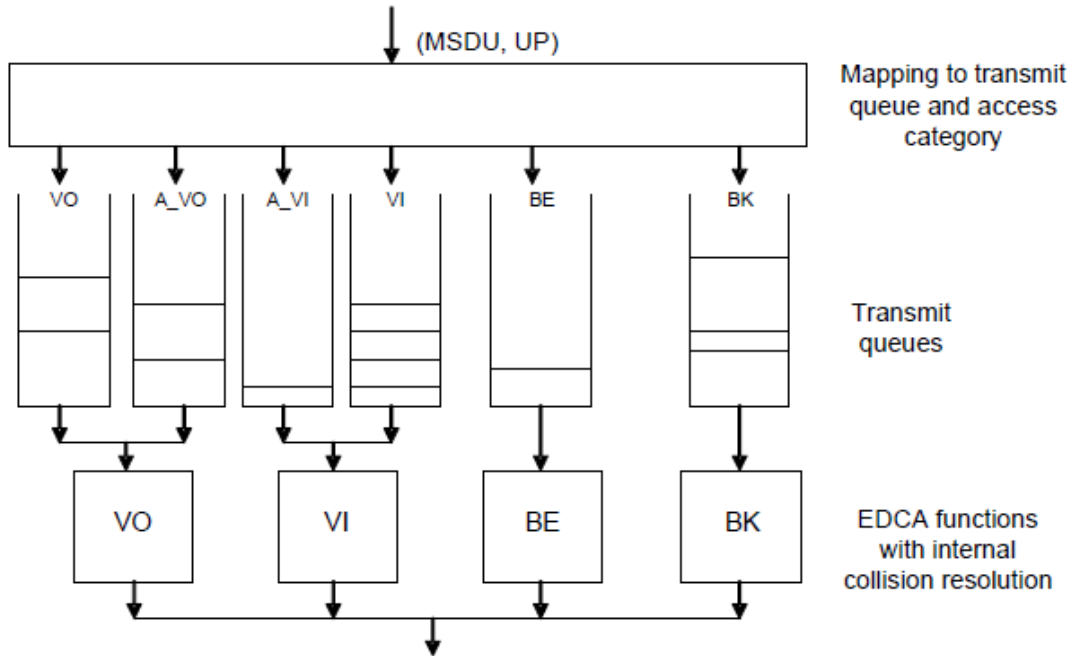


Figure 2.4: Reference Alternate EDCA architecture. From [17, p. 1378]

2.2.2 IEEE 1609.4 for multi-channel operations

The IEEE 1609.4 standard defines an "upper MAC layer", related to channel coordination, while channel access is already managed by 802.11.

IEEE 1609.4 defines, in fact, an extension to the IEEE 802.11 MAC layer.

Two types of channel are defined: service channels (SCH) and one control channel (CCH). As shown in figure 2.2, they use 10 MHz of bandwidth each and their IEEE channel numbers range from 172 to 184 (even numbers only), with 178 being the number of CCH.

The control channel is used for service advertisements (the “*WAVE Service Advertisements*”, transmitted using the WSMP protocol) and management data. It can carry only WSMP data.

All the other messages are transmitted on the service channels, which can be used to manage both WSMP and IPv6 data.

Since there is more than one channel, their access must be regulated. A “*MAC Sub-layer Management Entity*” (MLME) is defined: as reported in [30], “*this management entity provides the layer management service interface through which layer management functions may be invoked*”.

Three channel switching modes are defined by the standard, after dividing the time into slots, with a default duration of 50 ms:

- a. Continuous channel access: access is done continuously on the same channel, in an “always-on safety channel” mode.
- b. Alternating channel access: two different channels are accessed during two consecutive time slots. It can happen between CCH and SCH or between two SCHs.
- c. Immediate channel access: after receiving a WSA on the CCH, the device switches immediately to the SCH of interest, without respecting the slot boundaries. After it has completed its activity, the device can switch back to the initial channel.

The channel switching mode may depend on the final usage scenario of the system [30].

The channel routing (i.e. the selection of the proper channel and AC) is also defined by the standard. This is performed thanks to the definition of two MAC layer entities, one

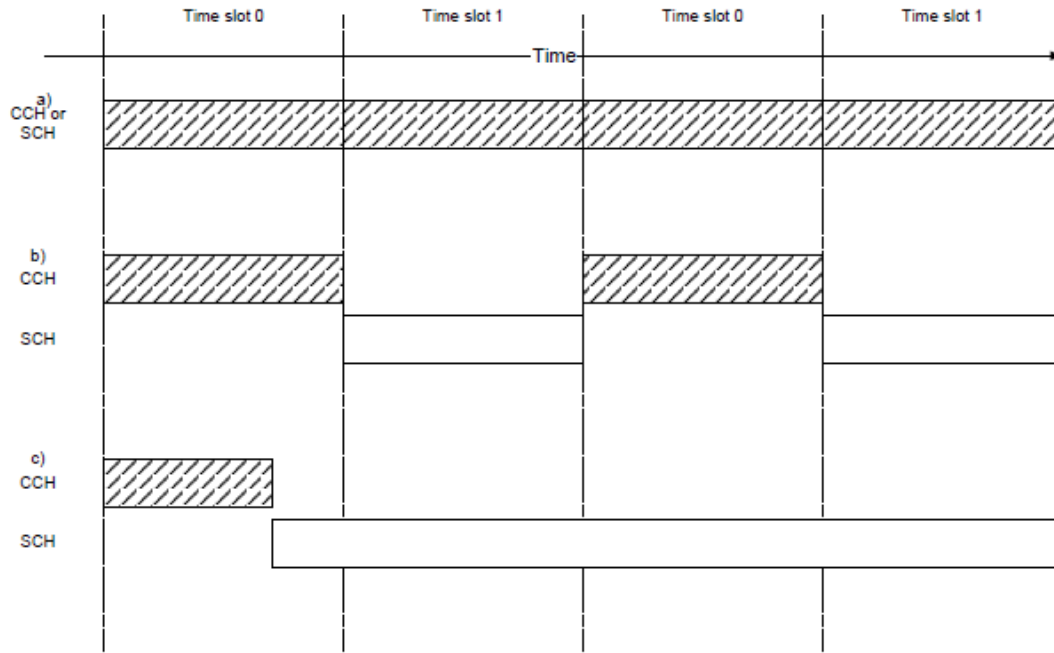


Figure 2.5: Channel access modes. From [30, p. 26]

related to the CCH and accepting only WSMP data and one related to SCHs, accepting IPv6 data too.

The choice of the proper entity and AC is performed depending on the user priority (0 to 7), which is mapped to an AC (BK to VO), and on the service the message belongs to. Looking at WSMP message, which can be transmitted on both SCHs and CCH, the channel is specified when a higher layer wants to transmit data: through a proper *Service Primitive* a *Channel Identifier* is given to the MAC layer and it is used for channel routing [30].

The conceptual internal WAVE MAC multi-channel architecture is shown in figure 2.6. This diagram, as reported in the standard, may depart from the actual implementation but it should be used as a reference as it facilitates the specifications of the of all the multi-channel and data prioritization operations.

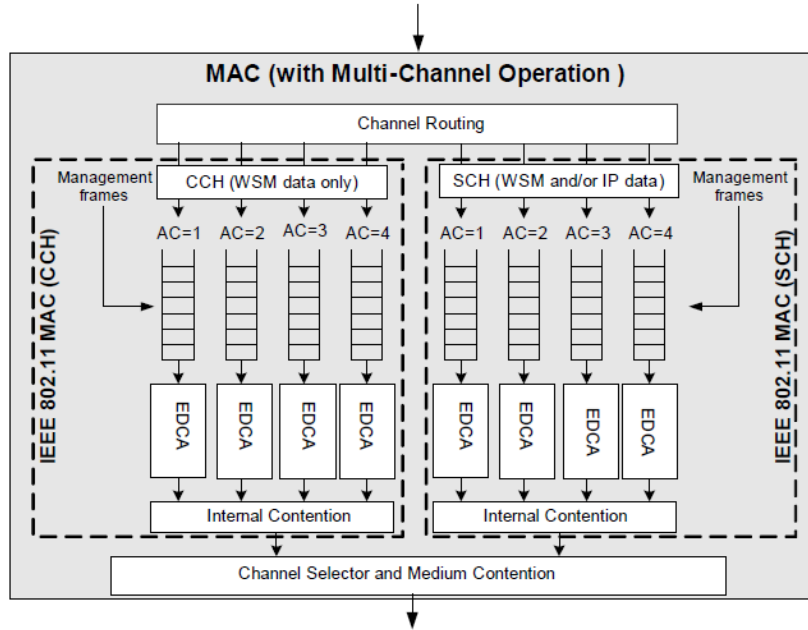


Figure 2.6: Conceptual transmit side MAC layer architecture. From [30, p. 17]

One last aspect to be considered is time synchronization, specified by IEEE 1609.4. Since there are now time slots that regulates the multi-channel usage, this aspect gains even more importance. Alternating channel access, for instance, should be synchronous, taking also into account that the message transmission should occur and complete within the allowed time slot, before a switch takes place (otherwise an SDU should be queued until the proper channel is activated again).

All the synchronization features are based on a common time reference, which is *UTC* (*Coordinated Universal Time*) time modulo 1 second, derived from any source able to provide it. For instance, GPS can be used for such a purpose, highlighting again the importance of having a positioning system as an essential component of VANETs.

The common time reference is derived from an UTC estimate, taking into account a possible tolerance and the estimate error standard deviation (for example by means of a Kalman Filter). The estimation, in a more complex implementation, can take into account the information received through *Time Advertisement* beacon frames that are exchanged

(as specified in IEEE 802.11-2016).

Due to time synchronization tolerance, radio switching effects and timing inaccuracies, each time slot starts with a guard interval.

When a device switches channel during a guard interval (as in the case of alternating channel access), it should consider the medium as busy during that interval and perform backoff; this is done to prevent multiple devices transmitting simultaneously at the end of the guard interval itself [30].

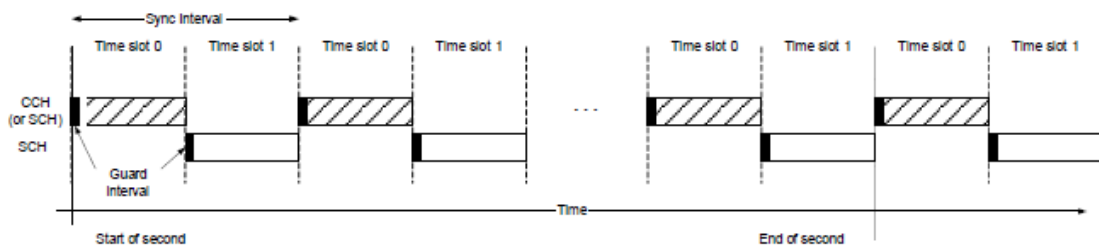


Figure 2.7: Time slots and guard intervals. From [30, p. 18]

Chapter 3

ALIX boards, UNEX DCMA 86P2, OpenWrt and Linux wireless subsystem

3.1 ALIX boards and development host

After an introduction about VANETs and IEEE WAVE protocols, the next chapters will be related to the implementation and integration work on the *PC Engines boards*, presenting the hardware, the tools used and the results obtained.

The boards presented in this chapter are the *ALIX 3d2* boards, which hardware characteristics are summarized below:

- CPU: AMD Geode LX800, x86 500 MHz processor designed for embedded applications
- RAM: 256 MB DDR DRAM
- Storage: Compact Flash (two 8 GB CF cards have been used: one MLC from Kingston and one SLC from PC Engines, ensuring a longer life for the card)

- Power: DC jack 7V to 20V
- Expansion: two miniPCI slots for up to two wireless cards and LPC “*Low Pin Count*” bus
- Connectivity: 1 Ethernet channel (Via VT6105M 10/100)
- Ports and I/O: 2 USB ports and a DB9 serial port
- Board size: 100 x 160 mm
- Firmware: *tinyBIOS*
- Other: 3 LEDs, optional RTC battery (not included in the boards used in this work)

One of the ALIX 3d2 boards is shown in figure 3.1. Two post-its have been attached to the two boards to distinguish them and have all the main parameters (such as their WLAN and Ethernet configured IP addresses).

In order to show all the main components of the ALIX boards, two figures are presented below:

It is possible to notice the following main hardware components:

1. AMD Geode LX800 x86 CPU
2. 256MB DRAM
3. First miniPCI slot with the UNEX DCMA 86P2 wireless card
4. From left to right: Ethernet connector (with a 10/100 Mbit/s Fast Ethernet Controller), DC power connector, RP-SMA antenna connector, DB9 serial port
5. Two USB ports on the other side of the board
6. AMD Geode CS5536 companion chip, optimized to work with the AMD Geode LX800 processor and providing controllers such as the ones for USB 2.0, IDE, AC97 audio, DMA, including real time clock features, Periodic Interval Timers (PIT) and



Figure 3.1: ALIX 3d2 board view from outside, with the enclosure mounted on. It is possible to distinguish the Ethernet port, the DB9 serial connector, a 50 ω impedance RP-SMA connector connected to the wireless card and the DC power connector.

many I/O functions; together with the AMD Geode CPU, it provides a system-level solution suitable for embedded boards [34]

7. LPC header
8. Second miniPCI slot to accommodate a second wireless card (it was not used during this work)
9. Compact Flash connector with a Kingston 8 GB MLC CF card inserted in; this card contains the openWRT Linux distribution that has been previously flashed using the host computer

The Ethernet port is used to connect the boards to the development host through the SSH protocol, practically allowing to run any shell command and command line program from the latter.

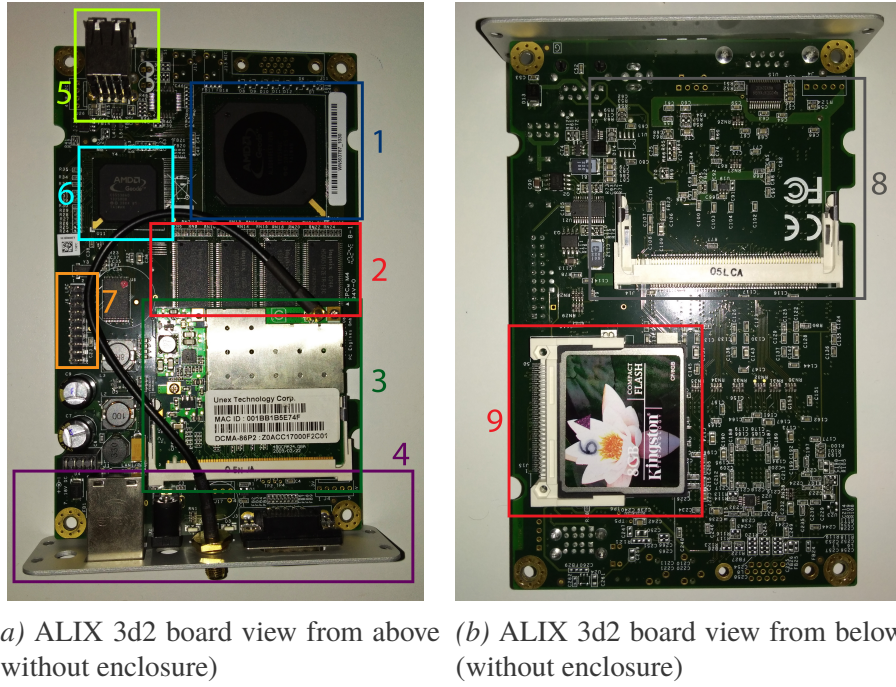


Figure 3.2: ALIX 3d2 main components

As host, an HP Envy j111sl notebook has been used, running Windows 10 as main operating system. Since a lot of work needed to be done on a host Linux distribution too (for example to cross-compile in a more practical way for the embedded boards), a virtual machine with Linux Mint 18.3 (64 bits) has been installed on a separate SSD, allowing to work in a flexible way both with Linux and Windows at the same time.

Of course this is just a personal configuration. Any other host PC configuration, for example with Linux only as main operating system, would allow to do almost the same operations described here, obtaining the same results.

As will be described later on, in order to be able to work with the boards configured for a different Ethernet subnet than the one used for Internet access, an ASUS USB to RJ45 Ethernet adapter (with 10/100 Mbit/s support) has been used to practically add a second Ethernet NIC to the development host.

This allowed to have both the boards and the local gateway to be connected to the host

computer, ensuring board connectivity and Internet connectivity at the same time, even when no local WiFi network was available (as it happened most of the time), without the need of changing the ALIX 3d2 configuration to match the local modem/router settings. This configuration is shown in figure 3.3.

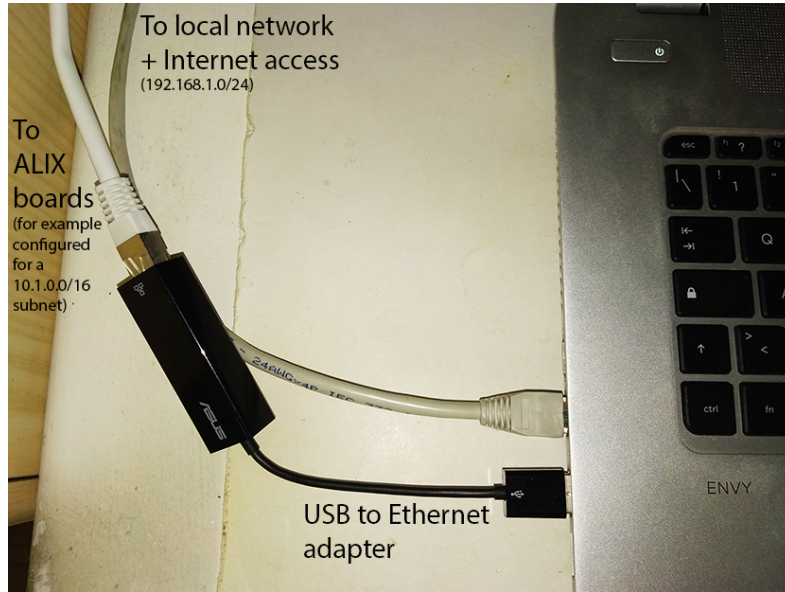


Figure 3.3: Ethernet connections on the development host.

Using two Ethernet NIC offered a flexible approach too, by giving the possibility to connect the boards to the local network (with Internet access) when needed, by just assigning Ethernet IP addresses in the range 192.168.1.0/24 and disconnecting the ASUS adapter configured for another subnet, and to any other subnet, by assigning different IPs to the Ethernet interfaces (for example in the private network range 10.1.0.0/16) and actually using the USB-RJ45 device configured for the same address range.

The range 192.168.1.0/24 mentioned above refers to the local network configuration that was present at the time of writing and implementing all the work on the boards. In case the same work has to be repeated, the correct local modem/router IP address range should be used instead of the one mentioned here.

3.2 Desk configuration and connections

Most of the work with the ALIX and APU boards has been developed working at home, after properly setting the desk and the surrounding area to work in the best way possible for all the development and testing work. Part of the work, though, was developed in Politecnico di Torino in a laboratory inside the DET department, in which WiFi Internet connectivity and an Ethernet switch were provided.

The configuration used at home can be seen in figure 3.4. After the first weeks, to work in a much more comfortable and polite way, some cables and the additional Ethernet switch have been moved to pass behind and under the desk and some electrical plugs have been added to connect the boards' power supply, keeping them farther away from the development PC. The connections, however, practically remain the same.

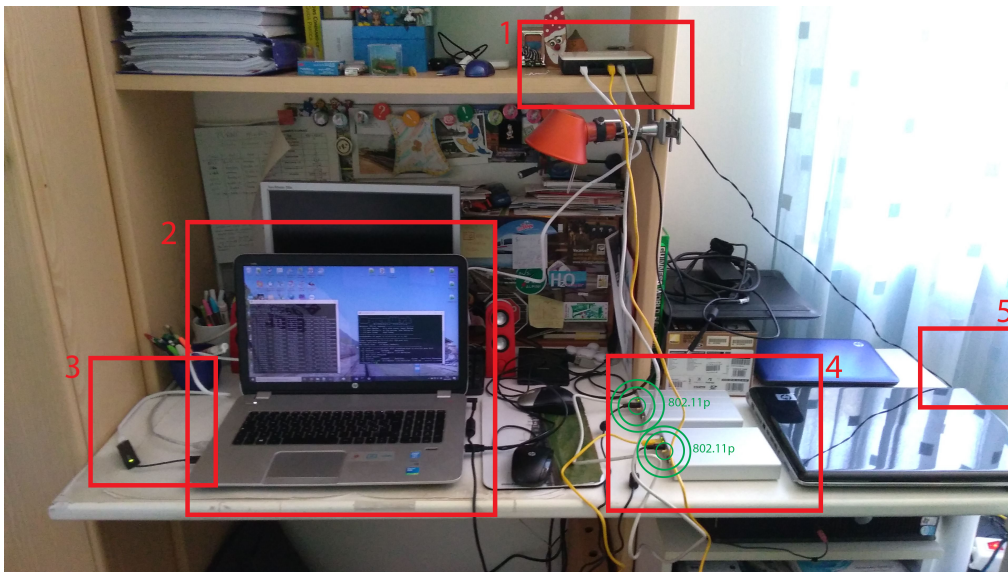


Figure 3.4: Desk configuration to work with the PC Engines boards.

It is possible to distinguish:

1. A *Sitecom* switch used to connect both the boards (supporting the connection of up to 4 boards) to the development host PC; it allows to connect and control the two

ALIX boards at the same time using two different SSH sessions

2. The development PC connected to the two boards
3. The Ethernet connections shown in figure 3.3
4. The two ALIX boards, communicating through 802.11p frequencies over the radio medium (conceptually represented by the green circles)
5. See figure 3.5 for a side view

In figure 3.5 a more recent side view of the desk is shown. As mentioned before, some cables now are put behind and under the desk and some plugs have been added on the wall. The boards are placed farther away from the development PC and the Ethernet switch connecting the two boards is now located inside the black box.

The power supply for the Ethernet switch connecting the boards can be located, as in the older desk configuration, in the power strip visible in the lower part of the photograph.

A second *Netgear* switch can be seen: it is the one connecting several devices to the local network with Internet access, including the category 5 cable labeled as “*To local network*” in figure 3.3.

When connecting the boards from their own subnet to the home network and to the



Figure 3.5: Side view of more recent desk configuration to work with the PC Engines boards.

Internet (without the use of the ASUS adapter described above), other than changing their IP addresses, one Ethernet cable is used to connect together the two switches, practically putting both the boards, connected to the Sitecom switch, inside the local network.

The connection to the modem/router, located in another room, happens through Powerline.

3.3 UNEX DCMA 86P2

The used ALIX boards are both equipped with an UNEX DCMA 86P2 V2X wireless card. This card, produced by the Taiwanese produced Unex, in conjunction with Wistron Corporation, is a device specifically designed for V2X applications, able to provide high power in the 5.850 to 5.925 GHz range, thanks to how it is designed and to additional hardware filtering.

Due to this, it should be best used in the DSCR range of frequencies, although also other 802.11a frequencies can be selected.

It is based on the Atheros *AR5414* dual-band multi-mode WLAN chipset, which is supported under the Linux operating system thanks to the *ath5k* driver, which will be mentioned multiple times later on in this work. This chipset supports all the most used modulations¹.

There are several research works, in literature, using this card [2] [6] [8] [24].

Now it seems to be out of production and more modern cards supported by newer and more stable drivers are available (such as the DHXA-222 used in conjunction with the APU boards), but it is still worth working with it, due to being apparently well-suited for DSRC communications and allowing us to compare V2X embedded devices, such as the old solutions with ALIX boards and the newer solutions with APU boards.

Both the ALIX boards and the DCMA 86P2 are not so recent, though; this fact leads for example to some IEEE WAVE MAC layer features being tested and some measurements

¹As described in the AR5414 chipset datasheet

being performed only on the newer APU boards, as detailed in chapter 5.

3.4 Used antennas

During all this work, we used indoor omni-directional antennas, with a gain of 5 dBi and an RP-SMA connector. They are targeted at both the 2.4 and 5 GHz frequency bands.

dBi is a relative power measurement unit, that measures the antenna gain with respect to an isotropic source, seen as an ideal “*perfect omnidirectional radiator*” [35].



Figure 3.6: Two 5 dBi omni-directional antennas

3.5 The OpenWrt Linux distribution

In order to properly work with the ALIX boards, an operating system is needed, providing the necessary basic services to the applications running over it.

These basic services actually include the CPU manager and the process scheduler, the memory and file manager, the network manager and all the device drivers needed to support external peripherals. The *ath5k* driver mentioned above refers to this part of the operating system, providing the support needed to use the UNEX card.

One important feature to look into the choice of operating systems for embedded applications like this is the support to *loadable kernel modules*, mainly when more complex operating systems are considered.

They are driver modules that can be loaded at run time inside the operating system kernel, without the need of recompiling the whole kernel every time (which, from personal experience, can take up to 2 hours when there are no object files which have been already compiled before).

These modules allow to work with the OS in a much more flexible way, with the disadvantage of introducing more easily, in flat architectures and monolithic kernels, bugs that can crash the whole system due to faulty loadable modules, that have direct access to the kernel space.

In this work both the loadable modules and the kernel recompilation methods are used, since most of the object files do not need to be recompiled every time, allowing a faster kernel recompilation with the advantages of producing standalone images as output.

The current use case requires then a strong basic support at least for all the standard 802.11 protocols: rewriting from scratch a complete PHY+MAC stack when there are already solid and well-maintained solutions would be very ineffective from this work point of view.

Considering that, embedded Linux distributions are chosen, since they actually provide support to loadable drivers, include several useful system programs for working with them, include a strong 802.11 support thanks to Atheros (i.e. *ath5k*, *ath9k* and *ath10k*) and Broadcom drivers which are constantly maintained and, most important, they are typically completely open source, which is a very important requirement in the wireless and network communications research field.

The most evident advantage, as reported in [36], is the flexibility open source software provides: there are already existing high quality commercial solutions, supporting the whole IEEE WAVE stack, as mentioned in chapter 1, but they are rather inflexible and it is typically difficult to modify their firmware to match the research and experimentation needs. With open source software it is instead much easier to extend the existing protocol stack with new features and updates, sharing then the obtained code and results with the

vehicular networking research community in an easy way².

All the Linux distributions are based on the Linux monolithic kernel, in which modules can be loaded by means of the “modprobe” program, which takes as input a kernel object file (.ko), containing the compiled driver.

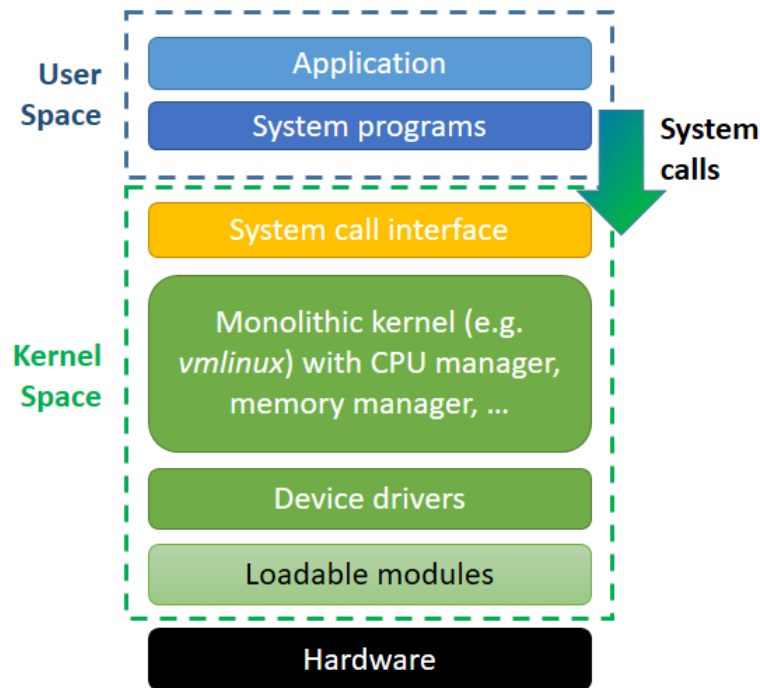


Figure 3.7: Monolithic kernel basic architecture

Having a monolithic kernel, in which the OS services are provided by a single executable (“vmlinux” or “vmlinuz” when it is compressed), is also a good compromise between OS safety (there is a good separation between the user and kernel space) and less performance overhead in the communication between user space and kernel space.

Between the various available embedded Linux distributions (including Voyage Linux, OpenWrt, and others), OpenWrt has been chosen as a base for this work.

²For example using the services provided by GitHub: <https://github.com/>

This does not mean that this is absolutely the best choice, but it was possible to find some advantages that led to OpenWrt in place of other distributions:

- It is a distribution especially targeted to embedded devices, very well suitable for routers, switches and other networking devices, as the ALIX boards used in the 802.11 communication context; it is used a lot, for example, as a replacement firmware for routers
- It supports a wide range of hardware devices, from ARMv8 processors to x86 ones [37]
- It has been used in a good number of research works in the past, and both the 2011 GCDC (*Grand Cooperative Driving Challenge*) and University of Bologna used it as a base for delivering vehicular communication services [6]
- The OpenC2X open source experimental and prototyping platform for vehicular communications, supporting all the ETSI ITS-G5 stack and being developed by a team in Paderborn University actually runs on standard Linux and there's a version based on LEDE, which was a branch of OpenWrt, now merged again into the main project; this platform with LEDE (and an updated version) will be used after the first tests with an older OpenWrt version [38]
- It allows a very good level of customization before (and after) building the system, allowing it to fit even in less powerful devices such as routers and to tailor to the own needs of the user; it can be fit in flash memories as low as 4 MB and with a minimum RAM of 32 MB [39]
- Their website³ documents in a quite complete and exhaustive way how it works and what steps to be followed to reach the desired goals

³<https://openwrt.org/docs/start>

In other works and use cases OpenWrt may be a worse choice if compared to other available embedded distributions. Voyage Linux, an x86 embedded distribution based on Debian, can be for instance another very good choice, as reported in [25].

The OpenWrt project was started in 2004. In 2016 LEDE was started as a branch of OpenWrt; in 2017 LEDE 17.01 was released but few months later the project was merged again to OpenWrt with the rules established by the LEDE development team (that is why, sometimes, it is possible to find “LEDE/OpenWrt” as project name), with the release of the OpenWrt 18.06 stable version [40] [41]. The latter has been released very recently at the time of writing (August 2018). All the latest OpenWrt releases are based on the fourth version of the Linux kernel.

The OpenWrt build process uses a custom build system, based on *Buildroot*, which is an open source build system for embedded Linux distributions, making all the process easier and generating Linux images through cross-compilation on a host development PC.

The build system comprises a set of patches and makefiles to build a complete embedded Linux environment, including generating a cross-compilation toolchain to compile programs for the target boards (such as the ALIX boards), creating a root file system, generating a bootloader and compiling the Linux kernel [42].

It is also based on the Linux Kconfig system, in which KConfig (*Kernel Configuration*) files (written in a special configuration language) are distributed in the build directory tree and are used by a kernel configurator to build the final system and check for dependencies between the different configuration options and included modules. Three states are possible for every configuration option:

- y: enable the option (for example to include a certain program directly into the kernel)
- n: do not enable the option
- m: build as a module (in this case the package is compiled, but it must be manually

installed inside the final system, if needed, with “`opkg update`”, followed by “`opkg install <package_name>`”, when an Internet connection is available)

Both a user interface based and a textual kernel configurators are made available to the user, respectively through the commands `make menuconfig` and `make config`.

`make menuconfig` displays an “ncurses”⁴ text version of the configurator and it is the most used command for building the kernel in this work [43].

3.5.1 The Linux wireless subsystem

The main components of any Linux wireless subsystem, which deals with the MAC and PHY layers of 802.11 standards, are schematized in figure 3.8 [44].

In this work the *mac80211* framework is considered: it is a framework for writing SoftMAC drivers.

SoftMAC refers to devices (i.e. WNICs) where the MLME is managed in software, allowing a better control of the hardware and the MAC frame management to happen directly in software.

Today, most of the WNIC devices are based on this architecture [45].

User applications and system programs are running over all the operating system services. Applications and system programs can access the general networking subsystem by means of proper system calls, opening for example sockets with `socket()` or using the “netlink” interface through *nl80211*, which provides a socket-based interface to transfer information between the user space and the kernel space, where the wireless subsystem lies.

It is important, before proceeding further, to note that there is a difference in the way in which management and data frames are handled.

⁴“ncurses” is a library enabling the creation of text-based user interface directly on terminal windows - url: <https://www.gnu.org/software/ncurses/ncurses.html>

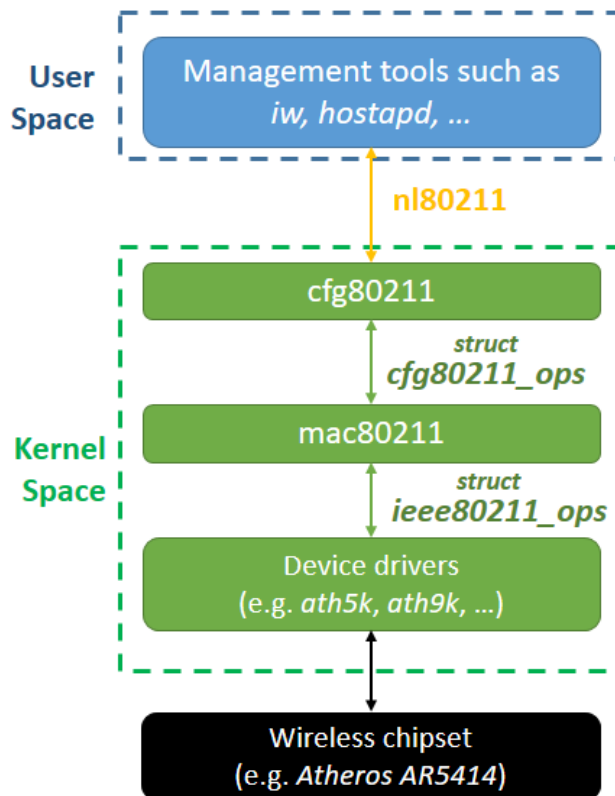


Figure 3.8: Linux 802.11 wireless framework main components, with SoftMAC drivers and devices

Management frames and configuration

This subsection deals on how management of wireless interfaces is performed in Linux based systems.

Wireless management tools such as *iw*, which is a command-line utility for wireless configuration and management, use the *nl80211* interface mentioned before to communicate with the “*cfg80211*” module.

The kernel modules are then the following:

- *cfg80211*: it provides a configuration API for 802.11 devices, bridging user space and drivers; in order to make any driver offer a consistent API this module must be

used by all the new wireless drivers written for Linux

- *mac80211*: it provides a framework for drivers for SoftMAC devices (making a driver API available); it depends on *cfg80211* for registration and configuration operations [45]
- Device drivers: they are the actual drivers providing access to the hardware (i.e. to the wireless chipset used to communicate at the PHY layer); examples include *ath5k* and *ath9k*

All the high level blocks shown in figure 3.8 actually support transparency, which means that they are as most as possible independent.

Introducing a modification, for instance, in the driver should not require any change to *mac80211* block. This is very useful, since patches can be introduced in the driver without any major or complex change in the *mac80211* module [46].

This is achieved thanks to C function pointers and callback functions, acting as handlers for the operations that can be performed on the wireless hardware through *mac80211*.

mac80211 provides to the device drivers a *struct ieee80211_ops*, containing a good number of function pointers, that must be filled by the lower level (i.e. by the driver) with the proper handlers for the various operations that can be performed on the wireless device. The *mac80211* block will then be able to invoke any proper handler without knowing its name and how it is implemented by the driver itself [46].

These handler are registered by the driver using a call to *ieee80211_alloc_hw()*, which is an *inline* function calling *ieee80211_alloc_hw_nm()* which effectively writes the information and the handlers passed by the driver to a *mac80211* local structure:

main.c

```
493 struct ieee80211_local *local;
```

To better clarify this concept, it is possible to look at an example.

For instance, *mac80211.h* defines *struct ieee80211_ops* containing several function pointers:

mac80211.h

```

3500 struct ieee80211_ops {
3501     void (*tx)(struct ieee80211_hw *hw,
3502               struct ieee80211_tx_control *control,
3503               struct sk_buff *skb);
3504     int (*start)(struct ieee80211_hw *hw);
3505     void (*stop)(struct ieee80211_hw *hw);
3506     #ifdef CONFIG_PM
3507     int (*suspend)(struct ieee80211_hw *hw, struct cfg80211_wowlan *
        wowlan);
3508     int (*resume)(struct ieee80211_hw *hw);
3509     void (*set_wakeup)(struct ieee80211_hw *hw, bool enabled);
3510     #endif
3511     int (*add_interface)(struct ieee80211_hw *hw,
3512                          struct ieee80211_vif *vif);
3513     int (*change_interface)(struct ieee80211_hw *hw,
3514                             struct ieee80211_vif *vif,
3515                             enum nl80211_iftype new_type, bool p2p);
3516     void (*remove_interface)(struct ieee80211_hw *hw,
3517                              struct ieee80211_vif *vif);
3518     int (*config)(struct ieee80211_hw *hw, u32 changed);
3519     void (*bss_info_changed)(struct ieee80211_hw *hw,
3520                              struct ieee80211_vif *vif,
3521                              struct ieee80211_bss_conf *info,
3522                              u32 changed);
3523
3524     ...

```

The *tx* function pointer has the following description (taken directly from *mac80211.h*):

“ * @tx: Handler that 802.11 module calls for each transmitted frame.

* *skb* contains the buffer starting from the IEEE 802.11 header.

* The low-level driver should send the frame out based on

* configuration in the TX control data. This handler should,

** preferably, never fail and stop queues appropriately.*

** Must be atomic.”*

mac80211 will actually call the handler registered at *tx* for transmitting frames, thus making this field one of the most important among the others.

ath5k, inside the *mac80211-ops.c* file, fills an *ieee80211_ops* structure with its own transmit handler:

mac80211-ops.c

```
782 const struct ieee80211_ops ath5k_hw_ops = {
783     .tx      = ath5k_tx,
```

ath5k_tx() is a function calling *ath5k_tx_queue()*, which is defined as follows inside the main *base.c* code base for the *ath5k* driver:

base.c

```
1610 void
1611 ath5k_tx_queue(struct ieee80211_hw *hw, struct sk_buff *skb,
1612     struct ath5k_txq *txq, struct ieee80211_tx_control *control)
```

This function is the transmission handler for the *ath5k* driver.

The driver then calls, for example in case of a WNIC connected to a PCI bus:

pci.c

```
250 hw = ieee80211_alloc_hw(sizeof(*ah), &ath5k_hw_ops);
```

This call will registers the handlers, including the *tx* one.

Every time a frame (both data or management, as it will be detailed later on) has to be transmitted on the wireless interface with a device supported by *ath5k*, *mac80211* will call, through a certain number of nested function calls, *drv_tx()* which calls itself:

driver-ops.h

```
36 local->ops->tx(&local->hw, control, skb);
```

effectively using the handler registered by the driver, which can be called hiding all the lower details to the current block.

A similar mechanism is used to link the *cfg80211* and *mac80211* blocks, with *mac80211* registering handlers to *cfg80211* through *struct cfg80211_ops*.

Tools such as *iw* use the *nl80211* library to communicate with the kernel space and with *cfg80211*. Then, *mac80211* is called to perform the needed configuration operations on the driver (which, of course, must support them).

Data frames

Data frames, transmitted through standard sockets, are instead managed in a different way, which is schematized in figure 3.9.

The transmission of data frames is briefly described here, as a meaningful example on how things work.

For what concerns the reception of packet, the inverse process occurs, either using polling or interrupts.

Inside the application a socket is typically created (either with or without broadcast permissions) specifying the type of socket and the protocol family.

System calls such as `socket()` are then used to pass the control to the kernel space. The socket layer will take care of managing the application created socket.

This block, which is “*protocol agnostic*”, will then pass the data to the proper *Network protocol* block, depending on what the user specified as protocol family during the socket creation. This block is responsible for parsing and decoding the packet header too [46].

The control is then given to the *net_dev* layer, linking network protocols to the desired hardware devices, including wireless chipsets.

A mechanism similar to the one described before is implemented, using a `struct net_device_ops` structure.

Concentrating on wireless transmissions, *mac80211* defines a structure:

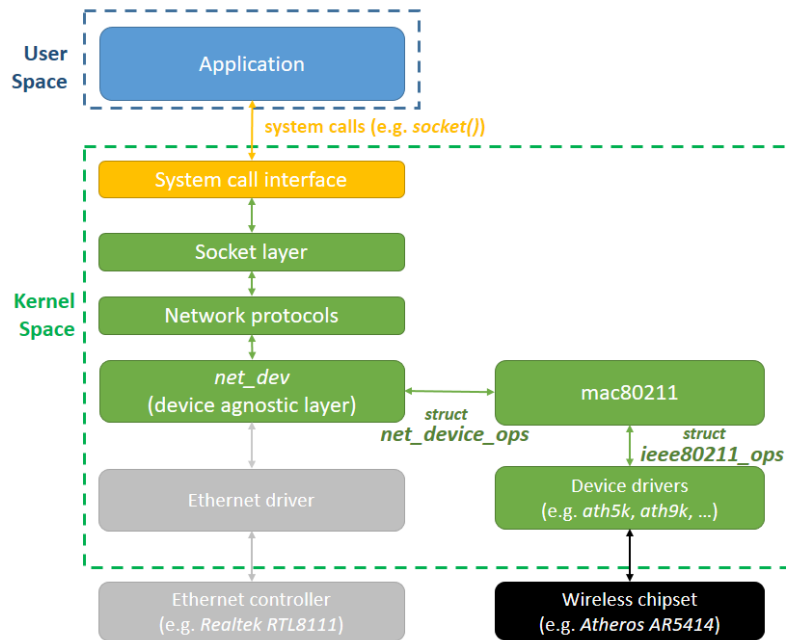


Figure 3.9: Linux networking structure when sending data frames over WiFi [46]

iface.c

```

1166 static const struct net_device_ops ieee80211_dataif_ops = {
1167     .ndo_open      = ieee80211_open,
1168     .ndo_stop      = ieee80211_stop,
1169     .ndo_uninit    = ieee80211_uninit,
1170     .ndo_start_xmit = ieee80211_subif_start_xmit,
1171     .ndo_set_rx_mode = ieee80211_set_multicast_list,
1172     .ndo_set_mac_address = ieee80211_change_mac,
1173     .ndo_select_queue = ieee80211_netdev_select_queue,
1174     .ndo_get_stats64 = ieee80211_get_stats64,
1175 };
  
```

used to register its handlers to *net_dev*.

The handler registration happens with a *mac80211* call to `register_netdevice()` (defined in *net_dev*), which accepts as argument a `struct net_device`, containing itself the structure defined above:

netdevice.h (in net_dev module)

```
1717 const struct net_device_ops *netdev_ops;
```

The registration function is called after another *net_dev* function is invoked, allocating and initializing the `struct net_device`; this function is also responsible for calling a void `(*setup)(struct net_device *)` *mac80211* callback, which sets the `netdev_ops` pointer to the `ieee80211_dataif_ops` structure.

This registration procedure, even if more complex, resembles then one described before between the driver and *mac80211*.

Then, every time a frame has to be transmitted using a wireless interface, the transmit handler `ieee80211_subif_start_xmit()` is called by *net_dev* to give control to *mac80211*, which will end up calling `drv_tx()` for data transmission, as described before.

Another useful handler which is worth being mentioned is *ndo_select_queue*, which is “called to decide which queue to use when device supports multiple transmit queues”.

Setting this with `ieee80211_netdev_select_queue` results in setting a wrapper to the `ieee80211_select_queue()` function of the WME *mac80211* module, which is responsible for choosing, in our case, the 802.11p EDCA queues.

Few notes on packet reception

According to an analysis work by Fred Chou on the Linux.com blog [46], which has been then directly verified on the Linux kernel source code, the reception of packets happens in an inverse way with respect to transmission.

When the hardware makes the system aware that a new frame has been received (either through polling or through interrupt), the driver performs some fast operation on the received data and passes it to *mac80211* for more complex processing, typically through a function called `ieee80211_rx()` or a similar variant.

Frames can then take two different ways:

1. Data frames are passed to *net_dev* and to higher level blocks, using `netif_receive_skb()` (called more than once inside the *rx* module of *mac80211*), which is also invoked by the Ethernet drivers to handle received messages. Unless specific options or a monitor interface are set, the frames seems to be converted to the 802.3 format (by means of `ieee80211_data_to_8023()` and `__ieee80211_data_to_8023()` calls in *rx.c*) before being passed to higher level modules.
2. Management frames, such as the ones used, for instance, in the authentication and association process with a BSS with infrastructure may instead be handled over to *cfg80211* and possibly sent to the user space through *nl80211*. Some management frames may also end their life inside *mac80211* [46].

One final note

One important final note about the Linux wireless subsystem is related to time-critical functionalities such as acknowledgments (when needed) and backoff timers: they are normally managed by the WNIC hardware and can be controlled through writing and reading the registers the device sets available to the developer.

3.6 Connecting with the boards

Two main software tools, under Microsoft Windows 10, has been used to connect with the boards and control them, launching tests, loading scripts and programs, and so on.

These tools have been used both for the ALIX and for the APU boards.

3.6.1 PuTTY

PuTTY is a free and open source SSH and Telnet client running on Microsoft Windows and Linux.

Since the boards, though OpenWrt and their Ethernet interface, are running an SSH

server (thanks to the *Dropbear* software package), it is possible to connect to them using PuTTY, in order to display a command line interface.

The command line can be then used to interact with the boards and the operating system, for example launching programs and shell scripts.

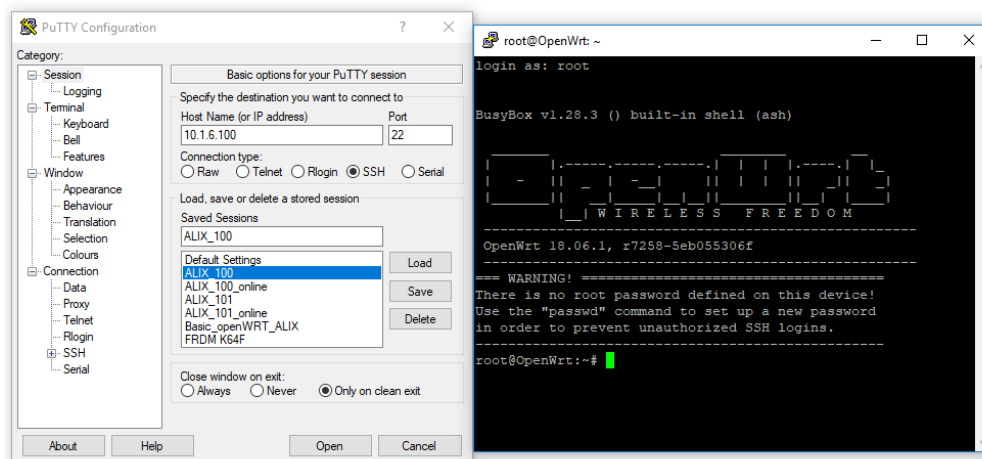


Figure 3.10: PuTTY interface before and after establishing an SSH connection with one of the ALIX boards, running OpenWrt

Through the *vi* command line text editor it is also possible to edit and create new files; however, since more advanced tools can be used to write files into the boards, this solution has been almost never used in this work.

In order to use SSH (after selecting the corresponding option), the IP address of the board has to be specified, together with port 22.

Sessions (with connection type + IP address + port) can be saved and quickly run again by means of a simple double click on the desired saved session.

3.6.2 WinSCP

In order to manage the files that are store inside the boards memories (either with the SquashFS or ext4 file systems) in a more comfortable way, the user can also rely on the SCP protocol, using SSH for a secure data transfer process.

This protocol, from the board side, is provided by the *dropbear* package, which is typically used as a lightweight SCP/SSH package in embedded systems.

From the development PC side a software tool able to support SCP is needed.

Under Microsoft Windows, the open source WinSCP program can be used. It provides support to FTP, SFTP, SCP, WebDAV and Amazon S3 protocols to securely transfer files between the local computer and a remote device⁵.

It offers a very practical user interface through which it is possible to access the boards memory storage almost as if the files were stored in the local PC; there's also the possibility of running more than one connection at the same time, seamlessly switching between the different connections by means of tabs.

Drag and drop operations are supported to copy the files from/to the remote device and the user can use a text editor of his choice to directly open and edit the files from within WinSCP.

In order to setup a connection, just like in PuTTY, the user must specify:

- The protocol (SCP)
- The port (22)
- The IP address of the SCP server running on the board (using the *eth0* interface address, not the wireless interface one)
- User name and password, if the user want to access the board in a faster way (specifying the password may have security implications, though)

Each configuration can be then saved for any future use.

Under Linux there are other alternative programs which can be used in place of WinSCP.

⁵Home page: <https://winscp.net/eng/docs/lang:it>

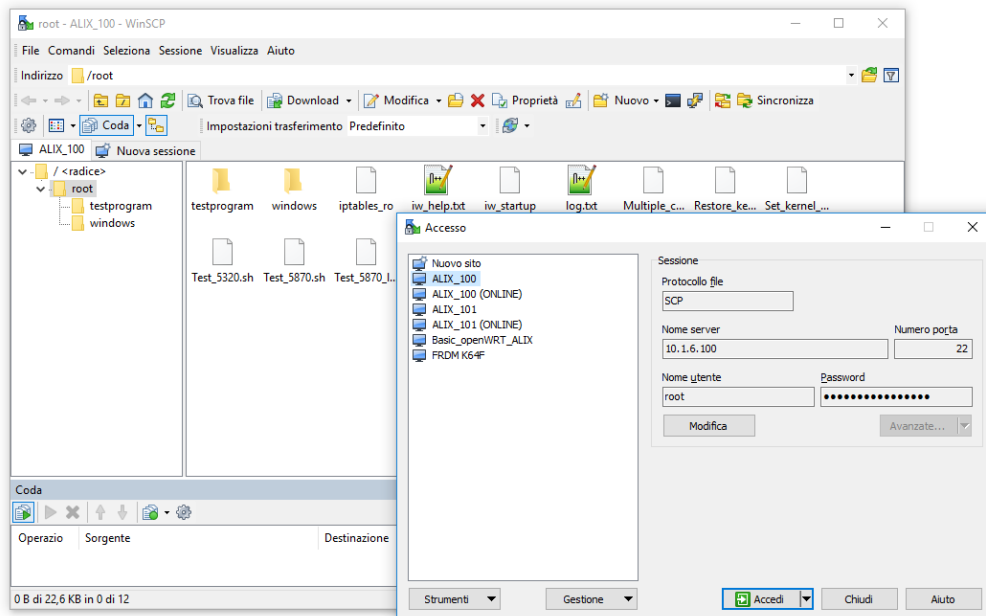


Figure 3.11: WinSCP new connection window and user interface

3.7 OpenWrt GCDC 2011 and network configuration

The first part of the work with the ALIX boards was to integrate an older OpenWrt distribution (version 10.03), patched for the GCDC 2011 competition and already used by *Università di Bologna* for some ITS related works and experimentations [6].

We were given directly a SquashFS + ext2 disk image to flash on the ALIX Compact Flash memory, containing the whole distribution with the 802.11p patch.

This patch was limited, however, to the 10 MHz channels at the ITS frequencies, allowing the selection of 802.11p channels in IBSS mode, which was used instead of the more proper OCB mode since no patches for OCB were available at the time.

The rest of the IEEE protocol was partly implemented, instead, by *Università di Bologna* in a separate Java code, using CALM-FAST libraries and a proper daemon in order to interface with the boards [6].

This older OpenWrt version was, in any case, useful to test the ALIX boards capabilities when trying to communicate over the ITS channels.

The patch introduced within this distribution was part of the 2011 GCDC competition, in which teams from different nations were challenged to create cooperative and autonomous driving solutions.

Both the 802.11p patch and the basic version of the Java program were released for this competition. *Università di Bologna* then added some custom classes to match their needs.

This version was also still based on the Linux “*Wireless extension*”, providing tools for wireless configuration such as `iwconfig`, which are now deprecated since the *mac80211* infrastructure, together with `iw`, should be now used by all drivers and Linux based wireless systems.

3.7.1 Configuration of the boards

The patched version of OpenWrt was flashed inside the two ALIX boards. Then, connecting one board only to the development PC through Ethernet, its default IP address, coming from a previous configuration, was found out.

The initial address was belonging to the 10.1.0.0/16 private address range, which was probably the one used internally by WiLAB, in *Università di Bologna* [6].

We decided to keep this network as a local network between the boards in the newer OpenWrt versions, too.

To detect the board default IP address, programs such as *Nmap* (possibly with the *Zenmap* GUI) can be used to scan the local network for connected devices, after configuring the PC Ethernet card to work on the proper subnet (in our case 10.1.0.0/16).

In order to configure the development host for connecting with the ALIX boards, when they are set to work in the 10.1.0.0/16 subnet, the following batch script has been written, affecting the Ethernet NIC corresponding to the Asus adapter (in our case it was assigned the *Ethernet 6* name):

Set_10_1_x_x_network.bat

```
1 @echo off
2 echo This file must be launched as administrator. If this is the case,
   press any key to continue.
3 pause
4 netsh interface ip set address "Ethernet 6" source=static address
   =10.1.11.60 mask=255.255.0.0 gateway=10.1.0.1 gwmetric=0
5 echo Ethernet 6 set to 10.1.0.0/24 subnet
6 pause
```

This script uses the *netsh* Windows utility to set the “Ethernet 6” card to use the following settings:

- IP address of the development host: 10.1.11.60
- Subnet mask: 255.255.0.0 (/16)
- Gateway (unused in our case): 10.1.0.1
- Gateway metric: can be left to 0

Writing a script instead of manually change the settings allows faster operations in case this configuration needs to be temporarily changed and then restored to the initial values.

Since any newly built OpenWrt image uses the 192.168.1.1 IP address as default one for the Ethernet interface, another similar script to properly configure the ASUS Ethernet to USB device was written:

Set_192_168_1_x_network.bat

```
1 @echo off
2 echo This file must be launched as administrator. If this is the case,
   press any key to continue.
3 pause
4 netsh interface ip set address "Ethernet 6" source=static address
   =192.168.1.199 mask=255.255.255.0 gateway=192.168.1.0 gwmetric=0
```

```
5 echo Ethernet 6 set to local 192.168.1.0/24 subnet
6 pause
```

The necessity of writing a second script was due to the fact that 192.168.1.1 is actually belonging to the 192.168.1.0/24 local subnet which is used at home (with Internet access), but it would be conflicting with the local modem/router address, which defaults at 192.168.1.1.

This configuration was temporarily used to access the boards running a newly built OpenWrt image and to change the address to a value inside the chosen range (i.e. 10.1.0.0/16).

However, newly built images will be used only later on, thus this script has been never used in conjunction with the OpenWrt GCDC 2011 builds.

Before configuring the boards, it was necessary to find out the root password, which is reported in [6]:

- Username: root
- Password: gcdc2011

Then, in order to configure the boards for the connection with the development PC, one file needed to be modified to set their Ethernet interface IP addresses.

This file is */etc/config/network*. It could be easily accessed through WinSCP and it was already partly configured by some projects in *Università di Bologna* [6].

We decided to give the boards specific IP addresses, which were then reported on two post-its applied to the boards, in order to easily recognize them. In particular:

- Board 1: *ALIX_100*⁶, with eth0 IP address: 10.1.6.100, wlan0 IP address: 10.10.6.100
(when connecting it to the local network with Internet access: eth0 address: 192.168.1.180)
- Board 2: *ALIX_101*, with eth0 IP address: 10.1.6.101, wlan0 IP address: 10.10.6.101
(when connecting it to the local network with Internet access: eth0 address: 192.168.1.181)

⁶Names are just given to recognize the boards, they do not play any role inside the integrated software and in any communication between the boards

The content of an */etc/config/network* file is reported below:

```
config 'interface' 'loopback'
    option 'ifname' 'lo'
    option 'proto' 'static'
    option 'ipaddr' '127.0.0.1'
    option 'netmask' '255.0.0.0'

config 'interface' 'lan_wilab'
    option 'ifname' 'eth0'
    # option 'type' 'bridge'
    option 'proto' 'static'
    option 'netmask' '255.255.0.0'
    option 'ipaddr' '10.1.6.100'
    option 'gateway' '10.1.0.1'
```

The only modification we needed to perform was related to changing the option `'ipaddr' '10.1.6.100'` line with the proper IP address (10.1.6.100 for the ALIX_100 board, and 10.1.6.101 for the ALIX_101 board).

Luca Nisi, in [6], had already changed some other options, commenting out the option `'type' 'bridge'` line and setting “proto” to “static”.

The other file that was actually already configured, but that needed a second check, was */etc/config/wireless*.

This file contains the board wireless configuration.

The ALIX_100 file is reported below:

```
config wifi-device radio0
    option type mac80211
    option channel 5
    option macaddr 00:1B:B1:B5:E7:4F
    option hwmode 11g

# REMOVE THIS LINE TO ENABLE WIFI:
option disabled 0
```

```
config wifi-iface
    option device      radio0
    option network     lan
    option mode        ap
    option ssid        WiLABALIX_43
    option encryption  none
```

After verifying that the line `option disabled` was set to 0 (setting this to 1 would disable the wireless interface), it was necessary to change `option macaddr` with the MAC address of the UNEX card (both for ALIX_100 and ALIX_101).

As a side note, it is possible to notice the line `option hwmode11g`. This option can be set to *11g* to use the 2.4 GHz range and to *11a* to use the 5 GHz range (even in the case of 802.11n or 802.11p).

This setting seems to be mismatching with the 802.11p frequencies, but the patch and the GCDC image settings actually take care of creating another wireless configuration using the 5 GHz frequency range, no matter what the user writes in the `/etc/config/wireless` file.

The MAC address can be found by using the `ifconfig` utility. Its output for the ALIX_100 board is reported below:

```
eth0      Link encap:Ethernet  HWaddr 00:0D:B9:1E:C0:AC
inet addr:10.1.6.100  Bcast:10.1.255.255  Mask:255.255.0.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:499  errors:0  dropped:0  overruns:0  frame:0
TX packets:313  errors:0  dropped:0  overruns:0  carrier:0
collisions:0  txqueuelen:1000
RX bytes:55582 (54.2 KiB)  TX bytes:59072 (57.6 KiB)
Interrupt:10  Base address:0xa000

lo        Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
UP LOOPBACK RUNNING  MTU:16436  Metric:1
```

```
RX packets:3 errors:0 dropped:0 overruns:0 frame:0
TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:279 (279.0 B)  TX bytes:279 (279.0 B)

wlan0      Link encap:Ethernet  HWaddr 00:1B:B1:B5:E7:4F
inet addr:10.10.6.100  Bcast:10.10.255.255  Mask:255.255.0.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

The MAC address is reported after wlan0, as HWaddr.

The boards are configured to run some GCDC specific services, by placing an initialization script inside */etc/init.d*:

gcdc

```
#!/bin/sh /etc/rc.common
# Copyright (C) 2008 OpenWrt.org

START=45

start() {
. /lib/gcdc/gcdc-init.sh
}

stop() {
. /lib/gcdc/gcdc-stop.sh
}
```

This script runs some additional scripts (on start and stop of the corresponding service) inside */lib/gcdc*, launching, among other things, the CALM daemon.

More important, a script placed in the root user folder, called *iw_startup* was configured to be run at startup.

This script performed all the configuration steps, with *iw* and *iwconfig*, needed to setup the ITS frequencies' communication.

With minor modifications to the original one (mainly related to setting the chosen IP address for the WLAN interface), the ALIX_100 *iw_startup* script is reported below:

iw_startup

```
1  #!/bin/sh
2
3  echo "Wifi Down"
4  wifi down
5  sleep 1
6  echo "Wifi Up"
7  wifi up
8  sleep 1
9  echo "Netherlands Frequency Register Set"
10 iw reg set NL
11 sleep 1
12 ifconfig wlan0 down
13 sleep 1
14 echo "Set Mode Ad-Hoc"
15 iwconfig wlan0 mode ad-hoc
16 ifconfig wlan0 up
17 sleep 1
18 echo "Set Frequency 5890 MHz"
19 iw dev wlan0 set freq 5890
20 iw dev wlan0 ibss leave
21 iw dev wlan0 ibss join ITS43 5890 fixed-freq ff:ff:ff:ff:ff:ff beacon
   2
22 echo "IP address"
23 ifconfig wlan0 10.10.6.100 netmask 255.255.0.0
24 echo "Set Rate 6M and Power 0dBm"
```



```
25 iwconfig wlan0 txpower 0
26 iwconfig wlan0 rate 6M
```

The script is described below:

1. It starts turning off and on the wifi network to avoid possible problems due to previous configurations (this step may also be avoided in most cases, provided that the wireless interface is already up).
2. Through the `iw` utility it is possible to set the wireless regulatory domain, which, thanks to a built-in database, defines the law limits for transmit power and frequency ranges in each country.

There is, in fact, a *Central Regulatory Domain Agent (CRDA)*, which is a Linux tool “for communication between the kernel and userspace for regulatory compliance” [45], used to set a certain regulatory domain, which has the purpose to check whether the current settings are compliant with the current country, set using `iw reg set <code>`.

It relies on the *wireless-regdb* database, containing, for each country, the allowed transmit power, frequencies, maximum antenna gain, and so on.

In order to display the current regulatory domain, the user can rely on the `iw reg get` command.

To set a different regulatory domain, `iw reg set <code>` is used, specifying the desired country (US seems to be selected as default by OpenWrt).

Since the Italian regulatory domain is not updated to include the ITS frequencies, nor was the US one, the GCDC 2011 image used the NL (Netherlands) flag, which was patched to include the 802.11p frequencies as allowed ones (up to 27 dBm, without antenna gain limit, with mandatory DFS to recognized interference from radars and automatically change channel).

3. It turns down the `wlan0` wireless interface to change its mode

4. It sets the wireless interface mode to *Ad-hoc* (OCB mode was not yet available), corresponding to IBSS.
5. It turns up again the *wlan0* wireless interface.
6. It leaves a possible IBSS (to ensure no other IBSS is active when joining a new IBSS).
7. It joins an IBSS, with name (ESSID) “ITS43” (any other name should be fine), fixed frequency 5890 MHz (CCH - it can be changed later on), BSSID equal to the broadcast MAC address (all binary ones) and beaconing period of 2 *ms* (this can be specified thanks to a proper *iw* patch included in the GCDC 2011 image).
8. Using *ifconfig* the WLAN IP address and netmask are set.
9. We then decided to start with a small transmit power (0 dBm), increasing it only when necessary. When the boards are one near to each other, however, they are completely able to communicate. So, 0 dBm is set as initial “*txpower*”.
10. The last line is used to set the target bitrate the UNEX cards will use. Since there was no native support for 802.11p rates and everything was working thanks to a patch, the user has to specify a double rate (as if 802.11a was used instead of 802.11p) with respect to the desired one. For instance, *iwconfig wlan0 rate 6M* was used to set the 3 *Mbit/s* 802.11p mode.

The ALIX_101 board included the same script, with a different WLAN IP address:

```
ifconfig wlan0 10.10.6.101 netmask 255.255.0.0
```

In order to display the current wireless configuration, it is possible to use *iwconfig*.

Since the script is run at startup, when ready the boards are already configured to transmit on the 10 MHz wide CCH:

iwconfig output after startup

```
lo          no wireless extensions.
```

```
eth0      no wireless extensions.

wlan0      IEEE 802.11a  ESSID:"ITS43"
Mode:Ad-Hoc  Frequency:5.89 GHz  Cell: Invalid
Tx-Power=0 dBm
RTS thr:off   Fragment thr:off
Encryption key:off
Power Management:off
```

The user can turn off the boards with `poweroff` (although it seems to be bugged in this OpenWrt version, practically breaking the PuTTY SSH connection but not turning off the whole operating system) and reboot them with `reboot`.

Some configuration parameters can also be easily changed; in particular:

- In order to change the transmission power, it is possible to use `iwconfig wlan0 txpower <power in dBm, greater or equal than 0>`
- To change the desired bitrate: `iwconfig wlan0 rate <desired bitrate x 2>`
- To change frequency (i.e. to change channel): `iw dev wlan0 ibss leave` followed by `iw dev wlan0 ibss join ITS43 <central frequency in MHz> fixed-freq ff:ff:ff:ff:ff:ff beacon 2`; we were not able to set, however, the standard 802.11a frequencies, other than the ITS ones, probably due to how the patch works

As an additional note, it can be useful to point out that any new script copied to the board should be made executable, by moving to its directory with `cd` and executing `chmod +x <script name>`.

3.7.2 iPerf

The `iperf` tool allows the user to measure the network throughput (called “bandwidth” inside the program), by means of a server running on one board and a client running on the other board.

This measurement tool will be used also with the newer OpenWrt versions.

It is a cross-platform tool, able to work with different layer 4 protocols, such as UDP, TCP and SCTP; it supports multiple simultaneous connections, it can be configured to run for a certain time or after a certain amount of data has been sent, possibly printing periodic reports at configurable intervals.

It is able to measure the network throughput, the packet loss and the jitter [47].

A complete rewrite of the program, by the same development team, is available under the name `iperf3`, representing the latest *iPerf* version. This program did not exist at the time of the 2011 GCDC competition, thus it is not available in the corresponding OpenWrt version.

The UDP based measurements will be used to test the throughput achievable by the boards; after capturing some packets from a client-server communication using *iPerf*, the packet loss measurement seems to be performed thanks to the UDP payload, in which some sequential numbers are fit.

This is confirmed looking at the *iPerf* source code:

Reporter.c

```
1113 // packet loss occurred if the datagram numbers aren't sequential
1114 if ( packet->packetID != data->PacketID + 1 ) {
1115     if (packet->packetID < data->PacketID + 1 ) {
1116         data->cntOutOfOrder++;
1117     } else {
1118         data->cntError += packet->packetID - data->PacketID - 1;
1119     }
1120 }
```

Sniffing happened through the `tcpdump` tool, which is better described when looking at the newer OpenWrt versions (in section 3.8.6).

In order to check if the boards are able to communicate with each other on the CCH, an *iPerf* server has been run on one board (ALIX_100) and an *iPerf* client has been run on the other board (ALIX_101).

Both the boards were running with the initial settings of 0 dBm transmit power and mode 1 (3 *Mbit/s*).

The used commands are the following:

- `iperf -s -u -i 2`, to start a server (-s), set the use of UDP datagrams (-u), trigger periodic reports for throughput, jitter and packet loss every 2 seconds (-i 2); since no datagram size is specified (with the -l option), 1470 B datagrams, as default size, will be expected; the port to be used can be specified by “-u <port number>” (default: port 5001).
- `iperf -c 10.10.6.100 -u -t 120 -i 2 -b 10M`, to start a client connecting to the server with IP address 10.10.6.100 (-c 10.10.6.100), using UDP (-u), running for 120 seconds (-t 120), with periodic interval every 2 seconds (-i 2) and specifying a target UDP bandwidth to send at equal to 10 *Mbit/s* (much more than what is achievable, to really test the obtainable throughput in mode 1); as it is not specified, 1470 B UDP datagrams will be sent; the port to be used can be specified by “-u <port number>” (default: port 5001).

The client side report is reported here:

log_iperf_client_gcdc.txt

```
-----  
Client connecting to 10.10.6.100, UDP port 5001  
Sending 1470 byte datagrams  
UDP buffer size: 106 KByte (default)  
-----
```

```
[ 3] local 10.10.6.101 port 43574 connected with 10.10.6.100 port
5001
```

[ID]	Interval	Transfer	Bandwidth
[3]	0.0- 2.0 sec	622 KBytes	2.55 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	2.0- 4.0 sec	508 KBytes	2.08 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	4.0- 6.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	6.0- 8.0 sec	512 KBytes	2.10 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	8.0-10.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	10.0-12.0 sec	553 KBytes	2.26 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	12.0-14.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	14.0-16.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	16.0-18.0 sec	505 KBytes	2.07 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	18.0-20.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	20.0-22.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	22.0-24.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	24.0-26.0 sec	510 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	26.0-28.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	28.0-30.0 sec	554 KBytes	2.27 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	30.0-32.0 sec	510 KBytes	2.09 Mbits/sec

[ID]	Interval	Transfer	Bandwidth
[3]	32.0-34.0 sec	560 KBytes	2.29 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	34.0-36.0 sec	510 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	36.0-38.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	38.0-40.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	40.0-42.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	42.0-44.0 sec	553 KBytes	2.26 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	44.0-46.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	46.0-48.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	48.0-50.0 sec	508 KBytes	2.08 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	50.0-52.0 sec	558 KBytes	2.29 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	52.0-54.0 sec	507 KBytes	2.08 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	54.0-56.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	56.0-58.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	58.0-60.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	60.0-62.0 sec	554 KBytes	2.27 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	62.0-64.0 sec	510 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	64.0-66.0 sec	560 KBytes	2.29 Mbits/sec

[ID]	Interval	Transfer	Bandwidth
[3]	66.0-68.0 sec	508 KBytes	2.08 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	68.0-70.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	70.0-72.0 sec	510 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	72.0-74.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	74.0-76.0 sec	518 KBytes	2.12 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	76.0-78.0 sec	554 KBytes	2.27 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	78.0-80.0 sec	512 KBytes	2.10 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	80.0-82.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	82.0-84.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	84.0-86.0 sec	514 KBytes	2.11 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	86.0-88.0 sec	554 KBytes	2.27 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	88.0-90.0 sec	508 KBytes	2.08 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	90.0-92.0 sec	557 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	92.0-94.0 sec	556 KBytes	2.28 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	94.0-96.0 sec	511 KBytes	2.09 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	96.0-98.0 sec	561 KBytes	2.30 Mbits/sec
[ID]	Interval	Transfer	Bandwidth
[3]	98.0-100.0 sec	511 KBytes	2.09 Mbits/sec


```

[ ID] Interval          Transfer          Bandwidth
[  3] 100.0-102.0 sec    557 KBytes    2.28 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 102.0-104.0 sec    510 KBytes    2.09 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 104.0-106.0 sec    556 KBytes    2.28 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 106.0-108.0 sec    554 KBytes    2.27 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 108.0-110.0 sec    510 KBytes    2.09 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 110.0-112.0 sec    558 KBytes    2.29 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 112.0-114.0 sec    512 KBytes    2.10 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 114.0-116.0 sec    556 KBytes    2.28 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 116.0-118.0 sec    511 KBytes    2.09 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3] 118.0-120.0 sec    554 KBytes    2.27 Mbits/sec
[ ID] Interval          Transfer          Bandwidth
[  3]  0.0-120.1 sec   31.5 MBytes    2.20 Mbits/sec
[  3] Sent 22461 datagrams
[  3] Server Report:
[ ID] Interval          Transfer          Bandwidth          Jitter    Lost/Total
      Datagrams
[  3]  0.0-120.2 sec   31.5 MBytes    2.20 Mbits/sec    15.954 ms
      1/22462 (0.0045%)

```

The client and the server running side by side are instead shown in figure 3.12.

As it is possible to see, very few frames are lost (1 over 22462, 0.0045%), which is in line with the results obtained in [6]. The jitter, computed by *iPerf* with continuous measurements (thanks to a “64 bit second/microsecond timestamp” recorded by the client

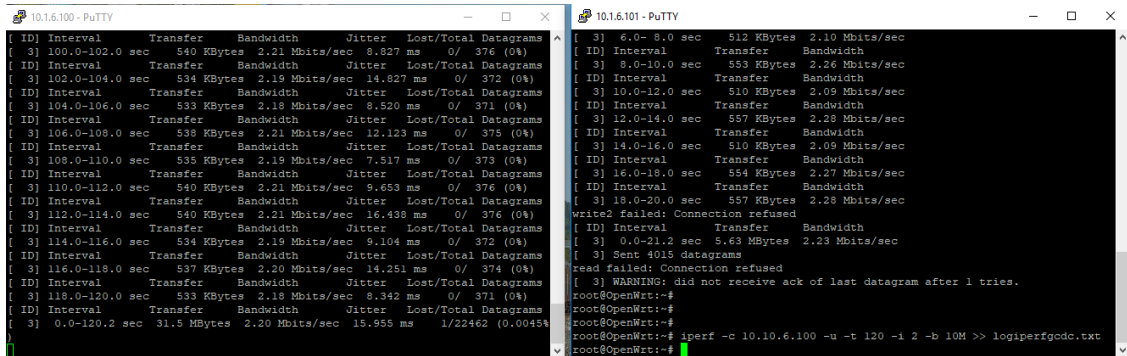


Figure 3.12: *iPerf* server running on one ALIX board (on the left) and *iPerf* client running on the other board (on the right, with output redirection to a log file)

[47]) and a final smoothed mean, is around 15 ms.

These results are obtained transferring around 550 KB every 2 seconds, with an achievable bandwidth of 2.20 *Mbit/s* on IBSS mode, which is quite good with respect to the theoretical value of 3 *Mbit/s*.

Trying to change channel on one board only, the communication could no more happen, as expected. Changing the channel to both boards (for example setting both the boards to work at 5.650 GHz) the communication could happen again with very similar results.

If one board, without any installed antenna, then, was put inside a cupboard while running *iPerf* tests, the communication was interrupted, proving the fact that with 0 dBm transmit power communication can happen only as long as the boards are quite near to each other, without major obstacles in between. Installing the antenna, as expected the maximum distance between the boards increased.

It was also possible to check the power received by one board with respect to another board. With 0 dBm, no antennas, and the boards placed very near to each other, it was possible to measure a received power of -76 dBm, while, increasing the effective transmission power to 24 dBm, the received power increased to -49 dBm. More detailed and systematic measurements will be taken with the newer platforms.

In order to view the power one board receives with respect to a second board, we used

the following command:

```
iw dev wlan0 station get "00:1B:B1:B5:E6:BE"
```

launched on ALIX_100 (running an *iPerf* server) with the ALIX_101 UNEX card MAC address (running an *iPerf* client).

3.7.3 Testing the VTL Java code

The next step was to quickly test the VTL Java code from [4], using the CALM daemon to interface with the boards, as a possibility to deliver advanced services with 802.11p programmed boards.

This code is able both to implement a VTL system for field tests and to implement simulations, with virtual coordinates. We actually tested it using the second approach.

Few modifications needed to be performed to the code, in order to make it accept the IP addresses of the boards, which are different than the ones accepted for the GCDC competition.

To test the VTL algorithm with the boards we had, we launched two instances of the Eclipse program, running on two different workspaces. Each is working with its own version of the code, which remains almost the same except for few small modifications.

The modifications are reported below, with the edited lines highlighted by comments:

it.unibo.trafficlights - VirtualTrafficLightVisualizer.java (ALIX_100 and ALIX_101)

```
42 public VirtualTrafficLightVisualizer(int positionOnScreen, String
    vehicleName) {
43     if(positionOnScreen==0) {
44         vtlActive = false;
45         return;
46     } else {
47         vtlActive = true;
48     }
49     previousColour=-1;
50     imgFrameVTL = new JFrame(vehicleName);
```

```

51  imgFrameVTL.setSize(400, 400);
52  this.ModifyPosition(positionOnScreen);
53
54  if(vehicleName.contains("0")) { // Edited to enable ALIX_100 IP
        address
55      imgFrameVTL.setBackground(Color.GRAY);
56  } else if(vehicleName.contains("37")) {
57      imgFrameVTL.setBackground(Color.RED);
58  } else if(vehicleName.contains("1")) { // Edited to enable ALIX_101
        IP address
59      imgFrameVTL.setBackground(Color.BLUE);
60  } else if(vehicleName.contains("35")) {
61      imgFrameVTL.setBackground(Color.WHITE);
62  } else if(vehicleName.contains("41")) {
63      imgFrameVTL.setBackground(Color.GREEN);
64  } else if(vehicleName.contains("45")) {
65      imgFrameVTL.setBackground(Color.PINK);
66  } else if(vehicleName.contains("49")) {
67      imgFrameVTL.setBackground(Color.YELLOW);
68  } else if(vehicleName.contains("47")) {
69      imgFrameVTL.setBackground(Color.BLACK);
70  } else {
71      CustomizedError.printError("Id veicolo non previsto: " +
        vehicleName);
72  }

```

it.unibo.communications - Main.java (ALIX_100)

```

31  /*****
32  * Settings
33  */
34  static private int deviceID = 00; // Edited by to support ALIX_100
35  //deviceID: choose # if you are using ALIX#, e.g. choose 35 if you are
        using ALIX35

```

```

36 static private int application = APP_VIRTUALTRAFFICLIGHT; //
    APP_VIRTUALTRAFFICLIGHT;
37 private static boolean useOfGPS = true;
38 private static boolean useOfMaps = true;
39 public static String city="Bologna";
40 private static boolean isRSU=false;

```

it.unibo.communications - Main.java (ALIX_101)

```

31 /*****
32 * Settings
33 */
34 static private int deviceID = 01; // Edited by to support ALIX_101
35 //deviceID: choose # if you are using ALIX#, e.g. choose 35 if you are
    using ALIX35
36 static private int application = APP_VIRTUALTRAFFICLIGHT; //
    APP_VIRTUALTRAFFICLIGHT;
37 private static boolean useOfGPS = true;
38 private static boolean useOfMaps = true;
39 public static String city="Bologna";
40 private static boolean isRSU=false;

```

it.unibo.communications - Main.java (ALIX_100 and ALIX_101)

```

170 public static void main(String[] args) throws Exception
171 {
172 new Config();
173
174 startingTime=System.currentTimeMillis();
175
176 // Setting of the device name
177 deviceName = "10.1.6.1" + String.format("%02d",getDeviceID()); //
    Edited to support our ALIX boards
178 System.out.println(deviceName);

```

The *deviceName* is used to set up a socket to connect with the boards.

It was then necessary to edit the *config.cfg* file, setting the virtual coordinates for simulating two cars reaching a VTL intersection. The two cars have been put near to an intersection by changing the respective *startingLat* and *startingLong* parameters (lines 23-24 of the configuration file). The *deviceId* line had to be changed too, setting it to 00 for ALIX_100 and to 01 for ALIX_101, since this value actually overrides the ones specified inside *Main.java*.

After running two instances of the Java programs, connected to the two ALIX boards, it was possible to see that the boards were correctly communicating using 802.11p: one board, simulating one OBU, received a green signal, while the other a red one, as shown in figure 3.13.

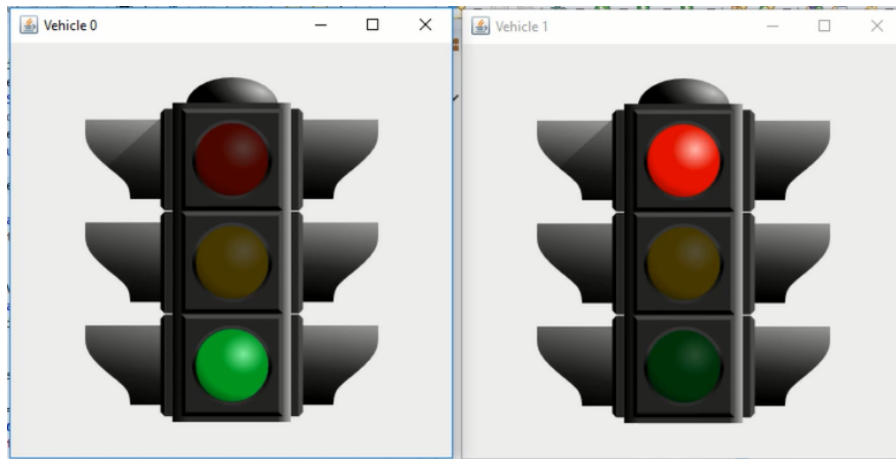


Figure 3.13: VTL simulation with the Java code connected to the boards: one simulated car (simulated on ALIX_100) getting green traffic light, the other (on ALIX_101) getting red traffic light

After the first one passed the intersection, the second one was properly notified and received a green semaphore too, as it is possible to see from the following logs:

Log from the Java code connected to the ALIX_100 board

```
.....  
Time: 00:00:28.618  
Time: 00:00:28.718
```

```
Time: 00:00:28.732, VTL message GREEN sent to 1.0 (with numVhc=2)
Time: 00:00:28.733, VTL message GREEN sent to 1.0 (with numVhc=2)
Time: 00:00:28.733, VTL message GREEN sent to 1.0 (with numVhc=2)
Time: 00:00:28.819
Time: 00:00:28.919
.....
```

Log from the Java code connected to the ALIX_101 board

```
.....
VTL: Threshold not yet exceeded
Time: 00:00:19.683
VTL: I set ORANGE
VTL: I set RED.
Time: 00:00:19.783
.....
VTL message received: GREEN (with numVhc=2.0)
VTL: HO RICEVUTO IL VERDE!!!! URL: fast://0x001bb1b5e74f:5
VTL: I set GREEN
VTL message received: GREEN (with numVhc=2.0)
VTL: I IGNORE A GREEN MESSAGE. I AM ALREADY GREEN!!!
Time: 00:00:25.709
VTL message received: GREEN (with numVhc=2.0)
VTL: I IGNORE A GREEN MESSAGE. I AM ALREADY GREEN!!!
Time: 00:00:25.810
.....
```

3.7.4 Moving to newer versions of OpenWrt

The previous sections show how the patched OpenWrt version can already be a quite stable solution for building and evaluating even complex applications, such as a VTL system, provided that some additional IEEE 802.11p features, other than the physical layer ones provided by the patch, are implemented in an external application.

There are, however, several drawbacks that lead us to select more updated solutions, based on newer versions of OpenWrt and on the version 4 of the Linux kernel:

- The 2011 GCDC images are based on an older version of the Linux kernel (2.6.32), which is actually missing over 7 years of updates and bug-fixes, thus being also more difficult to update as newer versions of both the operating system and the kernel come out. The capability of updating the software may play an important role in future connected vehicles, which may run Linux-based software solutions.
- As mentioned before, everything is working thanks to a GCDC patch, which is not officially included inside the OpenWrt distributions. This represented a valid solution, but it was limited to the 10 MHz channels at the 802.11p frequencies: other features were not fully included and/or supported and needed to be performed by an additional application. They could also have been coded with an additional patching work, but, in any case, it would have been related to an old OpenWrt version, with difficulties in porting it to more updated distributions (which may already include some of the desired features too), increasing the number of patches, which may be conflicting with new kernel features, at some point.
- Due to the patch, it seemed as if non-ITS frequencies, at least with a UNEX DCMA 86P2 card installed in, were no more selectable, limiting a bit the flexibility of the system.
- It was not possible, even though various different tests, to make the EDCA queues work; this was instead possible with newer platforms.
- The newer versions of OpenWrt and the Linux kernel seem to already integrate several 802.11p features introduced by projects such as the *CTU-IIG 802.11p-linux* one [48], including the OCB mode, which, even if it is similar to the IBSS architecture used in the GCDC patch, is the right mode established by the standard (as described in chapter 2).

- GCDC patches were mainly based on the *ath5k* driver. However, now there are several wireless cards supporting ITS frequencies which are supported by the *ath9k* driver. Moreover, as reported in [25], now many cards are based on the PCIe interface (other than the mini-PCI one used in the ALIX boards): this means moving from *ath5k* to *ath9k* and other drivers (at least for Atheros chipsets).
- Newer patches could provide a better support for GeoNetworking implementations, as reported in [25].
- Finally, using newer versions of the software platform allows the user to access new packages as they are introduced, together with newer versions of the existing ones. This, together with suitable hardware (faster, with more storage, possibly based on SSD - as in the APU boards, and more RAM), may lead to the integration of build systems directly on the ITS systems; for instance it is very easy to integrate gcc inside an OpenWrt distribution, by just selecting it after running “make menuconfig”.

For the aforementioned reasons, we investigated how to integrate two new OpenWrt versions: LEDE 17.01, together with the OpenC2X embedded platform [38], and the latest OpenWrt 18.06.1.

3.8 LEDE 17.01 and OpenC2X

The first platform we investigated is the OpenC2X embedded platform, developed by CCS Labs in University of Paderborn (Germany).

It includes a LEDE 17.01 distribution integrating the OpenC2X ETSI compliant platform.

This platform implements features of the ETSI protocol stack including DCC, CAM, DENM, the so-called “*Local Dynamic Map*”, offering support for GPS and even OBD-II connectivity to link the ITS device with the internal vehicle network (usually based on CAN).

It is the first open source platform, running over Linux, “*supporting most ETSI ITS-G5 features*” [36].

Being open source, it also offers a high degree of stabilizability and flexibility for what concerns modifications and updates to the protocol stack.

OpenC2X embedded is based on a patched LEDE distribution, a branch of OpenWrt which is now merged again inside the main project, with native support, coming directly from OpenWrt, to OCB mode and EDCA queues (for which, however, a patch was needed and it was introduced by the Paderborn University team).

This platform, in the default configuration, also includes the *LuCI* web user interface, accessible from the connected PC at the same IP address which was set for the board Ethernet interface (it is sufficient to write this address in any web browser).

This web interface was actually modified in order to enhance it with specific OpenC2X features, for example related to transmission and reception of CAM and DENM messages.

All the downloading and building work happen on the Linux Mint virtual machine.

The LEDE distribution was first of all downloaded from the following website: <https://github.com/florianklingler/OpenC2X-embedded>, using Git⁷:

```
git clone https://github.com/florianklingler/OpenC2X-embedded.git
```

The distribution was cloned inside the *OpenC2X-embedded* directory.

Before proceeding, several prerequisite packages needed to be installed in the development PC, where the build system resides. This can be done by using [49]:

```
sudo apt-get install build-essential subversion libncurses5-dev zlib1g-dev gawk gcc-multilib flex git-core gettext libssl-dev
```

Then, in order to prepare the distribution, it was necessary to update and download the available feeds (which are OpenWrt collections of packages, sharing a common location). This can be done thanks to a script included in every OpenWrt distribution (called *feeds*):

⁷Git is a “free and open source distributed version control system” which helps developers to manage software projects of any size. Home page: <https://git-scm.com/>

```
cd OpenC2X-embedded
./scripts/feeds update -a
./scripts/feeds install -a
```

This script can be also used to clean the downloaded feeds data: `./scripts/feeds clean`.

After updating the feeds, it was possible to load an OpenC2X embedded default configuration, already selecting the needed packages without the need of looking for them inside the kernel configuration window executed through `make menuconfig`. Since the ALIX boards are based on the x86 Geode processor, the corresponding configuration was selected:

```
./create_config.sh x86_geode
```

The `create_config.sh` script is an addition by the CCS Labs team, not included in official OpenWrt releases. It simplifies the creation of default configurations, by copying already generated configuration files contained in `./configs` in `.config`, which is the hidden configuration file normally generated after launching `make menuconfig`.

It was then necessary to run a `make` command to generate a full system configuration before the build process, including dependencies check.

```
make defconfig
```

It was then possible to run the Linux kernel configuration by executing:

```
make menuconfig
```

Most of the useful packages were already selected, but it could be useful to add a few.

In particular:

- Network -> `tcpdump`, for packet sniffing
- Network -> firewall -> `iptables` -> `iptables-mod-tee`, again for packet sniffing
- Utilities -> `strace`, which installs a the *strace* tool, very useful for debugging, since it is able to display the system call trace of whichever command

- Utilities -> `grep`, to install the *grep* search utility also onto the boards' system
- Network -> Time Synchronization -> `chrony`, an utility that uses NTP to synchronize the system clock, when the boards are connected to the Internet
- Utilities -> Shells -> `bash`, installing the *bash* shell inside the boards; in very low memory devices a user may want to disable this option and keep the existing *sh* shell, but the PC Engines boards are, in general, quite capable in terms of hardware and we will need this to perform some measurements (see chapter 6), mainly due to its support to arrays inside scripts

`iperf` and `iperf3` should be already included in the default configuration.

Including `make` and `gcc` could be useful too, mainly to develop small programs directly on the ITS device, with the condition that sufficient RAM and memory storage are available. We decided, however, to stick with the classical embedded solution of cross-compilation on the development PC (which is the solution that could be adopted in case low memory cheap devices are used.).

After including the desired package it is possible to compile the whole kernel. The first compilation may take, depending on the available hardware in the development PC, over two hours:

```
make -j1 V=s
```

The `V=s` argument enables verbose output, useful to identify possible compilation error. `-j1` instead sets up the compilation for working with one job only.

Although this has been done multiple times when building LEDE, a better solution, which has been successfully tested with OpenWrt 18.06.1, is to use:

```
make download  
make -j10 V=s
```

The first command downloads all the dependencies source code, enabling multi-core and multi-job compilation, which would otherwise result in a compilation error. The second command compiles the OpenWrt images generating 10 jobs, which can be a good choice

for 4 cores, 8 thread CPUs such as the Intel Core i7 one available on the development PC [50].

Multi-core compilation allows the development PC to finish the build process in even less than one hour, under some circumstances.

It has, however, the drawback of making it more difficult to detect a compilation error when the compilation stops before completion, with respect to the `-j1` case.

The user can also clean up the build environment, by means of:

- `make clean`, to do a basic clean, deleting the already compiled sources in the `./bin` and `./build_dir` directories.
- `make dirclean`, to perform a full clean operation, deleting also the `./staging_dir` and `./toolchain` folders (it deletes the OpenWrt toolchain for cross-compilation).
- `make distclean`, to perform a deep clean operation, equal to the previous case, but also clearing out all the feeds and configuration files.

After the build process finishes, two main compilation products are generated:

- The toolchain, including a gcc version to cross-compile for the boards
- The OpenWrt images, ready to be flashed onto the board's memory, generated thanks to the OpenWrt *Image Builder*, able to integrate the desired packages into flashable images.

3.8.1 LEDE toolchain

The toolchain binaries, containing tools for cross-compiling for the embedded boards are located in the `./staging_dir` folder. This is actually valid for every LEDE/OpenWrt distribution.

In the LEDE 17.01 distribution supplied with the OpenC2X embedded platform, the toolchain binaries, including gcc, is located in `./staging_dir/toolchain-i386_pentium_gcc-5.4.0_musl-1.1.16/bin/`.

Other toolchains, such as the one for MIPS, may be present inside the *./staging_dir*; however, we are interested on x86 system, thus using only the *i386* tools.

In order to make the use of the toolchain *gcc* easier and more flexible, some lines were added to the *.bashrc*⁸ file of the Linux operating system running on the development PC, as reported also in [51]:

.bashrc

```
# PATH environmental variable settings for OpenWrt toolchain
PATH=$PATH:/home/francesco/openwrt/staging_dir/toolchain-
    i386_pentium_gcc-7.3.0_musl/bin/
export PATH

# Set STAGING_DIR environment variable for the toolchain
STAGING_DIR=/home/francesco/openwrt/staging_dir/toolchain-
    i386_pentium_gcc-7.3.0_musl/bin
export STAGING_DIR

alias compileboard="i486-openwrt-linux-musl-gcc"

# Function to compile for the PC Engines boards
compileboardstatic() {
    i486-openwrt-linux-musl-gcc -o "$1" -static "$1.c"
}
```

More than one x86 version of *gcc* is actually being compiled: the choice of which version to use is up to the programmer, depending on his specific needs.

We decided to create an alias for compiling with *musl-gcc*, in order to support the *musl* C standard library implementation⁹, which may be useful in developing programs for the boards.

⁸The *.bashrc* file, under any Linux distribution using *bash*, is stored inside the main user directory (i.e. in */home/<username>*) and it contains commands to be executed every time a non-login *bash* shell is started

⁹Home page: <https://www.musl-libc.org/>

Using `musl-gcc` requires static linking (which is performed quite efficiently by this library), otherwise we were not able to run the obtained binaries on the target boards.

After saving the `.bashrc` file, it was then possible to compile custom C programs, such as the ones presented in section 3.8.5. To compile simple C programs we could use, directly from the terminal:

```
compileboard -o <binary file> -static <C file(s)>
```

or:

```
compileboardstatic <binary name>
```

which assumes the `.c` file has the same name as the final binary file. Of course more complex aliases could be defined to compile more complex projects.

To compile simple C programs, in any case, we always used `compileboardstatic`.

As a side note older versions of the toolchain can actually be used to cross-compile programs which will run on newer OpenWrt versions, but it is nevertheless suggested to always use updated binaries.

3.8.2 LEDE images, SquashFS and ext4

After the compilation finishes, two main OpenWrt images are generated, compressed inside `.tar.gz` archives and located inside `./bin/targets/x86/geode/` (or a similar directory inside `./bin`).

They are all relying on the GRUB2 boot loader for the boot process.

One is based on the ext4 file system (*lede-x86-geode-combined-ext4.img*), the other uses SquashFS+JFFS2, which are merged into a single file system thanks to *OverlayFS* (*lede-x86-geode-combined-squashfs.img*).

SquashFS is a read only, compressed file system, with the advantage of taking little space in embedded systems, with low overhead and easy recovery, since it is not actually possible to write on it.

Thanks to OverlayFS, JFFS2 is union mounted¹⁰ with SquashFS, providing also a writable compressed file system with journaling and flash wear leveling support. Thanks to JFFS2 the user can actually write data on the device. It is a good solution for raw NOR flash devices, such as some routers.

Due to SquashFS not having any bad block management and requiring all the blocks in order, it is typically not so suitable for raw NAND flash devices, in which UBIFS can be used instead [52].

ext4 is instead the more commonly used Linux file system, which optionally supports journaling.

Due to ALIX boards using Compact Flash memories, working with ATA controllers, and APU boards using SSDs, ext4 seemed to be a good choice.

Looking more in depth at the possible filesystems could be a future improvement to this work.

For now, the ext4 images will be always used.

After building the images, they were flashed into the Compact Flash memories, then mounted on the ALIX boards.

Their IP address was then changed to match the settings presented in section 3.7, editing the */etc/config/network* file.

Two additional modifications were needed to:

- Commenting out the line option 'type' 'bridge' by placing an # in front of it
- Adding the line option gateway '10.1.0.1' which is not present by default

An example of *network* configuration file is reported below, for the ALIX_100 board:

```
config interface 'loopback'
    option ifname 'lo'
    option proto 'static'
```

¹⁰Union mount actually mounts two combined file systems into what appears to be a single directory to the end user, generating a new new virtual root filesystem in /


```
option ipaddr '127.0.0.1'
option netmask '255.0.0.0'

config globals 'globals'
    option ula_prefix 'fd49:d667:37fc::/48'

config interface 'lan'
# option type 'bridge'
    option ifname 'eth0'
    option proto 'static'
    option ipaddr '10.1.6.100'
    option netmask '255.255.0.0'
    option gateway '10.1.0.1'
# option ip6assign '60'
```

After logging in to the root account, it was possible to test the 802.11p capabilities of the LEDE distribution, integrated inside the ALIX boards.

Before working with the boards on the ITS frequencies, some basic configuration needs to be performed, though.

The first configuration operation is related to the file */etc/config/wireless*, in which the wireless interface was enabled by setting option `disabled` to 0.

Then, we decided to keep the approach of using an `iw_startup` script, with the difference of not making it automatically executed at startup, to leave more freedom for the final user.

The script, adapted to use only `iw` and not the deprecated `iwconfig` (which is also no more included in default configurations), is reported below (in the ALIX_101 board only the WLAN IP address is different):

`iw_startup` (ALIX_100 board)

```
1 #!/bin/sh
2
3 echo "Wifi Down"
```

```
4 wifi down
5 sleep 1
6 echo "Wifi Up"
7 wifi up
8 echo "Waiting 5 seconds..."
9 sleep 5
10 echo "Pacthed Italian Frequency Register Set"
11 iw reg set IT
12 sleep 1
13 echo "Turn off wlan0"
14 ifconfig wlan0 down
15 sleep 1
16 echo "Set Mode OCB"
17 iw dev wlan0 set type ocb
18 echo "Turn on wlan0"
19 ifconfig wlan0 up
20 sleep 1
21 echo "Leave possibile OCB..."
22 iw dev wlan0 ocb leave
23 echo "Set Frequency 5890 MHz (CCH) and channel width 10MHz (802.11p)"
24 iw dev wlan0 ocb join 5890 10MHz
25 echo "IP address set to 10.10.6.100"
26 ifconfig wlan0 10.10.6.100 netmask 255.255.0.0
27 echo "Set Rate 6M and Power 0dBm, using iw"
28 iw dev wlan0 set bitrates legacy-5 6
29 iw dev wlan0 set txpower fixed 0
30 echo "Completing configuration, waiting 8 seconds..."
31 sleep 8
32 echo "Configured as: "
33 iw dev
```

Now OCB is supported by the kernel and it can be used instead of the IBSS mode, also with non-ITS frequencies (although it would be probably less meaningful), which seemed to be, in the previous case, disabled by the patch.

Some additional `sleep` commands have been added too, to let the system complete any previous configuration before issuing new commands.

One useful feature of the OpenWrt build system is the possibility of including custom files directly inside the built images, making them available as soon as the system is integrated into the boards.

In order to include them, a `files` directory has been created inside the main LEDE build directory. Every file contained in `./files/<dir>` will be included inside `./<dir>` [50].

Using this feature, the `iw_startup` has been included in the root directory, by placing it inside `./files/root` (after creating the directory itself).

Other useful commands, which are now different than before, are:

- `iw dev`, used to show the current Wi-Fi configuration, in place of the old `iwconfig`
- `iw dev <interface> station dump` (for example `iw dev wlan0 station dump`), showing useful statistic information such as inactive time, received and transmitted packets, received signal power and the last transmit bitrate; some information such as the bitrate values, as will be detailed later on, seems to be very unreliable, though
- `iw dev <interface> set txpower fixed <value in mBm>` (for example `iw dev wlan0 set txpower fixed 0`), to set a fixed output effective transmit power, in mBm (*milliBel per mW*), knowing that $100\text{ mBm} = 1\text{ dBm}$
- `iw phy <phy name> info`, to show information about the specified hardware device (for example `iw phy phy0 info`)

This system resulted to be quite complete, but few additional patches were needed in order to work with the ALIX boards (with WNICs still supported by *ath5k*) and perform some measurements.

One first modification was introduced to the wireless regulatory file, in order to enable the ITS frequencies also for Italy (IT flag), taking as reference the NL country in the GCDC 2011 OpenWrt version.

The file which will be used, with name *regdb.txt*, is available in the *./package/kernel/mac80211/files/* directory and it can be easily modified, at least in the LEDE version supplied with OpenC2X (it will be different in OpenWrt 18.06.1).

It already enables the ITS frequencies for Germany (DE flag), but in order to add them to Italy as well, the line (5842 – 5932 @ 20), (27), DFS was added to the Italian country:

```
country IT: DFS-ETSI
(2402 – 2482 @ 40), (20)
(5170 – 5250 @ 80), (20), AUTO-BW
(5250 – 5330 @ 80), (20), DFS, AUTO-BW
(5490 – 5710 @ 160), (27), DFS
(5842 – 5932 @ 20), (27), DFS
# 60 GHz band channels 1-4, ref: Etsi En 302 567
(57000 – 66000 @ 2160), (40)
```

The modifications to this file becomes active only after a recompilation process.

The frequency band that has been set ranges from 5842 *MHz* to 5932 *MHz* to take into account the fact that the regulatory database calculations are made considering 20 MHz wide channels, instead of the 10 MHz 802.11p channels. So, some MHz of margin have been added to the allowed frequency range (the reference is the GCDG 2011 patch).

It is important, however, to keep in mind that this is still a patch: more normative research is needed in case a similar system is used outside of research activities or it is really deployed onto vehicles.

Then, before introducing the true patches, the patching tool *quilt* is briefly described.

3.8.3 Patch creation and management: quilt

The *quilt* software is a non GNU utility¹¹ that can be used to keep track of the modification to source code files, by means of patches, which can be applied and un-applied like in

¹¹Home page:<http://savannah.nongnu.org/projects/quilt>

a stack, created and removed, refreshed and so on; this utility is actually well documented in the Linux Man Pages¹²; this tool can be efficiently used from the terminal.

Patches are included in *.patch diff* files, containing information about the modified lines of code and applied by *quilt* or during compilation.

The Linux patch tool, although not used in this work, can be used to apply patches to source code starting from these *diff* files.

It is included in the LEDE/OpenWrt build system to create and manage patches to the Linux kernel and packages code.

In order to configure quilt to generate patches in the correct format, as suggested in the OpenWrt documentation, a proper configuration file (*.quiltrc*) must be created in the user home directory (the same where *.bashrc* is stored) [53].

This file contains also the preferred editor to use when editing source code files to generate patches; in order to dynamically and easily change this field, a script in turn generating the required *.quiltrc* file has been created on the Linux virtual machine in the development PC (the same would apply with an installed Linux partition).

This script accepts as argument the name of the editor.

quiltrcmaker.sh

```
1  #!/bin/bash
2
3  echo "Creating .quiltrc file in your home directory..."
4  cat > ~/.quiltrc <<EOF
5  QUILT_DIFF_ARGS="--no-timestamps --no-index -p ab --color=auto"
6  QUILT_REFRESH_ARGS="--no-timestamps --no-index -p ab"
7  QUILT_SERIES_ARGS="--color=auto"
8  QUILT_PATCH_OPTS="--unified"
9  QUILT_DIFF_OPTS="-p"
10 EDITOR="$1"
11 EOF
```

¹²<https://www.systutorials.com/docs/linux/man/1-quilt/>

```
12 echo "Process completed!"
```

The `.quiltrc` file has been then created in the user home directory, by moving to the directory where the script has been saved (in our case the current user home directory) and by invoking:

```
cd --
chmod +x quiltrcmaker.sh
./quiltrcmaker.sh subl
```

The second line should be run only once to set the execution permission on the script before launching it.

As editor, we decided to use Sublime Text (*subl*), which is a multi platform, advanced and commercial source code editor. Other free and open source alternatives can be specified, such as *nano*.

After creating the configuration file, it is possible to start using *quilt*.

The workflow in order to create a new patch is briefly summarized here, taking as reference [53]:

1. Move to the build system main directory using `cd`.
2. Prepare the source directory and existing patches:

```
make package/example/{clean,prepare} V=s QUILT=1.
```

Example with *iPerf*:

```
make package/network/utils/iperf/{clean,prepare} V=s QUILT=1.
```
3. Move to the prepared source directory inside `./build_dir`, looking for the prepared source has been placed (it should be printed by the previous command):

```
cd build_dir/target-*/example-*
```

Example with *iPerf*:

```
cd build_dir/target-i386_pentium_musl-1.1.16/iperf-2.0.9/.
```
4. Apply all the pre-existing patches before creating a new one:

```
quilt push -a.
```

5. Show the list of all the existing patches:

```
quilt series.
```

6. Create a new patch: `quilt new <xxx>-<patch_name>.patch`, where `<xxx>` should be a number, higher than all the other patch numbers (we noticed that the alphabetical order play a role too, so, creating a patch with number 1010 while all the other patches are named with three digits numbers does not ensure that it will be applied after patch #999, unless all the other patches are renamed with four digits and a zero in front); *quilt* manages the patches like if they were put inside a stack; the newly created patch is actually “put on top” and applied after all the other underlying patches.

7. Associate the files to edit to the patch and modify them: `quilt edit <path/ filename>`. This command will associate a file and open the preferred editor to edit that file; the *path* is relative to the position of the package prepared source directory.

Example with an *iPerf* file: `quilt edit src/Locale.c`

8. Optionally review the changes: `quilt diff`

9. Update the patch with the modifications: `quilt refresh`.

10. Repeat points from 6 to 9 to create other patches, if needed.

11. Return to the build system main directory using `cd` (example: `cd ../../../../`)

12. Update the package:

```
make package/example/update V=s.
```

Example with *iPerf*:

```
make package/network/utils/iperf/update V=s.
```

13. Rebuild the package to compile the changes:

```
make package/example/{clean,compile} package/index V=s.
```

Example with *iPerf*:

```
make package/network/utils/iperf/{clean,compile} V=s.
```

In order to make this workflow easier to follow, we created a script for the development PC which is able to prepare, push all the existing patches, update and build a package for LEDE/OpenWrt, adding then a corresponding alias inside *.bashrc* to call it from any terminal window.

The script uses a *quiltmake.config* file inside the user home directory, which must contain, on two separate lines:

- The OpenWrt build system directory
- The desired target directory name

quiltmake.sh

```
1  #!/bin/bash
2
3  if [ $# -ne 2 ]
4  then
5      echo "Error using program. Expected two parameters: "
6      echo "1) Package name (either its name or its compiled name)."
7      echo "2) -p for \"prepare\" or -q for cd and quilt push or -u for
           update or -b for build (includes -u)"
8  exit 1
9  fi
10
11 IFS=$'\r\n' # Internal field separator setting for reading the .config
           file
12 filename="quiltmake.config"
13
14 cd ~
15 i=0
16 while read -r line
```



```
17 do
18     config[$i]="$line"
19     i=$((i+1))
20 done < $filename
21 openwrtpath=${config[0]}
22 target=${config[1]}
23
24 echo "openWRT path: $openwrtpath"
25 echo "target name: $target"
26
27 if [ $2 = "-p" ]
28 then
29     echo "Preparing package: $1"
30     cd $openwrtpath
31     make package/$1/{clean,prepare} V=s QUILT=1
32 elif [ $2 = "-q" ]
33 then
34     echo "Moving to folder: $openwrtpath/build_dir/$target/$1"
35     cd $openwrtpath/build_dir/$target/$1
36     echo "Pushing existing patches..."
37     quilt push -a
38     echo "Now you can start creating your patch with \"quilt new <number
39         >-<filename>.patch\""
40     echo " and \"quilt edit <src file>\"."
41     echo "Review your changes with \"quilt diff\" and apply them with \"
42         quilt refresh\"."
43     echo "Please use now \"cd $openwrtpath\" and \"cd build_dir/$target/
44         $1\" to start working"
45     echo " as reported above."
46 elif [ $2 = "-b" ]
47 then
48     echo "(Re)building package $1"
49     echo "Updating..."
50     cd $openwrtpath
```

```
48 make package/$1/update V=s QUILT=1
49 echo "Cleaning, compiling and installing"
50 make package/$1/{clean,compile,install} package/index V=s
51 echo "Ok. Now, if you do not see any error, you can regenerate your
    image using, for instance, \"make -j1 V=s\""
52 elif [ $2 = "-u" ]
53 then
54 echo "Updating package $1"
55 echo "Updating..."
56 cd $openwrtpath
57 make package/$1/update V=s QUILT=1
58 echo "Package updated."
59 else
60 echo "Unrecognized argument $2. The script will terminate now."
61 exit 1
62 fi
63 echo "Done."
```

Example of quiltmake.config file

```
/home/francesco/openc2x_embedded/OpenC2X-embedded/
target-i386_pentium_musl-1.1.16
```

Added line to .bashrc

```
alias quiltmake="~/quiltmake.sh"
```

Thanks to this script, creating patches, for example for the *iPerf* package, was made simpler. A sample workflow, opening a terminal on the LEDE main build directory, for *iPerf* patching is shown below:

```
quiltmake network/Utils/iperf -p
quiltmake iperf-2.0.9 -q
cd build_dir/target-i386_pentium_musl-1.1.16/iperf-2.0.9/
quilt new ....
quilt edit ....
```

```
quilt refresh
quiltmake network/utils/iperf -b
```

3.8.4 Patching ath5k and iperf

The OpenC2X platform was designed to flawlessly work with the *ath9k* driver. In order to adapt the platform to properly work with the *ath5k* based UNEX cards, a patch needed to be coded.

In fact, looking at the hardware properties supported by the driver with `iw dev phy phy0`, two channels (182 and 184) were actually missing from the list of the supported ones, even though, from previous experience, we knew that they were actually supported.

Part of the hardware information command output is reported below:

```
.....
* 5860 MHz [172] (14.0 dBm)
* 5870 MHz [174] (33.0 dBm)
* 5880 MHz [176] (33.0 dBm)
* 5890 MHz [178] (33.0 dBm)
* 5900 MHz [180] (33.0 dBm)
valid interface combinations:
* #{ managed } <= 2048, #{ AP, mesh point } <= 4, #{ IBSS } <= 1,
  total <= 2048, #channels <= 1
.....
```

In order to enable the support for them, the driver needed to be modified. Thanks to the transparency mechanism introduced in section 3.5.1, it was sufficient to change the driver source code, without touching, for instance, the *mac80211* layer.

The patch file is reported here:

999-ITS-G5D-channels-fix.patch (in package/kernel/mac80211)

```
1 --- a/drivers/net/wireless/ath/ath5k/base.c
2 +++ b/drivers/net/wireless/ath/ath5k/base.c
3 @@ -319,7 +319,22 @@ ath5k_setup_channels(struct ath5k_hw *ah
```

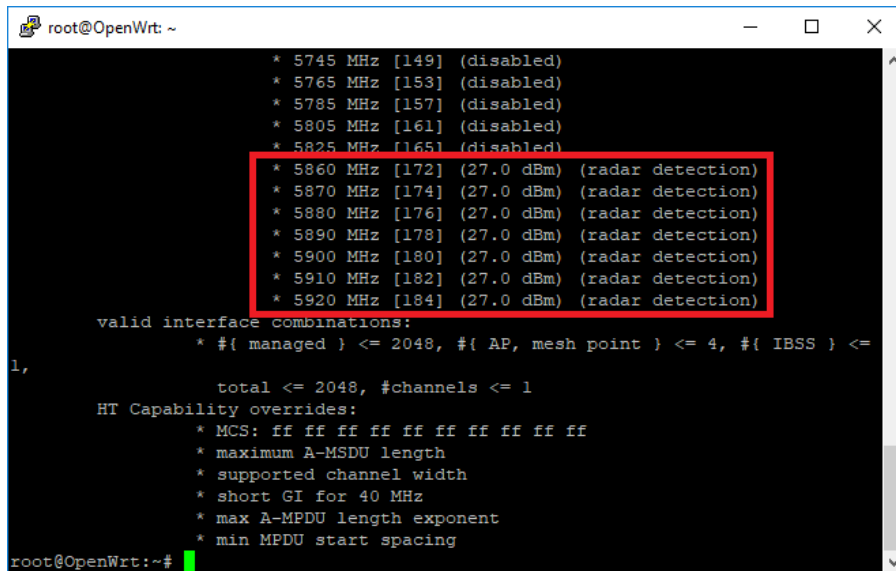
```
4
5 count = 0;
6 for (ch = 1; ch <= size && count < max; ch++) {
7 -   freq = ieee80211_channel_to_frequency(ch, band);
8 +   /* ITS-G5D fix: added by Politecnico di Torino to enable channels
      182
9 +       and 184 on UNEX DCMA 86P2 cards */
10 +   if(ch == 182 || ch == 184) {
11 +       switch(ch) {
12 +           case 182:
13 +               freq=5910;
14 +               break;
15 +           case 184:
16 +               freq=5920;
17 +               break;
18 +           default:
19 +               break;
20 +       }
21 +   } else {
22 +       freq = ieee80211_channel_to_frequency(ch, band);
23 +   }
24
25 if (freq == 0) /* mapping failed - not a standard channel */
26 continue;
```

The 182 and 184 channels, representing the ITS-G5D ones in the ETSI standard, were disabled due to the fact that *ath5k* was using, inside the *ath5k_setup_channels()* function (responsible for channels' setup) a call to *ieee80211_channel_to_frequency()*, a utility function provided by *cfg80211*. This function, defined in *net/wireless/util.c* in the Linux kernel code, performs the mapping between the channel number and the corresponding central frequency.

It is, however, mapping channels from 182 to 196 to the 4.9 GHz range, instead of the 5.9 GHz range.

In order not to modify the *cfg80211* source code, which may be used by other devices, the driver was modified to perform a manual mapping between channels 182 and 184 and their respective center frequencies (5910 and 5920 MHz).

Figure 3.14 shows the driver correctly recognizing channels 182 and 184.



```

root@OpenWrt: ~
* 5745 MHz [149] (disabled)
* 5765 MHz [153] (disabled)
* 5785 MHz [157] (disabled)
* 5805 MHz [161] (disabled)
* 5825 MHz [165] (disabled)
* 5860 MHz [172] (27.0 dBm) (radar detection)
* 5870 MHz [174] (27.0 dBm) (radar detection)
* 5880 MHz [176] (27.0 dBm) (radar detection)
* 5890 MHz [178] (27.0 dBm) (radar detection)
* 5900 MHz [180] (27.0 dBm) (radar detection)
* 5910 MHz [182] (27.0 dBm) (radar detection)
* 5920 MHz [184] (27.0 dBm) (radar detection)

valid interface combinations:
* #{ managed } <= 2048, #{ AP, mesh point } <= 4, #{ IBSS } <=
1,
    total <= 2048, #channels <= 1
HT Capability overrides:
* MCS: ff ff ff ff ff ff ff ff
* maximum A-MSDU length
* supported channel width
* short GI for 40 MHz
* max A-MPDU length exponent
* min MPDU start spacing
root@OpenWrt:~#

```

Figure 3.14: Channels 182 and 184 now available

Another feature that was missing was *iPerf* support to the EDCA traffic classes, to perform measurements with prioritized traffic using a simple tool, without the need of coding other external applications.

The IP ToS field (*iperf* option *-S*) could have been used, but it appeared to be ignored in frame prioritization, due to the “999-Enable-queueing-in-all-4-ACs-BE-BK-VI-VO.patch” OpenC2X *mac80211* patch, which, however, introduces a very useful workaround that makes it possible to directly set the traffic class by acting on any created socket options.

This patch actually modifies the *ieee80211_select_queue()* function, in *mac80211*, mentioned when speaking about the Linux wireless subsystem.

Its code can be seen, as an updated version to work with OpenWrt 18.06.1, in Appendix B.

Following the workflow presented before for the *quilt* tool, a patch for the *iPerf* program has been created, adding a new, non-mandatory, option `-A` to specify the traffic class which should be used to send packets (it accepts as argument “BK”, “BE”, “VI” or “VO”).

Then, after fetching the access category from the user, the patched binary can set the socket, created by *iPerf*, to use the corresponding user priority.

The patch file, including comments to the modified and newly introduced lines, is reported below:

001-iperf-MAC_AC-patch.patch (in package/network/utils/iperf)

```

1  --- a/src/PerfSocket.cpp
2  +++ b/src/PerfSocket.cpp
3  @@ -136,8 +136,18 @@ void SetSocketOptions( thread_Settings *
4                      (char*) &tos, len );
5          WARN_errno( rc == SOCKET_ERROR, "setsockopt IP_TOS" );
6      }
7  +
8  #endif
9
10 + // set MAC AC (access category) field, if specified only (i.e. if
    mMACUP != -1)
11 + // AC is set starting from user priorities (UP)
12 + if ( inSettings->mMACUP >= 0 ) {
13 +     int up = inSettings->mMACUP;
14 +     Socklen_t len = sizeof(up);
15 +     int rc = setsockopt( inSettings->mSock, SOL_SOCKET,
        SO_PRIORITY, (char*) &up, len );
16 +     WARN_errno( rc == SOCKET_ERROR, "setsockopt SO_PRIORITY" );
17 + }
18 +
19     if ( !isUDP( inSettings ) ) {
20         // set the TCP maximum segment size
21         setsock_tcp_mss( inSettings->mSock, inSettings->mMSS );
22  --- a/src/Settings.cpp

```

```

23 +++ b/src/Settings.cpp
24 @@ -107,6 +107,7 @@ const struct option long_options[] =
25     {"realtime",          no_argument, NULL, 'z'},
26
27     // more esoteric options
28 +{"accesscategory",    required_argument, NULL, 'A'},
29     {"bind",            required_argument, NULL, 'B'},
30     {"compatibility",    no_argument, NULL, 'C'},
31     {"daemon",          no_argument, NULL, 'D'},
32 @@ -152,6 +153,7 @@ const struct option env_options[] =
33     {"IPERF_REPORTSTYLE",required_argument, NULL, 'y'},
34
35     // more esoteric options
36 +{"IPERF_MACAC",       required_argument, NULL, 'A'},
37     {"IPERF_BIND",      required_argument, NULL, 'B'},
38     {"IPERF_COMPAT",    no_argument, NULL, 'C'},
39     {"IPERF_DAEMON",    no_argument, NULL, 'D'},
40 @@ -172,7 +174,7 @@ const struct option env_options[] =
41
42     #define SHORT_OPTIONS()
43
44 -const char short_options[] = "1b:c:def:hi:l:mn:o:p:rst:uvw:x:y:zB:CDF
45     :IL:M:NP:RS:T:UVWZ:";
46 +const char short_options[] = "1b:c:def:hi:l:mn:o:p:rst:uvw:x:y:zA:B:
47     CDF:IL:M:NP:RS:T:UVWZ:";
48
49 /*
50
51     -----
52
53     * defaults
54 @@ -233,6 +235,7 @@ void Settings_Initialize( thread_Setting
55     //main->mThreads          = 0;          // -P,
56     //main->mRemoveService = false;        // -R,

```

```

52      //main->mTOS          = 0;          // -S,  ie. don't set type
      of service
53 +   main->mMACUP           = -1;          // -A (set to an invalid
      number as default -> with -1 no setsockopt will be called for AC)
54      main->mTTL            = 1;          // -T,  link-local TTL
55      //main->mDomain       = kMode_IPv4;  // -V,
56      //main->mSuggestWin = false;        // -W,  Suggest the window
      size.
57 @@ -679,6 +682,24 @@ void Settings_Interpret( char option, co
58      mExtSettings->mTOS = strtol( optarg, NULL, 0 );
59      break;
60
61 +   case 'A': // 802.11p/802.11e MAC layer access categories
62 +       // Mapping between UP (0 to 7) and AC (BK to VO)
63 +       if(strcmp(optarg,"BK") == 0) {
64 +           mExtSettings->mMACUP=1; // UP=1 (2) is AC_BK
65 +       } else if(strcmp(optarg,"BE") == 0) {
66 +           mExtSettings->mMACUP=0; // UP=0 (3) is AC_BE
67 +       } else if(strcmp(optarg,"VI") == 0) {
68 +           mExtSettings->mMACUP=4; // UP=4 (5) is AC_VI
69 +       } else if(strcmp(optarg,"VO") == 0) {
70 +           mExtSettings->mMACUP=6; // UP=6 (7) is AC_VO
71 +       } else {
72 +           // Leave to default (-1), i.e. no AC is set to socket
73 +           , and print error
74 +           fprintf( stderr, "Invalid AC specified with -A\nValid
      ones are: BK, BE, VI, VO\nNo AC will be set\n");
75 +       }
76 +       //mExtSettings->mMACUP = strtol( optarg, NULL, 0 );
77 +       break;
78 +
79      case 'T': // time-to-live for multicast
80      mExtSettings->mTTL = atoi( optarg );

```



```

81         break;
82 --- a/include/Settings.hpp
83 +++ b/include/Settings.hpp
84 @@ -129,6 +129,7 @@ typedef struct thread_Settings {
85     // int's
86     int mThreads;                // -P
87     int mTOS;                    // -S
88 +   int mMACUP;                  // -A
89     int mSock;
90     int Extractor_size;
91     int mBufLen;                 // -l
92 --- a/src/Locale.c
93 +++ b/src/Locale.c
94 @@ -111,6 +111,7 @@ Client specific:\n\
95     -n, --num          #[KM]      number of bytes to transmit (instead of -t
96                                   )\n\
97     -r, --tradeoff      Do a bidirectional test individually\n\
98     -t, --time          #          time in seconds to transmit for (default
99                                   10 secs)\n\
100 +   -A, --accesscategory <AC> Forces a certain EDCA MAC access category
101                                   to be used\n\
102     -B, --bind [<ip> | <ip:port>] bind src addr(s) from which to
103                                   originate traffic\n\
104     -F, --fileinput <name> input the data to be transmitted from a
105                                   file\n\
106     -I, --stdin          input the data to be transmitted from
107                                   stdin\n\
108 --- a/include/version.h
109 +++ b/include/version.h
110 @@ -1,2 +1,2 @@
111 -#define IPERF_VERSION "2.0.9"
112 +#define IPERF_VERSION "2.0.9 OpenC2X patch"
113 +#define IPERF_VERSION_DATE "9 Sept 2016"

```

In order to test the added functionality, two *sh* scripts using *iperf* have been written, to open a *client+server* on one board, connecting to a *server+client* on the other board, with the possibility of setting two different traffic classes for the two clients connecting to the two servers.

server6000client7000.sh

```
1  #!/bin/sh
2
3  if [ $# -ne 3 ]
4  then
5      echo "Error using program. Expected 3 parameters: client IP, AC, inj
        . data rate."
6      exit 1
7  fi
8
9  iperf -s -u -i 2 -p 6000 &
10 sleep 2
11 echo 'Starting client...'
12 iperf -c $1 -u -b $3 -i 2 -t 60 -A $2 -p 7000 >> Client7000.txt
13 sleep 2
14 echo 'Killing iperf server...'
15 killall iperf
16 echo 'Done.'
```

server7000client6000.sh

```
1  #!/bin/sh
2
3  if [ $# -ne 3 ]
4  then
5      echo "Error using program. Expected 3 parameters: client IP, AC, inj
        . data rate."
6      exit 1
7  fi
```

```
8
9 iperf -s -u -i 2 -p 7000 &
10 sleep 2
11 echo 'Starting client...'
12 iperf -c $1 -u -b $3 -i 2 -t 60 -A $2 -p 6000 >> Client6000.txt
13 sleep 2
14 echo 'Killing iperf server...'
15 killall iperf
16 echo 'Done.'
```

They should be run one on one board and the other on the second board, specifying as arguments:

1. The WLAN IP address of the other board to connect to (for instance 10.10.6.100 or 10.10.6.101)
2. The traffic class (a string containing “BK”, “BE”, “VI” or “VO”)
3. The data rate at which the UDP transmission should happen (for example *10M*)

During this work, they were also included inside all the newly built LEDE/OpenWrt images, using the custom files feature described in section 3.8.2.

As is it possible to notice, the new `-A` option is used (together with the typical *iPerf* commands presented before) and two client log files are saved into *Client6000.txt* and *Client7000.txt*.

More detailed measurements with *iPerf*, on the APU boards, be presented in chapter 6.

3.8.5 C programs for broadcast transmissions

The next step foresaw the creation of two command line C programs for broadcast (and possibly also unicast) transmissions, which were then cross-compiled for the boards.

The two programs are based on raw sockets, in order to make them as much flexible as possible, even for the implementation of non-IP protocols, such as WSMP.

In order to properly test them, UDP, which is IP-based, was chosen, but we tried to keep our approach as general as possible, in order to enable any future improvement with other protocols and packet formats.

The disadvantage is that the packet should be manually built: for instance, considering UDP, the application has to build the Ethernet header, the IP header and the UDP header, including the IP and UDP checksums, which should be then checked by the receiver.

In order to make it easier to build the UDP packet and to separate the application from all the functions to manage and build the frame, we wrote a custom library, which contains:

- Some useful constants, such as additional Ethertypes
- Some useful macros, for instance to compute the size of a standard IP/UDP packets (with no options in the IP header)
- Additional types, including “`macaddr_t`”, to contain mac addresses as 1 B arrays with 6 places, “`byte_t`”, to define in a more friendly manner byte arrays and variables (which are actually unsigned char) and “`ethertype_t`”, as a new, more friendly, name for unsigned short, to be used to specify a certain Ethertype
- General utility functions, including “`wlanLookup()`”, which can be used to automatically look for a wireless interface on the device, two functions to print a given packet with a certain length (both in hexadecimal mode - “`display_packet()`” and in character mode - “`display_packetc()`”) and functions to manage MAC address arrays (“`macaddr_t`” type)
- Functions to populate an Ethernet header (with source MAC, destination MAC and Ethertype) and to encapsulate an SDU inside an Ethernet packet, combining header and data coming from higher layers
- Functions to populate and manage IPv4 headers and packets
- Functions to populate and manage UDP headers and packets

- Two function which can be useful when receiving UDP datagrams through a raw socket: “UDPgetpayloadsize()”, allowing to get the payload size of a specified UDP datagram (given its pointer), and “UDPgetpacketpointers()”, to obtain, given a certain buffer containing an UDP packet, the pointer to the headers and payload sections

The library is divided into two main files, called “rawsock.h” and “rawsock.c” plus two additional files “ipsum_alth.h” and “ipsum_alth.c”, which contain a copy of the “|”ip_fast_csum()| function from the 4.19.1 version of the Linux kernel; this function is able to compute the IP checksum is a fast and reliable way.

This function, even though it is available in the Linux kernel, was copied inside two additional files to make it easier to compile programs using the library, since this IP checksum calculation function is contained in kernel header files (in particular, inside *net/checksum.h*), which are not normally available for inclusion like other headers.

This library currently supports IP and UDP only, with very few constants related also to WSMP.

We built its structure, however, in the hope that it could be easily extended to other non-IP protocols without having too many issues to deal with.

Its code and an example on how it can be used (i.e. the packet sending program) are reported in the appendix mentioned at the end of this section.

The two programs are represented by one periodic sender and one receiver, printing the information which are read from the wireless medium, either broadcast or directed to a specific board:

- The sending program (*testprogram_broadcastSend*) accepts as arguments: the port to be used, IP address (255.255.255.255 can be used for broadcast flooding transmissions), user priority (which will be mapped to an access category), period to use for sending each message (in seconds, values less than 1 are accepted), payload of the message (the program could be enhanced to use a different payload, not specified by the user though the terminal).

- The receiving program (*testprogram_broadcastReceive*) does not accept any argument (as it can receive all UDP packets), but it can be easily enhanced to received parameters from the user, such as source IP address, port, and so on.

In the current version of the program, which is set for broadcast transmissions only (at the moment), the IP address specified by the user is actually ignored. This parameter, however, has been kept to enable future versions of the program to easily support both unicast and broadcast transmissions.

They are both able to automatically look for available wireless interfaces thanks to the functions contained in the library described before.

The “`wlanLookup()`” function has the following prototype:

```
62 rawsockerr_t wlanLookup(char *devname, int *ifindex, macaddr_t mac,  
    unsigned int index);
```

When only one interface is available and “0” is specified as index, that interface name is returned inside “devname”. Then, if the other two arguments are not NULL, the interface index and the corresponding source MAC address (if available) are returned.

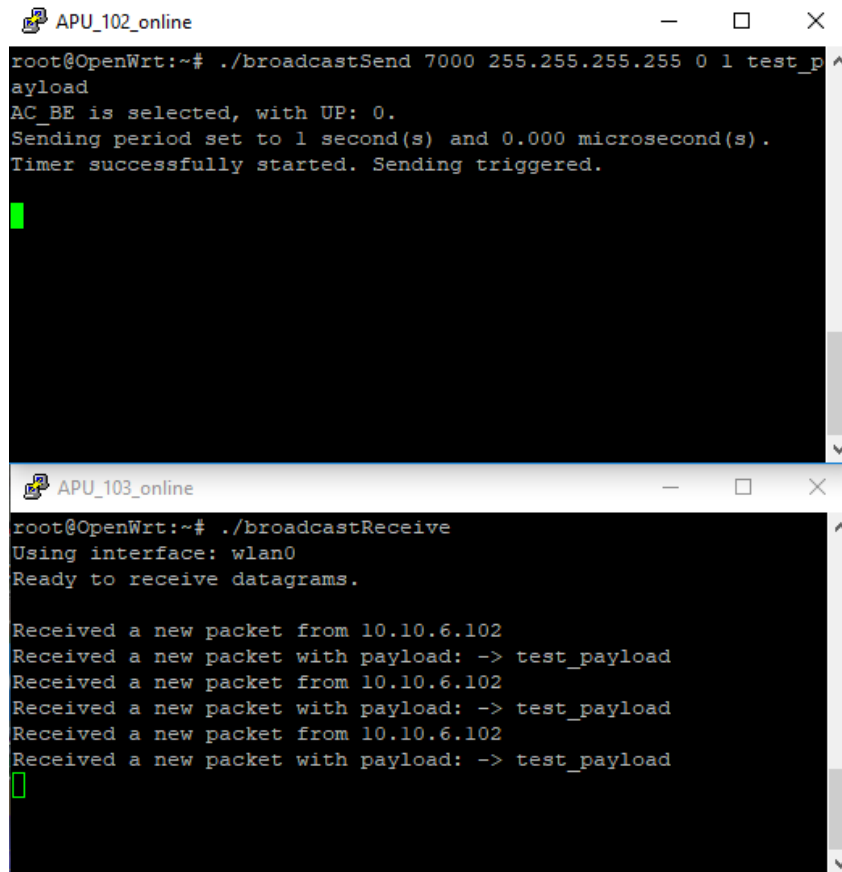
If more than one interface is present, the number of available interfaces is returned by the function and the index is used to point to a specific interface (for instance “index=1” can be used to point to a possible “wlan1” interface).

As mentioned before, the programs are actually using UDP as higher layer protocol, but they can be used as a basic skeleton to build more complex projects, possibly supporting non-IP protocols which can be implemented in the future, such as WSMP.

The processing done by the receiving program is then limited, at this level, to reading and printing the received information, but, again, they can be extended to perform further processing, use nl80211 to manage the wireless stack, and so on.

The sending program can also be used as a base to implement a broadcast sending software parsing information from files and from other structures, other than simply receiving the information from the user via terminal (both for what concerns the payload and the interval/user priority).

The time precision of the Linux timers used to send periodic messages, taking also into account the program execution time, allows the user to easily reach around 1 *ms* of precision; this is fine for ITS applications like sending periodic BSMs.



The image shows two terminal windows. The top window, titled 'APU_102_online', shows the execution of a program that sends broadcast messages. The bottom window, titled 'APU_103_online', shows the execution of a program that receives broadcast messages.

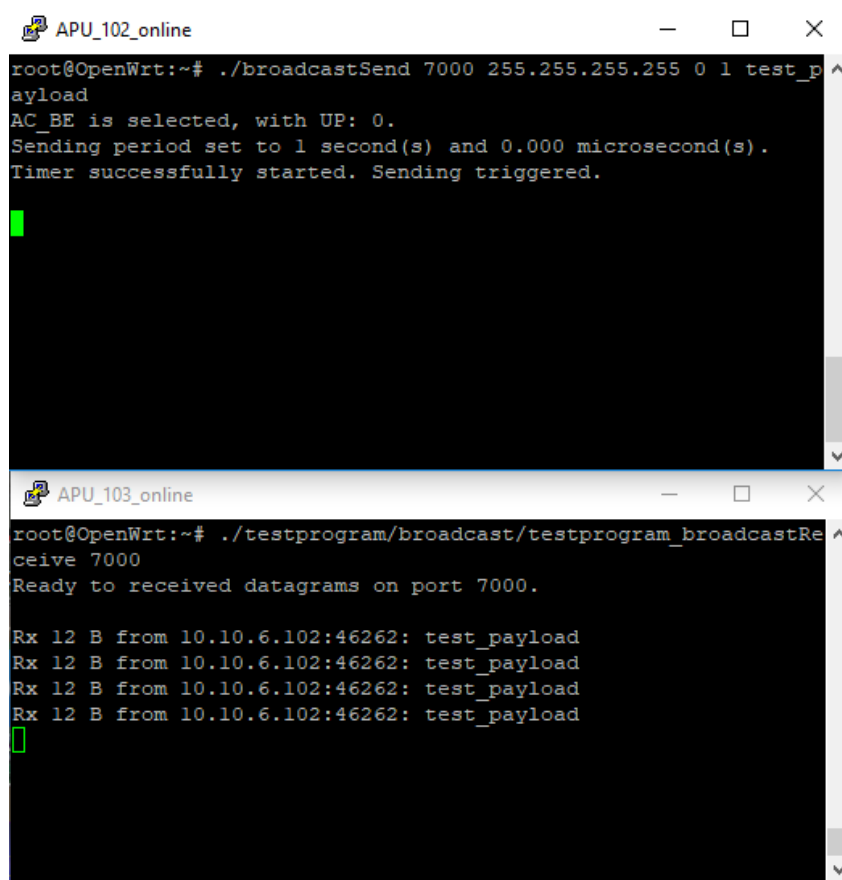
```
APU_102_online
root@OpenWrt:~# ./broadcastSend 7000 255.255.255.255 0 1 test_p
ayload
AC_BE is selected, with UP: 0.
Sending period set to 1 second(s) and 0.000 microsecond(s).
Timer successfully started. Sending triggered.

APU_103_online
root@OpenWrt:~# ./broadcastReceive
Using interface: wlan0
Ready to receive datagrams.

Received a new packet from 10.10.6.102
Received a new packet with payload: -> test_payload
Received a new packet from 10.10.6.102
Received a new packet with payload: -> test_payload
Received a new packet from 10.10.6.102
Received a new packet with payload: -> test_payload
```

Figure 3.15: The two C programs for broadcast transmission and receptions, running respectively on the two boards (the image has been recreated in a second moment: that’s the only reason why the APU boards are shown in the title of each window)

In order to validate, at least for what concerns UDP, our implementation, we also tested the sending program together with a receiver program using UDP datagram sockets, showing that our manually built UDP packets are actually received and that both the library and the sender/received program should be working properly.



```
APU_102_online
root@OpenWrt:~# ./broadcastSend 7000 255.255.255.255 0 1 test_p
ayload
AC_BE is selected, with UP: 0.
Sending period set to 1 second(s) and 0.000 microsecond(s).
Timer successfully started. Sending triggered.

APU_103_online
root@OpenWrt:~# ./testprogram/broadcast/testprogram_broadcastRe
ceive 7000
Ready to received datagrams on port 7000.

Rx 12 B from 10.10.6.102:46262: test_payload
Rx 12 B from 10.10.6.102:46262: test_payload
Rx 12 B from 10.10.6.102:46262: test_payload
Rx 12 B from 10.10.6.102:46262: test_payload
```

Figure 3.16: Validating the sender program with a receiver program using a simple UDP datagram socket

The C source code is reported in Appendix A.

3.8.6 Packet sniffing: methodologies

One important feature when developing and researching on networking applications, is the possibility to perform packet sniffing.

Three ways of performing packet sniffing on the *PC Engine* boards have been studied, trying to maintain them as general as possible, as general method to do sniffing on networking embedded systems, typically missing an user interface and unable to run *Wireshark*.

In order to perform packet sniffing in an optimal way, the chosen board should be

configured to use a monitor wireless interface.

Thanks to how the Linux networking subsystem works, with virtual interfaces, a board can manage both monitoring and, for instance, transmission of information over the radio medium.

In order to create a new virtual monitor interface, alongside the main wireless interface (which, in our work, was always called *wlan0*), the following command have to be issued:

```
iw dev wlan0 interface add wlan1 type monitor
# or: iw phy phy0 interface add wlan1 type monitor
ifconfig wlan1 up
```

It is important to notice that this is possible only if the wireless card supports interface combinations.

A board can also be configured in monitor mode only, which is always the preferred option.

In case two modes are used simultaneously, however, the monitor interface will depend on the master (managed) one, for example for what concerns the configured frequency.

(1) Using tcpdump

`tcpdump` is a packet analysis command line tool, able to report all the captured packets over a certain wireless interface (it is typically used in conjunction with monitor mode interfaces).

Its use is well documented inside the Linux Man Pages [54]. This section will concentrate on the useful options for packet sniffing operations.

When included into the boards, it can be used both for printing out the captured packets or to save them into a file, which can be transferred to the development PC and visualized inside *Wireshark*.

In the first case, it is sufficient to use, on the boards (so, by means of PuTTY):

```
tcpdump -i <interface> -l
```

Where `<interface>` is the monitor interface, for example, with reference to the previous section, `wlan1`. This option is meaningful only when few packets are received and to check whether communication can really happen between different devices.

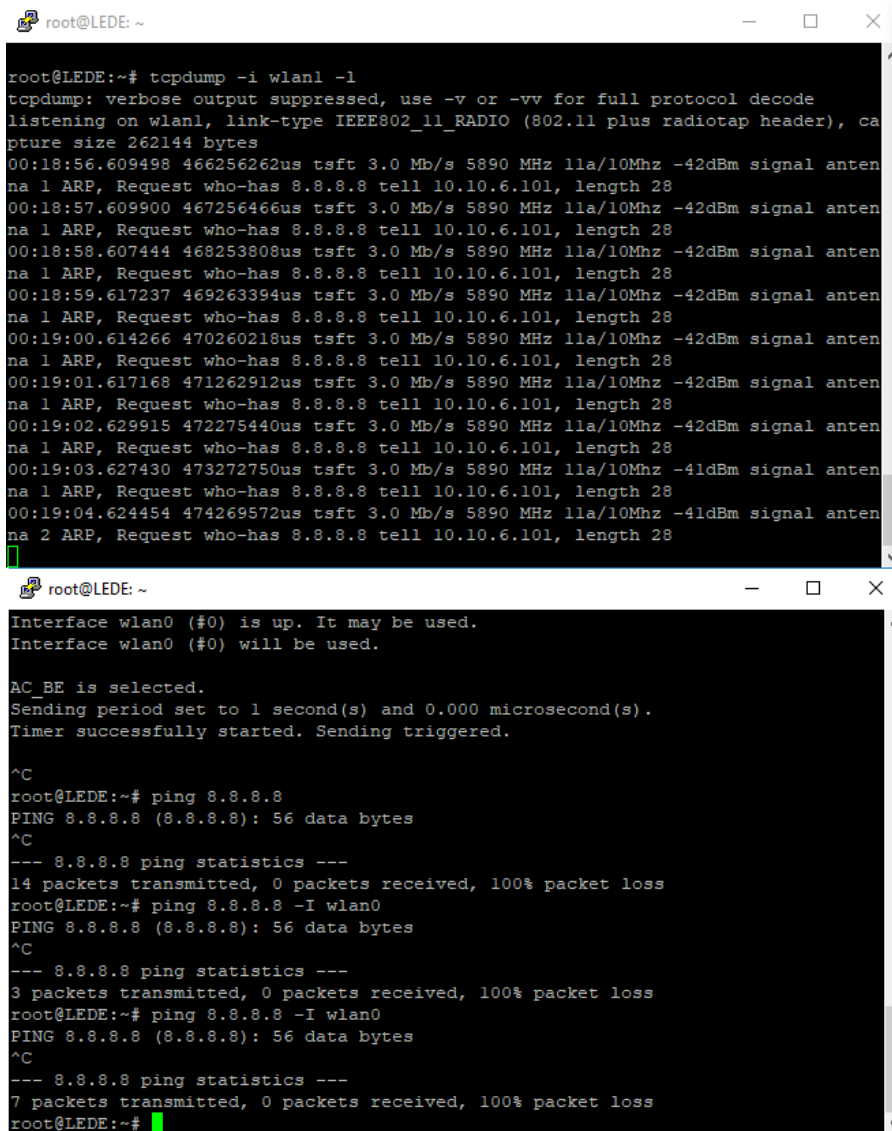
`-l` makes the output line buffered for a better visualization.

In order to capture network traffic and be able to better analyze it, it is possible to make `tcpdump` write the raw packets to a file, instead of parsing and printing them out. It can be read both by `tcpdump`, with the `-r <file>` option, or transferred to the development PC and analyzed with *Wireshark*.

The command to be used is:

```
tcpdump -i <interface> -w <file>
```

The *.pcap* extension is the suggested one for every capture file [54].



```

root@LEDE:~# tcpdump -i wlan1 -l
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan1, link-type IEEE802_11_RADIO (802.11 plus radiotap header), ca
pture size 262144 bytes
00:18:56.609498 466256262us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:18:57.609900 467256466us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:18:58.607444 468253808us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:18:59.617237 469263394us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:19:00.614266 470260218us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:19:01.617168 471262912us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:19:02.629915 472275440us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -42dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:19:03.627430 473272750us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -41dBm signal anten
na 1 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28
00:19:04.624454 474269572us tsft 3.0 Mb/s 5890 MHz 11a/10Mhz -41dBm signal anten
na 2 ARP, Request who-has 8.8.8.8 tell 10.10.6.101, length 28

root@LEDE:~#
Interface wlan0 (#0) is up. It may be used.
Interface wlan0 (#0) will be used.

AC_BE is selected.
Sending period set to 1 second(s) and 0.000 microsecond(s).
Timer successfully started. Sending triggered.

^C
root@LEDE:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
^C
--- 8.8.8.8 ping statistics ---
14 packets transmitted, 0 packets received, 100% packet loss
root@LEDE:~# ping 8.8.8.8 -I wlan0
PING 8.8.8.8 (8.8.8.8): 56 data bytes
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
root@LEDE:~# ping 8.8.8.8 -I wlan0
PING 8.8.8.8 (8.8.8.8): 56 data bytes
^C
--- 8.8.8.8 ping statistics ---
7 packets transmitted, 0 packets received, 100% packet loss
root@LEDE:~#

```

Figure 3.17: `tcpdump` running over a monitor interface in the ALIX_100 board, capturing failed ARP requests from the ALIX_101 board, directly using *stdout* over the command line

(2) Using `tcpdump` and `plink`

One useful option for packet sniffing foresees the use of *plink*, which is a command line interface to PuTTY.

We used *plink* to open a session with the board chosen for wireless monitoring, capturing traffic with the *tcpdump* program and sending it to *Wireshark*, running on the Windows development host, through a pipe.

This allowed to have almost real time monitoring of the wireless traffic directly on the development PC, as long as it was connected with the board (through Ethernet).

To simplify the process, a simple batch script, for Windows, has been written:

```
1 @echo off
2 title Wireshark PC Engines capture
3 color F0
4 set /p monIP="Insert the IP address of the board (use the Ethernet
   interface IP here!): "
5 set /p iface="Insert the monitor interface (for ex. 'wlan0'):"
6 echo "IP: %monIP%    Port: 22    Interface: %iface%"
7 :: using tcpdump on:
8 :: -i <specified interface>
9 :: -w <standard output, '-' is used to indicate stdout>
10 :: -U with packet-buffered output
11 :: piping (|) to wireshark with:
12 :: -k (start capture immediately)
13 :: -i - (read from stdin ('-'))
14 plink.exe -l root %monIP% -P 22 "tcpdump -i %iface% -U -w -" | "C:\
   Program Files\Wireshark\Wireshark.exe" -k -i -
15 echo "Script terminated"
16 pause
```

It asks the user to specify the Ethernet IP address of the board monitoring the wireless medium (the same IP address used to connect with PuTTY and SSH) and the interface to be used (in our case wlan1).

While the *tcpdump* options are already commented in the code, the main *plink* options are listed below:

- `-l <username>`: connect with the specified username (e.g. *root*)

- `-pw <password>`: use the specified password, if set on the target board
- `-P <port>`: to connect to the specified port (e.g. 22 for SSH connections)

Figure 3.18 shows a capture session using the script and *Wireshark*, while the C programs (plus some failed ARP requests) presented in the previous section are running.

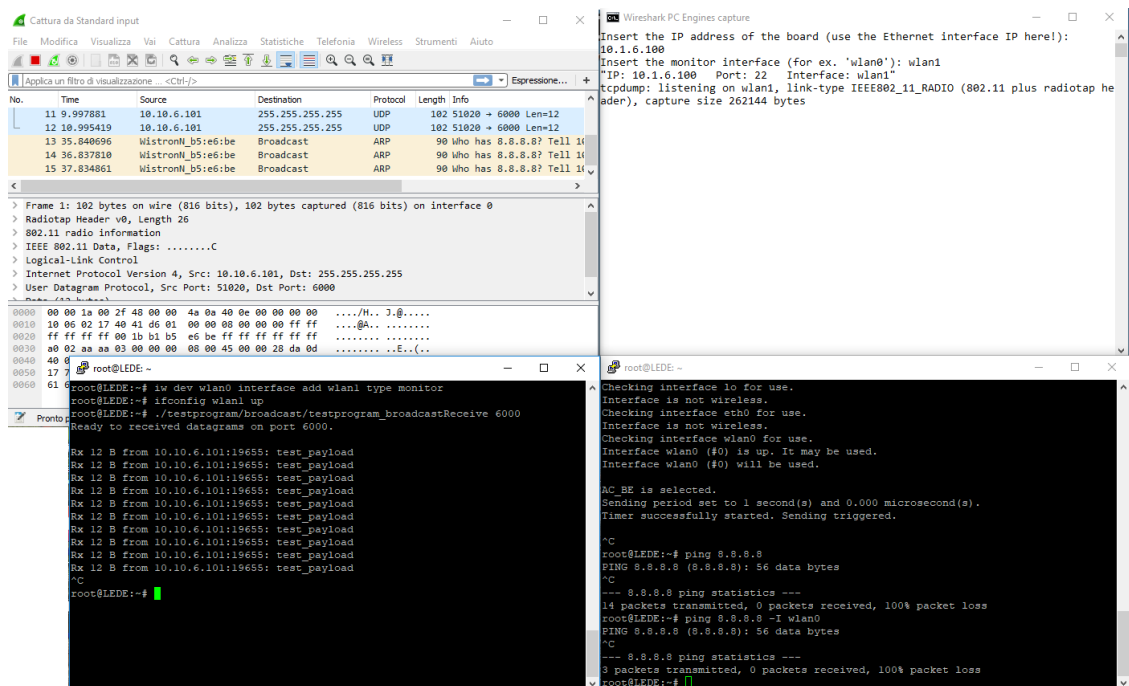


Figure 3.18: Packet sniffing using method (2)

(3) Using iptables-mod-tee

The third possibility, which can be used only when a development PC is connected to the board monitoring the selected channel, is to rely on the *iptables* firewall, with a target extension called *tee*.

This target extension “*will clone a packet and redirect this clone to another machine on the local network segment*” [55], actually cloning, in our case, all the packets received by the board and sending them to the development PC, which should be already inside

the boards subnet when interacting with them through PuTTY or WinSCP.

It is then possible to open *Wireshark* and, through proper filtering, filter out all the SSH information and view the wireless traffic received by the board.

In this case, only the boards needs to be configured. To simplify the setup procedure, an *sh* script has been written, to be run on the board itself. This script contains also the relevant *iptables* commands, used to setup the so called *mangle*¹³ table for packet cloning and forwarding. It will also check whether *iptables* and the *TEE* extension are installed before proceeding with the configuration.

iptables_route

```
1  #!/bin/sh
2
3  # Check correct parameter usage and eventually flush the mangle table
4  if [ $# -ne 3 ]
5  then
6      if [ $# -eq 1 -a $1 = 'flush' ]
7      then
8          echo "Flushing mangle table..."
9          iptables -t mangle -F
10     else
11         echo "Error using program. Expected two parameters."
12         echo "1)  IP address to monitor (IP or 'any')"
13         echo "2)  IP address of the PC running Wireshark"
14         echo "3)  Interface to be used (for ex. wlan0)"
15         echo "One parameter only is accepted only if it is 'flush'"
16         echo " to flush the whole iptables mangle table"
17     fi
18     exit 1
19 fi
20
```

¹³Packet mangling refers to the modification of packets

```
21 # Check if the required package iptables is installed.
22 # If it is not, abort the execution.
23 # You can install it by using opkg install iptables.
24 opkg list | grep -w "iptables" >/dev/null 2>&1
25 if [ $? -ne 0 ]
26 then
27     echo "iptables is not installed. Please install it before
        proceeding further."
28     exit 1
29 fi
30
31 # Check if the required package iptables-mod-tee is installed.
32 # If it is not, abort the execution.
33 # You can install it by using opkg install iptables-mod-tee.
34 opkg list | grep "iptables-mod-tee" >/dev/null 2>&1
35 if [ $? -ne 0 ]
36 then
37     echo "iptables-mod-tee is not installed. Please install it before
        proceeding further."
38     exit 1
39 fi
40
41 # $1 - IP of device to monitor
42 # $2 - IP of gateway (i.e. of the PC running Wireshark)
43 echo "Setting POSTROUTING..."
44 if [ $1 = 'any' ]
45 then
46     iptables -t mangle -A POSTROUTING -o $3 -j TEE --gateway $2
47 else
48     iptables -t mangle -A POSTROUTING -o $3 -s $1 -j TEE --gateway $2
49 fi
50 if [ $? -eq 0 ]
51 then
52     echo "Done."
```

```
53 else
54     echo "iptables -t mangle -A POSTROUTING <...> reported an error.
        The script will terminate now."
55     exit 1
56 fi
57 echo "Setting PREROUTING..."
58 if [ $1 = 'any' ]
59 then
60     iptables -t mangle -A PREROUTING -i $3 -j TEE --gateway $2
61 else
62     iptables -t mangle -A PREROUTING -i $3 -d $1 -j TEE --gateway $2
63 fi
64 if [ $? -eq 0 ]
65 then
66     echo "Done."
67 else
68     echo "iptables -t mangle -A PREROUTING <...> reported an error.
        The script will terminate now."
69     exit 1
70 fi
71
72 echo "iptables routing has been configured for $3 on mangle table,
        using TEE package."
73 echo "On wireshark, connect to the network corresponding to $2."
74 if [ $1 != 'any' ]
75 then
76     echo "You can use as filter: (ip.src == $1) || (ip.dst == $1)"
77 fi
```

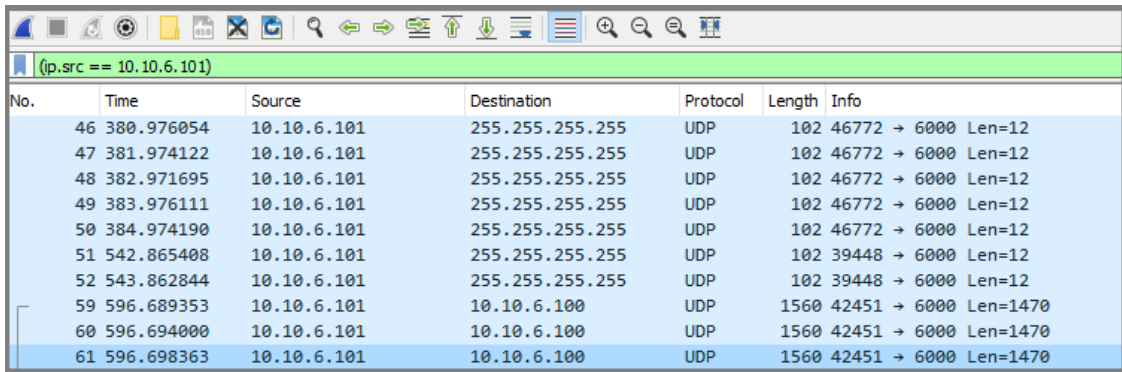
The script accepts as parameters either `flush`, to clear the firewall table and the existing forwarding settings, or a sequence of IP address to use when copying traffic to the development PC (or “any” to monitor any address, including broadcast), IP address of the PC running *Wireshark* (in our case it should be 10.1.11.60, as defined in section 3.7.1)

and the interface to be used.

For instance:

```
./iptables_route any 10.1.11.60 wlan0
```

In order to filter the meaningless traffic inside Wireshark (which should be set to capture on the Ethernet interface where the boards are connected), filters such as IP based ones can be used, as in figure 3.19.



No.	Time	Source	Destination	Protocol	Length	Info
46	380.976054	10.10.6.101	255.255.255.255	UDP	102	46772 → 6000 Len=12
47	381.974122	10.10.6.101	255.255.255.255	UDP	102	46772 → 6000 Len=12
48	382.971695	10.10.6.101	255.255.255.255	UDP	102	46772 → 6000 Len=12
49	383.976111	10.10.6.101	255.255.255.255	UDP	102	46772 → 6000 Len=12
50	384.974190	10.10.6.101	255.255.255.255	UDP	102	46772 → 6000 Len=12
51	542.865408	10.10.6.101	255.255.255.255	UDP	102	39448 → 6000 Len=12
52	543.862844	10.10.6.101	255.255.255.255	UDP	102	39448 → 6000 Len=12
59	596.689353	10.10.6.101	10.10.6.100	UDP	1560	42451 → 6000 Len=1470
60	596.694000	10.10.6.101	10.10.6.100	UDP	1560	42451 → 6000 Len=1470
61	596.698363	10.10.6.101	10.10.6.100	UDP	1560	42451 → 6000 Len=1470

Figure 3.19: Wireshark with filter (ip.src == 10.10.6.101), to display only the traffic generated by the “ALIX_101” board

The drawback of this method is that it can only monitor packets directly received by the board (either targeted to the specific board or multicast/broadcast); that is why we actually preferred method (2).

This method does not seem to work with monitor interfaces, so, it should be used only with managed interfaces, such out wlan0 (not wlan1).

3.9 Packet sniffing: RadioTap

Every time an interface is set in monitor mode it will be able to capture packets at the 802.11 level, including an additional *RadioTap* header.

This additional encapsulation, added during capture, is able to provide the user more

useful information about the captured frame, such as signal level at the antenna (*SSI Signal*), channel frequency, some channel flags telling whether the 2.4 or 5 GHz has been used and the channel width, and so on [56].

It is able, in some sense, to capture the state of the radio medium and of the hardware at the moment in which the packet was received and it can be reviewed using *Wireshark*.

This additional “header” is typically inserted by the driver, which then informs the *mac80211* layer [57].

3.10 OpenWrt 18.06.1

After working with LEDE 17.01, we decided to try to move to the most updated version of OpenWrt, i.e. to OpenWrt 18.06.1, released in August 2018 (less than two months before the moment of writing), with Linux kernel version 4.14.63 (4.14 is the 18th LTS release of the Linux kernel [58]).

We used as a base the OpenC2X LEDE distribution, downloading OpenWrt 18.06.1 and applying all the already existing OpenC2X patches to it, with very few modifications with respect to their LEDE version, which is coded by the CCS Labs team in Paderborn.

We adapted also all the custom patches described before, which required almost no modifications to the code (although some line offsets occurred due to newer versions of the packages).

All the patches, including the original OpenC2X ones adapted to work in the new OpenWrt version, are reported in Appendix B, since they include quite a lot of code.

To ensure compatibility, they all have been recreated using *quilt*, after updating the *quiltmake.config* file with the new directory and target name:

Updated quiltmake.config file

```
/home/francesco/openwrt-18.06.1  
target-i386_pentium_musl
```

Other than the patches presented before, three new patches have been added:

- An existing OpenWrt patch for the new version of `iw` (*200-reduce_size.patch*) was excluding from compilation the `ocb` module, thus making the `ocb leave` and `ocb join` commands unavailable. This patch was probably introduced to reduce the size of `iw` for maintaining support to older devices with low memory [59] [60]. The new patch restores the OCB module when building `iw`.
- The wireless regulatory database file, which was included inside *mac80211*, has been moved and it is now managed in *./package/firmware/wireless-regdb/*. Instead of directly editing the file and placing it as a “custom file” (which is now called *db.txt*), we decided to introduce two patches: one adding the ITS frequencies to IT, and one adding them to DE (in the same way in which there were introduced by the Paderborn University team), to maintain complete compatibility with the OpenC2X platform.

At the moment our OpenWrt 18.06.1 platform does not include the OpenC2X modules as in the previous LEDE solution, but, thanks to OpenC2X standalone, they can actually be inserted and they should work as intended.

A future improvement of this work could consider the integration and testing of these modules also inside the new OpenWrt version.

Since OpenC2X is a very useful software solution for delving into ITS systems, however, it can be important to keep the whole solution compatible with it.

The commands and solutions that can be applied to OpenWrt 18.06.1 are then the same as the ones described before: it is possible to compile C code and run the previously described programs, to do packet sniffing, to perform measurements with *iPerf* just like before, and so on.

When building the system, since no “*./create_config.sh*” script was available (it was included in LEDE by the CCS Labs team), it was necessary, before setting the main configuration with “*make menuconfig*” and in place of running the aforementioned script, to:

1. Set the target only (x86, x86_geode) after issuing “make menuconfig” for a first time
2. Generate the default configuration: “make defconfig”
3. Run again “make menuconfig” to select the desired packages and configuration

Then, during make menuconfig we tried to select all the packages that were present in a default OpenC2X configuration, including all the additional packages presented in 3.8.

Finally, two modifications have been performed inside the OpenWrt build system, other than settings their IP addresses in *etc/config/network* as in 3.7.1 and 3.8.2:

- The custom files described before, such as *iw_startup*, have been included in the *files* folder, just like in LEDE/OpenC2X embedded.
- The *.bashrc* file was updated to point to the new toolchain location; the modified lines are shown in Appendix B.

It can be important to remember that the “poweroff” command can be used to shut down the boards in a safe manner.

On the ALIX boards it seems to work only partially, while on the APU boards it will be able to effectively turn them completely off (just like a desktop PC).

As a final note, when building the OpenWrt image, it is always important to install all the prerequisite packages inside the development PC [49]:

```
sudo apt-get install build-essential subversion libncurses5-dev zlib1g
-dev gawk gcc-multilib flex git-core gettext libssl-dev
```

Chapter 4

Analyzing the DSRC spectrum

4.1 Introduction

Before proceeding further with the work, moving to the newer APU boards, we decided that it could be interesting to use a spectrum analyzer tool, in order to achieve two main results: first, to verify that no other DSRC radio sources were interfering with the boards' communications (this was quite likely, due to DSRC system not being so widespread as of today and due to tests being performed, at this phase, mostly indoors), second, to validate the 10 MHz wide channel usage when one board is transmitting information, verifying also that the guard band is kept free by the UNEX cards.

This will let us discover few problems with the ALIX boards.

These tests can be, of course, repeated for any embedded system running an 802.11p stack, provided that it can give the possibility to change the channel in which the information is transmitted. Possibly any other supported analyzer device can be used too.

Spectrum analyzers are tools that are able to capture signals at different frequencies, thus specializing on the frequency domain (unlike oscilloscopes that are time domain based) thanks to specialized hardware.

These instruments are very important in electronic and telecommunication engineering to test RF circuits and radio transmission. They can be nevertheless used to look at the WiFi frequency spectrum in a certain location, helping for example to detect possible devices interfering with the one used for transmitting information.

Spectrum analyzer can be in a portable or bench-top format. The latter are similar to oscilloscopes analysing the captured spectrum of certain signals: it's the case, for instance, of the Keysight X-Series Signal Analyzers. The evolution of technology allowed, however, to create miniaturized circuits, leading to the birth of portable analyzer almost as precise and powerful as the bench-top lab-grade solutions [61].

The typical plot that is shown by a spectrum analyzer comprises the signal magnitude (usually in dBm) on the y axis and the signal frequency on the x axis.

4.2 MetaGeek Wi-Spy DBx and Chanalyzer

In this work, a professional portable analyzer produced by *MetaGeek* has been used.



Figure 4.1: MetaGeek Wi-Spy DBx

This device is the *Wi-Spy DBx* (version 3) dual band spectrum analyzer. As it is possible to see in figure 4.1 it is composed by the device itself, including an antenna, and an

USB cable to connect it to a PC.

The PC should be running a software able to capture the device raw output and present it to the user in a more readable format.

Wi-Spy DBx is actually targeted at locating source of interferences which are acting against desired WiFi communications and at testing the existing coverage of WiFi networks.

The main characteristics of the device are the following [62]:

- Default frequency range: 2.400 to 2.495 GHz - 5.150 to 5.850 GHz
- Supported frequency range: 2.300 to 2.700 GHz - 4.900 to 5.925 GHz
- Amplitude range: $-100\ dBm$ to $-6.5\ dBm$
- Amplitude resolution: $0.5\ dBm$
- Bandwidth resolution: 58.036 to 812.500 KHz (2.4 GHz band, default: 214.286 KHz) and 53.571 to 750.000 KHz (5 GHz band, default: 464.286 KHz)

The device is actually tuned using the default frequency range, but it can be used and it works well also within the full supported range, which includes all the DSRC frequencies [62].

It comes bundled with a professional software: *Chanalyzer*. This program, running under Windows, is able to capture raw data and show it the user, drawing various plots useful to monitor the different frequencies in the 2.4 and 5 GHz bands, including a typical frequency-magnitude spectrum plot (in which average, peak and current magnitude values are shown) and a 3D plot in which time is plotted on the y axis, frequency on the x axis and the magnitude is shown by means of a heat map (a red marker means high power, a blue one mean low power).

Utilization graphs (with % utilization vs frequency) are made available to the user too and it is possible to record captures that happened in time, directly from within the program.

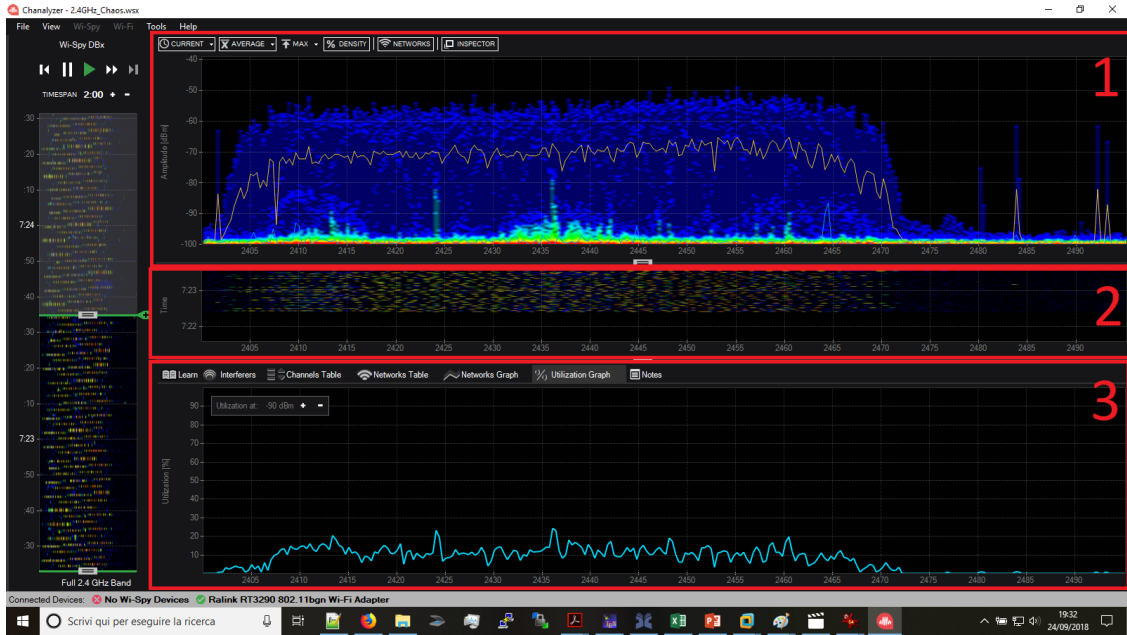


Figure 4.2: MetaGeek Chanalyzer user interface during a capture session on the 2.4 GHz band. It is possible to distinguish (1) a *density view* showing “how often a signal is detected at a specific amplitude” [63] (2) a *waterfall view*, showing also the time axis and (3) a *utilization plot*

This is, however, a commercial program. It has the big advantage of offering professional-grade tools just out of the box, with many options available to the users, but it may lack of flexibility and customizability when it comes to new features which may be needed in research.

Inside Chanalyzer almost every useful feature for WiFi monitoring is already implemented, but in case new features (and/or small customizations) need to be added they should be requested to Metageek and cannot be coded by the user.

Moreover, Chanalyzer fully supports the DSRC frequencies in the 5 GHz range, but they are made available to the user only when a particular plug-in (the so called “*Lab Accessory*”) is bought and installed.

Since this additional license was not available with the standard Chanalyzer license, we decided to look at an open source software solution, interfacing with the Wi-Spy device,

at least for a part of the work with the spectrum analyzer.

4.3 Kismet Spectools

This software solution is represented by Kismet Spectrum-Tools (in short *Spectools*), which is the only spectrum analysis open source program we were able to find.

At the moment of writing it is lacking a good number of useful features which are instead available in the Metageek program (including the very useful session recording tool), but being open source, as stated multiple times in this work, it offers a high degree of flexibility, enabling the user to code its own features when needed and improve this solution, possibly sharing its code with other users.

It is written completely in C and it is released under the GNU GPL license, which enables modifications to the code. It runs under Linux and it uses the GTK and Cairo libraries to create a user interface in which some of the plots presented before are drawn [64].

It includes, as reported in [64], “userspace drivers for the hardware itself, a graphing UI built GTK and Cairo, network protocols for remote device capture, and simple utilities for developing additional tools”.

The planar view of the GUI seems to be very similar to the Chanalyzer one but it computes the average values in a different way, making them less meaningful for our purposes (they are always very low dBm values). The other values are instead computed in the same way as Chanalyzer does.

In a future improvement to this work the way in which Spectrum-Tools computes the average can be studied in a much deeper way, checking why it is giving different values with respect to the Chanalyzer average ones.

The code managing this feature is in any case quite complex.

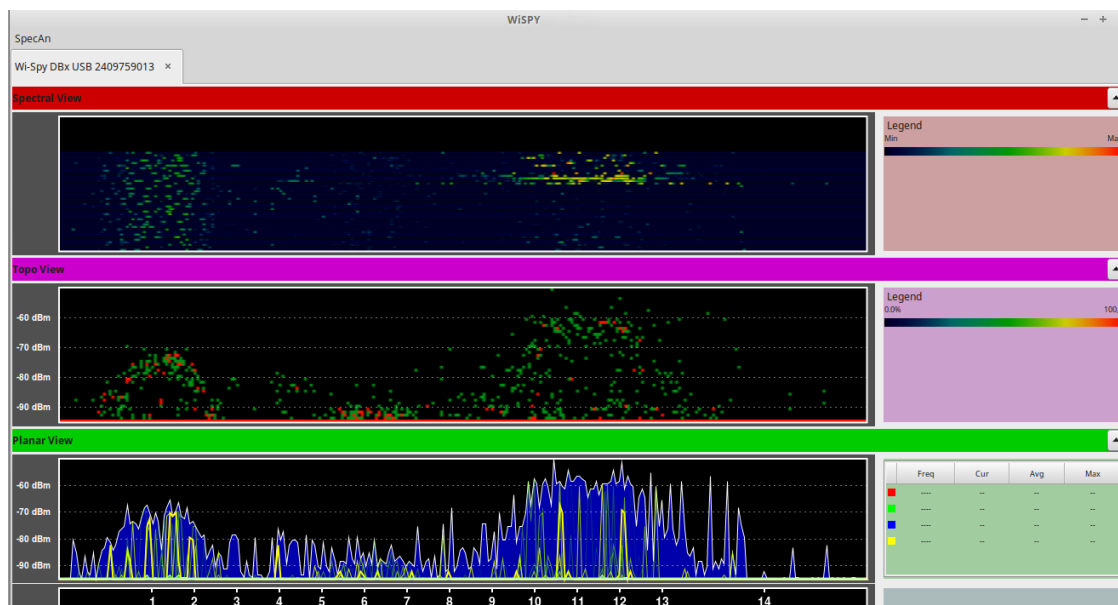


Figure 4.3: Spectrum-Tools GUI, providing three views: a *Spectral View*, equal to the Chanalyzer waterfall view (but with no information provided when moving the cursor over the plot), an additional *Topo View*, showing signal peaks over time at a given frequency and a *Planar View*, similar to the Chanalyzer one

4.3.1 Patching Kismet Spectools

In order to add the support for DSRC frequencies the Spectools software needed to be patched, since it provided only default frequency ranges just after being compiled and executed.

Using the Linux Mint virtual machine running on the development PC, first of all we downloaded the latest development tree of the software, through Git. The following command has been used:

```
git clone https://www.kismetwireless.net/spectools.git
```

The Spectools code was downloaded inside the "spectools" directory.

Moving to the spectools directory, it was possible to prepare the source and generate

the needed Makefiles by using a *GNU autoconf*¹ generated script:

```
cd spectools
./configure
```

It is important to note that the source code compilation requires the USB development library "libusb-dev", otherwise the configuration will fail. In order to install it inside Linux Mint, we used the following "apt-get" command:

```
sudo apt-get install libusb-dev
```

After the configuration, it was possible to build Spectrum-Tools for the first time using:

```
make
```

Every successive compilation was then performed by means of

```
make clean
```

```
make
```

to be sure to recompile all the files.

When compiling Spectrum-Tools, three executables are generated:

- **spectool_curses**: a tool extracting data from Wi-Spy and Ubertooth devices, with a text user interface
- **spectool_raw**: a tool extracting raw data from the devices mentioned above; it can be used to build other spectrum analysis tools and scripts [65]
- **spectool_gui**: the most complex tool, providing a GUI interfacing with the spectrum analyzer and showing the signal magnitude in time and in the frequency domain in a similar way as Chanalyzer does; this work focuses on this program

¹More details are available here: <https://www.gnu.org/software/autoconf/autoconf.html> - "*Autoconf is an extensible package [...] that produce shell scripts to automatically configure software source code packages*"

Part of the source code was then analyzed and patched in order to add the following features to *spectool_gui*:

- Possibility to choose the dBm ranges in the plots which are drawn in the GUI. Before, the range was fixed to $-95\text{ dBm} \rightarrow -50\text{ dBm}$. These ranges are a window parameter, so they can be changed on any new window without affecting the other open ones.
- Support for the DSRC frequencies (channel 172-184), adding the possibility to highlight the 10 MHz wide IEEE 802.11p channels in the planar view and inserting two new ranges for capture sessions: one including almost the full 5 GHz band and adding the 802.11p channels and one specifically showing the DSRC channels only.

In order to launch the GUI of Spectrum-Tools in a faster way the following line was added to the user *.bashrc* file, defining an alias every time the shell is started:

.bashrc

```
#set alias to run spectools gui
alias spectoolgui="sudo ~/spectools/spectool_gtk"
```

The alias already includes the *"sudo"* command to launch the program with root privileges; this is needed in order to avoid *"Operation not permitted"* errors.

The patched code is presented below: for each file the most significant lines of code for the patching work are shown. All the added or edited lines are introduced by in-line comments starting with *\\Patch: .*

wispy_hw_dbx.c

wispy_hw_dbx.c

```
315  if (model == WISPYDBx_MODEL_DBxV1 ||
316  model == WISPYDBx_MODEL_DBxV2 ||
317  model == WISPYDBx_MODEL_DBxV3) {
```

```

318 // DBX devices get 2.4, 2.4 turbo, 5, 5-1, 5-2, 5-3
319 *num_ranges = 8; // Patch: increased by 2 the number of ranges for
    WiSpy DBx devices (it was 6)
320 } else if (model == WISPYDBx_MODEL_24i ||
321 model == WISPYDBx_MODEL_24xV2) {
322 // 24i and x are 2.4 only, and get just the 2.4 and 2.4 turbo
323 *num_ranges = 2;
324 } else if (model == WISPYDBx_MODEL_900x ||
325 model == WISPYDBx_MODEL_900xV2) {
326 // 900x series gets 2 900mhz-ish ranges
327 *num_ranges = 2;
328 } else if (model == WISPYDBx_MODEL_950x) {
329 // 950x only does one range
330 *num_ranges = 1;
331 } else {
332 // We've failed somehow
333 fprintf(stderr, "FAILURE: Couldn't determine device model\n");
334 return;
335 }
336
337 *ranges = (spectool_sample_sweep *) malloc(sizeof(
    spectool_sample_sweep) * *num_ranges);
338
339 if (model == WISPYDBx_MODEL_DBxV1 ||
340 model == WISPYDBx_MODEL_DBxV2 ||
341 model == WISPYDBx_MODEL_DBxV3 ||
342 model == WISPYDBx_MODEL_24i ||
343 model == WISPYDBx_MODEL_24xV2) {
344
345 wispydbx_create_settings_from_preset(&((*ranges)[0]),
346 "Full 2.4GHz Band", 2400.0f, 2495.0f,
347 333.3f, 200, model);
348 wispydbx_create_settings_from_preset(&((*ranges)[1]),
349 "Full 2.4GHz Band (Turbo)", 2400.0f, 2495.0f,

```

```

350     1000.0f, 500, model);
351 }
352
353 if (model == WISPYDBx_MODEL_DBxV1 ||
354     model == WISPYDBx_MODEL_DBxV2 ||
355     model == WISPYDBx_MODEL_DBxV3) {
356     wispydbx_create_settings_from_preset(&((*ranges)[2]),
357     "Full 5GHz Band", 5150.0f, 5836.0f,
358     1497.070f, 428, model);
359     wispydbx_create_settings_from_preset(&((*ranges)[3]),
360     "UNII Low/Mid (ch. 36-64)", 5150.0f, 5350.0f,
361     748.535f, 428, model);
362     wispydbx_create_settings_from_preset(&((*ranges)[4]),
363     "UNII Low/Mid (ch. 100-140)", 5470.0f, 5725.0f,
364     1122.070f, 428, model);
365     wispydbx_create_settings_from_preset(&((*ranges)[5]),
366     "UNII Low/Mid (ch. 149-165)", 5725.0f, 5836.0f,
367     375.0f, 428, model);
368     // Patch: add a full 5 GHz range, including DSRC frequencies (U-NII
369     // -4)
370     wispydbx_create_settings_from_preset(&((*ranges)[6]),
371     "UNII Low/Mid/DSRC (ch. 149-184)", 5150.0f, 5925.0f,
372     1497.070f, 428, model);
373     // Patch: add a DSCR/ITS range (U-NII-4 only) - the frequency
374     // resolution parameter actually
375     // influences the accuracy at which frequencies are read and
376     // requires more investigation/fine-tuning
377     // setting default value from https://support.metageek.com/hc
378     // /en-us/articles/203802010-Wi-Spy-Data-Sheet
379     wispydbx_create_settings_from_preset(&((*ranges)[7]),
380     "UNII DSRC/ITS (ch. 172-184)", 5850.0f, 5925.0f,
381     464.286f, 428, model);
382 }

```

This code is used by Spectrum-Tools to manage the Wi-Spy DBx device, acting as a driver for the spectrum analyzer. Through the function:

```
307 void wispydbx_create_settings_from_preset(spectool_sample_sweep *range
    , char *name, float startFrequencyMHz, float stopFrequencyMHz,
    float frequencyResolutionkHz, float filterBandwidthkHz, int model)
```

new settings for the DBx device, such as frequency ranges, are defined. Every setting is then written into a `spectool_sample_sweep` structure, which is used as a base for capturing and managing data coming from Wi-Spy DBx.

In order to enable the selection of ITS frequencies, which are nevertheless supported by the analyzer, two additional calls to `wispydbx_create_settings_from_preset()` have been added. These calls are added just after the other similar ones, inside the `wispydbx_add_supportedranges()` function, acting as a “*channel range adder*” for the device.

This is sufficient to enable a very basic support for the 802.11p channels.

spectool_container.h

In order to allow these channels (from 172 to 184) to be displayed and correctly highlighted in the GUI (taking into account that they are 10 MHz wide), another file has been modified.

spectool_container.h

```
248 struct spectool_channels {
249     /* Name of the channel set */
250     char *name;
251     /* Start and end khz for matching */
252     int startkhz;
253     int endkhz;
254     /* Number of channels */
255     int chan_num;
256     /* Offsets in khz */
257     int *chan_freqs;
```

```

258  /* Width of channels in khz */
259  int chan_width;
260  /* Text of channel numbers */
261  char **chan_text;
262 };
263
264 /* Some channel lists */
265 static int chan_freqs_24[] = {
266     2411000, 2416000, 2421000, 2426000, 2431000, 2436000, 2441000,
267     2446000, 2451000, 2456000, 2461000, 2466000, 2471000, 2483000
268 };
269
270 static char *chan_text_24[] = {
271     "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13",
272     "14"
273 };
274
275 static int chan_freqs_5[] = {
276     5180000, 5200000, 5220000, 5240000, 5260000, 5280000, 5300000,
277     5320000,
278     5500000, 5520000, 5540000, 5560000, 5580000, 5600000, 5620000,
279     5640000,
280     5660000, 5680000, 5700000, 5745000, 5765000, 5785000, 5805000,
281     5825000
282 };
283
284 static char *chan_text_5[] = {
285     "36", "40", "44", "48", "52", "56", "60", "64", "100", "104",
286     "108", "112", "116", "120", "124", "128", "132", "136", "140",
287     "149", "153", "157", "161", "165"
288 };
289
290 // Add ITS frequencies (patch)
291 static int chan_freqs_ITS[] = {

```



```

288     5860000, 5870000, 5880000, 5890000, 5900000, 5910000, 5920000
289 };
290
291 // Add ITS frequencies (patch)
292 static char *chan_text_ITS[] = {
293     "172", "174", "176", "178", "180", "182", "184"
294 };
295
296 static int chan_freqs_900[] = {
297     905000, 910000, 915000, 920000, 925000
298 };
299
300 static char *chan_text_900[] = {
301     "905", "910", "915", "920", "925"
302 };
303
304 /* Allocate all our channels in a big nasty array */
305 static struct spectool_channels channel_list[] = {
306     { "802.11b/g", 2400000, 2483000, 14, chan_freqs_24, 22000,
307       chan_text_24 },
308     { "802.11b/g", 2402000, 2480000, 14, chan_freqs_24, 22000,
309       chan_text_24 },
310     { "802.11a", 5100000, 5832000, 24, chan_freqs_5, 20000, chan_text_5
311       },
312     { "802.11a UN-II", 5100000, 5483000, 14, chan_freqs_5, 20000,
313       chan_text_5 },
314     { "900 ISM", 902000, 927000, 5, chan_freqs_900, 5000, chan_text_900
315       },
316     { "802.11p", 5860000, 5920000, 7, chan_freqs_ITS, 10000,
317       chan_text_ITS }, // Add ITS frequencies (patch)
318     { NULL, 0, 0, 0, NULL, 0, NULL }
319 };
320
321 // Patch: struct definition for managing the min and max dBm settings

```

```

316 typedef struct _dbmranges {
317     int min;
318     int max;
319 } dbmranges;
320
321 #endif

```

This file (together with the linked `.c` file) acts a container class, where structures related to the available channels are defined, among other things, including structures used for sample sweeps and device data/API callbacks management.

The structure array `static struct spectool_channels channel_list[]` contains a list of available channels. Each member of the array is a structure, as can be seen from lines 248-262 of the displayed source code, containing the name of the channel set, the frequency range, the number of channels, the central frequencies in terms of an integer array (`int *chan_freqs;`), the channel width and the text to be shown for each channel when they are displayed inside the GUI. All the frequencies are specified in *kHz*.

The modified code inserted two additional arrays containing the central frequencies of the ITS channels and their IEEE numbers as strings to be displayed in the GUI.

These arrays are then used to define a new element inside the structure array mentioned before, choosing a 10 *MHz* (10000 *kHz*) channel width as requested by 802.11p.

The size of the array is not initially specified and the array is scanned inside `spectool_gtk_channel.c` using a `for` loop, until a terminating `NULL` element is found. This allowed to add elements inside the array with flexibility, without the need of doing anything else than adding a new line in its definition.

The `for` loop mentioned before is the following one:

`spectool_gtk_channel.c`

```

96 /* Try to figure out the channels we use for this spectrum */
97 for (x = 0; wwidget->chanopts != NULL &&
98     channel_list[x].name != NULL && wwidget->chanopts->chanset == NULL;
99     x++) {

```

A new `dbmranges` structure has been defined too, and it is used to store the minimum and maximum dBm ranges which, after the patch, are completely selectable from the GUI.

spectool_gtk.c

The dBm ranges patch required more work and more files to be modified. The relevant lines of code are presented below and then briefly described.

spectool_gtk.c

```
84 typedef struct _nb_aux {
85     GtkWidget *nbvbox, *nodev_vbox;
86     GtkWidget *nblabel;
87     GtkWidget *mindbm_gtk, *maxdbm_gtk; // Patch
88     wg_aux *auxptr;
89
90     GtkWidget *planar, *spectral, *topo, *channel;
91     SpectoolWidgetController *p_con, *s_con, *t_con;
92
93     gint pagenum;
94
95     SpectoolChannelOpts *chanopts;
96
97     spectool_phy *phydev;
98     int wdr_slot;
99     GList *wdr_menu;
100 } nb_aux;
101
102 /* fwd defs */
103 static nb_aux *build_nb_page(GtkWidget *notebook, wg_aux *auxptr);
104
105 static void main_devopen(int slot, void *aux) {
106     nb_aux *nbaux = (nb_aux *) aux;
```

```

107 dbmranges range; // Patch
108
109 g_return_if_fail(aux != NULL);
110
111 // Patch: read the dBm ranges from the GUI
112 range.min=atoi(gtk_combo_box_text_get_active_text(nbaux->mindbm_gtk)
113 );
114 range.max=atoi(gtk_combo_box_text_get_active_text(nbaux->maxdbm_gtk)
115 );
116
117 // Patch: a new function spectool_widget_set_ranges accepts as
118 // parameters the GtkWidget and
119 // a dbmranges structure (defined in spectool_container.h)
120 spectool_widget_set_ranges(nbaux->planar,range); // Patch
121 spectool_widget_set_ranges(nbaux->topo,range); // Patch
122 spectool_widget_set_ranges(nbaux->spectral,range); // Patch
123 spectool_widget_set_ranges(nbaux->channel,range); // Patch
124
125 spectool_widget_bind_dev(nbaux->planar, nbaux->auxptr->wdr, slot);
126 spectool_widget_bind_dev(nbaux->topo, nbaux->auxptr->wdr, slot);
127 spectool_widget_bind_dev(nbaux->spectral, nbaux->auxptr->wdr, slot);
128 spectool_widget_bind_dev(nbaux->channel, nbaux->auxptr->wdr, slot);
129
130 nbaux->phydev = wdr_get_phy(nbaux->auxptr->wdr, slot);

```

spectool_gtk.c

```

260 static nb_aux *build_nb_page(GtkWidget *notebook, wg_aux *auxptr) {
261     nb_aux *nbaux = (nb_aux *) malloc(sizeof(nb_aux));
262     GtkWidget *temp, *hbox, *arrow, *closebutton, *closeicon;
263     // Patch: minimum dBm values allowed (it is sufficient to change
264     // this in order to add new admitted values)
265     // Valid values: from -100 to -70
266     const char *mindBm[] = {"-100", "-95", "-90"};

```

```

266 // Patch: maximum dBm values allowed (it is sufficient to change
    this in order to add new admitted values)
267 // Valid values: from -50 to 0
268 const char *maxdBm[] = {"-50","-40","-30","-20","-10"};
269 // Patch: index to iterate through the previously defined arrays
270 int i;
271 nbaux->auxptr = auxptr;
272
273 /* Default label for the tab, packed into a hbox */
274 hbox = gtk_hbox_new(FALSE, 1);
275 gtk_widget_show(hbox);
276 nbaux->nblabel = gtk_label_new("No device");
277 gtk_box_pack_start(GTK_BOX(hbox), nbaux->nblabel, FALSE, FALSE, 0);
278
279 closebutton = gtk_button_new();
280 closeicon = gtk_image_new_from_stock(GTK_STOCK_CLOSE,
    GTK_ICON_SIZE_MENU);
281 gtk_container_add(GTK_CONTAINER(closebutton), closeicon);
282 gtk_button_set_relief(GTK_BUTTON(closebutton), GTK_RELIEF_NONE);
283 gtk_widget_show(closebutton);
284 gtk_widget_show(closeicon);
285 gtk_box_pack_end(GTK_BOX(hbox), closebutton, FALSE, FALSE, 0);
286
287 gtk_signal_connect(GTK_OBJECT(closebutton), "clicked",
288 GTK_SIGNAL_FUNC(close_nb_button), nbaux);
289
290 nbaux->nbvbox = gtk_vbox_new(FALSE, 1);
291 nbaux->pagenum = gtk_notebook_append_page(GTK_NOTEBOOK(notebook),
    nbaux->nbvbox,
292 hbox);
293
294 /* Make the device picker buttons and label */
295 nbaux->nodev_vbox = gtk_vbox_new(FALSE, 0);
296 temp = gtk_label_new("No device selected...");

```

```
297     gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), temp, FALSE, FALSE,
298         4);
299
300     /* Build the arrow for using an open device */
301     hbox = gtk_hbox_new(FALSE, 0);
302     gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), hbox, FALSE, FALSE,
303         2);
304
305     temp = gtk_button_new_with_label("Open Device");
306     g_signal_connect_swapped(G_OBJECT(temp), "clicked",
307         G_CALLBACK(main_menu_spawnpicker),
308         nbaux);
309     gtk_box_pack_start(GTK_BOX(hbox), temp, TRUE, TRUE, 0);
310     gtk_widget_show(temp);
311
312     temp = gtk_button_new();
313     arrow = gtk_arrow_new(GTK_ARROW_DOWN, GTK_SHADOW_OUT);
314     gtk_container_add(GTK_CONTAINER(temp), arrow);
315     g_signal_connect_swapped(G_OBJECT(temp), "event",
316         G_CALLBACK(main_nodev_menu_button_press),
317         nbaux);
318     gtk_box_pack_start(GTK_BOX(hbox), temp, FALSE, FALSE, 0);
319     gtk_widget_show(temp);
320     gtk_widget_show(hbox);
321     gtk_widget_show(arrow);
322
323     temp = gtk_button_new_with_label("Open Network Device");
324     g_signal_connect_swapped(G_OBJECT(temp), "clicked",
325         G_CALLBACK(main_menu_spawnnetpicker),
326         nbaux);
327     gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), temp, FALSE, FALSE,
328         2);
329     gtk_widget_show(temp);
```

```
328
329 // Patch: adding combo box for setting minimum dBm value
330 hbox = gtk_hbox_new(FALSE, 0);
331 gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), hbox, FALSE, FALSE,
    2);
332
333 temp = gtk_label_new("Minimum dBm value (default: -95 dBm): ");
334 gtk_box_pack_start(GTK_BOX(hbox), temp, FALSE, FALSE, 2);
335 gtk_widget_show(temp);
336
337 nbaux->mindbm_gtk = gtk_combo_box_text_new();
338
339 gtk_box_pack_start(GTK_BOX(hbox), nbaux->mindbm_gtk, FALSE, FALSE,
    2);
340 for (i=0; i<G_N_ELEMENTS(mindBm); i++) {
341     gtk_combo_box_text_append_text (GTK_COMBO_BOX_TEXT(nbaux->
        mindbm_gtk), mindBm[i]);
342 }
343
344 gtk_combo_box_set_active(GTK_COMBO_BOX(nbaux->mindbm_gtk), 1); //
    -95 dBm is the default value
345 gtk_widget_show(nbaux->mindbm_gtk);
346
347 gtk_widget_show(hbox);
348
349 // Patch: adding combo box for setting maximum dBm value
350 hbox = gtk_hbox_new(FALSE, 0);
351 gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), hbox, FALSE, FALSE,
    2);
352
353 temp = gtk_label_new("Maximum dBm value (default: -50 dBm): ");
354 gtk_box_pack_start(GTK_BOX(hbox), temp, FALSE, FALSE, 2);
355 gtk_widget_show(temp);
356
```

```

357  nbaux->maxdbm_gtk = gtk_combo_box_text_new();
358
359  gtk_box_pack_start(GTK_BOX(hbox), nbaux->maxdbm_gtk, FALSE, FALSE,
360                      2);
361  for (i=0; i<G_N_ELEMENTS(maxdBm); i++) {
362      gtk_combo_box_text_append_text (GTK_COMBO_BOX_TEXT(nbaux->
363          maxdbm_gtk), maxdBm[i]);
364  }
365
366  gtk_combo_box_set_active(GTK_COMBO_BOX(nbaux->maxdbm_gtk), 0); //
367      -50 dBm is the default value
368
369  gtk_widget_show(nbaux->maxdbm_gtk);
370
371  gtk_widget_show(hbox);
372
373  /*
374  temp = gtk_button_new_with_label("Close Tab");
375  g_signal_connect_swapped(G_OBJECT(temp), "clicked",
376  G_CALLBACK(gtk_widget_destroy), G_OBJECT());
377  gtk_box_pack_start(GTK_BOX(nbaux->nodev_vbox), temp, FALSE, FALSE,
378                      2);
379  gtk_widget_show(temp);
380  */
381
382  gtk_box_pack_start(GTK_BOX(nbaux->nbvbox), nbdev_vbox, FALSE,
383                      FALSE, 0);
384
385  gtk_widget_show(nbaux->nodev_vbox);
386
387  gtk_widget_show(nbaux->nblabel);
388
389  /* Make the inactive devices */
390  nbdev_opts = (SpectoolChannelOpts *) malloc(sizeof(
391      SpectoolChannelOpts));

```



```
385 spectoolchannelopts_init(nbaux->chanopts);
386
387 nbaux->channel = spectool_channel_new();
388 spectool_widget_link_channel(nbaux->channel, nbaux->chanopts);
389 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->channel, FALSE,
    FALSE, 0);
390
391 nbaux->planar = spectool_planar_new();
392 spectool_widget_link_channel(nbaux->planar, nbaux->chanopts);
393 nbaux->p_con = spectool_widget_buildcontroller(GTK_WIDGET(nbaux->
    planar));
394 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->planar, TRUE, TRUE,
    0);
395 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->p_con->evbox, FALSE,
    FALSE, 2);
396
397 nbaux->topo = spectool_topo_new();
398 spectool_widget_link_channel(nbaux->topo, nbaux->chanopts);
399 nbaux->t_con = spectool_widget_buildcontroller(GTK_WIDGET(nbaux->
    topo));
400 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->topo, TRUE, TRUE, 0)
    ;
401 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->t_con->evbox, FALSE,
    FALSE, 2);
402
403 nbaux->spectral = spectool_spectral_new();
404 spectool_widget_link_channel(nbaux->spectral, nbaux->chanopts);
405 nbaux->s_con = spectool_widget_buildcontroller(GTK_WIDGET(nbaux->
    spectral));
406 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->spectral, TRUE, TRUE
    , 0);
407 gtk_box_pack_end(GTK_BOX(nbaux->nbvbox), nbaux->s_con->evbox, FALSE,
    FALSE, 2);
408
```

```
409 spectool_channel_append_update(nbaux->channel , nbaux->planar);
410 spectool_channel_append_update(nbaux->channel , nbaux->topo);
411 spectool_channel_append_update(nbaux->channel , nbaux->spectral);
412
413 gtk_widget_show(nbaux->nbvbox);
414
415 auxptr->num_tabs++;
416
417 return nbaux;
418 }
```

spectool_gtk.c

```
472 gtk_window_set_title(GTK_WINDOW(window), "WiSPY (Patch)"); //
    Patch: now the title shows that this is a patched version
```

This file contains the code managing the GTK graphical interface of *spectool_gui*.

It is using the GTK+ library², providing a toolkit for coding graphical user interfaces.

GTK+ is based itself on *GObject (GLib Object System)*, which provides an object based system to non object-oriented languages, such as C.

GUIs are based in this case on “GTK widgets”, such as text boxes, combo boxes, arrows, vertical and horizontal boxes acting as containers for other widgets, and so on.

The main modifications, as it can be seen from the reported code, are the following:

- The function `main_devopen()`, which is called every time a device is opened from the GUI, now reads the minimum and maximum dBm values to be displayed from two GTK “*Combo Box*” widgets³. These values are stored inside inside a `dbmranges` structures, as explained before.

They are then passed as second argument to a newly defined function

`spectool_widget_set_ranges()`, which sets these values as attributes of the

²Home page: <https://www.gtk.org/>

³Added later on inside the code

GtkWidget passed as first argument. Being them specific to a certain GtkWidget graphical element, they fulfill the goal of being window independent.

In order to store the pointers to the combo boxes containing the user settings for the dBm ranges, two new elements have been added to the `nb_aux` structure. This is needed since the GUI is built from within another function and it is necessary to keep track of the pointers to the proper widgets in order to retrieve the dBm ranges data.

- `build_nb_page()` is the function which builds the GUI. Two new string arrays are defined to set the user choices for the minimum and maximum dBm values to be displayed.

Defining these two arrays at the beginning of the function makes the code quite flexible for future fast modifications, since it is sufficient to add a value within the allowed range (highlighted in the comments) to effectively enable a new admitted value. Inside this function the new combo boxes are defined, containing the choice for the minimum and maximum dBm values, which are obtained scanning the two previously defined string arrays.

It is possible to notice that `gtk_hbox_new()` has been used to create a container for the text label (“*Minimum/Maximum dBm value (default: -xx dBm)*”) and the linked combo box. This function is now deprecated and should be substituted with `gtk_grid_new()`, creating a new and more up-to-date type of container. We decided, however, to keep the old function in order to maintain consistency with the rest of the code.

- One last small modification was related to the window title: it was changed adding “(Patch)” in order to clearly distinguish the patched `spectool_gui` version.

spectool_gtk_widget.h/c

spectool_gtk_widget.h

```
77 struct _SpectoolWidget {
78     GtkBinClass parent;
79
80     int hlines;
81     /* Top of DB graph, max sample reported */
82     int base_db_offset;
83     /* Bottom of db graph, min sample reported */
84     int min_db_draw;
85
86     /* Patch: added to store in a SpectoolWidget the dBm ranges read
87        from the GUI
88
89     /* Conversion data */
90     int amp_offset_mdbm;
91     int amp_res_mdbm;
92
93     /* Graph elements we've calculated */
94     int g_start_x, g_start_y, g_end_x, g_end_y,
95     g_len_x, g_len_y;
96     int dbm_w;
97     double wbar;
98
99     gint timeout_ref;
```

spectool_gtk_widget.c

```
260 static void spectool_widget_wdr_sweep(int slot, int mode,
261 spectool_sample_sweep *sweep, void *aux) {
262     SpectoolWidget *wwidget;
263
```

```

264 g_return_if_fail(aux != NULL);
265 g_return_if_fail(IS_SPECTOOL_WIDGET(aux));
266
267 wwidget = SPECTOOL_WIDGET(aux);
268
269 wwidget->dirty = 1;
270
271 /* Generic sweep handler to add it to our cache, all things get this
272    */
273 if ((mode & SPECTOOL_POLL_ERROR)) {
274     wwidget->phydev = NULL;
275     if (wwidget->sweepcache != NULL) {
276         spectool_cache_free(wwidget->sweepcache);
277         wwidget->sweepcache = NULL;
278     }
279     wdr_del_ref(wwidget->wdr, wwidget->wdr_slot);
280     wwidget->wdr_slot = -1;
281 } else if ((mode & SPECTOOL_POLL_CONFIGURED)) {
282     if (wwidget->sweepcache != NULL) {
283         spectool_cache_free(wwidget->sweepcache);
284         wwidget->sweepcache = NULL;
285     }
286
287     if (wwidget->sweep_num_samples > 0) {
288         wwidget->sweepcache =
289             spectool_cache_alloc(wwidget->sweep_num_samples,
290                                 wwidget->sweep_keep_avg,
291                                 wwidget->sweep_keep_peak);
292     }
293
294     wwidget->amp_offset_mdbm =
295         spectool_phy_getcurprofile(wwidget->phydev)->amp_offset_mdbm;
296     wwidget->amp_res_mdbm =
297         spectool_phy_getcurprofile(wwidget->phydev)->amp_res_mdbm;

```

```

297
298     /*
299     wwidge->base_db_offset =
300         SPECTOOL_RSSI_CONVERT(wwidge->amp_offset_mdbm, wwidge->
301             amp_res_mdbm,
302             spectool_phy_getcurprofile(wwidge->phydev)->rssi_max);
303     */
304     // Patch: now dBm draw and offset are set from the GUI
305     wwidge->base_db_offset = wwidge->dbm_store.max;
306     wwidge->min_db_draw = wwidge->dbm_store.min;
307     //wwidge->base_db_offset = -50;
308     // wwidge->min_db_draw = SPECTOOL_RSSI_CONVERT(wwidge->
309         amp_offset_mdbm, wwidge->amp_res_mdbm, 0);
310     // printf("debug - min db draw %d\n", wwidge->min_db_draw);
311     //wwidge->min_db_draw = -95;
312     //}
313
314 } else if (wwidge->sweepcache != NULL && sweep != NULL) {
315     spectool_cache_append(wwidge->sweepcache, sweep);
316     /*
317     wwidge->min_db_draw =
318     SPECTOOL_RSSI_CONVERT(wwidge->amp_offset_mdbm, wwidge->
319         amp_res_mdbm,
320         sweep->min_rssi_seen > 2 ?
321         sweep->min_rssi_seen - 2: sweep->min_rssi_seen);
322     */
323     // Patch: now dBm draw and offset are set from the GUI
324     //wwidge->min_db_draw = -95;
325     wwidge->min_db_draw = wwidge->dbm_store.min;
326 }
327
328 /* Call the secondary sweep handler */
329 if (wwidge->wdr_sweep_func != NULL)
330     (*(wwidge->wdr_sweep_func))(slot, mode, sweep, aux);

```

328 }

spectool_gtk_widget.c

```

1075 // Patch: function to set the dBm ranges in a SpectoolWidget
1076 void spectool_widget_set_ranges(GtkWidget *widget, dbmranges ranges) {
1077     SpectoolWidget *wwidget;
1078
1079     // Check for correct object type
1080     g_return_if_fail(widget != NULL);
1081     g_return_if_fail(IS_SPECTOOL_WIDGET(widget));
1082
1083     // Check for correct ranges being read
1084     g_return_if_fail(ranges.min >= -100 && ranges.min <= -70);
1085     g_return_if_fail(ranges.max >= -50 && ranges.max <= 0);
1086
1087     wwidget = SPECTOOL_WIDGET(widget);
1088
1089     wwidget->dbm_store=ranges;
1090 }

```

The modifications to this file include:

- The definition of the new function `spectool_widget_set_ranges()`, which sets the dBm ranges as attributes of the GTK widget specified as first argument. This pointer actually refers to a *SpectoolWidget* object, which is derived from *GtkBin*, which is itself, in the object hierarchy, derived from *GtkWidget*. This object actually includes some additional attributes, including the newly inserted dBm ranges, as it is possible to see from line 87 of *spectool_gtk_widget.h*.
- The function `spectool_widget_wdr_sweep()` now uses the dBm ranges contained in the specified *SpectoolWidget* structure, instead of the previously fixed values of -95 and -50. Since this function manages the data captured from the device

and it is used at least a couple of times as a callback, its structure has not been modified and the dBm ranges are set directly in `spectool_widget_set_ranges()`.

spectool_gtk_planar.c,spectool_gtk_topo.c

spectool_gtk_planar.c

```
741 static void spectool_planar_init(SpectoolPlanar *planar) {
742     SpectoolWidget *wwidget;
743     GtkWidget *scrollwindow;
744
745     GtkCellRenderer *cell;
746     GtkTreeViewColumn *column;
747     GtkTreeIter iter;
748     GtkTreeSelection *selection;
749
750     GtkWidget *clabel;
751     PangoAttrList *attr_lst;
752     PangoAttribute *attr;
753
754     GtkWidget *bhb;
755
756     GdkColor c;
757     GtkStyle *style;
758
759     spectool_planar_marker *mkr;
760
761     wwidget = SPECTOOL_WIDGET(planar);
762
763     wwidget->sweep_num_samples = SPECTOOL_PLANAR_NUM_SAMPLES;
764     wwidget->sweep_keep_avg = 1;
765     wwidget->sweep_keep_peak = 1;
766
767     wwidget->sweep_num_aggregate = 2;
```



```
768
769     wwidget->hlines = 8;
770     //wwidget->base_db_offset = -50; // Patch: commented this out (it
       should be unnecessary)
771
772     wwidget->graph_title = "<b>Planar View</b>";
773     wwidget->graph_title_bg = "#00CC00";
774     wwidget->graph_control_bg = "#A0CCA0";
```

spectool_gtk_topo.c

```
457 static void spectool_topo_init(SpectoolTopo *topo) {
458     SpectoolWidget *wwidget;
459
460     GtkWidget *temp;
461     GtkWidget *legendv, *legendh;
462     PangoAttrList *attr_list;
463     PangoAttribute *attr;
464
465     wwidget = SPECTOOL_WIDGET(topo);
466
467     wwidget->sweep_num_samples = 60;
468
469     wwidget->sweep_keep_avg = 0;
470     wwidget->sweep_keep_peak = 0;
471
472     /* Aggregate a big chunk of sweeps for each peak so we do less
       processing */
473     wwidget->sweep_num_aggregate = 1;
474
475     wwidget->hlines = 8;
476     //wwidget->base_db_offset = -50; // Patch: commented this out (it
       should be unnecessary)
477
```

```

478 wwidget->graph_title = strdup("<b>Topo View</b>");
479 wwidget->graph_title_bg = strdup("#CC00CC");
480 wwidget->graph_control_bg = strdup("#CCA0CC");

```

The modifications to these files, defining the *SpectoolTopo* and *SpectoolPlanar* objects, derived from *SpectoolWidget*, are very marginal.

The only modification involves commenting out the `wwidget->base_db_offset = -50;` initialization lines, which should be no more necessary, since the maximum dBm value is now specified by the user through the GUI.

4.3.2 Results of the patching work

Launching the program with `spectoolgui`, it is possible to see that the user interface is showing two new options for setting the dBm ranges in the plots.

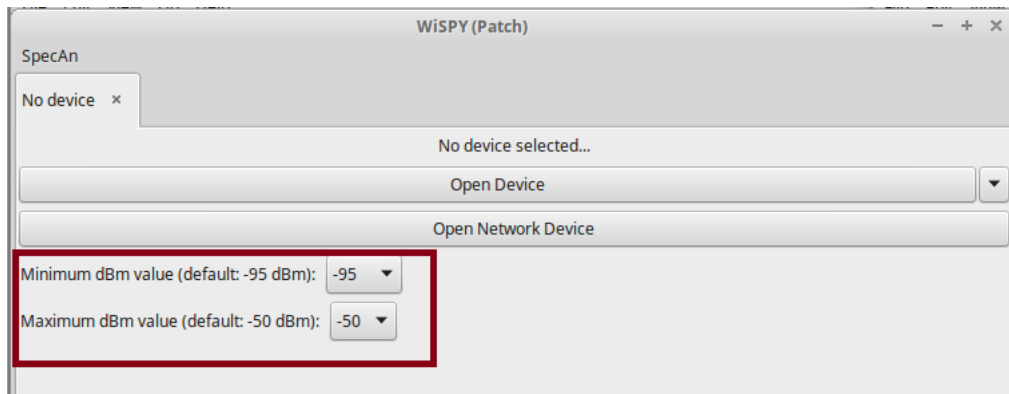


Figure 4.4: Spectrum-Tools GUI, now showing the options for the dBm values

Moreover, when connecting the Wi-Spy device to the PC, two new options are selectable for the ITS frequencies, as shown in figure 4.5.

When starting a capture using one of the additional options, it is then possible to select the 10 MHz wide channels from 172 to 184, for which data related to the frequency spectrum can be now collected and displayed.

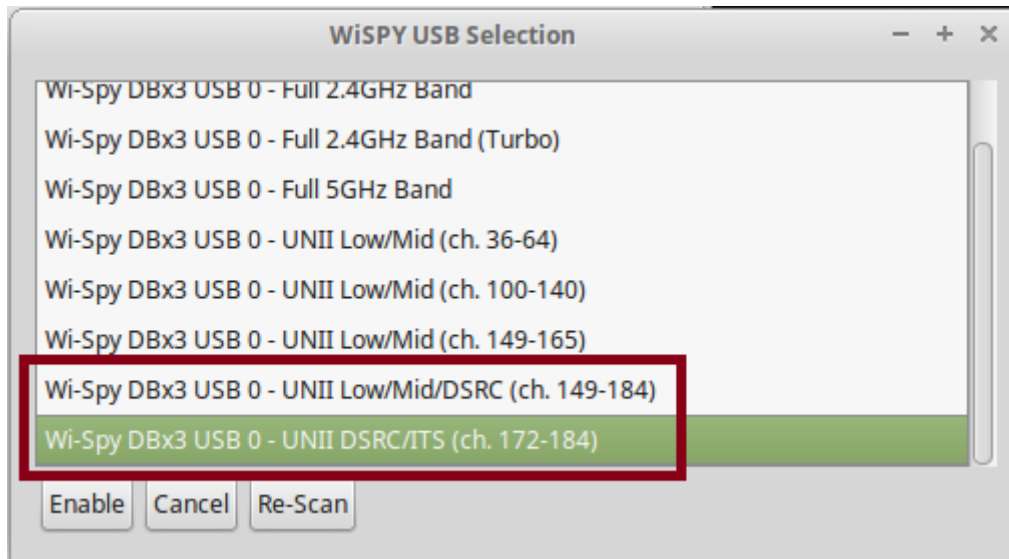


Figure 4.5: New options for the 802.11p channels' spectrum analysis

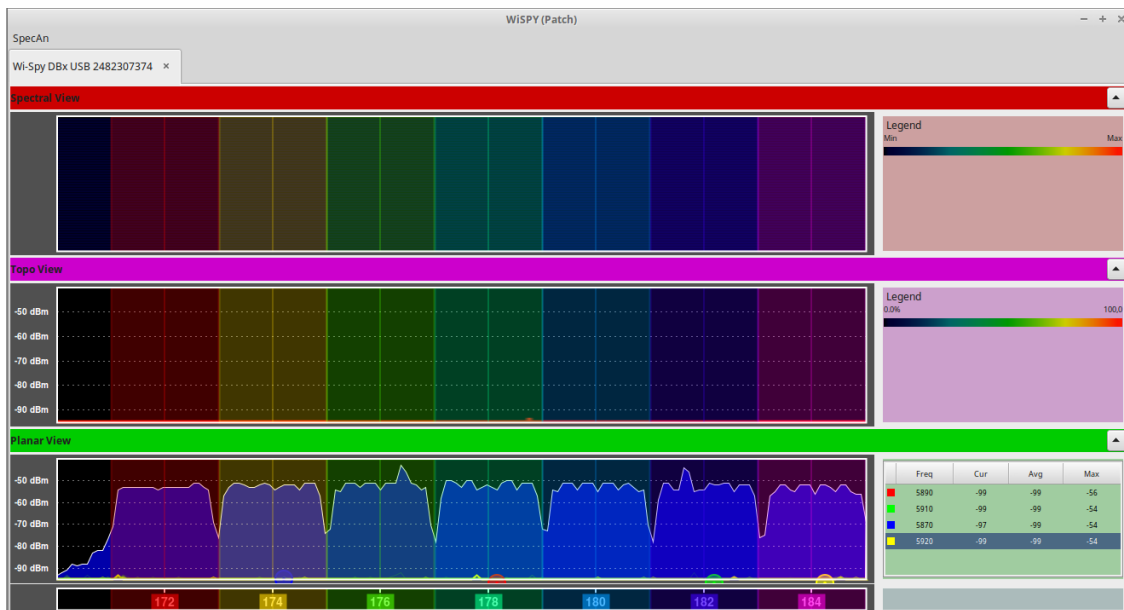


Figure 4.6: ITS channels inside *spectool_gui*

4.4 Looking for possible interferences

This verification has been performed on three different days, looking at what Wi-Spy actually captured in around 15 minutes sessions.

The device has been positioned very near to the position, at home, where the boards are usually placed.

The results are reported in the following table:

Date	Maximum dBm value over all the channels
2018-09-24	-93 dBm
2018-09-25	-93 dBm
2018-09-29	-92 dBm

The signal level resulted to be always less than -92 dBm, which can be considered as electrical noise.

So, as it is also possible to see from figure 4.7, no interfering sources over the ITS channels have been found.

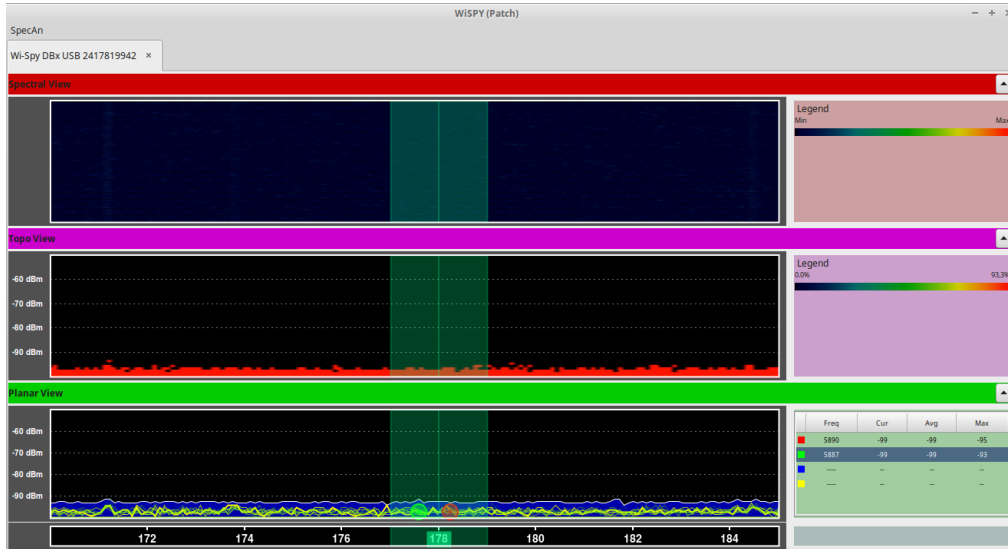


Figure 4.7: Absence of interference during the 2018-09-24 test (channel 178 is highlighted)

4.5 ALIX spectrum usage and transmission problems

For what concerns the spectrum usage by the UNEX DCMA 86P2 WLAN cards, some problems have arisen and they could be clearly visualized using the *Wi-Spy* device.

These problems were less visible without using any spectrum analyzer device, but they were nevertheless present.

As it will be detailed later, these issues were not impeding at all the communication to happen on the selected channel (thus all the previous results in which the boards communicate remain valid), but they could be the cause of possible random packet losses.

The first problem which was found out is shown in figure 4.8. When setting a high output power out of the UNEX DCMA 86P2 cards (around 24 dBm set in *iw*, with the 5 dBi antenna), the filtering was probably not performed correctly, and caused the board to invade frequencies outside the 10 MHz band, mainly when other devices are placed nearby.

The figure shows the incorrect behavior when transmitting over channel 172 at 24 dBm, compared to the maximum values of the spectrum for a lower power transmission on channel 178, which happened before.

When transmitting at lower power, in fact, the problem was hardly present and the boards behaved almost correctly, inside their 10 MHz wide channels.

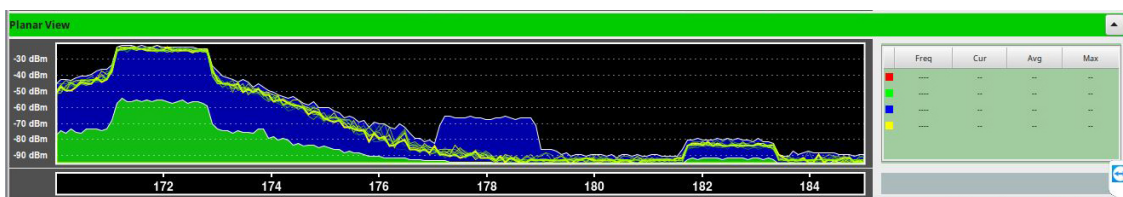


Figure 4.8: Incorrect frequencies for high power transmission with UNEX DCMA 86P2

Since future tests with the APU boards will show a much more polite spectrum usage, with the same software platform installed inside (OpenWrt 18.06.1), even though the maximum power of the newer UNEX cards is far below 24 dBm (the maximum allowed

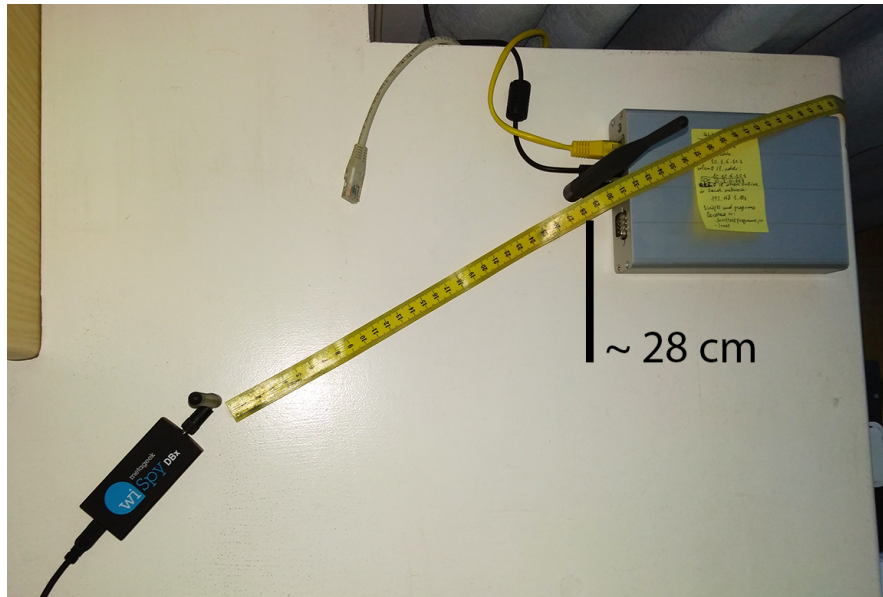


Figure 4.9: Configuration which was used for the previously described test, with the Wi-Spy device placed about 28 cm away from the antenna mounted on the ALIX board; the test has been also repeated with iPerf and two ALIX boards, with similar results

by the driver is 18 dBm), the problem has to be researched probably in how the hardware of the WLAN cards is designed and has been produced.

It is possible to have an evidence of this issue also when using the ALIX boards together with the newer APU boards.

We set the two couples of boards as follows:

	<i>software level txpower</i>	channel	<i>iPerf</i> port	connected antenna?
ALIX	27 dBm (high)	178 (5890 MHz)	7000	no
APU	3 dBm (low)	176 (5880 MHz)	6000	no

Running a couple of *iPerf* server+client of each of them and trying to record data in an almost synchronized way.

It was possible to detect a strong interference from the ALIX boards, with an evident drop in the measured throughput for what concerns the APU communication, even though the channels were different. The results are reported in figure 4.10.

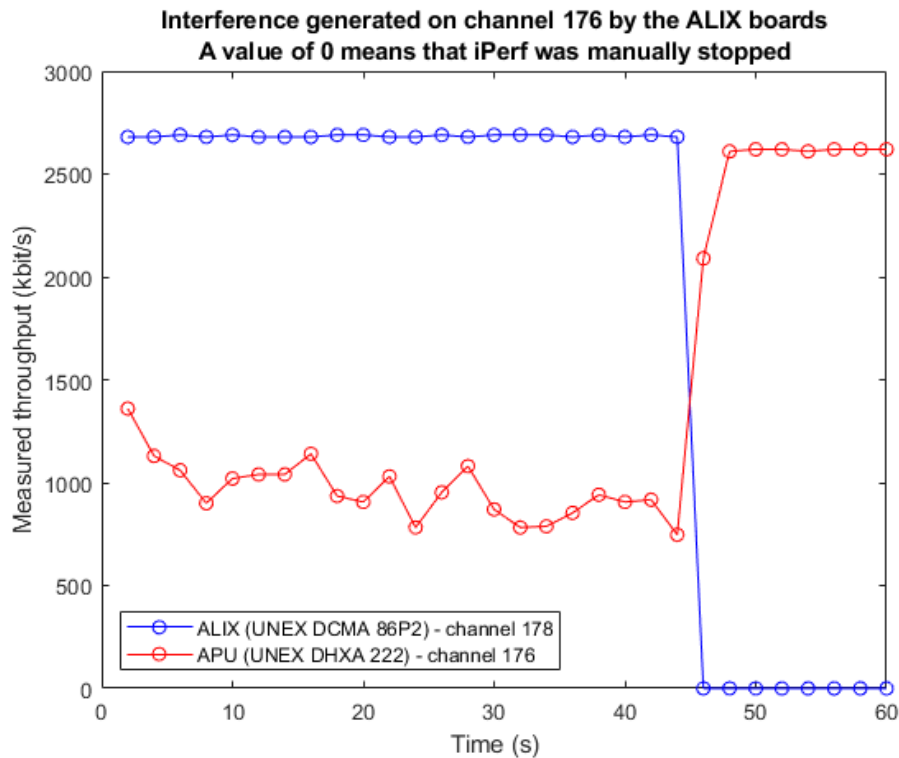


Figure 4.10: ALIX boards interfering, at high power, even with adjacent channels, making other boards drop their throughput even though they are not communicating at the same frequency of the ALIX ones

The second problem is even worse, since it can cause random packet losses, and we were not able to solve it by looking at the complex source code of the *ath5k* driver.

It can be described as follows: when transmitting, after an amount of time for which we were unable to find a well defined pattern, sometimes the transmission power could be unexpectedly increased to a very high value (showing also the problem described before), after a short interruption of the transmission of typically less than 5/6 seconds.

Everything occurred while the software seemed to be unaware of the power increase, since it always continued to show the previously set transmit power, when issuing “iw dev”.

It was completely solved only when resetting the power with commands like (with a previously set transmission power of 0 dBm):

```
iw dev wlan0 set txpower fixed 1500 # or any other mBm value different
    than 0
iw dev wlan0 set txpower fixed 0
```

Changing channel would lower the overall power received by the Wi-Spy device, but in any case, as we observed, at a higher level with respect to what was received before.

This behavior was observed, when setting a non ITS frequency of 5320 MHz (channel width 10 MHz), both using the patched Spectrum Tools and Chanalyzer, excluding the fact that it could be a problem of the Wi-Spy device interfacing with the development PC.

When triggered during any packet transmission test (with the C programs or with *iPerf*), it caused packet loss, as it is possible to see from figures 4.11 and 4.12.

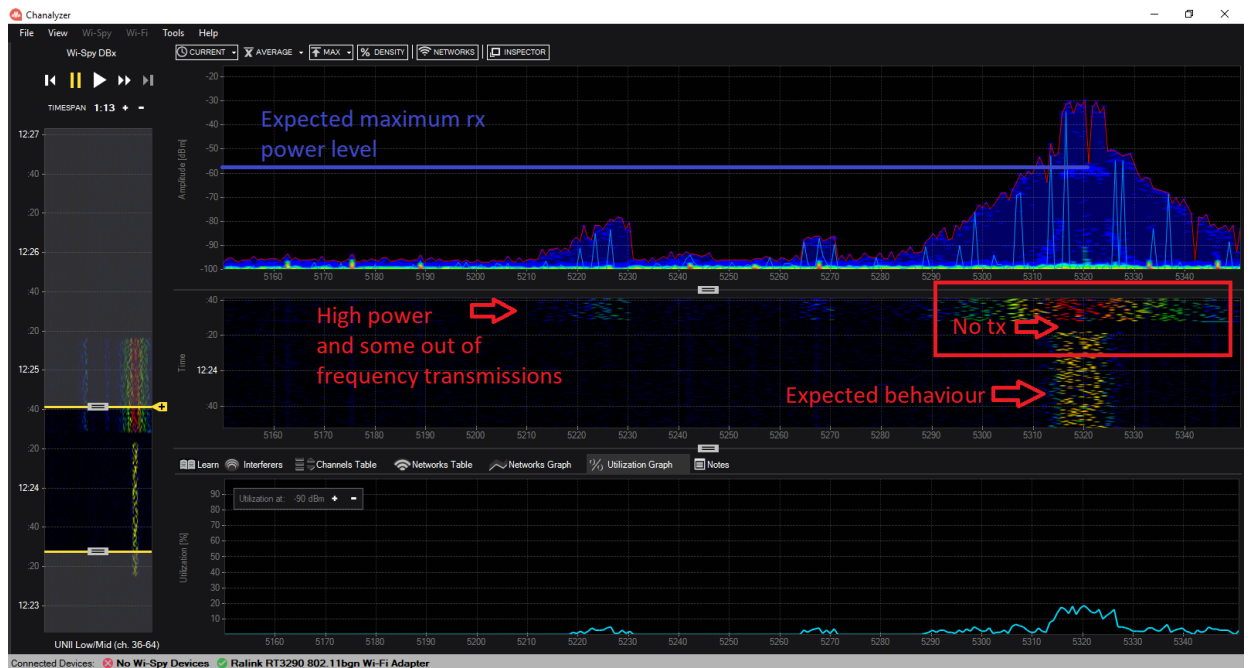


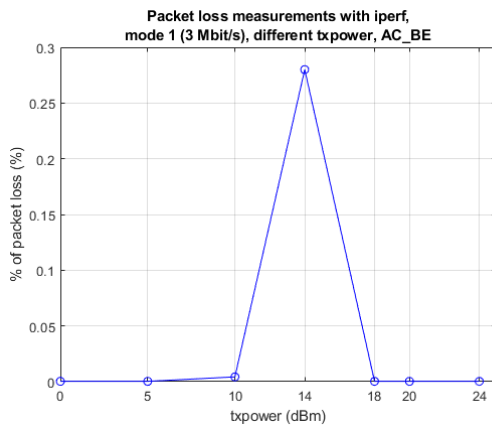
Figure 4.11: The problem of the unexpected power increase, as shown by Chanalyzer. It was possible to detect the same problem also with Spectrum Tools, on true ITS channels (172 to 184); the frequency “invasion” described before can also be seen

[3]	58.0-60.0 sec	640 KBytes	2.62 Mbits/sec	6.159 ms	4/	450 (0.89%)
[3]	60.0-62.0 sec	642 KBytes	2.63 Mbits/sec	10.548 ms	2/	449 (0.45%)
[3]	62.0-64.0 sec	640 KBytes	2.62 Mbits/sec	8.526 ms	3/	449 (0.67%)
[3]	64.0-66.0 sec	642 KBytes	2.63 Mbits/sec	5.694 ms	3/	450 (0.67%)
[3]	66.0-68.0 sec	642 KBytes	2.63 Mbits/sec	8.728 ms	2/	449 (0.45%)
[3]	68.0-70.0 sec	642 KBytes	2.63 Mbits/sec	7.449 ms	4/	451 (0.89%)
[3]	70.0-72.0 sec	640 KBytes	2.62 Mbits/sec	5.521 ms	2/	448 (0.45%)
[3]	72.0-74.0 sec	642 KBytes	2.63 Mbits/sec	12.496 ms	4/	451 (0.89%)
[3]	74.0-76.0 sec	640 KBytes	2.62 Mbits/sec	7.275 ms	2/	448 (0.45%)
[3]	76.0-78.0 sec	642 KBytes	2.63 Mbits/sec	5.006 ms	2/	449 (0.45%)
[3]	78.0-80.0 sec	642 KBytes	2.63 Mbits/sec	10.968 ms	4/	451 (0.89%)
[3]	80.0-82.0 sec	640 KBytes	2.62 Mbits/sec	6.618 ms	2/	448 (0.45%)
[3]	82.0-84.0 sec	642 KBytes	2.63 Mbits/sec	5.784 ms	4/	451 (0.89%)
[3]	84.0-86.0 sec	642 KBytes	2.63 Mbits/sec	9.501 ms	2/	449 (0.45%)
[3]	86.0-88.0 sec	640 KBytes	2.62 Mbits/sec	7.786 ms	3/	449 (0.67%)
[3]	88.0-90.0 sec	642 KBytes	2.63 Mbits/sec	5.530 ms	3/	450 (0.67%)
[3]	90.0-92.0 sec	148 KBytes	606 Kbits/sec	6.626 ms	0/	103 (0%)
[3]	92.0-94.0 sec	0.00 Bytes	0.00 bits/sec	0.000 ms	0/	0 (0%)
[3]	94.0-96.0 sec	355 KBytes	1.45 Mbits/sec	9.687 ms	30/	277 (11%)
[3]	96.0-98.0 sec	642 KBytes	2.63 Mbits/sec	7.877 ms	4/	451 (0.89%)
[3]	98.0-100.0 sec	642 KBytes	2.63 Mbits/sec	5.452 ms	2/	449 (0.45%)
[3]	100.0-102.0 sec	642 KBytes	2.63 Mbits/sec	11.512 ms	4/	451 (0.89%)
[3]	102.0-104.0 sec	640 KBytes	2.62 Mbits/sec	6.924 ms	2/	448 (0.45%)

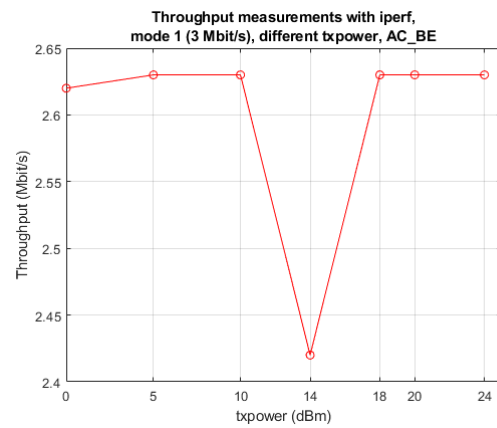
Figure 4.12: Packet loss as seen by an iPerf client just after an unexpected transmission power increase occurred

The following figure (figure 4.13) shows instead two plots, which came out from a succession of tests performed with the two ALIX boards placed at a distance of 3.25 m one from each other.

The tests involved an *iPerf* client running on one board and an *iPerf* server running on the other one, recording packet loss and recored bandwidth, after setting the 3 Mbit/s data rate.



(a) Packet loss



(b) Decreased average throughput

Figure 4.13: Problems during the 14 dBm test

As it is possible to see, a random packet loss occurred at a certain point, very probably caused by this problem.

This second problem was not present with the old *ath5k* version used in the GCDC 2011 OpenWrt images; the driver has, however, changed a lot since then, and it was very difficult to compare portions of code to try to find out where the problem could reside. It was also using IBSS instead of the more proper OCB mode.

The presence of this behavior, which seemed difficult to predict and which could be caused by some instability inside the combination of the latest version of *ath5k* and UNEX DCMA 86P2, represented another reason to move to the newer APU boards, which are described in the next chapter.

Chapter 5

APU boards and UNEX

DHXA-222

5.1 The APU boards

After integrating an open source 802.11p system inside the ALIX boards and doing some considerations on how the ITS frequency spectrum can be analyzed, it was possible to move to the newer PC Engines boards.

Moving to a newer system is actually justified not only by the fact that it is possible to work on more up to date, still in production, hardware; in fact:

- Moving to newer hardware solves the instability and unreliability of the *ath5k* driver used in conjunction with the UNEX DCMA 86P2 cards (described in 4.5), taking also into account that several newer cards are now supported by the *ath9k* driver.
- miniPCI WLAN cards, such as the ones used in the ALIX boards, are become more difficult to find, being replaced by PCIe cards, such as *UNEX DHXA-222*. This means moving to hardware supporting PCIe (or mini PCIe) is very desirable [25].
- The APU boards are able to deliver much better performances when it comes to

running operating systems or applications on top of them. They allow a much faster system startup, more resource for more complex applications and possibly also the possibility to switch to more demanding operating systems, such as Debian-based ones, if needed. Here we will concentrate on OpenWrt, though, trying to test a platform that could be integrated in a wider range of hardware boards, from less performing to more powerful ones.

- The APU boards have been successfully used in several research projects, including the validation of the OpenC2X platform, using APU 1d4 boards [36].
- The newer boards are supporting SSDs instead of Compact Flash cards. SSDs are typically lasting long, in terms of read/write cycles, and offer higher speeds and capacities. As a side effect, thanks to the USB support, this makes the whole development process a bit more comfortable, since it is now possible to flash full system images inside the SSD directly from a live USB device, in a fast and reliable way, without the need of removing the memory, putting it inside an adapter and flashing from the development PC.

We used two *APU 1d* boards, which have the following hardware characteristics [66]:

- CPU: AMD G series T40E, 1 GHz dual core processor with 64 bit support, designed for embedded applications, with “32K data + 32K instruction + 512K L2 cache per core”
- RAM: 2 GB DDR3-1066 DRAM
- Storage: SD card slot, possibility to boot from external USB, mSATA SSD (we used a SATA III Transcend *MSA370* MLC NAND Flash SSDs for each board, with a capacity of 16 GB each)
- Power: 12V DC, with a power consumption of 6 to 12 W

- Expansion: 2 miniPCI express slots to accommodate up to two wireless cards, LPC bus (may require the selection of custom drivers inside the chosen Linux distribution), GPIO header, I2C bus, SIM slot, COM2 header
- Connectivity: 3 Gigabit Ethernet channels (Via Realtek RTL8111E)
- Ports and I/O: 2 external USB ports (plus 2 internal USB), DB9 serial port, three LEDs and one push button (*SI*)
- Board size: 152.4 x 152.4 mm
- Firmware: *coreboot*
- Other: RTC battery, conductive cooling for the CPU and the south bridge to the external enclosure, using “a 3 mm alu heat spreader”

One of the APU 1d boards is shown in figure 5.1. The attached post-it serves as a reminder for the IP addresses which are set on the board and to easily recognize them.



Figure 5.1: ALIX 1d board view from outside, with the UNEX card and the enclosure mounted on. It is possible to distinguish the DB9 serial connector, the three Ethernet ports, two USB ports and the DC power connector, together with two RP-SMA connectors for two antennas.

The main components of these embedded boards are shown in figure 5.2.

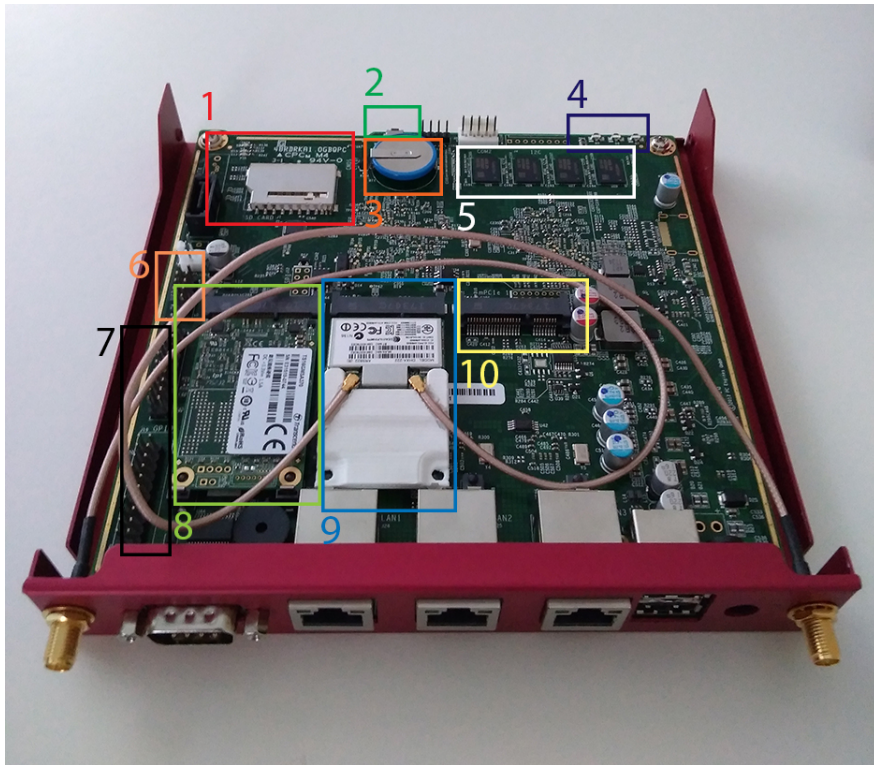


Figure 5.2: APU 1d board, without enclosure, with UNEX card installed in

It is possible to distinguish the following main hardware components, along the ports that are shown in figure 5.1, with reference to figure 5.2:

1. SD card slot
2. SI push button
3. RTC battery
4. Three LEDs
5. DRAM memory (half of the memory chips, the other half can be seen when viewing the board from below)
6. 9 pin USB header (providing support for the 2 internal USB)

7. LPC and GPIO connectors
8. Transcend 16GB MLC SSD, mounted on the mSATA slot
9. UNEX DHXA-222 WLAN card, with two *U.FL* to *RP-SMA* pigtail connectors, mounted on one of the two mini PCIe slots (PCIe slot 2 has been used, because it allowed an easier connection of the pigtail cables)
10. Additional mini PCIe slot
11. Below the board, it would be possible to see the AMD CPU, the south bridge, the other half of the DRAM silicon memory chips and the SIM slot

The UNEX cards came in an half size mini PCIe format, while the APU boards are designed for full size chips.

In order to adapt the UNEX cards to fit inside the APU boards, an half to full size adapter has been printed using a *MakerBot* 3D printer and then further processed to dig two small conduits where the pigtail cables could be placed, once connected to the WLAN card.

The Ethernet ports are mapped by operating systems into different interfaces and this fact gains more importance now, since more than one port is available.

The OpenWrt system actually maps the Ethernet ports to the same number of interfaces, according to the following scheme [67]:

```
-----  
--0---|DB9|---|eth0|---|eth1|---|eth2|-|USB|---(DC)----0--  
-----
```

5.2 UNEX DHXA-222

The DHXA-222 WLAN cards used with the APU boards are produced by UNEX, just like the UNEX DCMA 86P2 cards presented before.

They are half size mini PCI express (version 1.2) chips, with two *U.FL* antenna connectors for MIMO 2x2 operations, supporting all the ITS frequencies (not officially declared, but they are actually used as ITS cards, mostly in research works) and dual-band operations.

They also support Bluetooth 4.0.

Being based on the Atheros *AR9462* chipset, they are supported by the *ath9k* driver, which is actually used a lot in ITS research activities, including the use of the OpenC2X environment [36] [25].

In particular, the CCS Labs team in Paderborn has released few patches targeted specifically to this driver, in order to enable OCB operations and make it possible to manage the 802.11p channels.

They provide a maximum output power which is lower than the DCMA 86P2 cards, which allowed the system to output even over 30 dBm of effective transmission power. Now, the maximum output power is declared to be 17 dBm (2 dBm for Bluetooth operations).

Other characteristics reported by the producer are summarized below [68]:

- Operation voltage: $3.3V\ DC \pm 5\%$
- Wi-Fi receiver sensitivity: $-71\ dBm \rightarrow -93\ dBm, \pm 2\ dBm$ tolerance
- Operation temperature range: $-10^\circ \rightarrow +70^\circ$

The minimum settable transmit power seems to be, now, 3 dBm, at least when using systems based on OpenWrt.

5.3 Installing a Linux based operating system on the SSD

In order to install an operating system inside the SSD, the procedure is a bit more complex than the previous case (in which the CF card was simply removed and flashed

from the development PC), but, once done for the first time, it allows to more comfortably load system images on the mSATA SSD.

In order to install OpenWrt, or possibly any other embedded Linux distribution, the user can make use of the USB ports which are available in the APU boards.

In particular, to integrate a Linux distribution such as OpenWrt inside the SSD, the following step shall be performed:

1. Download TinyCore Linux from the PC Engines website: <http://www.pcengines.ch/tinycore.htm>. This is a very lightweight Linux distribution, able to boot quickly, for basic operations such as flashing something inside the SSD or update the BIOS.
2. Download the proper file, depending on which kind of operating system you are using: *apu-tinycore-usb-installer.exe* for Microsoft Windows or *apu_tinycore.tar.bz2* for Linux/MacOS.

Under Windows, it is sufficient to run the installer after inserting an USB stick in the development PC.

Under Linux/MacOS, additional steps may be required, as detailed in the link presented before.

3. Copy the desired operating system image or the system installation files into the USB stick with TinyCore; we always put them directly on the main directory, and it worked fine. For what concerns OpenWrt, the *gzipped* image file should be copied, for instance: *openwrt-x86-64-combined-ext4.img.gz*.
4. Insert the USB stick in the board, connect the development PC to the APU board using a serial cable¹ and finally connect the APU board to the main through a proper 12V power supply.
5. It may be necessary to navigate, the first time, through the APU BIOS menu, in order to set the boot order putting the proper USB port on top of the list.

¹Further details on the serial connection are reported afterwards

6. After booting from the USB device and connecting through a suitable terminal window or terminal emulator (such as PuTTY in *Serial* mode or *screen*), it will be possible to interact with TinyCore.
7. If you find out that the BIOS firmware is outdated, in order to update it, run: `flashrom -w apu140908.rom` (or any other `*.rom` file which is available inside the PC Engines TinyCore distribution); we will not, however, focus on BIOS updates in this work.
8. It is possible instead to partition the SSD and run all the necessary files to unpack and install a Linux distribution inside the SSD, thanks to the files which were added to the USB stick main directory [25].

For what concerns OpenWrt, issue the `fdisk -l` command, to find out which is the name assigned to the SSD device. In our case, it always resulted to be `/dev/sda`.

Then, run this command to start the extraction process, practically integrating the OpenWrt image inside the SSD: `zcat <image_name>.gz | dd of=/dev/sda bs=16k`

In our case, the command was the following one: `zcat openwrt-x86-64-combined-ext4.img.gz | dd of=/dev/sda bs=16k`

9. Rebooting the board (issuing a `poweroff` command and disconnecting/reconnecting the DC power supply - now `poweroff` should really turn the board completely off), it should be now possible to interact with OpenWrt using the SSH server just like in the ALIX boards case.

It may be interesting to point out also few additional considerations:

- We were not able to boot TinyCore from newer USB sticks with a capacity of 8 GB and more. We were instead able to use an older 4 GB Transcend USB stick, shown in figure 5.3. This device was then kept apart to perform any future installation process.

- Using USB sticks with a fat form factor may be a bad idea, since it may result impossible to fit it together with the DC power cable, which is placed near to the 2 external USB ports.
- In case, for any reason, the APU board keeps booting on the *MemTest+* environment with the serial console disabled, refusing to show the standard boot menu and/or to boot from SD, USB or SSD, a possible solution may be to push the *S1* button, and boot the board keeping it pushed. This button allows the user to obtain the setup options even when the serial console is disabled (as in the APU2 boards) [69].
- Since the `zcat` command actually extracts the image to standard output, which is then piped to `dd` to copy the data inside the SSD, it is sufficient to have an USB stick that is able to store the *gzipped* image; it is not necessary to have enough free space also for the uncompressed **.img* file (which may take over 1 GB of disk space, depending on the configuration).



Figure 5.3: Transcend 4 GB USB stick with TinyCore and the OpenWrt image

5.3.1 Serial connection with the APU board

In order to connect with the boards for the first setup process, or to view the full boot process with all its debug messages, it may be necessary to rely on a serial to USB connection.

In our case, we used a *Manhattan* DB9 Male to USB cable, with a *Prolific* chip inside, together with a DB9 Female to Female cable.

This cable does not work under Windows 10, due to driver problems which will not be detailed here, but it works flawlessly when used from the Linux Mint virtual machine.

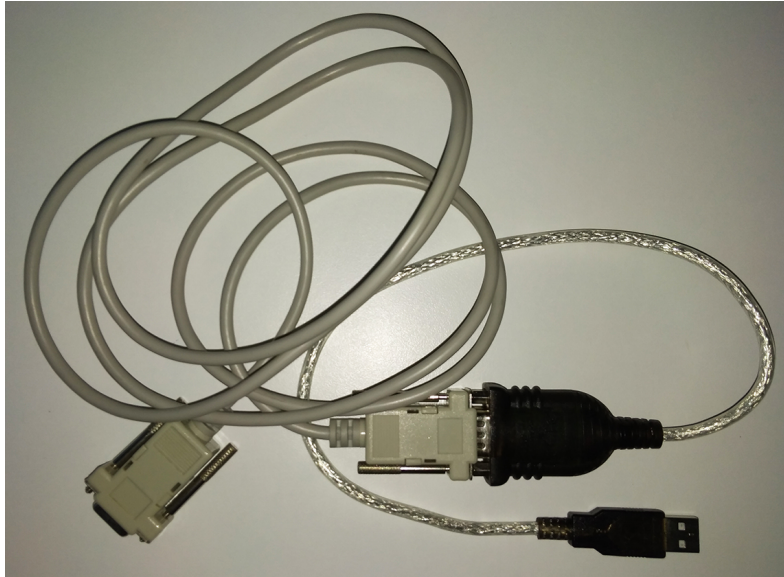


Figure 5.4: The RS232/DB9 to USB cable used to connect the development PC with the boards, in order to interact with them through serial console

In order to connect with the APU boards, the following settings should be used [25]:

- Baud rate: 115200
- Data bits: 8
- Stop bits: 1
- Parity: none

Any terminal emulation program can be used, including PuTTY and, under Linux, `minicom`.

We decided to use, however, the simpler `screen` utility, from within the Linux Mint virtual machine.

We installed it using: `sudo apt-get install screen`.

Then, it was necessary to find out the name assigned to the device, through the command: `dmesg | grep tty`

Looking at the line related to the *Prolific PL2303* chip, it is possible to find out the required information.

In our case, we obtained, inside the full output of the command, the following line:

```
[22832.076391] usb 3-3.1: pl2303 converter now attached to ttyUSB0
```

telling that the serial device had been mapped to `/dev/ttyUSB0`.

After that, it was possible to start the connection with: `sudo screen /dev/ttyUSB0 115200`.

To detach from the current session, a sequence of *Ctrl+A* and *Ctrl+D* can be used.

It is important to note that, unless specific user permissions are set, `sudo` is always required, under Linux, to open any serial port.

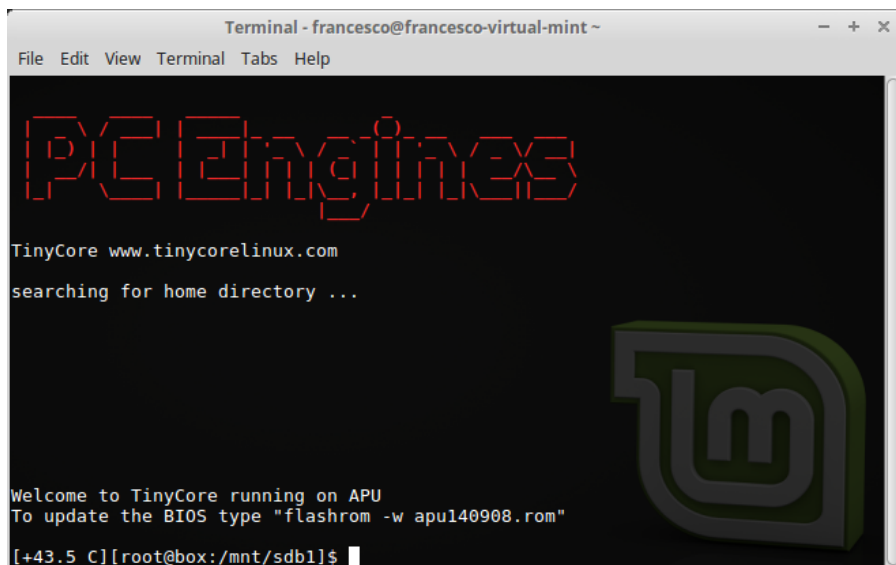


Figure 5.5: TinyCore running on one of the APU boards, through a live USB stick, with screen showing the serial console on the Linux Mint (Xfce) terminal emulator

Connecting the boards through a serial cable has also a further advantage: just after integrating OpenWrt, the boards are given, for their Ethernet ports, the 192.168.1.1/24 IP address.

It may be uncomfortable to access it when the boards are connected inside a local network where 192.168.1.1 is already assigned to a gateway, but by means of the serial cable it is possible to access the network configuration file and change it before actually connecting with SSH.

It can be done, for instance, with the “vi” text editor, by launching: “vi /etc/config/network”.

5.4 Compiling OpenWrt for the APU boards

In order to integrate OpenWrt 18.06.1 on the APU boards, some modifications to the `.config` file where needed, through the `make menuconfig` configurator.

Before modifying the configuration file to match with the APU boards, the old ALIX configuration was copied, in order to keep it for future reference and to compile images more optimized for the ALIX boards:

```
~/openwrt-18.06.1 $ cp .config config_ALIX.config
```

In order to compile an optimized image, which can make use the 64 bit support coming from the AMD CPU, a new target had to be selected:

- Target system: x86
- Subtarget: x86_64
- Target Profile: Generic

After changing the target, we tried to make our configuration match most of the one presented in the official OpenWrt page about the APU boards [67], while keeping all the modifications and additional packages presented in chapter 3.

One important step is to select the needed kernel modules listed in [67], to allow proper operation with the most important hardware modules:

- *r8169*: to enable support for the RTL8111/8168B PCI Express Gigabit Ethernet controller

- *i2c_core* + *i2c_piix4*: although not used in our case, it can be important for any application requiring I2C
- *usbcore* + *usb_storage*: for USB support
- *kmod-ata-ahci*: SATA support
- *kmod-sp5100_tco*: Watchdog support, which, as reported in [67], should “*allow waking up*” (though an image without this package has not been tested)

We then tried to include all the packages we thought to be useful from the configuration file listed in the “*Custom OpenWrt image*” section in [67].

Our configuration file is reported in Appendix C.

We were then able to compile the new images, to be integrated inside the APU boards following the procedure described before in section 5.3, with:

```
make clean
make -j10 V=s
```

The new images were then available in “*./bin/targets/x86/64/*”.

5.5 Configuring the APU boards

After installing OpenWrt inside the boards, it was possible to configure them, giving them the proper IP addresses, changing the *iw_startup* script, and so on, in a similar manner with respect to the ALIX boards.

As with the older boards, we decided to give them specific IP addresses (for both the Ethernet and WLAN interfaces), which were then reported on two post-its applied to the boards, in order to easily recognize them. In particular:

- Board 1: *APU_102*², with:

²Again, names are just given to recognize the boards and they do not play any role inside in the boards’ communication

- *eth0* local IP address: 10.1.6.102
 - *wlan0* IP address: 10.10.6.102
 - *eth0* internet IP address: 192.168.1.182
- Board 2: *APU_103*, with:
 - *eth0* local IP address: 10.1.6.103
 - *wlan0* IP address: 10.10.6.103
 - *eth0* internet IP address: 192.168.1.183

We decided to start giving the boards two Ethernet IP addresses related to the local network with Internet access (i.e. 192.168.1.182 and 192.168.1.183), instead of the 10.1.0.0/16 network, since they make it easier to perform the measurements, install new packages with `opkg` and to, for instance, synchronize the date and time through NTP, as described in the next section.

This required adding two cables to the *Netgear* switch connected to the local network with Internet access, and plugging them into the APU boards; the ASUS adapter was in this case no longer used, as reported in the desk configuration description in section 3.2.

In order to set these IP addresses, the `/etc/config/network` file has been modified just like before, and the WLAN IP address was changed inside the `iw_startup` script, which seems to work fine without any modification from the previous cases except:

- The aforementioned WLAN address
- Changing the line setting the initial transmission power to use, for instance, 15 dBm (1500 mBm) instead of 0 dBm, which seems now unavailable (with the UNEX DHXA-222 cards the minimum seems to be 3 dBm, but setting it directly with `iw` does not seem to work all the times):

```
iw dev wlan0 set txpower fixed 1500
```


- Removing the lines “wifi up” and “wifi down”, which seemed to create problems with the configuration, other than being now, probably, completely unnecessary

We also modified the `/etc/config/wireless` file in order to enable the wireless interface, by setting the line “option disabled '1'” to “option disabled '0'”.

All the configuration files for the two APU boards are reported in Appendix D, which should be taken as reference for any future configuration operation.

After performing all the necessary configuration steps, it was then possible to start OpenWrt on the boards, login to the `root` user, launch the `iw_startup` script and start working with them in OCB mode.

5.6 NTP synchronization with chrony

In order to perform any indoor measurement, mainly when a GPS device providing the UTC time (as reported in the IEEE standards) is not available yet, it can be very useful to enable an NTP time synchronization system on the boards, using the standard Internet connection through the Ethernet ports.

For instance, it can be used when there is the need of launching two *iPerf* clients together, at the same time, or to synchronize any kind of measurement.

The same procedure can be applied when building and integrating OpenWrt images on the ALIX boards (and possibly also several other boards) and it can be useful also for a wide range of applications, not only for indoor measurements.

There are many tools for NTP synchronization in OpenWrt, including an already available system NTP daemon (included in *busybox*) and *ntp-utils*, for instance [70].

We will focus on how to configure OpenWrt for an NTP client functionality, getting the time from the Internet and using it to adjust the system clock.

Before describing how to set up the boards to synchronize over NTP, it is important to note that, after synchronization, the current time and date can be maintained with no Internet connection (even if possible drifts and offsets will not be compensated anymore)

as long as the board is connected to the DC power supply, or, if available, as long as the RTC battery has enough power to keep the RTC running (the latter is the case of the APU boards).

Among the available tools, we decided to give up using the pre-installed NTP daemon, and move to *chrony*, which has to be manually included when preparing the build system with `make menuconfig` (see 3.8). This decision was motivated by the fact that *chrony* offers more advanced functionalities and quite precise time synchronization, accounting for possible intermittent connection too (which may be the case of our boards, under some situations); it can also be used to synchronize with reference clocks, such as GPS data, as reported in [71].

According to the *chrony* website³, this tool is also smaller than other NTP synchronization utilities, trying to reduce the memory and CPU usages, which may be useful in less performing embedded systems.

The steps to be followed to set up the APU boards with *chrony* are presented below (they are based on OpenWrt 18.06.1, but they should be completely repeatable for LEDE 17.01 too):

1. After including *chrony*, as reported in section 3.8, and booting OpenWrt, it is possible to use WinSCP (or similar programs) to navigate the boards file system and locate the `/etc/config/chrony` configuration file.
2. Change the line “`option hostname '<server>'`” inserting your preferred NTP server (or leave the default OpenWrt server, which should be already there); we decided to opt for one of the INRiM NTP servers, which are located in Turin (quite near us) and are taking their reference from a very precise cesium atomic clock. So, in our case the “*hostname*” line should be: “`option hostname 'ntp1.inrim.it'`” [72].

Our file is reported in Appendix E.

³<https://chrony.tuxfamily.org/comparison.html>

3. In order to have only one NTP daemon running on the APU boards, it is suggested to disable the pre-installed one (and this may be required, for instance, when using other tools such as the *ntpd* package, as reported in [70]). It can be done by changing few lines inside the */etc/config/system* configuration file.

In particular, the line “`option enabled '1'`”, after “`config timeserver 'ntp'`”, should be set to “`option enabled '0'`”, in order to disable the system NTP client.

If the user wants to change the timezone too, the string after “`option timezone`” can be modified from “UTC” (which is the default option) to any other valid value⁴. For instance, to set the Italian time, the string “CET-1CEST,M3.5.0,M10.5.0/3” can be used. We decided, in any case, to leave this to UTC (as required by the standard, the UTC time should be used).

4. In order to completely achieve the previous goal, open an SSH connection (for example using PuTTY) and disable the *sysntpd* service, enabling the *chronyd* one, which is a daemon provided by *chrony*:

```
service sysntpd stop
service sysntpd disable
service chronyd enable
```

Reboot then the boards, for example shutting them off, removing the DC power supply and giving them power again:

```
poweroff
```

If it works correctly, the more proper “reboot” command can be used too, without the need of performing a full power cycle.

5. Valid DNS servers are required by *chrony* to resolve the server name specified before. Change the file */etc/dnsmasq.conf* adding, at the end of the file, the lines:

⁴Valid values are listed, as of October 2018, here: https://openwrt.org/docs/guide-user/base-system/system_configuration

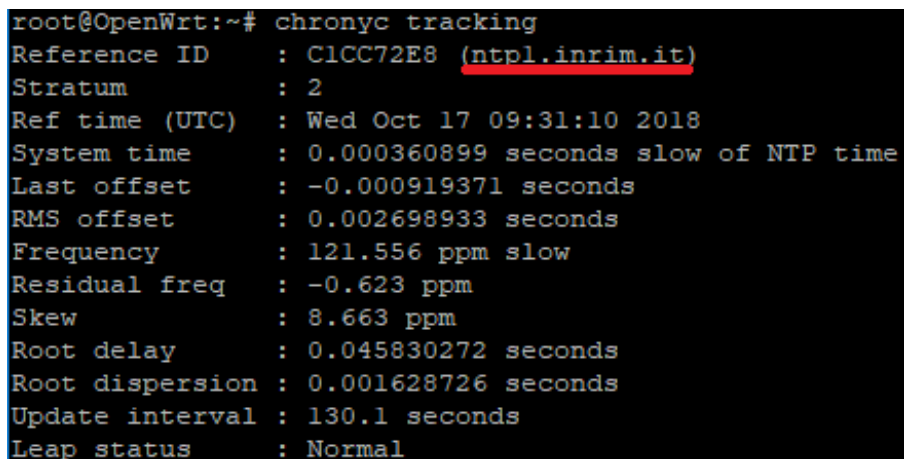
```
# DNS servers for APU boards
server=8.8.8.8
server=8.8.4.4
```

These lines have been used to add the Google Public DNS servers.

6. Now *chrony* should be able to synchronize the date and time when the boards are connected to the Internet.

In order to display useful information, the following commands can be used:

- `date`: it can be used to display the current date and time
- `chronyc tracking`: it can be used to display the current synchronization and system clock performance data (which should now include the correct NTP server, as it can be seen in figure 5.6).
- `chronyc sourcestats -v`: it can be used to print information “*about the drift rate and offset estimation process for each of the sources currently being examined by [the chrony daemon] chronyd*” [73]



```
root@OpenWrt:~# chronyc tracking
Reference ID    : C1CC72E8 (ntp1.inrim.it)
Stratum        : 2
Ref time (UTC)  : Wed Oct 17 09:31:10 2018
System time     : 0.000360899 seconds slow of NTP time
Last offset     : -0.000919371 seconds
RMS offset      : 0.002698933 seconds
Frequency       : 121.556 ppm slow
Residual freq   : -0.623 ppm
Skew            : 8.663 ppm
Root delay      : 0.045830272 seconds
Root dispersion : 0.001628726 seconds
Update interval : 130.1 seconds
Leap status     : Normal
```

Figure 5.6: Output of the “`chronyc tracking`” command after configuring *chrony* with the INRiM NTP 1 server and synchronizing the date and time

When using OpenWrt, a *chrony.conf* file, inside */etc/chrony/*, should be present too. This file is in a format which is documented in the *chrony* documentation and it should be included in the final *chrony* configuration, together with the other configuration file.

This file was not modified, in our case, when setting up the APU boards.

5.7 Sniffing with an APU board

Packet sniffing can be now performed using any board as a monitor device, capturing traffic that other boards are exchanging, without being directly involved in this kind of traffic.

We tested this configuration by using one of the APU boards (for example “APU_102”), with a monitor interface *wlan1*, capturing traffic coming from the two ALIX boards, in which a sample client and server were running (they were written during one of the courses in Politecnico and they are quite effective in testing whether connectivity is present or not): the client is a simple socket C program requesting the current date, only once, to a server running on the other board.

The client was, in our tests, running on the ALIX_100 board, while the server was running on the ALIX_101 board.

The only effective packet sniffing method, in this case, involved the use of *tcpdump*, and the best method resulted to be the use of *plink*, *tcdump* and *Wireshark* (as detailed in section 3.8.6).

By using this configuration, as it is possible to see from figure 5.7, it was possible to capture all the TCP traffic between the client and the server, with the APU board listening on the same frequency (5.89GHz).

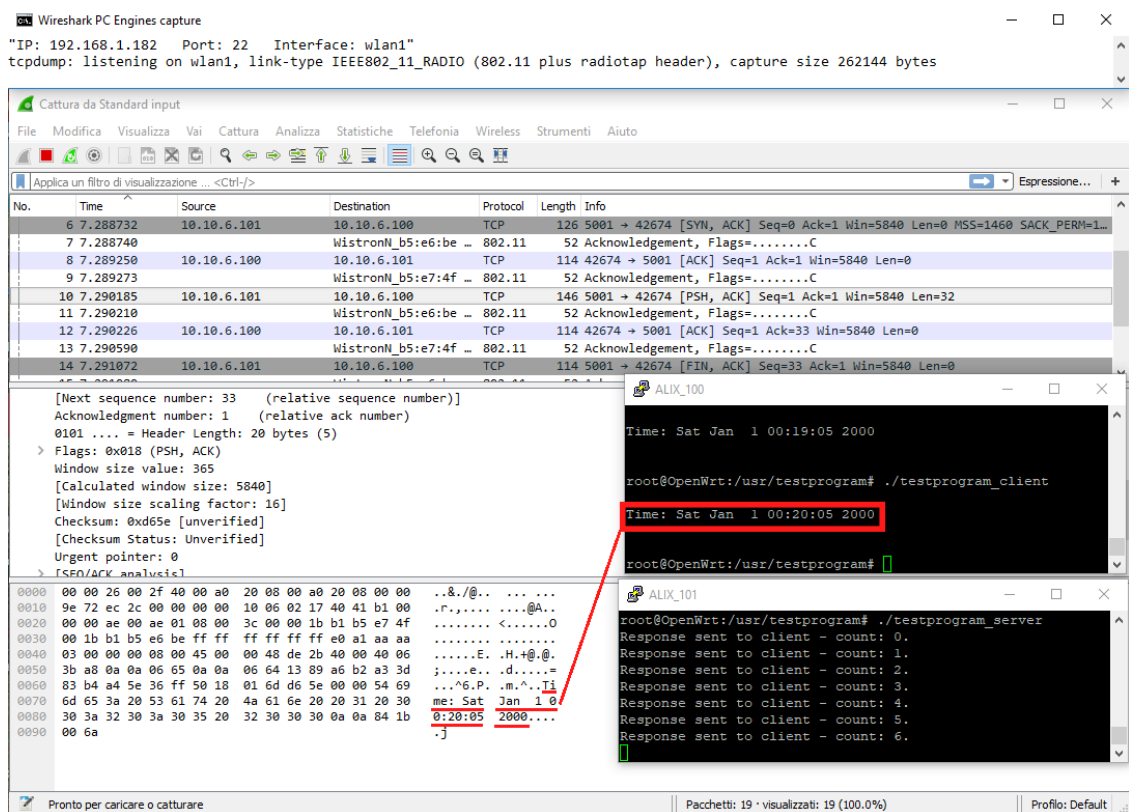


Figure 5.7: Packet sniffing using one of the APU boards, capturing traffic between the ALIX boards (the date and time in the ALIX boards was not synchronized over Internet, that's why a wrong date is reported by the server)

5.8 Bitrates on the APU boards and correct way of measuring them

While working with the APU boards, we noticed that the output of the command:

```
iw dev wlan0 station dump
```

which should report useful station statistics (such as the received power and the current transmit bitrate), was giving us, sometimes, some unreliable information about data bitrates, at least when used with this particular hardware and the OCB mode.

In particular, the information about the last transmit bitrate proved to be quite unreliable, sometimes giving a different value both with respect to what we expected and to what is actually reported in RadioTap headers, which can be captured using *plink*, *tcpdump* and *Wireshark*.

After performing some tests, we concluded that the information about the bitrate given by “`iw dev wlan0 station dump`” should be, when possible, discarded, since it seems to provide, at least in certain situations, wrong values. This does not happen, as we observed, all the times, but it can be observed as long as measurements are performed.

The information given by RadioTap headers instead seems to be always the expected one, thus capturing traffic with *Wireshark* (using a second board) and analyzing the RadioTap headers should be the best way of measuring the current physical transmit bitrate of the sending board.

In order to better verify that what we mentioned before is actually correct, we set up few additional tests, checking both the bitrate reported by `iw` and the one that is captured by means of *Wireshark*.

In particular, we imposed a certain bitrate to the boards using:

- `iw dev wlan0 set bitrates legacy-5 6`, for 3 MBit/s
- `iw dev wlan0 set bitrates legacy-5 12`, for 6 MBit/s
- `iw dev wlan0 set bitrates legacy-5 24`, for 12 MBit/s

Other 802.11p values seems to be unsupported as of now: they cannot be selected and they always return an “*Invalid Argument*” error.

A future improvement to this work could concentrate on the 802.11p bitrates and analyze how they are managed both by the wireless subsystem and by the underlying driver (*ath5k* or *ath9k* or possibly other ones), trying to introduce proper patches enabling them, of course only if the connected WLAN cards provides the needed support.

After selecting a given bitrate, we used the same scripts which we wrote for the measurements presented in chapter 6 (section 6.1.2 will also introduce them more in detail),

adding a third script able to periodically report the rate read from “iw dev wlan0 station dump”.

This script is based on the NTP synchronization just like the ones used for the measurements⁵, in order to align the rate reports to the iPerf server and client periodic ones.

It is then able to save in a proper log file, every second (using a while loop with a “sleep 1” command inside), the rate which is extracted from “iw”, thanks to the “grep” and “cut” system programs.

rateLooker.sh

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 rates=( 0.5M 1M 1.5M 2M 2.5M 2.6M 2.7M 3M 5M 5.4M 5.5M 5.6M 5.7M 5.8M
13         5.9M 6M 7M 9M 10M 11M 11.5M 11.6M 11.7M 11.8M 11.9M 12M 100M 500M )
14
15 physrates=( 3 6 12 )
16
17 currwait=0
18 i=0 # Change this to match a specific test
19 p=0 # Change this to match a specific test
20 while [ $p -lt ${#physrates[@]} ]; do
21     while [ $i -lt ${#rates[@]} ]; do
22         echo "Iteration started at second $(date | cut -d$'\t' -f5 | cut -
23             d":" -f3 | cut -d" " -f1), waiting until second $currwait"
```

⁵More details are presented in chapter 6, in section 6.1.2


```
21     waituntil $currwait
22     echo "Logging started on stdout"
23     secs=61
24     while [ 0 -ne $secs ]; do
25         currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1
26             )
27         echo "$currsec - $(iw dev wlan0 station dump | grep "bitrate:" |
28             cut -f3 | cut -d" " -f1 | cut -d"." -f1) MBit/s"
29         sleep 1
30         secs=$((secs-1))
31     done
32     echo "Logging done."
33     i=$((i+1))
34     currwait=$(( (currwait+10)%60 ))
35 done
36
```

A monitor mode interface (*wlan1*) has also been set on the APU_102 board and used to capture the traffic with the development PC and Wireshark.

Looking at the tests in which the problem was present and taking for instance the test with 12 *Mbit/s* as physical bitrate and 7 *Mbit/s* as iPerf offered traffic (it is the “*UDP bandwidth to send at*”, specified by means of the *-b* option), this is the log obtained thanks to the *rateLooker.sh* script:

```
00 - 3 MBit/s
01 - 3 MBit/s
02 - 3 MBit/s
03 - 3 MBit/s
04 - 3 MBit/s
05 - 3 MBit/s
06 - 3 MBit/s
07 - 3 MBit/s
```

08 - 3 MBit/s
09 - 3 MBit/s
10 - 3 MBit/s
11 - 3 MBit/s
12 - 3 MBit/s
13 - 3 MBit/s
14 - 3 MBit/s
15 - 3 MBit/s
16 - 3 MBit/s
17 - 3 MBit/s
18 - 3 MBit/s
19 - 3 MBit/s
20 - 3 MBit/s
21 - 3 MBit/s
22 - 3 MBit/s
23 - 3 MBit/s
24 - 3 MBit/s
25 - 3 MBit/s
26 - 3 MBit/s
27 - 3 MBit/s
28 - 3 MBit/s
29 - 3 MBit/s
30 - 3 MBit/s
31 - 3 MBit/s
32 - 3 MBit/s
33 - 3 MBit/s
34 - 3 MBit/s
35 - 3 MBit/s
36 - 3 MBit/s
37 - 3 MBit/s
38 - 3 MBit/s
39 - 3 MBit/s
40 - 3 MBit/s
41 - 3 MBit/s

```

42 - 3 MBit/s
43 - 3 MBit/s
44 - 3 MBit/s
45 - 3 MBit/s
46 - 3 MBit/s
47 - 3 MBit/s
49 - 3 MBit/s
50 - 3 MBit/s
51 - 3 MBit/s
52 - 3 MBit/s
53 - 3 MBit/s
54 - 3 MBit/s
55 - 3 MBit/s
56 - 3 MBit/s
57 - 3 MBit/s
58 - 3 MBit/s
59 - 3 MBit/s
00 - 3 MBit/s
01 - 3 MBit/s

```

As it is possible to see, it clearly reports a bitrate of 3 *Mbit/s*, even though 12 *Mbit/s* was set before.

Having a look at the iPerf server log, it is possible to already understand that there should be something wrong with the “iw dev <interface> station dump” output:

```

-----
Server listening on UDP port 7000
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
-----

[  3] local 10.10.6.102 port 7000 connected with 10.10.6.103 port
    43265
[ ID] Interval          Transfer    Bandwidth      Jitter    Lost/Total
      Datagrams

```

[3]	0.0– 2.0	sec	1.75	MBytes	7.35	Mbits/sec	0.152	ms	1/ 1251
	(0.08%)								
[3]	2.0– 4.0	sec	1.75	MBytes	7.34	Mbits/sec	0.170	ms	0/ 1248
	(0%)								
[3]	4.0– 6.0	sec	1.75	MBytes	7.34	Mbits/sec	0.137	ms	0/ 1248
	(0%)								
[3]	6.0– 8.0	sec	1.75	MBytes	7.34	Mbits/sec	0.110	ms	0/ 1249
	(0%)								
[3]	8.0–10.0	sec	1.75	MBytes	7.34	Mbits/sec	0.135	ms	0/ 1248
	(0%)								
[3]	10.0–12.0	sec	1.75	MBytes	7.34	Mbits/sec	0.114	ms	0/ 1248
	(0%)								
[3]	12.0–14.0	sec	1.75	MBytes	7.34	Mbits/sec	0.159	ms	0/ 1248
	(0%)								
[3]	14.0–16.0	sec	1.75	MBytes	7.34	Mbits/sec	0.152	ms	0/ 1249
	(0%)								
[3]	16.0–18.0	sec	1.75	MBytes	7.34	Mbits/sec	0.143	ms	0/ 1248
	(0%)								
[3]	18.0–20.0	sec	1.75	MBytes	7.34	Mbits/sec	0.132	ms	0/ 1249
	(0%)								
[3]	20.0–22.0	sec	1.75	MBytes	7.34	Mbits/sec	0.118	ms	0/ 1248
	(0%)								
[3]	22.0–24.0	sec	1.75	MBytes	7.34	Mbits/sec	0.119	ms	0/ 1248
	(0%)								
[3]	24.0–26.0	sec	1.75	MBytes	7.34	Mbits/sec	0.150	ms	0/ 1248
	(0%)								
[3]	26.0–28.0	sec	1.75	MBytes	7.34	Mbits/sec	0.123	ms	0/ 1249
	(0%)								
[3]	28.0–30.0	sec	1.75	MBytes	7.34	Mbits/sec	0.135	ms	0/ 1248
	(0%)								
[3]	30.0–32.0	sec	1.75	MBytes	7.34	Mbits/sec	0.102	ms	0/ 1248
	(0%)								
[3]	32.0–34.0	sec	1.75	MBytes	7.34	Mbits/sec	0.111	ms	0/ 1248
	(0%)								

[3]	34.0-36.0	sec	1.75	MBytes	7.34	Mbits/sec	0.128	ms	0/	1249
		(0%)									
[3]	36.0-38.0	sec	1.75	MBytes	7.34	Mbits/sec	0.164	ms	0/	1248
		(0%)									
[3]	38.0-40.0	sec	1.75	MBytes	7.34	Mbits/sec	0.132	ms	0/	1248
		(0%)									
[3]	40.0-42.0	sec	1.75	MBytes	7.34	Mbits/sec	0.131	ms	0/	1249
		(0%)									
[3]	42.0-44.0	sec	1.75	MBytes	7.34	Mbits/sec	0.125	ms	0/	1248
		(0%)									
[3]	44.0-46.0	sec	1.75	MBytes	7.34	Mbits/sec	0.131	ms	0/	1248
		(0%)									
[3]	46.0-48.0	sec	1.75	MBytes	7.34	Mbits/sec	0.124	ms	0/	1249
		(0%)									
[3]	48.0-50.0	sec	1.75	MBytes	7.34	Mbits/sec	0.172	ms	0/	1248
		(0%)									
[3]	50.0-52.0	sec	1.75	MBytes	7.34	Mbits/sec	0.156	ms	0/	1249
		(0%)									
[3]	52.0-54.0	sec	1.75	MBytes	7.34	Mbits/sec	0.135	ms	0/	1248
		(0%)									
[3]	54.0-56.0	sec	1.75	MBytes	7.34	Mbits/sec	0.146	ms	0/	1248
		(0%)									
[3]	56.0-58.0	sec	1.75	MBytes	7.34	Mbits/sec	0.149	ms	0/	1248
		(0%)									
[3]	0.0-60.0	sec	52.5	MBytes	7.34	Mbits/sec	0.127	ms	1/37450	
		(0.0027%)									

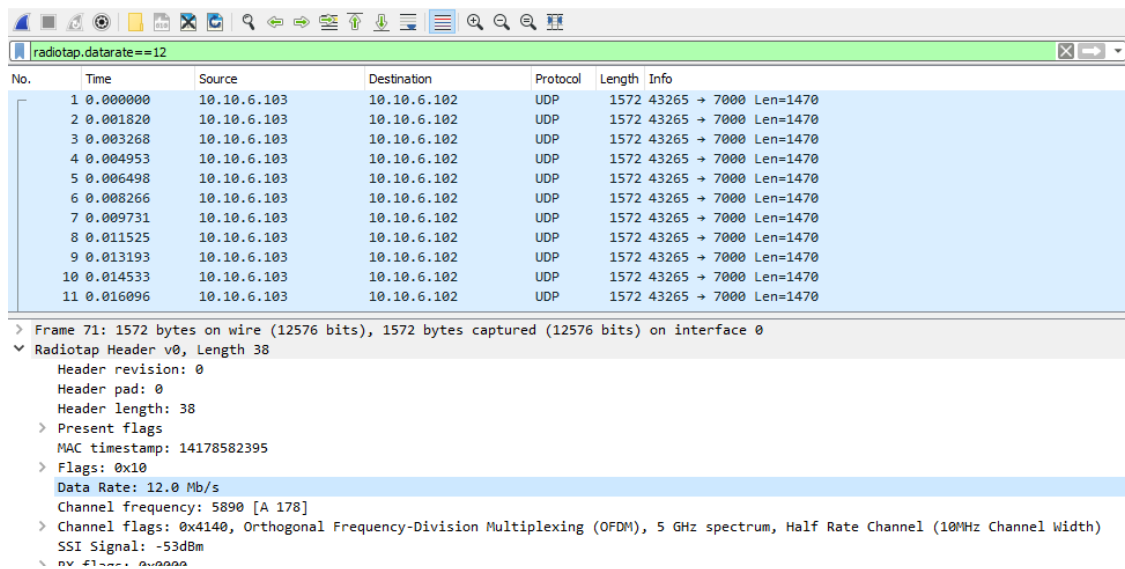
In fact reaching a throughput (“Bandwidth”) of 7.34 Mbit/s with 0% packet loss would be impossible with a 3 Mbit/s physical bitrate.

Moreover, analyzing the traffic captured by the APU_102 board and filtering for the RadioTap header’s field related to the data rate (as shown in figure 5.8), it was possible to observe that all the UDP frames are effectively transmitted by iPerf using the 12 Mbit/s mode, which is really the expected one.

This also confirms the fact that the RadioTap information should be the correct and reliable one.

As an important consideration, repeating exactly the 12M/7M test few days after, led to `iw` always giving the correct information (12 *Mbit/s*), which seems to confirm that the behavior we described here is not observed all the times and it is not so easy to predict.

As a workaround, using the RadioTap header information should always give the expected information about the data rate.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
2	0.001820	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
3	0.003268	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
4	0.004953	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
5	0.006498	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
6	0.008266	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
7	0.009731	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
8	0.011525	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
9	0.013193	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
10	0.014533	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470
11	0.016096	10.10.6.103	10.10.6.102	UDP	1572	43265 → 7000 Len=1470

> Frame 71: 1572 bytes on wire (12576 bits), 1572 bytes captured (12576 bits) on interface 0

▼ Radiotap Header v0, Length 38

- Header revision: 0
- Header pad: 0
- Header length: 38
- > Present flags
 - MAC timestamp: 14178582395
 - > Flags: 0x10
 - Data Rate: 12.0 Mb/s**
 - Channel frequency: 5890 [A 178]
 - > Channel flags: 0x4140, Orthogonal Frequency-Division Multiplexing (OFDM), 5 GHz spectrum, Half Rate Channel (10MHz Channel Width)
 - SSI Signal: -53dBm

Figure 5.8: Filtering for the data rate field of the RadioTap header in the test previously described, in Wireshark, after capturing the traffic coming from the APU_103 board with APU_102; using `radiotap.datarate==12` as filter shows that all the UDP packets should have been transmitted using a 12 *MBit/s* mode, which is not what `iw` was telling us in that specific case

The information about the received power seems to be, instead, much more reliable.

Chapter 6

Measurements on the APU boards and results

6.1 Throughput and packet loss measurements

After setting up the system with the APU board, the UNEX DHXA-222 WLAN cards and the latest OpenWrt version (18.06.1), which actually represents the final software platform of this work, we were able to perform more systematic measurements to assess the system performance. We heavily relied on iPerf, as measurement tool.

After finding out the bandwidth usage and instability problems of the ALIX boards, we decided to concentrate the tests on the newer *PC Engines* boards.

All the measurements will follow this scheme, after a short introduction: first, the network and desk configuration are shown, indicating all the conditions under which the data was collected, second, the used scripts and commands are reported, third, the obtained plots are shown and commented, fourth, if necessary, additional considerations are presented to the reader.

This first set of measurements aims at collecting and analyzing data for what concerns the reachable throughput and the packet loss for different values of offered traffic (the

iPerf client *-b* parameter); the measurement are repeated three times for three physical layer bitrate values: 3, 6 and 12 Mbit/s.

6.1.1 Network configuration and conditions

The first measurements were performed under the following conditions:

- Variable physical bitrate
- 5 dBi antenna (MIMO) connected to the boards
- 3 dBm txpower set in OpenWrt (so, this should be the transmitted power without the additional antenna gain of 5 dBi)
- Boards placed very near to each other, with 16 *cm* between the two enclosures
- Duration of each single test: 60 seconds
- iPerf port: 7000
- iPerf report interval: 2 seconds
- Layer 4 protocol: UDP
- Payload size: 1470 B (default iPerf value)
- UDP buffer size: 208 KB (default iPerf value)
- Channel: 178

UDP was chosen as protocol since it seemed to be the best choice both for simulating IP-based transmissions, being compatible with iPerf, which is a de-facto standard in network measurements, and for simulating future non-IP WSMs transmission when a proper stack will be implemented.

All these transmissions are based upon the OCB mode, according to the IEEE standards.

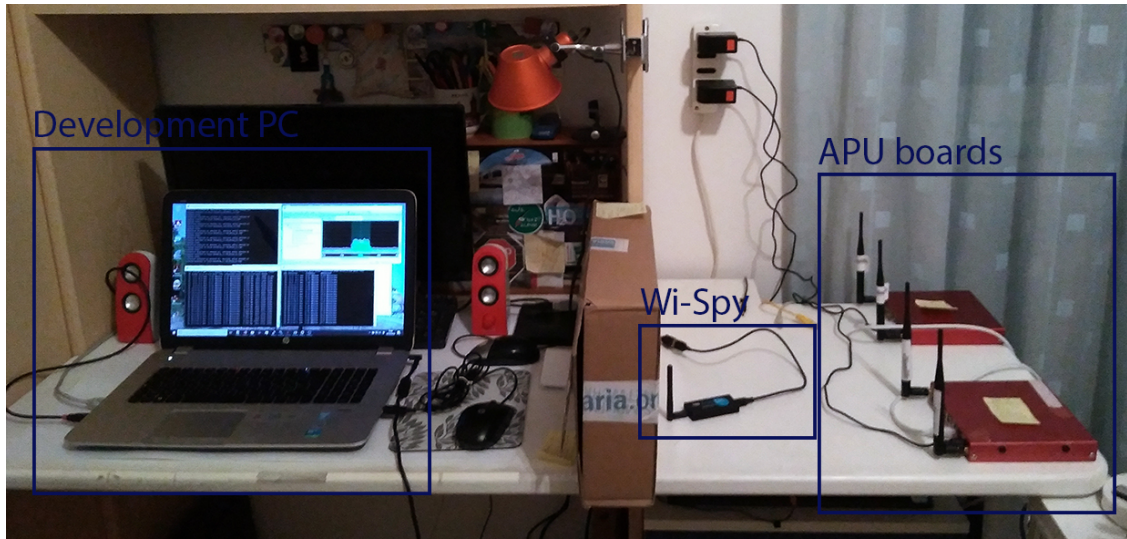


Figure 6.1: Configuration of the desk and network topology when performing the first throughput and packet loss measurements

The configuration of the desk during the tests is reported in figure 6.1

The APU_103 board was working as iPerf client, sending packets to the APU_102 board, in which an iPerf server was started.

6.1.2 Scripts and commands

The script we used is reported below. As most of the scripts presented in this chapter, it is based on the fact that now the date and time are synchronized thanks to *chrony*, as detailed in section 5.6, thus it needs the board to be connected to the Internet through its Ethernet ports or, in case of unavailable Internet connection during the tests, to be connected at least once to synchronize its time, which should then remain sufficiently synchronized for the duration of the tests themselves.

For these first measurements it is not strictly necessary to synchronize, but it was nevertheless very useful for example when we decided to study more in depth the current physical layer rate reported by “iw”, as mentioned in section 5.8.

iperfclient7000.sh

```

1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfclientb {
13     iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000
14 }
15
16 rates=( 0.5M 1M 1.5M 2M 2.5M 2.6M 2.7M 3M 5M 5.4M 5.5M 5.6M 5.7M 5.8M
17         5.9M 6M 7M 9M 10M 11M 11.5M 11.6M 11.7M 11.8M 11.9M 12M 100M 500M )
18
19 physrates=( 3 6 12 )
20
21 currwait=0
22 i=0
23 p=0
24 while [ $p -lt ${#physrates[@]} ]; do
25     wifiarate=$(( ${physrates[$p]} * 2 )
26     iw dev wlan0 set bitrates legacy-5 $wifiarate
27     echo "Rate set to ${physrates[$p]} (iw dev wlan0 set bitrates legacy
28         -5 $wifiarate)"
29     sleep 1
30     while [ $i -lt ${#rates[@]} ]; do
31         echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":"
32             " -f3 | cut -d" " -f1), waiting until second $currwait"
33         waituntil $currwait

```

```
30     echo "-----"
31     echo "Running test with -b ${rates[$i]} (iw dev wlan0 set bitrates
    ... ${physrates[$p]}")
32     iperfclientb ${rates[$i]}
33     echo "Test with -b ${rates[$i]} terminated"
34     echo "-----"
35     i=$((i+1))
36     currwait=$(( (currwait+10)%60 ))
37 done
38     sleep 1
39     p=$((p+1))
40     i=0
41 done
```

The script defines a *bash* function, called *waituntil* and accepting one argument, that waits until a certain “seconds’ value” (passed as first argument) occurs on the current time and date.

For instance, if the current time is 15:46:23 and the argument is 30, the script will wait for 7 seconds just as soon as the current time becomes 15:46:30.

Since this “seconds’ number” is specified inside the script, by using the same number on two boards (i.e. on two scripts) at the same time, they can perform synchronized tests, waiting until the same date and time occurs inside both of them.

Of course this is going to work only if both the boards have a reasonably synchronized time reference and that is where *chrony* plays its role.

The current number of seconds is obtained thanks to the “date” command, which is then filtered by means of “cut”, extracting the relevant data (i.e. the two digits indicating the current value of the seconds).

A second function is defined just to shorten each call to the iPerf client in the main script body.

Two arrays are then defined, containing all the offered traffic rates that should be tested and the physical rates to be used.

For each of the physical rates (which are set in lines 23 and 24), all the values specified in “*rates*” are tested.

The “sleep 1” command is just there to be sure that the system completed its configuration in any possible situation, before proceeding further.

Then, an iPerf client with a certain *-b* value (i.e. one of the values contained in “*rates*”) is started and it performs a 60 second lasting test.

The synchronization mechanism takes into account this test duration: the first measurement always starts when the time reaches “00” as seconds’ value (`currwait=0`); then, the script will wait until this value reaches “10”, then “20” at the following iteration, and so on, adding 10 seconds every time (`currwait=$(((currwait+10)%60))`). This avoids the client finishing just after this “00” value and have to wait for more than 50 seconds before starting the next test.

In the first version of the script, which was actually used to collect data for the 3 Mbit/s physical rate configuration, there was a different definition of the *rates* arrays, that was then improved for the later tests.

That is why it is possible to see a little mismatch between the point in the 3 Mbit/s plot and what is reported inside this array.

Additional measurements, by manually running an iPerf client and server on the two boards, have been then performed to further improve the final plots:

- we added 4 values of offered rates to the 6 Mbit/s tests: *4M*, *13M*, *14M*, *15M*
- we added 7 values to the 12 Mbit/s tests: *4M*, *13M*, *14M*, *15M*, *17M*, *19M*, *21M*

The values related to 100M and 500M were then discarded since they did not allow to properly show all the data in a single plot and since the results were almost the same as the previous measurements (*10M* or *15M* or *21M*).

This script was launched on the APU_103 board, working as client, while on APU_102 it was sufficient to start a continuously running server with:

```
iperf -s -u -i 2 -p 7000 2>&1 | tee -a "ServerLog.txt"
```

To briefly comment this command, it is possible to say that:

- An iPerf server is started on port 7000 (`-p 7000`), with periodic reports every 2 seconds (`-i 2`) and expecting UDP datagrams (`-u`)
- The standard error *stderr*, with descriptor 2, is redirected to the standard output *stdout*, with descriptor 1, in order to log everything the program is providing as output (`2>&1`)
- The “tee” command used to take the output of iPerf (both *stderr* and *stdout* at this point) and both display it on standard output (i.e. on the SSH console) and store it inside a log file (“ServerLog.txt”), appending new information to the already existing one (`-a`)

6.1.3 Plots and results

The results are shown in the following plots, comparing packet loss and reachable throughput for different values of offered traffic.

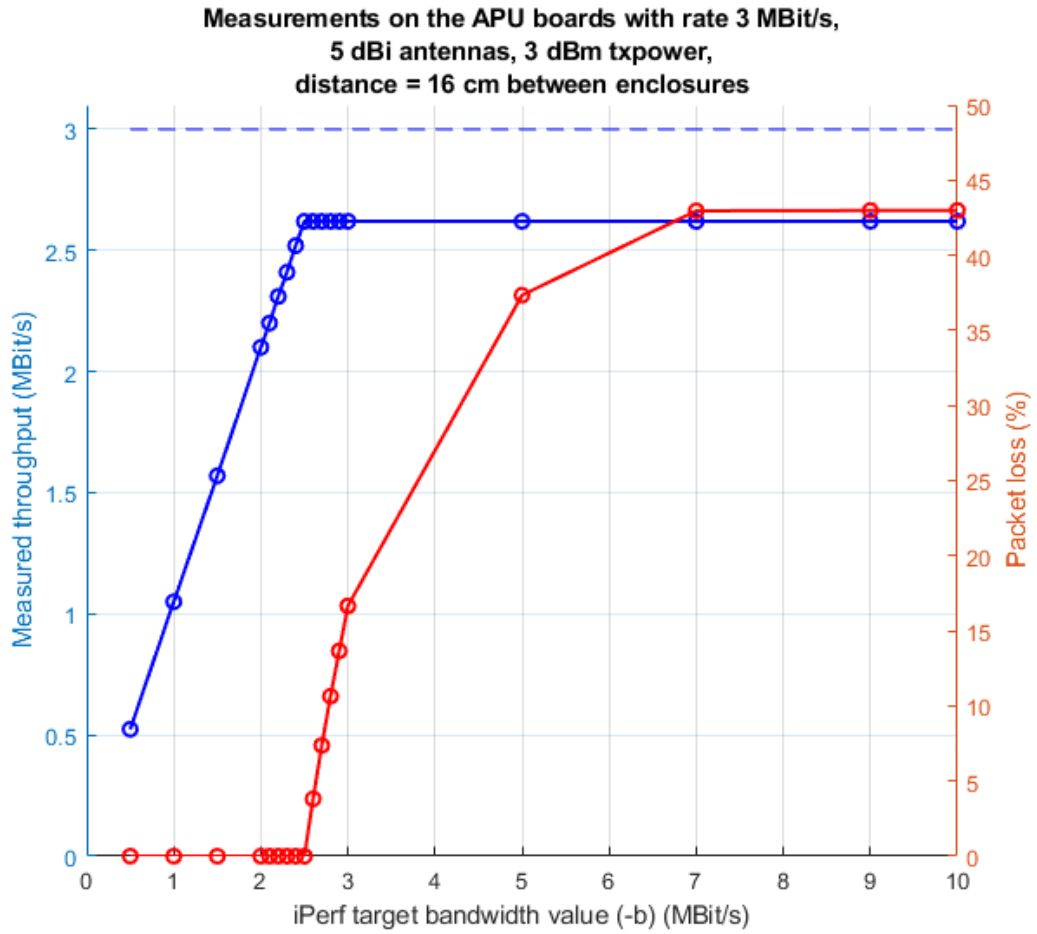


Figure 6.2: Measurements for 3 Mbit/s of physical layer bitrate

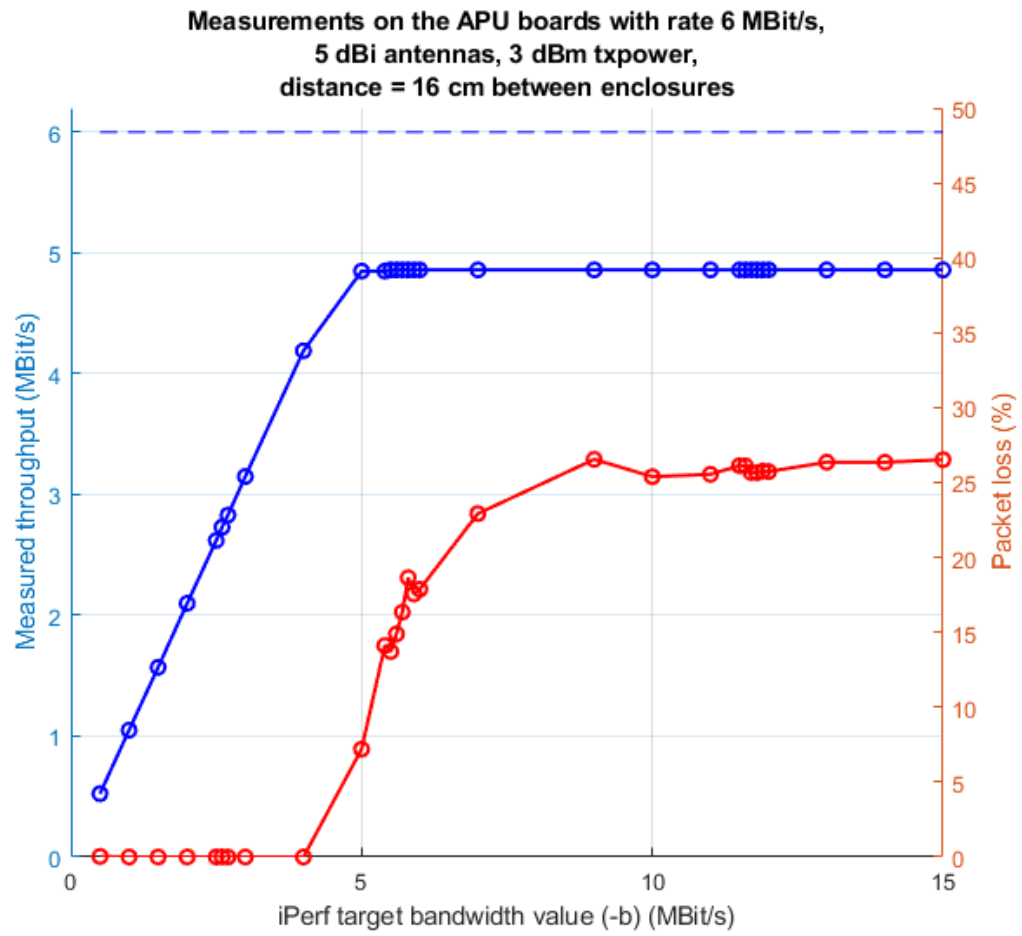


Figure 6.3: Measurements for 6 Mbit/s of physical layer bitrate

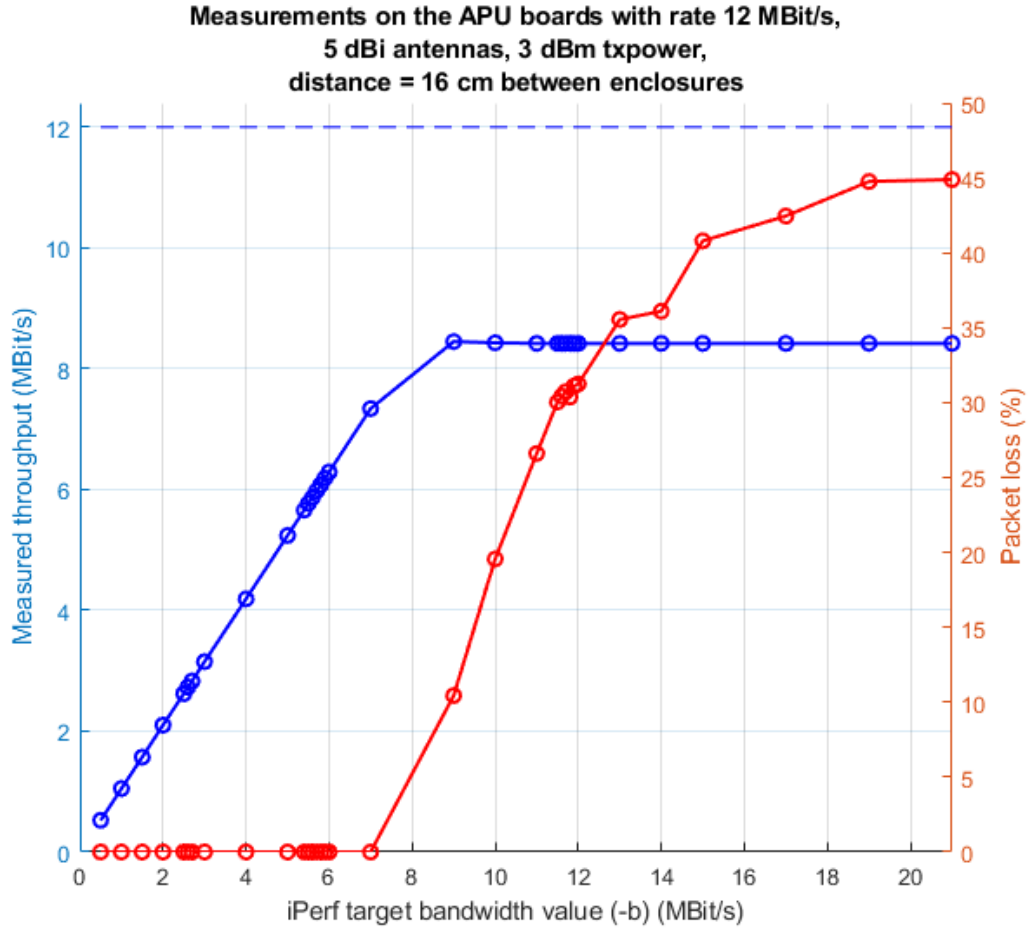


Figure 6.4: Measurements for 12 Mbit/s of physical layer bitrate

As it is possible to see from the plots, the ideal throughput is never reached, even with a quite big and “optimized” UDP payload as 1470B can be.

We noticed that as the physical rate is lower, we can arrive near to the ideal value, at least as far as what iPerf is reporting, while as it is higher, the gap between the ideal and real throughput becomes larger.

In particular, the following obtained values are worth to be mentioned:

Physical bitrate	Maximum throughput*	Maximum -b with 0% loss**
3 Mbit/s	2.62 Mbit/s	2.5M
6 Mbit/s	4.85 Mbit/s	4M
12 Mbit/s	8.45 Mbit/s	7M

Table 6.1: Important measurement data

*With the boards being placed quite near to each other

**It is not the absolute maximum: it's the maximum over the selected values

Looking at the packet loss values, it is possible to notice that they progressively increase, as the offered traffic is increased over the aforementioned 0% loss values.

These values of offered traffic which start to generate packet loss, of course, are always more than what the network can bear, generating a congested situation.

This trend stabilizes around a certain percentage as -b continues to grow, which is:

- ~43% for 3 Mbit/s
- ~26% for 6 Mbit/s
- ~45% for 12 Mbit/s

This fact can be explained, very probably, with the presence of a WNIC buffer (or a mac80211 buffer, or a driver buffer) between the iPerf application, *mac80211* and the physical hardware sending the frames over the air, which has its throughput limited to the measured values (for instance, it cannot ever send 100 Mbit/s of data with an 802.11p stack).

The exact position of this buffer has still to be analyzed more in details (the driver source code resulted to be very complex and it is quite difficult to find a sufficiently complete datasheet for the UNEX DHXA-222 cards); for the sake of simplicity, however, we will call it “driver buffer”.

Here comes the concept of “socket buffer” (or “UDP buffer size”)¹: iPerf uses sockets

¹Thanks to Robert McMahon, from the iPerf 2 forums, for helping me clarify this concept

to send data out over the wireless medium and it sets a certain socket buffer size which is used by higher layers to accept the data to be sent:

```
tcp_window_size.c
132 rc = setsockopt( inSock, SOL_SOCKET, SO_SNDBUF,
133   (char*) &newTCPWin, sizeof( newTCPWin ));
```

If this socket size is greater than the “driver buffer” size, which should be at a lower level and nearer to the WNIC signal processing hardware, the data will be accepted by the network stack as long as the iPerf buffer is not full.

The data, however, even if the kernel tries to deal with it as quick as possible, may be dropped due to this lower level buffer not being able to accept new frames, for example if the offered traffic is too much with respect to what the WNIC can deliver.

In the opposite case, when the buffer used by the iPerf’s socket has a lower size than the driver’s buffer (which seems also to be different between *ath5k*+UNIX DCMA 86P on the ALIX boards and *ath9k*+UNIX DHXA-222 on the APU boards), a lower loss should be observed, as it is the networking stack that it should now be preventing too much data to flow towards the driver, blocking new writes to the socket as long as the iPerf buffer is not able to accommodate the new data. The drawback is that this can limit the traffic, coming from the application, that is able to flow out from the board.

It is also important to take into account that the process of filling the buffer has a different evolution depending on the packet size, and, as we observed, this is not always linear with the size of the payload and the length of the iPerf buffer; moreover, drivers may also implement flow control mechanisms, which can influence how buffers are managed.

iPerf usually works, when in UDP mode, with a default buffer size of 208 *KB* (212992 *B*), which can be changed using the “-w” parameter.

Since iPerf reserves this space by means of a call to `setsockopt()`, the buffer size specified by the user is normally doubled by the kernel, to maintain a certain overhead

over the specified size².

The maximum (and minimum) value is actually limited by the kernel, and it can be changed by using the `sysctl` utility.

In particular, a script like the following one can be used:

Set_kernel_params.sh

```
1  #!/bin/sh
2  sysctl -w net.ipv4.udp_rmem_min=4096
3  sysctl -w net.ipv4.udp_wmem_min=4096
4  sysctl -w net.core.rmem_max=212992
5  sysctl -w net.core.rmem_default=212992
6  sysctl -w net.core.wmem_max=212992
7  sysctl -w net.core.wmem_default=212992
```

Here the default APU1d OpenWrt 18.06.1 values are reported, but any other value can be set by the user, depending on his or her needs.

In particular, the meaning of each kernel parameter is resumed below:

- **net.ipv4.udp_rmem_min**: “Minimal size, in bytes, of receive buffers used by UDP sockets in moderation”³
- **net.ipv4.udp_wmem_min**: “Minimal size, in bytes, of send buffer used by UDP sockets in moderation”⁴
- **net.core.rmem_max**: maximum receive buffer size for all the connections
- **net.core.rmem_default**: default receive buffer size for all the connections
- **net.core.wmem_max**: maximum send buffer size for all the connections
- **net.core.wmem_default**: default send buffer size for all the connections

²As reported in “man 7 socket”, in the part related to “SO_SNDBUF”

³From “man udp”.

⁴From “man udp”.

The effect of the buffer can also be visualized in the following plot:

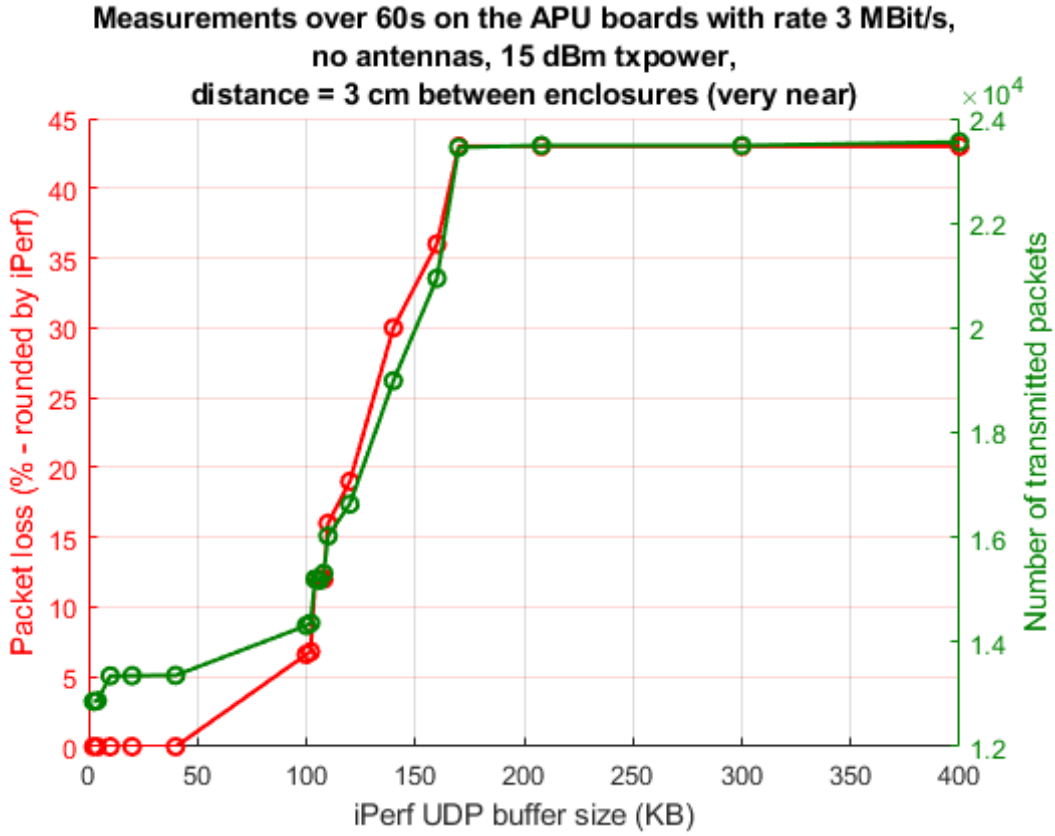


Figure 6.5: Evolution of packet loss and number of transmitted packets as the iPerf send socket buffer size is increased

This plot has been obtained by choosing a physical rate of 3 Mbit/s, “-b 10M”, OpenWrt txpower equal to 15 dBm and varying the buffer size with “-w”; no antennas were connected (but the boards were placed very near to each other).

Other than that, the conditions were the same of the main test reported in this section.

The following values of “-w” have been tested:

-w	Buffer size (KBytes)	Packet loss (%)	Transmitted packets	Throughput (Mbit/s)
1K	2.25	0.078%	12865	2.52
2K	4	0%	12875	2.52
5K	10	0%	13351	2.62
10K	20	0%	13356	2.62
20K	40	0%	13362	2.62
50K	100	6.6%	14312	2.62
51K	102	6.8%	14360	2.62
52K	104	12%	15202	2.62
53K	106	12%	15173	2.62
54K	108	12%	15310	2.62
55K	110	16%	16026	2.62
60K	120	19%	16633	2.62
70K	140	30%	18989	2.62
80K	160	36%	20946	2.62
90K	180	43%	23447	2.62
104K	208	43%	23485	2.62
150K	300	43%	23481	2.62
200K	400	43%	23550	2.62

As it is possible to see, the number of transmitted packets tends to increase as the buffer size is increased, letting iPerf try to write more data, until a certain value is reached.

The same trend is followed by the percentage packet loss, which is initially 0% or very near to this value and it is then increased as we let more data flow towards the driver.

With very low values of buffer size (2.25K, 4K) we also observed a small reduction in the throughput, which was lowered to a maximum of 2.52Mbit/s , which is a little less than the value measured before.

It is important to note that this test has been performed trying to offer around 10Mbit/s , which can never be fully transmitted when a physical datarate of 3Mbit/s is selected.

A better and more detailed characterization of the buffer behavior is presented in section 6.2.

6.1.4 Additional considerations

During all the previous tests (except the one related to the socket buffer), the Wi-Spy device has been kept connected to the development PC, in order to look at the spectrum usage.

We were able to verify that the ALIX boards’ issues with the frequency usage and the transmission power seemed not to affect the APU boards (with the UNEX DHXA-222 cards), since the channel usage was much more polite (with no interference on nearby channels, even though the transmission power was lower, in this case) and no “unexpected power increase” was ever observed, even after a lot of data points had been collected.

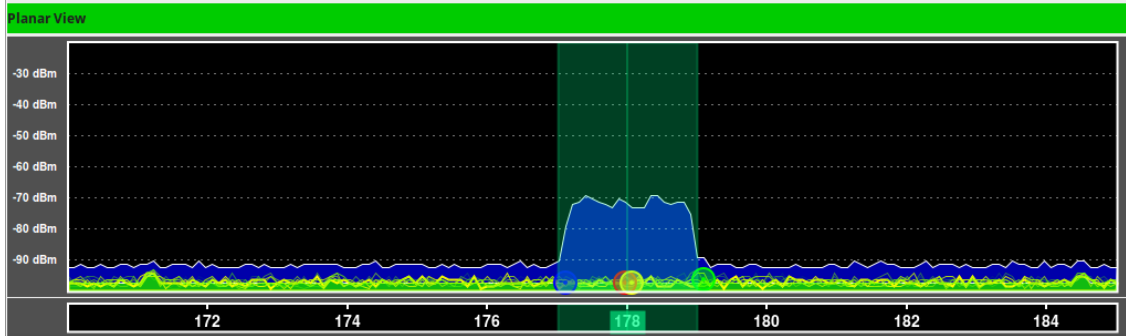


Figure 6.6: Planar view showing the maximum spectrum usage during the previous tests, which was quite the expected one

6.2 Characterization of the buffered transmission

After looking at the first packet loss and throughput measurements, we noticed a packet loss which is actually compatible with the presence of one or more buffers in the system, mainly WNIC/mac80211-side and also application-side, due to the presence of a

UDP transmission buffer, which can be set in any application, for instance, by means of “`setsockopt()`”.

In order to better characterize the buffered transmission behavior, we performed additional measurements using a purposely prepared script and a modified version of iPerf 2.0.12, patching its UDP send loop to output useful debug informations, which are normally unavailable to the user.

We first of all chose a physical rate of 6 Mbit/s and tried to measure the packet loss and number of transmitted packets, progressively increasing the “UDP window size” (i.e. the UDP socket send buffer size), for different values of packet size. Similar results are to be expected also for the other selectable vales of physical rate.

The value of `-w` was instead kept constant, at its default value (208 KB), during all the previous measurements.

We then tried to better explain this behavior by collecting and analyzing data related to both the evolution of the UDP send buffer and to the evolution of the iPerf “running delay”, which is better described in the section related to the “Scripts and commands, iPerf UDP send loop”.

6.2.1 Network configuration and conditions

The buffer related measurements were performed under the following conditions.

Packet loss and number of packets:

- 6 Mbit/s physical bitrate
- 5 dBi antenna (MIMO) connected to the boards
- 3 dBm txpower set in OpenWrt (so, this should be the transmitted power without the additional antenna gain of 5 dBi)
- Boards placed very near to each other, with 16 *cm* between the two enclosures
- Duration of each single test: 60 seconds

- iPerf port: 7000
- iPerf report interval: 2 seconds
- Layer 4 protocol: UDP
- Payload size: variable
- UDP buffer size: variable
- Channel: 178

Evolution of send buffer and running delay:

- 3 Mbit/s physical bitrate
- no antennas connected to the boards, which were in any case placed very near to each other
- 15 dBm txpower set in OpenWrt
- Boards placed very near to each other, with around 2.5 cm between the two enclosures
- Duration of each single test: 20 seconds
- iPerf port: 7000
- iPerf report interval: variable (0.1 seconds or 2 seconds)
- Layer 4 protocol: UDP
- Payload size: 1470 B (default)
- UDP buffer size: variable
- Channel: 178

The second set of measurements could be performed without the antennas, as the distance between the boards was small enough for any error-free communication over the air and the received or transmitted power was irrelevant in this case, provided that the communication could happen without errors due to the wireless medium.

All these transmissions are based again upon the OCB mode.

6.2.2 Scripts and commands, iPerf UDP send loop

For the first set of measurements, it was sufficient to set the physical rate to 6 Mbit/s (with `iw`) and manually launch couples of iPerf server and client with variable values of “-w” and “-l”.

In order to simplify and automatize the whole process, we wrote two scripts, one for the iPerf clients (to be launched on APU_103 and connecting to APU_102 with IP 10.10.6.102) and one for the iPerf servers (to be launched on APU_102).

iperfserver_wtest.sh (APU_102)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfserverblw {
13     echo "# iperf -s -u -i 2 -p 7000 -l $2 -w $3 -t 64" >> "Logs/
        ServerLog_b_$1_l_$2B_w_$3.txt"
14     iperf -s -u -i 2 -p 7000 -l $2 -w $3 -t 64 | tee -a "Logs/
        ServerLog_b_$1_l_$2B_w_$3.txt"
```

```

15 }
16
17 # Check if the Logs directory exists. If not, create it.
18 if [ ! -d "./Logs" ]; then
19     echo "Logs directory missing. It will be created now"
20     mkdir Logs
21 else
22     echo "Logs directory exists. Make a backup of its content before
        proceeding."
23 fi
24 read -p "Press enter to continue"
25
26 echo "Setting rate to 6 MBit/s..."
27 iw dev wlan0 set bitrates legacy-5 12
28 sleep 1
29
30 windows=( 2.25K 5K 10K 20K 50K 60K 70K 80K 90K 104K 150K 200K )
31 len=104
32 b=10M
33
34 currwait=0
35 i=0
36 while [ $i -lt ${#windows[@]} ]; do
37     echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1), waiting until second $currwait"
38     waituntil $currwait
39     echo "-----"
40     echo "Running test with -b $b -l $len -w ${windows[$i]}"
41     iperfserverblw $b $len ${windows[$i]}
42     echo "Test with -b $b -l $len -w ${windows[$i]} terminated"
43     echo "-----"
44     i=$((i+1))
45     currwait=$(( (currwait+10)%60 ))
46 done

```

iperfclient_wtest.sh (APU_103)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfclientblw {
13     echo "# iperf -c 10.10.6.102 -u -i 2 -t 60 -p 7000 -l $2 -b $1 -w $3
14         " >> "Logs/ClientLog_b_$1_l_$2B_w_$3.txt"
15     iperf -c 10.10.6.102 -u -i 2 -t 60 -p 7000 -l $2 -b $1 -w $3 | tee -
16         a "Logs/ClientLog_b_$1_l_$2B_w_$3.txt"
17 }
18
19 # Check if the Logs directory exists. If not, create it.
20 if [ ! -d "./Logs" ]; then
21     echo "Logs directory missing. It will be created now"
22     mkdir Logs
23 else
24     echo "Logs directory exists. Make a backup of its content before
25         proceeding."
26 fi
27 read -p "Press enter to continue"
28
29 echo "Setting rate to 6 MBit/s..."
30 iw dev wlan0 set bitrates legacy-5 12
31 sleep 1
```

```

30 windows=( 2.25K 5K 10K 20K 50K 60K 70K 80K 90K 104K 150K 200K )
31 len=104
32 b=10M
33
34 currwait=0
35 i=0
36 while [ $i -lt ${#windows[@]} ]; do
37     echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1), waiting until second $currwait"
38     waituntil $currwait
39     echo "-----"
40     echo "Running test with -b $b -l $len -w ${windows[$i]}"
41     sleep 2
42     iperfclientblw $b $len ${windows[$i]}
43     echo "Test with -b $b -l $len -w ${windows[$i]} terminated"
44     echo "-----"
45     i=$((i+1))
46     currwait=$(( (currwait+10)%60 ))
47 done

```

Both the scripts are based on the “iperfclient7000.sh” script presented in the previous section, synchronizing the execution of the server on one board (which is set to last for 64 seconds to allow the client to have some margin) and on the client on the other board, using NTP and the “waituntil” bash function. The client is started 2 seconds after the server to let the system completely configure the socket, in such a way that the client, when started, will find a ready server on the other board, avoiding any “cold start” packet loss.

Each script performs a set of measurements for different values of “-w”, given a certain packet size.

In particular, the following values are tested:

-w	Buffer size (KBytes)
2.25K	4
5K	10
10K	20
20K	40
50K	100
60K	120
70K	140
80K	160
90K	180
104K	208
150K	300
200K	400

Then, the packet size was changed by giving a different value to the “l” variable, on line 31 of both scripts (it may be important to set the same value both server and client side, so we always gave the same value to both of them).

The server and client outputs are logged respectively to two log files, using tee just like before:

- `ServerLog_b_10M_l_<length>_w_<buffer size in K>.txt`
- `ClientLog_b_10M_l_<length>_w_<buffer size in K>.txt`

They were saved inside a “Logs” subfolder, which is automatically created by the script if it was not already present (lines 17-24), and they have been both parsed to extract the

relevant data and stored for any future reference.

For what concerns the second set of measurements, instead, it was very difficult to simply use bash scripts or already available tools, since the evolution of the UDP transmit buffer size is not immediately obtainable, at least not in a synchronized way with iPerf sending packets over the networking stack.

For a first analysis of the evolution, in time, of the data inside the transmission queue, it was possible to write a script querying the “/proc/net/udp” file, which lists all the opened UDP sockets, including information about the size of the send and receive queues (respectively with the “tx_queue” and “rx_queue” fields).

```
root@OpenWrt:~# cat /proc/net/udp
sl local_address rem_address  st tx_queue rx_queue tr tm->when retr
53: 67060A0A:0035 00000000:0000 07 00000000:00000000 00:00000000 0000
53: 0100007F:0035 00000000:0000 07 00000000:00000000 00:00000000 0000
53: B701A8C0:0035 00000000:0000 07 00000000:00000000 00:00000000 0000
123: 00000000:007B 00000000:0000 07 00000000:00000000 00:00000000 0000
323: 0100007F:0143 00000000:0000 07 00000000:00000000 00:00000000 0000
728: 67060A0A:9AD8 66060A0A:1B58 01 0001A700:00000000 00:00000000 0000
740: 00000000:26E4 00000000:0000 07 00000000:00000000 00:00000000 0000
root@OpenWrt:~#
```

Figure 6.7: Part of the output of “cat /proc/net/udp”, when executed on the APU_103 board, in which an iPerf client was running, sending data to a server on the APU_102 board. The socket of interest is highlighted.

With reference to figure 6.7, it is possible to highlight the following important fields:

- The “local_address” field lists the local IP address in an hexadecimal little-endian format and the port; in this case 67060A0A corresponds to 10.10.6.103 (0A.0A.06.67) and 9AD8 to the client port (39640).
- The “remote_address” field lists the remote IP address in an hexadecimal little-endian format and the port; in this case 66060A0A corresponds to 10.10.6.102 (0A.0A.06.66) and 1B58 to the server port (specified with “-p 7000”)

- It is also possible to notice a non-empty “tx_queue” (with *0x0001A700* bytes occupied at the moment of querying the file, corresponding to 105 KB), possibly indicating that the application is generating data faster than what the network card is able to send; this field is equal to the kernel memory usage, in bytes, of packets which have been buffered by the application but not yet sent by the kernel’s wireless subsystem and the WNIC. It is expected, as iPerf, in this case, was set with “-b 10M” under a 3 Mbit/s physical rate.

In order to periodically log the “tx_queue” field, the following *bash* script has been written:

procudp_monitor.sh

```

1  #!/bin/bash
2
3  # The script expects as argument the name to append to the log file
   name (Log_txbuf_evolution_<name>.csv
4  if [ $# -ne 1 ]; then
5      echo "Error. One argument was expected."
6      exit 1
7  fi
8
9  # Create the log file in .csv format (to be easily imported inside
   Matlab)
10 echo "Logging on file Log_txbuf_evolution_$1.csv started"
11 echo "Date,Txbuf" > "Log_txbuf_evolution_$1.csv"
12
13 # Wait until the UDP socket is created (i.e. until grep returns
   positively while looking for a line containing the APU_102 and
   APU_103 IP addresses)
14 while true; do
15     cat /proc/net/udp | grep "67060A0A" | grep "66060A0A" 2>&1 > /dev/
       null
16     if [ $? -eq 0 ]; then

```

```

17     break
18 fi
19 done
20
21 # Considering an average 15.5 points/second (from previous tests) =
    403 point for 26 s. The loop iterates for a period which is
    estimated to be around 26 s: this value can then be used to plot
    under Matlab
22 count=0
23 while [ $count -lt 404 ]; do
24     # Extract the tx_queue field from the proper socket in /proc/net/udp
        , using sed (with the substitution command s/)to remove multiple
        consecutive spaces from the selected line (\s is a "space"),
        which can fool the cut utility, then using cat with " " (space)
        and then ":" as delimiters ('cut -d":" -f2' would have selected
        the rx_queue field)
25     txbufB=$(cat /proc/net/udp | grep "67060A0A" | grep "66060A0A" | sed
        's/\s\s\s*/ /g' | cut -d" " -f6 | cut -d":" -f1)
26     # Log this field after converting it to decimal ( $((0x$txbufB)) ),
        together with the current date and time
27     echo "$(date),$(echo $((0x$txbufB)))" >> "Log_txbuf_evolution_$1.csv"
28     # With this value of sleep (0.05 s), 15.5 pts/s seems to be
        collected on average
29     ./millisleep 0.05
30     count=$((count+1))
31 done

```

The script should collect points in time corresponding to an estimated span of 26 seconds from the moment in which the socket was created.

The time at which every point is collected is not accurate, being based on how many points every second several previous test measurements could collect under the same conditions, but it should be fine for general and more qualitative measurements.

For better and more precise measurements, as detailed later, it will be necessary to

modify the iPerf source code.

It is possible to notice that the scripts calls a “`millisleep`” binary, placed inside the same folder.

This is a custom implementation of a very simple sleep utility that is able to support waiting for less than a second. Its code is reported in Appendix F.

Before describing how we performed more precise measurements on the transmit queue and “running delay” evolution, it may be important to briefly describe how iPerf manages the client-side transmission of UDP packets.

We will concentrate on explaining what happens with the options that are actually used in this work; variations to this scheme may be expected as other options are selected.

We also considered iPerf 2.0.12: with iPerf 3 or any future version of iPerf 2 variations may occur.

Taking as reference the conceptual flow chart for the iPerf 2 send loop, reported in figure 6.8, the transmission of UDP packets at a certain “bandwidth” (i.e. at a certain value of “`-b`”) happens thanks to a “running delay”.

The value of offered traffic is translated in a delay that should occur between the transmitted packet with a payload of a certain size (for instance the default size of 1470 B), in order to respect what the user specified with “`-b`”.

This delay, called in the code “`delay_target`”, is computed according to the following formula:

$$t_d = \frac{L \cdot k_{s-ns} \cdot k_{B-b}}{b}$$

with:

- t_d : target delay [ns]
- L : payload size [B]
- $k_{s-ns} = 10^9$: constant for the conversion of seconds to nanoseconds [ns/s]
- $k_{B-b} = 8$: constant for the conversion of bytes to bits [bit/B]

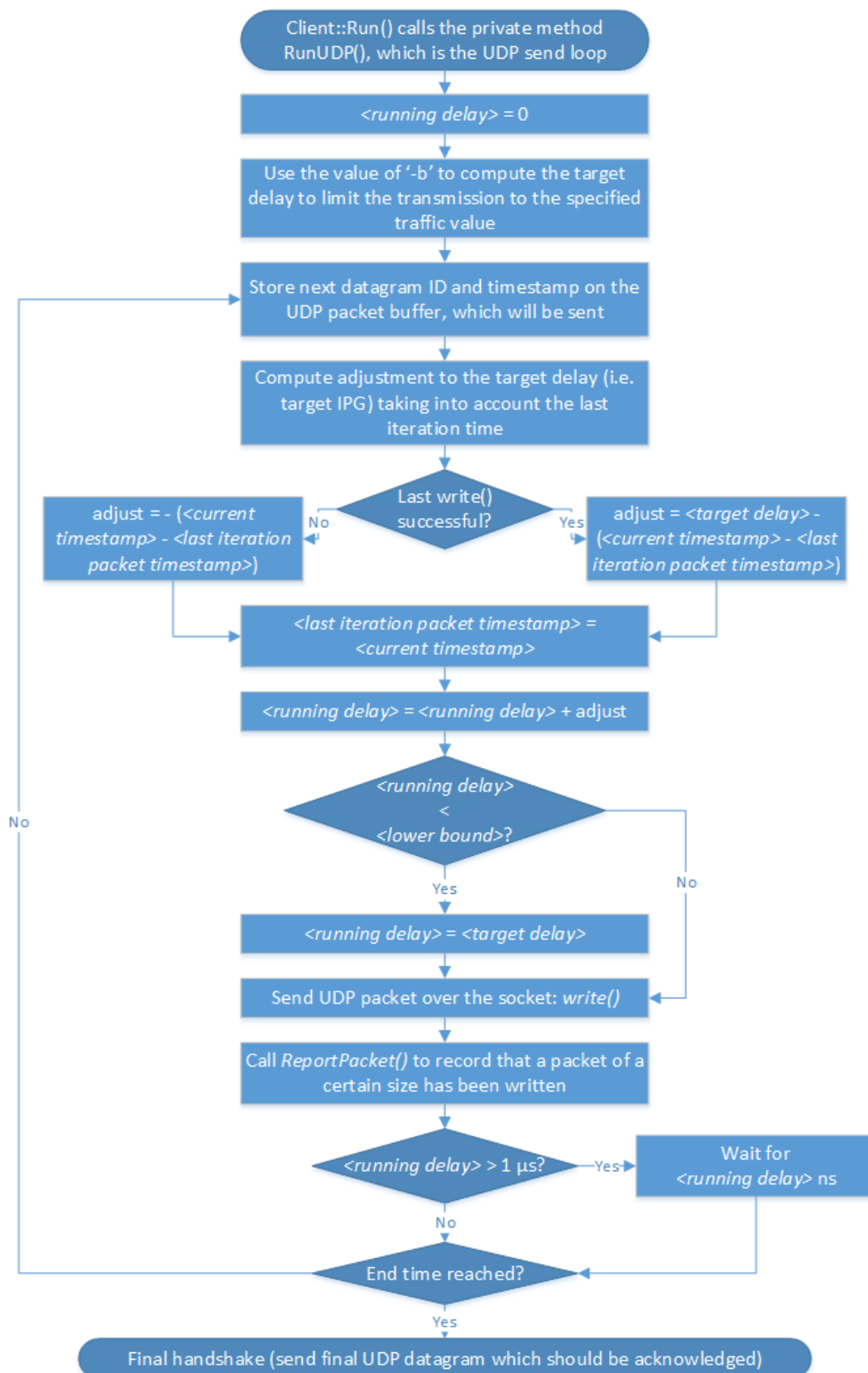


Figure 6.8: Conceptual flow chart of the iPerf 2 UDP send loop, when the -b, -i, -u, -t options are set, as in this work

- b : offered traffic [bit/s]

For instance, with “-b 10M”, with the default payload size the formula returns:

$$t_d = \frac{L \cdot k_{s-ns} \cdot k_{B-b}}{b} = \frac{1470 \cdot 10^9 \cdot 8}{10 \cdot 1024 \cdot 1024} = 1121521 ns \cong 1.1215 ms$$

In an ideal case, if a sleep of 1.1215 ms would be inserted between each packet, the transmission would be at around 10 Mbit/s, which is what the user specified.

However, each iteration of the loop takes time to be executed and this should be taken into account.

This is done thanks to a “running delay”, which is a delay value stored for the whole life of the thread running the client and adjusted at each iteration; it is called “delay” in the code.

The adjustment is computed, at each iteration, in the following way:

- Adjustment = Target delay - last loop iteration time, if the *write* was successful
- Adjustment = - last loop iteration time, if the *write* was unsuccessful (i.e. nothing was passed to the lower blocks of the wireless subsystem)

The loop time is computed thanks to the current timestamp (which will be inserted inside the current UDP packet for jitter computation) and to the last transmitted packet timestamp, which is stored at each iteration.

This value is then used to adjust the running delay, which was initialized to 0 at the beginning of the UDP send loop.

According to our understanding of this algorithm, it is possible to look at an example: let's suppose that the time for executing the instructions inside each iteration is always equal to $10 \mu s \pm 1 \mu s$, with a target delay of 1.1215 ms.

At the first iteration:

$$adjust = \langle target\ delay \rangle - \langle loop\ time \rangle = 1121521\ ns - 10000\ ns = 1111521\ ns$$

$$delay = delay + adjust = 0\ ns + 1111521\ ns = 1111521\ ns$$

At the second iteration, which will take the instruction time plus the previous delay (supposing an instruction execution time of 11 μ s):

$$\begin{aligned} \text{adjust} &= \langle \text{target delay} \rangle - \langle \text{loop time} \rangle = 1121521 \text{ ns} - (1111521 + 11000) \text{ ns} = -1000 \text{ ns} \\ \text{delay} &= \text{delay} + \text{adjust} = 1111521 \text{ ns} - 1000 \text{ ns} = 1110521 \text{ ns} \end{aligned}$$

Which is in fact a little less than the first iteration value, to take into account the slower iteration (11 μ s). And so on.

In case the second iteration took 10 μ s of instruction time rather than 11 μ s:

$$\begin{aligned} \text{adjust} &= \langle \text{target delay} \rangle - \langle \text{loop time} \rangle = 1121521 \text{ ns} - (1111521 + 10000) \text{ ns} = 0 \text{ ns} \\ \text{delay} &= \text{delay} + \text{adjust} = 1111521 \text{ ns} + 0 \text{ ns} = 1111521 \text{ ns} \end{aligned}$$

As it is possible to see, the delay is in this case equal to the first iteration value and it is exactly equal to the target delay minus the time to execute the instructions in the loop.

Thanks to this mechanism the delay should be constantly adjusted to restrict the “bandwidth” (the iPerf offered traffic) to the value specified by the user.

A negative value of this running delay means, as written in the code, that the “iperf app is behind” and could not be fast enough to maintain the required delays.

In order not to let this running delay grow to values which are too negative, in the aforementioned case, when it reaches a certain “lower bound” it is reset again to the target value, at least as far as we correctly understood the algorithm.

This “lower bound” is computed considering the report interval, thanks to the following formula:

$$\text{timer}_{\text{sosnd}} = \frac{i \cdot 1000000}{2}$$

$$t_{d_{LB}} = -\text{timer}_{\text{sosnd}} \cdot 10^3$$

with:

- $t_{d_{LB}}$: delay lower bound [ns]
- $\text{timer}_{\text{sosnd}}$: socket write timeout [μ s]
- i : report interval [s]

According to the comments inside the “Client.cpp” source file, the programs sets, when writing data, a socket timeout, in order to allow responsive reporting even in case a write() call would block the program for a long time, for any reason.

The delay lower bound is set to be equal to the negative of the socket timeout, in nanoseconds.

Finally, the running delay actually translates into a wait procedure only if it is greater than 1 μ s, otherwise, being it very small, the next iteration starts immediately.

The running delay can be measured together with the UDP transmit queue length, in order to better characterize the buffered communication which is used to perform most of the measurements on the APU boards.

In order to log all these values in a synchronized way (i.e. to obtain the delay value together with the current length of the transmit queue), the source code of a separate copy of iPerf 2.0.12 has been downloaded⁵ and the source files managing the client have been patched to properly output these values.

In particular, the extra information provided by the modified client is related to:

- UDP Tx buffer length, in bytes, before transmitting each packet (i.e. just before performing a *write* on the socket)
- Running delay value, in nanoseconds, for each iteration of the loop (i.e. for each transmitted packet, if all the transmissions are successful)
- The target delay, which is printed on standard output just after the client starts
- The delay lower bound, which is printed again on standard output, once after the client starts

The data about the evolution of the buffer and of the running delay is stored inside two arrays, which are printed on *stderr* only at the end of the UDP send loop execution, to avoid

⁵<https://sourceforge.net/projects/iperf2/files/iperf-2.0.12.tar.gz/download>

continuous write to files or on *stdout/stderr* which could potentially affect performance in an evident way.

As data, at each iteration (i.e. at each packet number), the status of the buffer is stored, together with the delay caused by that status plus all the instruction execution time.

This means that the delay should be stored inside the arrays with 1 index less than then the one used for the buffer status, since the delay corresponding to a certain situation in the buffer (making the *write* slower or faster) is computed only at the next iteration.

To better clarify this concept, let's suppose to have, at the moment in which packet 100 should be transmitted, a full buffer, which causes a large delay in performing the *write* operation. We will have:

- Iteration corresponding to packet 99: delay: *short (related to packet 98 loop time)*,
buffer: *not full*
- Iteration corresponding to packet 100: delay: *short (related to packet 99 loop time)*,
buffer: *full*
- Iteration corresponding to packet 101: delay: *long (related to packet 100 loop time)*,
buffer: *full/not full*

In order to align “delay is long” with “buffer is full”, given `arridx` as the index of the arrays, it is possible to store at each iteration the delay information in `dlyarray[arridx]` and the buffer size in `bufarray[arridx+1]`.

The modified code is presented more in details in Appendix F.

After renaming the compiled iPerf binary to “iperfdbg” and transferring it to the boards, a server and a client were respectively launched on the APU_102 and APU_103 boards, using commands like the following ones:

```
./iperfdbg -s -u -i 2 -p 7000 -w 400K
```

and

```
./iperfdbg -c 10.10.6.102 -u -i 2 -t 20 -p 7000 -b 10M -w 400K 2>  
logiperfdbg.txt
```

In both cases the “-i” and “-w” values were changed depending on the test.

The “2> logiperfdbg.txt” part of the second command allowed us to redirect the values stored inside the two arrays, normally printed on *stderr*, to a log file (“logiperfdbg.txt”).

In order to be able to test large values of “-w”, it was also necessary to increase the maximum send and receive buffer size, by modifying the *net.core.rmem_max* and *net.core.wmem_max* kernel parameters:

Set_kernel_params.sh

```
1 #!/bin/sh
2 sysctl -w net.ipv4.udp_rmem_min=4096
3 sysctl -w net.ipv4.udp_wmem_min=4096
4 sysctl -w net.core.rmem_max=15728640
5 sysctl -w net.core.rmem_default=212992
6 sysctl -w net.core.wmem_max=15728640
7 sysctl -w net.core.wmem_default=212992
```

6.2.3 Plots and results: first set

The results of the first set of measurements are shown in the following plots. It may be useful to note that during the first set of measurements the maximum buffer size has been kept at its default value.

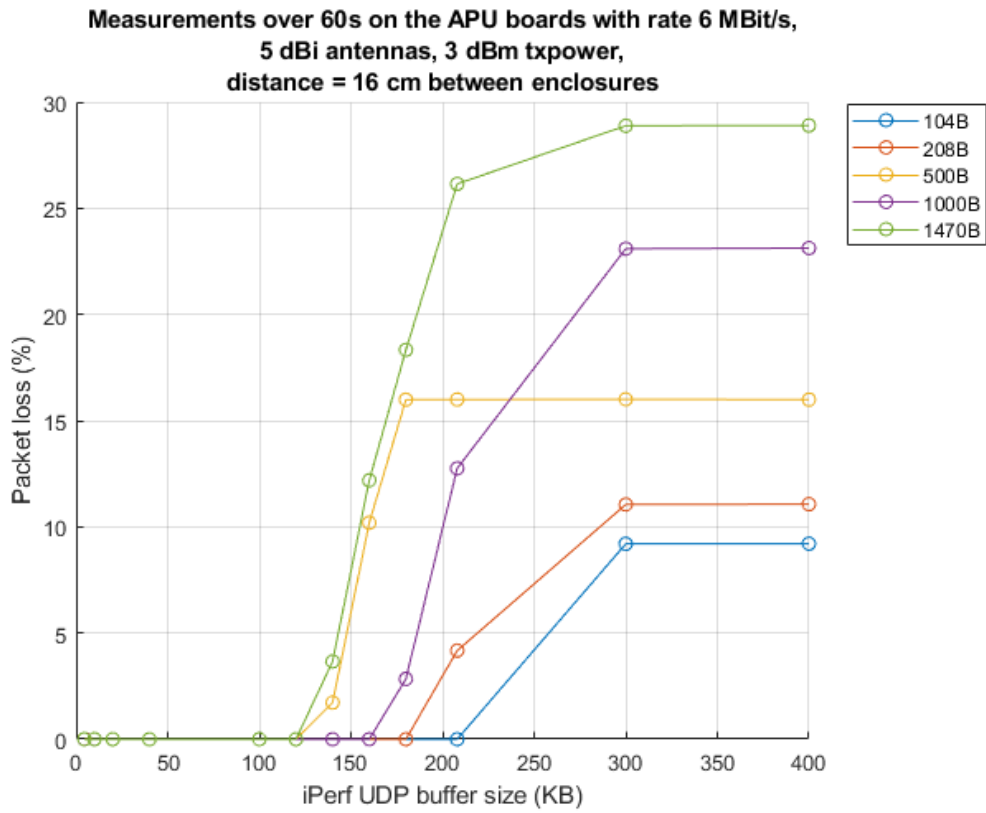


Figure 6.9: Packet loss measurements for 6 Mbit/s of physical layer bitrate, with different payload size and “UDP buffer size” values

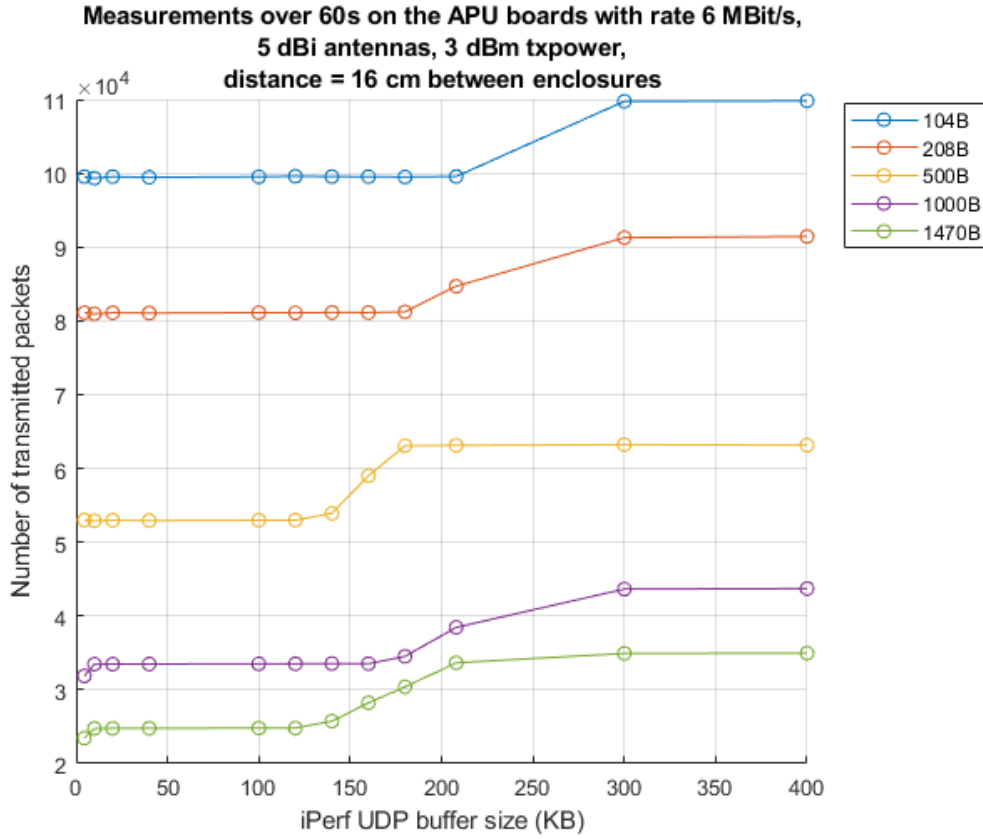


Figure 6.10: Number of transmitted packets for 6 Mbit/s of physical layer bitrate, with different payload size and “UDP buffer size” values

As it is possible to see, the losses are very small as a small buffer size is selected, limiting the overall traffic that iPerf is able to pass to the networking stack; they then increase, depending on the packet size, as the buffer is increased, since more data is now written by iPerf to the socket; this data, however, cannot be fully transmitted, as the selected offered traffic (–b 10M) is much more than what the network can deliver.

As the payload size is increased, the maximum buffer size guaranteeing 0% losses tends to decrease towards lower values, except for the 500 bytes case, showing a behavior which is not always linear, probably due to the frames fitting in a better or worse way inside the buffer and due to the combined effect of the UDP buffer at an application level

and of the behavior of the wireless subsystem when transmitting data.

In particular, we got the following data:

Payload size	Max buffer size with 0% loss*
104 B	208 KB
208 B	180 KB
500 B	120 KB
1000 B	160 KB
1470 B	120 KB

* Among the chosen values

For what concerns the number of transmitted frames, it is proportional to the payload size, as expected, meaning that a higher payload size will result in less independent frames being generated and transmitted.

It also tends to increase as the buffer size is increased, even though the number tends to remain quite stable for very low “-w” values and to stabilize, with a big packet loss value, as the configuration approaches much bigger values, possibly due to a sort of equilibrium between the dropped packets and what iPerf is trying to send, as detailed later on.

6.2.4 Plots and results: second set

For the second set of measurements, partly explaining the behavior observed in the first set, it can be useful to comment one plot at a time.

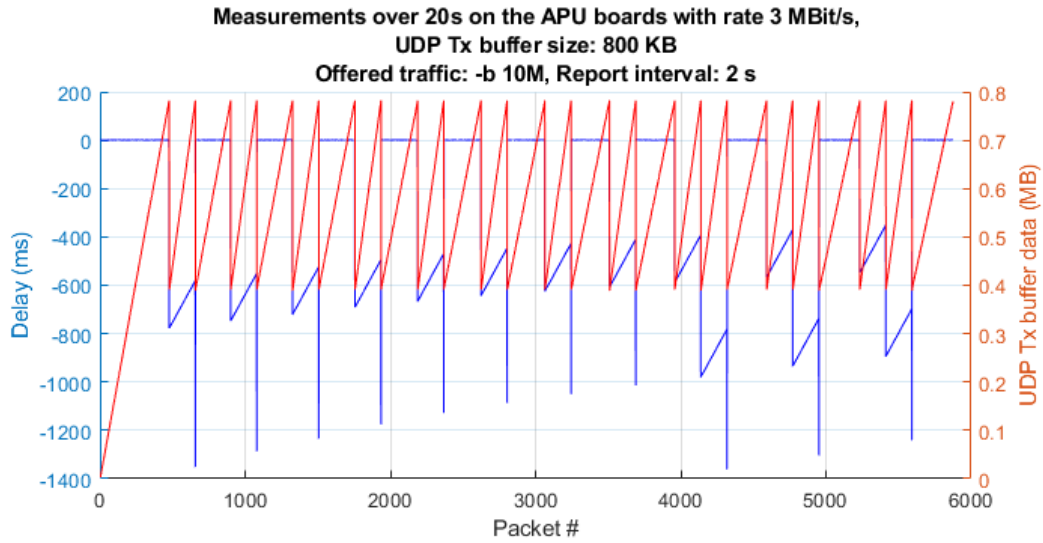


Figure 6.11: First buffer evolution plot

When specifying a high value of “-b”, with respect to the maximum reachable value, the buffer, depending on the moment, is actually completely filled in.

It starts by being gradually filled (with a lower packet loss in the moment in which the transmission starts), until it is completely full. Then, it starts following a “zig-zag” behavior, which limits the amount of traffic iPerf is able to push towards the lower levels.

Taking into account that in this case the target delay is equal to 1.121521 ms, the lower bound is -1000 ms and looking closer at the first section of the buffer evolution plot it is possible to better described the overall transmission behavior.

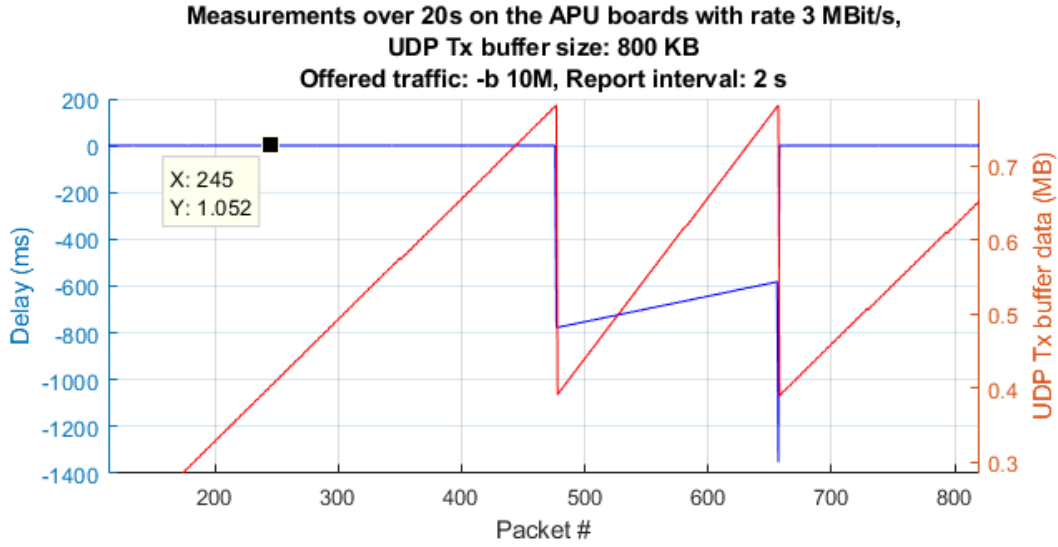


Figure 6.12: Zoom over a section of the first buffer evolution plot

It is possible to notice that the delay is initially around 1 ms, as the buffer gets filled in.

When it is full, the *write* operation takes much more time, being it blocking until the send timeout expires (which is not our case, here), dropping the delay down to a negative value.

This will let the WNIC transmit a certain amount of packets, allowing the buffer to gain some free space; the size of this “step” is very probably dependent on both the wireless subsystem and the WNIC+driver behavior.

As suggested by Robert Hancock, a user in the OpenWrt forums, the fact that the “*write()*” call blocks for more time with respect to the one that would be needed to free up the space for a single packet is reasonable, since packets may get aggregated (for instance for 802.11n and 802.11ac MPDU aggregation) and it would also be quite inefficient, from the application point of view, to “*wake up so frequently just to refill the socket buffer*” [74].

This will also make the program not to wait anymore, and try to push as much data as possible to try moving the running delay towards the target value.

As the data is generated, the buffer is filled again, until it becomes full again. This time the delayed *write* seems to have a much worse effect, causing the running delay to drop over the delay lower bound (as it was already negative), which resets it to the target value, making the process repeat in time.

Of course not all the packets can be transmitted by the WNIC, thus causing a packet loss which is not plotted here.

However, as the UDP buffer size approaches higher values, this “zig-zag” behavior seems to be quite similar between different values of “-w”, possibly explaining why the packet loss seems to stabilize around 26% or 43-45%.

Looking at a situation with a lower value of UDP buffer size (it may be important to remember that, during these tests, the server and client buffers are always aligned, as far as size is concerned):

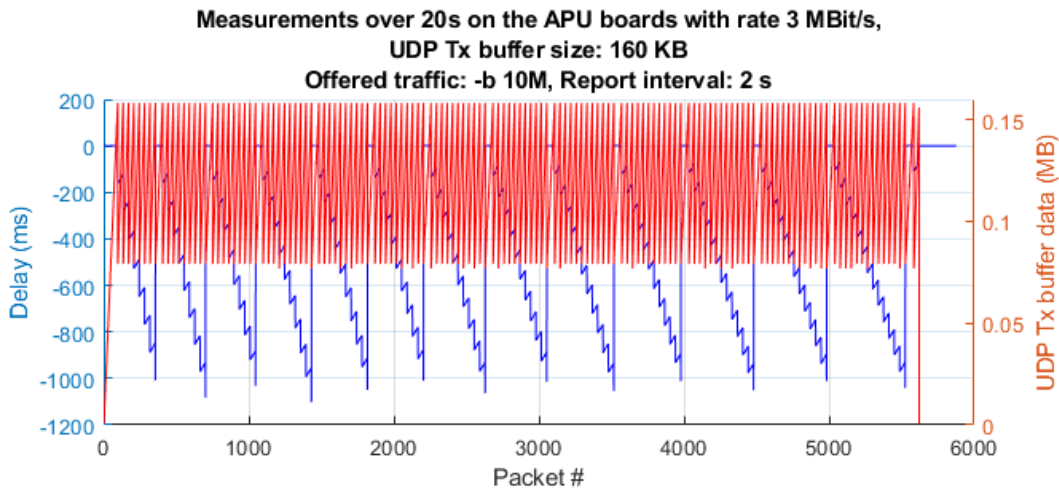


Figure 6.13: Second buffer evolution plot

It is possible to see that, as the buffer size is reduced, the “zig-zag” behavior becomes much faster, with smaller delays to due blocking writes, which occur every time the buffer is full. Thus, this seems both to limit the number of packets iPerf is able to transmit to its lower layers and to cause the running delay to reset less often.

This also seems to explain the behavior shown in figure 6.10.

This evolution is even more evident when further reducing “-w”, for instance using a 40KB buffer, which, according to the previous measurements, is able to guarantee a packet number limitation such that a 0% loss is measured:

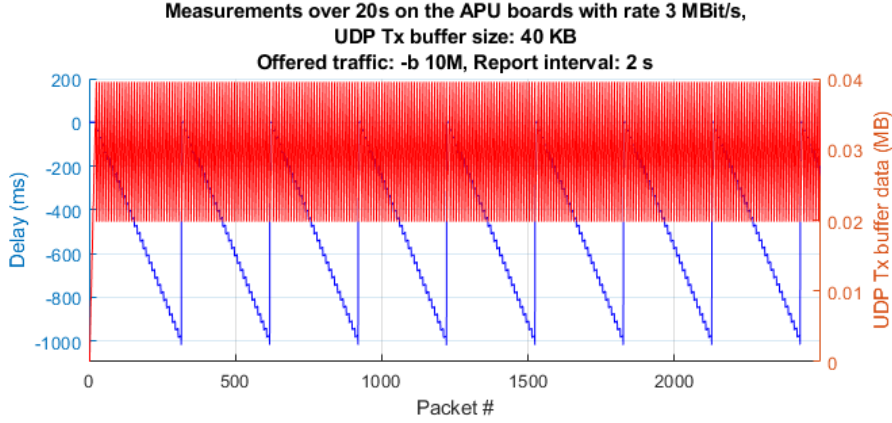


Figure 6.14: Third buffer evolution plot

It can be interesting now to show two extreme situations: one in which the offered traffic can all be managed by the WNIC with a 3 Mbit/s physical rate (choosing “-b 1M”) and one, with “-b 10M”, in which the buffer is so big that it is never filled in, letting iPerf transmit all the data it can and causing a very high packet loss.

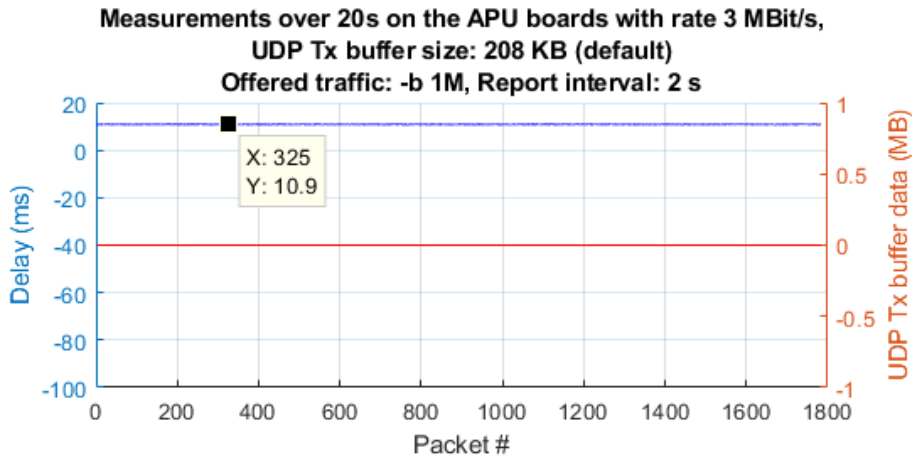


Figure 6.15: Fourth buffer evolution plot: low offered traffic; target delay: 11.215210 ms, lower bound: -1000 ms

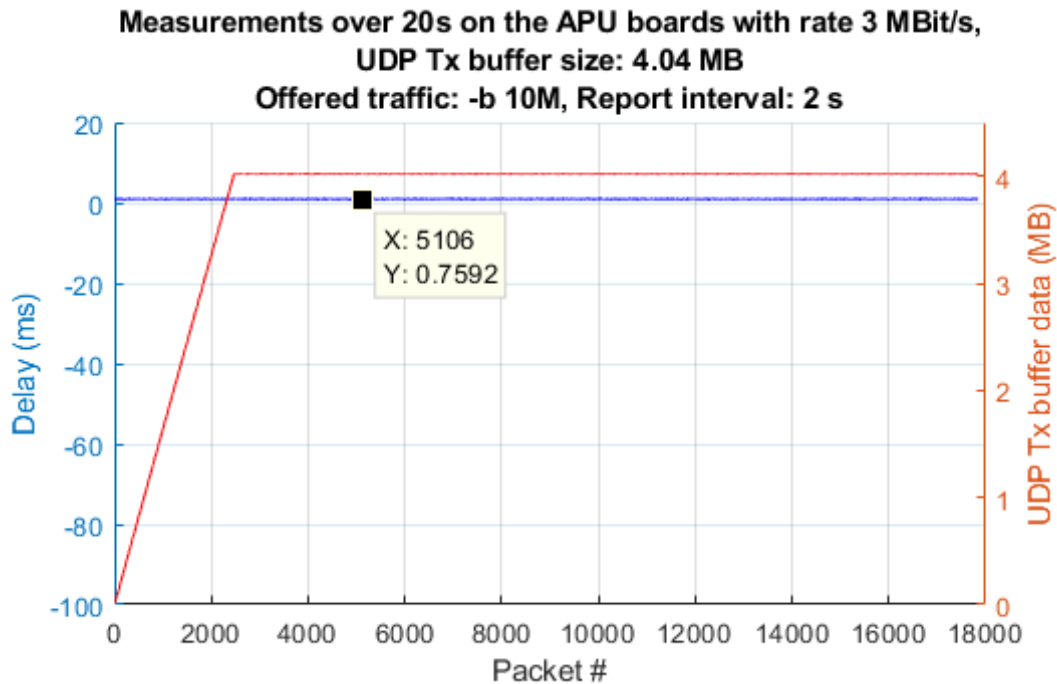


Figure 6.16: Fifth buffer evolution plot: very large UDP buffer size; target delay: 1.121521 ms, lower bound: -1000 ms

As it is possible to see, in the first case the buffer is always found empty by each *write* operation, thus the delay is always around the target value, as shown by the datatip. This is due to the WNIC always being able to delve with the offered volume of data.

In the second case, instead, the buffer is filled up to 4.02 MB, which seems to be the maximum that the application (iPerf) is able to provide with its UDP send loop, but it is never completely filled in. Exactly the same situation can be observed with any buffer size greater than 4.04 MB.

This causes the running delay to always oscillate around the target value, with no *write* operations causing larger delays. However, as there is never any large delay in the iPerf write operations, this causes a very large packet loss in the WNIC side, just as soon as this size is reached while increasing “-w”, as detailed later on.

As far as packet loss is concerned, it was possible to detect an abrupt step when passing from the “zig-zag” situation to the one shown in figure 6.16, in particular:

Buffer size	Packet loss	Number of transmitted packets
800 KB	43.18%	23571
1.56 MB	43.06%	24007
2.02 MB	45.37%	25662
2.04 MB	73.68%	53498
6 MB	73.68%	53499
15 MB	73.84%	53499

This data is obtained over 60 seconds, with 1470 B as payload and “-b 10M”.

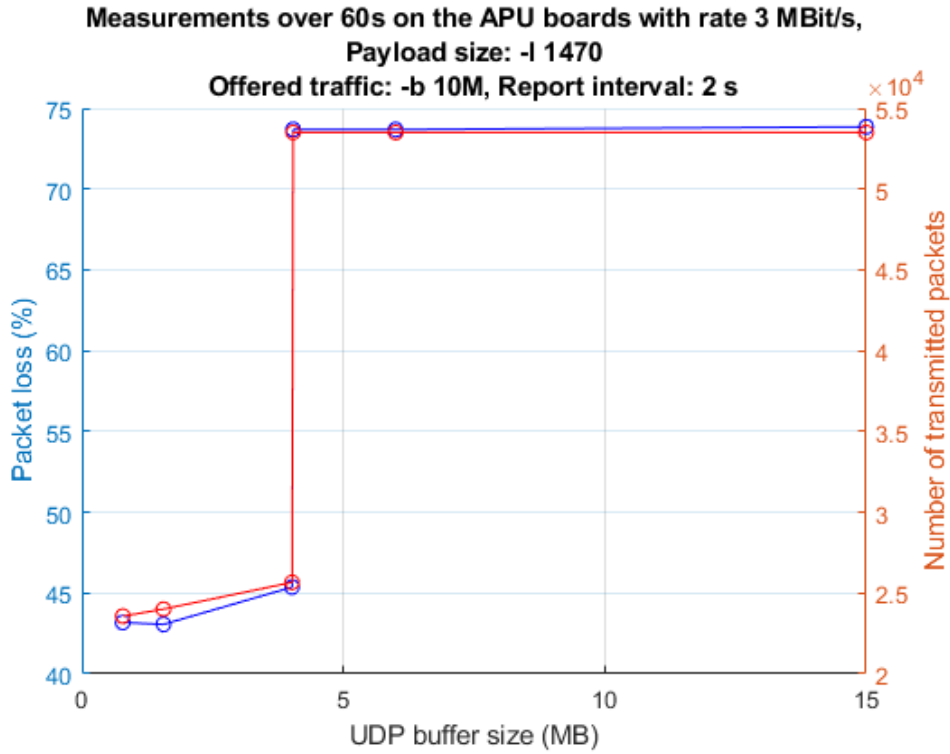


Figure 6.17: Packet loss and number of transmitted packets with large values of UDP buffer size

As a final observation, it can be useful to notice that the “zig-zag” behavior highlighted before depends also on the report interval.

Thus, the packet loss, due to the very peculiar algorithm used in the measurement application, can be influenced by the report interval too, although huge variations should not be expected.

This is probably due to the fact that the both the socket timeout and the delay lower bound are depending on the report interval time.

For instance, looking at the plot for an 800 KB socket buffer size, with a report interval of 0.1 seconds, and comparing it to the plot shown in figure 6.11, which was obtained for “-i 2”, it is possible to notice a different behavior both in the evolution of the buffer and in the one of the running delay.

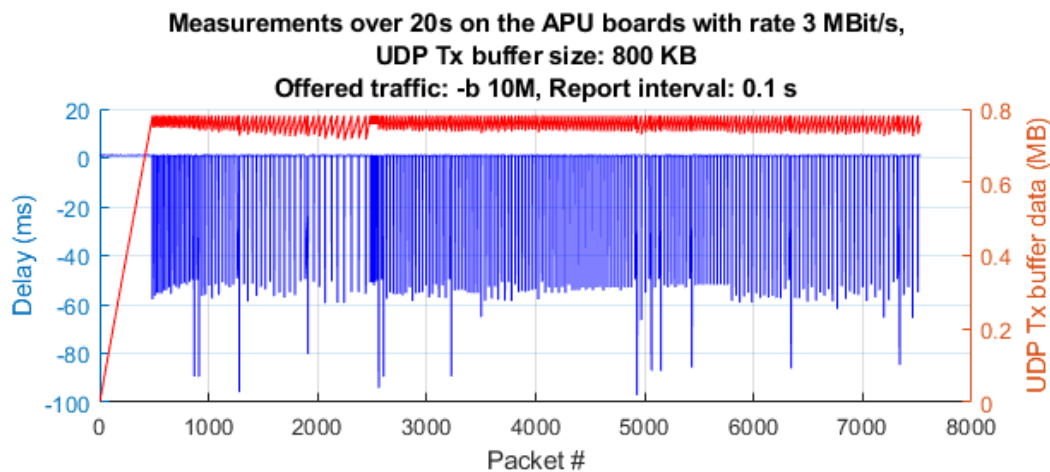


Figure 6.18: Sixth buffer evolution plot, with report interval equal to 0.1 s; target delay: 1.121521 ms, lower bound: -50 ms

Here, due to the smaller lower bound value, the delay is more often reset to its default value, always dropping around -50 ms when blocking writes occur.

This is probably suggesting us that, even if no timeout error occurs, the “write()” calls are actually being unblocked by the timeout expiration.

Our hypothesis is that no error is returned due to the buffer having freed up enough space to accommodate a single packet with 1470 B of payload.

6.3 More systematic throughput and packet loss measurements

After briefly looking at how the buffered transmission and reception behaves, it was possible to perform more systematic throughput and packet loss measurements, by choosing different values of physical data rates: 3 Mbit/s, 6 Mbit/s, 12 Mbit/s.

For each set of measurements, we measured packet loss and throughput, progressively increasing the amount of offered traffic (with “-b”).

For each physical rate, we also drew a different curve for 9 different payload length values, from a less optimized small payload of 16 B to a more optimized payload of 1470 B (which is also the default value of the measurement application): 16 B, 48 B, 104 B, 208 B, 500 B, 700 B, 1000 B, 1200 B, 1470 B.

6.3.1 Network configuration and conditions

The network configuration is very similar to the one presented in section 6.1:

- Variable physical bitrate
- 5 dBi antenna (MIMO) connected to the boards
- 3 dBm txpower set in OpenWrt (so, this should be the transmitted power without the additional antenna gain of 5 dBi)
- Boards placed very near to each other, with 16 *cm* between the two enclosures
- Duration of each single test: 60 seconds
- iPerf port: 7000
- iPerf report interval: 2 seconds
- Layer 4 protocol: UDP

- Payload size: variable, over 9 different values
- UDP buffer size: 208 KB (default iPerf value)
- Channel: 178

6.3.2 Scripts and commands

In order to perform all the measurements, we used two scripts, based on the NTP synchronization and “waituntil” function presented before.

We wrote one script for launching, in sequence, the different iPerf servers on the APU_102 board and one script for the different iPerf clients on the APU_103 board.

In particular, for each payload length all the offered traffic values are tested, moving then to the next length value, and so on.

Three different sets of very similar scripts have been used for each physical bitrate.

Since they involve several lines of code, they are reported in Appendix G.

These scripts produced one log for each 60 seconds test (both client-side and server-side); all the server logs were then parsed on the Linux Mint virtual machine by means of three purposely written data parsing *bash* scripts, extracting the meaningful information from a large number of logs and producing *.csv* files in output, which could be easily imported in Matlab.

These scripts are also reported in Appendix G.

6.3.3 Plots and results

The results are shown in the following plots, in which the physical rate is represented by a dotted line:

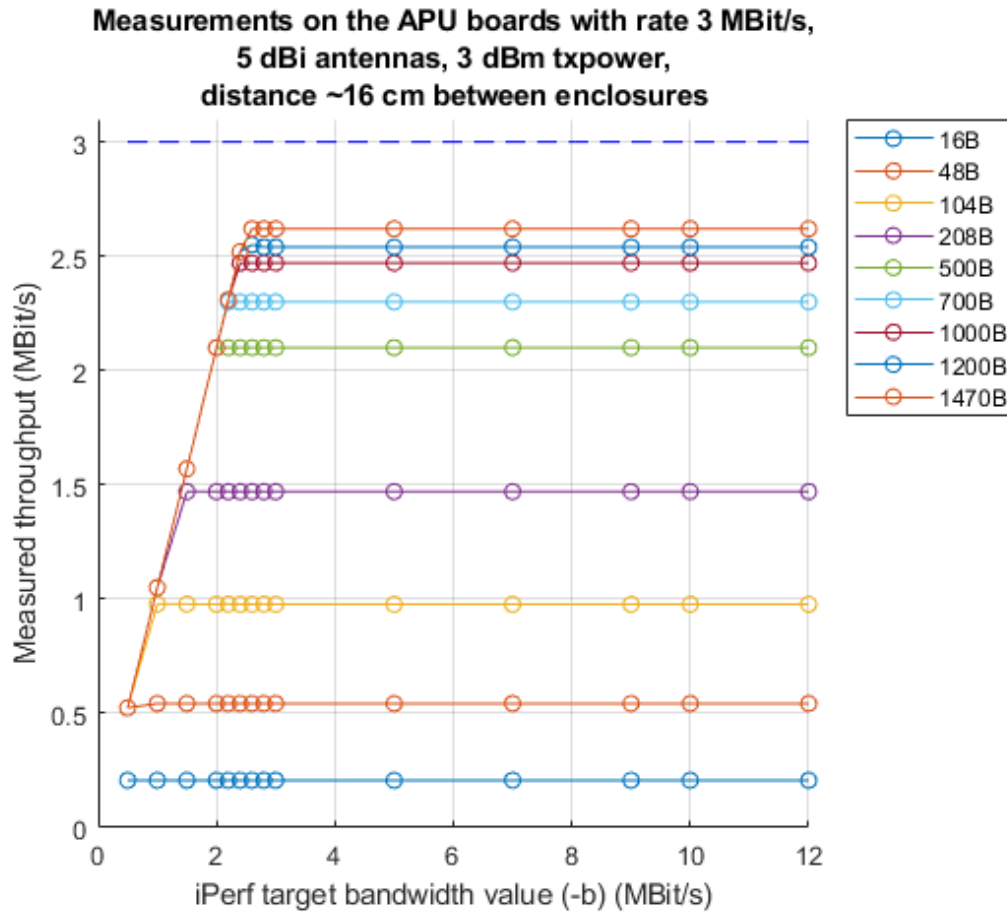


Figure 6.19: Throughput measurements for 3 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

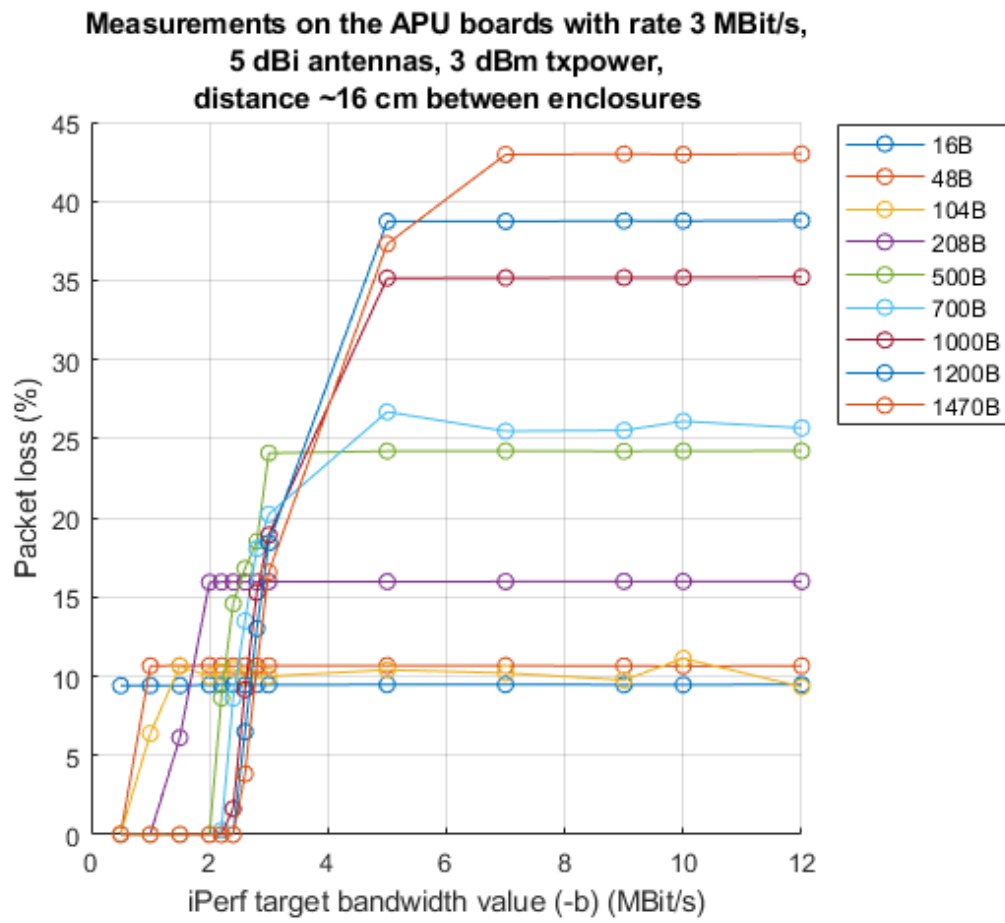


Figure 6.20: Packet loss measurements for 3 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

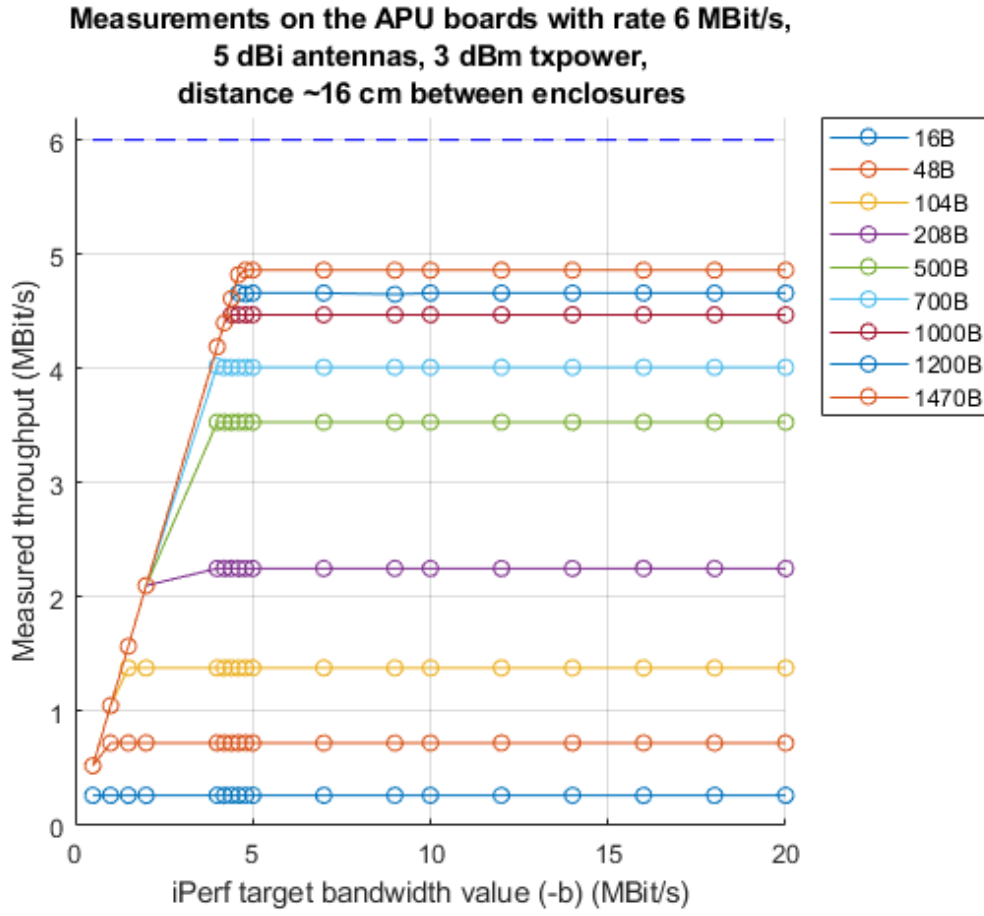


Figure 6.21: Throughput measurements for 6 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

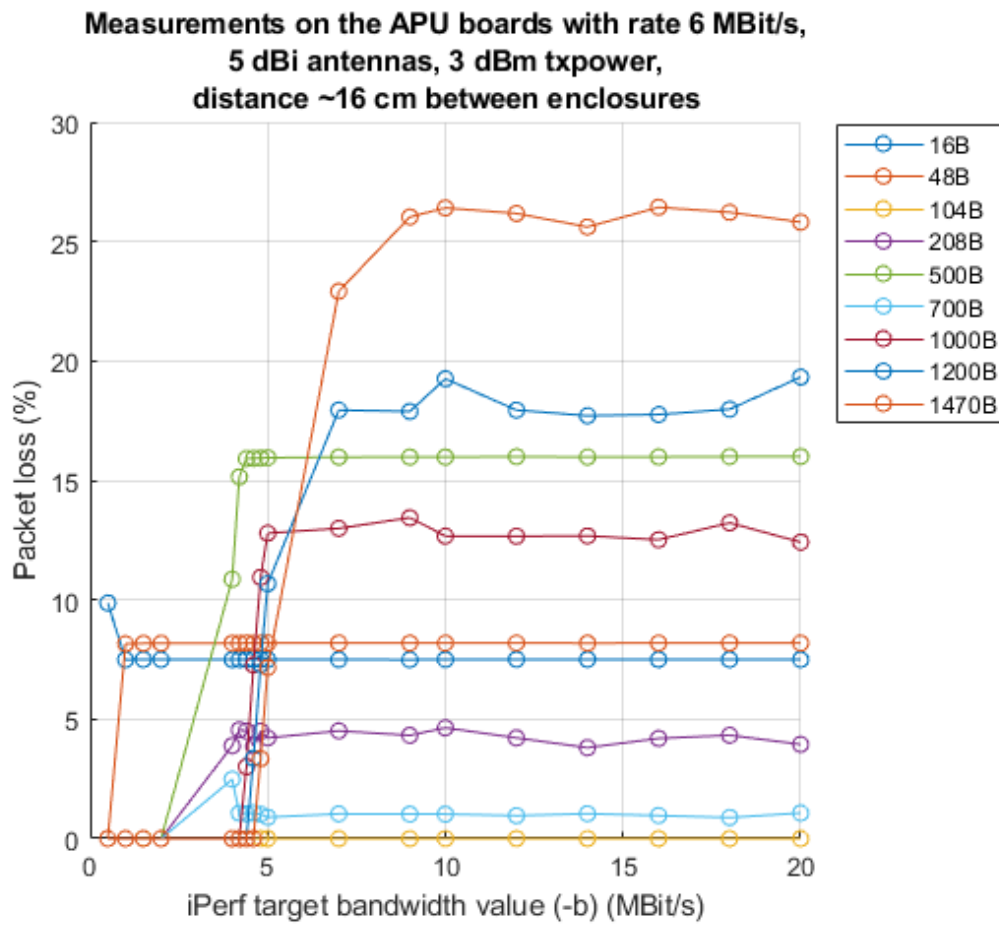


Figure 6.22: Packet loss measurements for 6 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

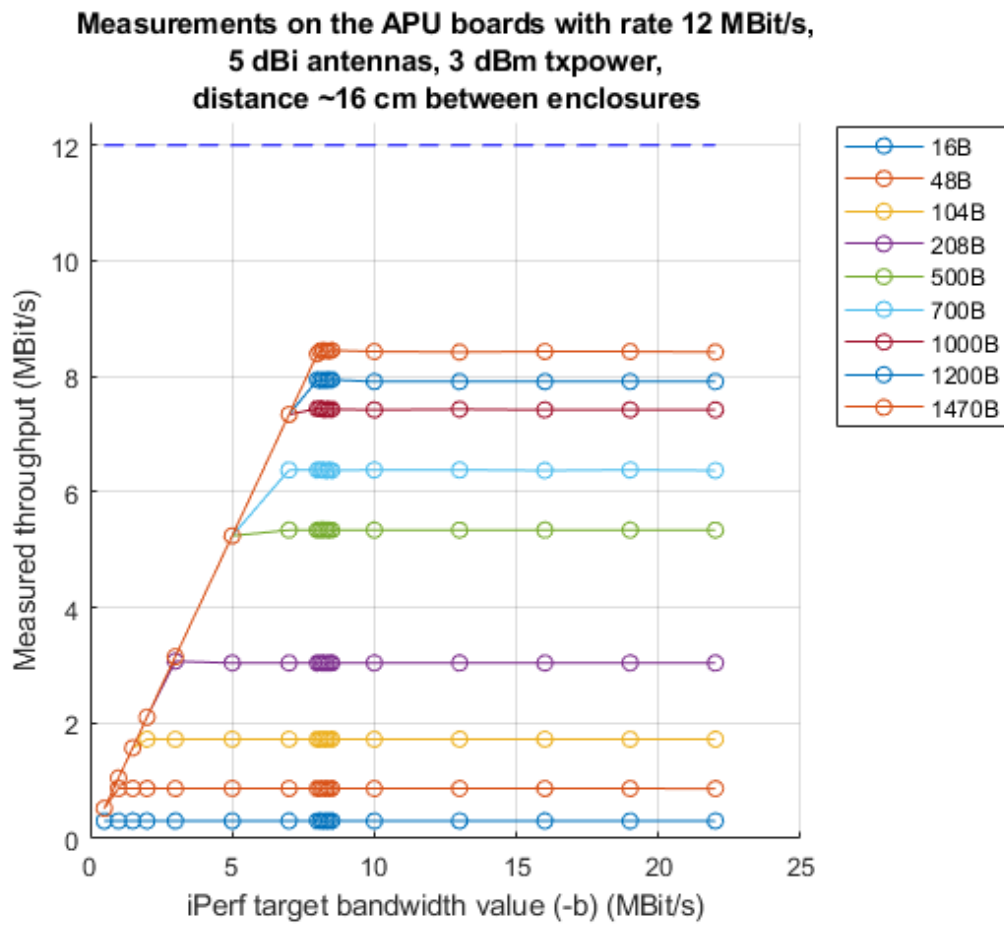


Figure 6.23: Throughput measurements for 12 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

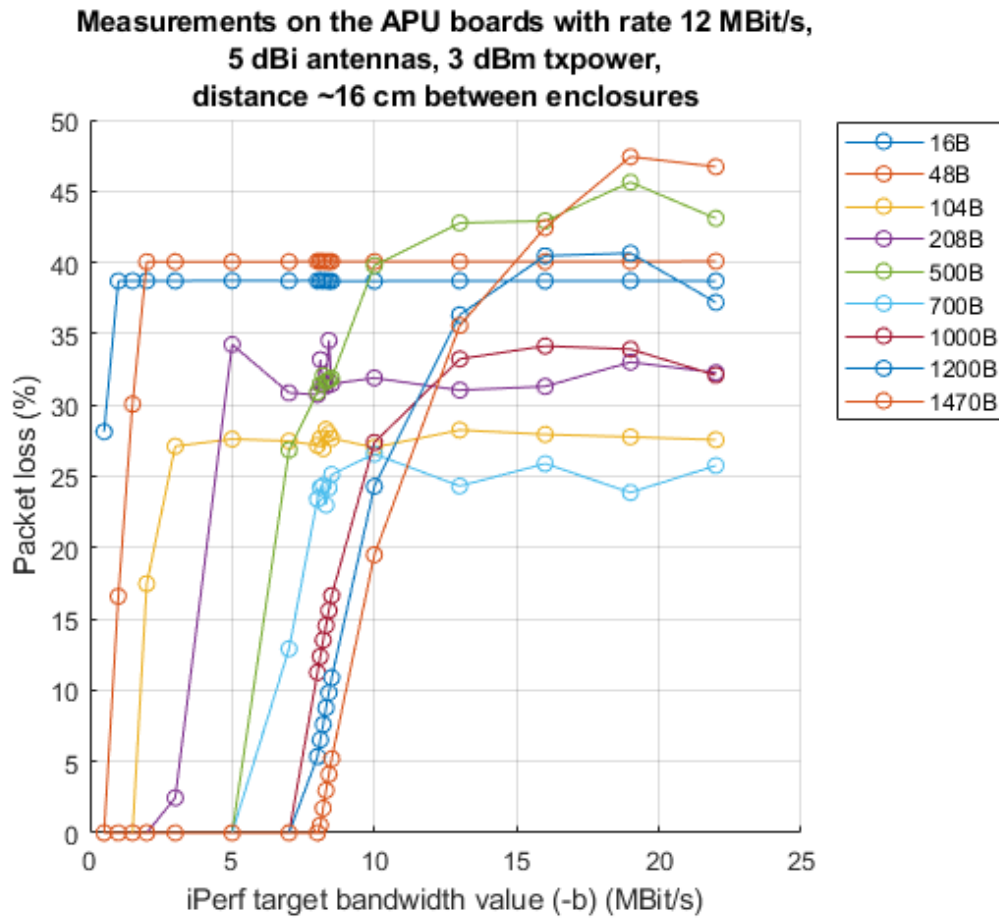


Figure 6.24: Packet loss measurements for 12 Mbit/s of physical layer bitrate, with different payload size and offered traffic values

As it is possible to see, the reachable throughput depends on the payload size which is used by the application: with the tested values, the more the payload size, the more a higher throughput can be reached.

Using a different protocol stack, for instance when a WSMP stack will be implemented, may cause some variations to these results; however, the trend should remain the same.

In particular, we were able to measure the following maximum throughput values:

Physical rate (Mbit/s)	Payload size (-l) (B)	Maximum throughput (Mbit/s)
3	16	0.207
3	48	0.543
3	104	0.977
3	208	1.470
3	500	2.100
3	700	2.300
3	1000	2.470
3	1200	2.540
3	1470	2.620
6	16	0.256
6	48	0.723
6	104	1.380
6	208	2.250
6	500	3.530
6	700	4.010
6	1000	4.470
6	1200	4.660
6	1470	4.860
12	16	0.306
12	48	0.866

12	104	1.720
12	208	3.040
12	500	5.340
12	700	6.370
12	1000	7.420
12	1200	7.910
12	1470	8.420

The packet loss trend is instead less linear, probably due to the combined effect of the UDP buffer and the WNIC transmitting packets over the wireless medium.

In any case, as expected, increasing the value of offered traffic always causes an increase in the packet loss values when the offered traffic is more than what the network can manage, except for few oscillations, which are always under 3.5%.

The oscillations can be observed when concentrating the analysis around a certain offered traffic value but, as stated before, they are never very big.

The exact amount of packet loss, when offering too much traffic, is also dependent on how the application manages the packet transmission (in this case, the reported values are always referring to the iPerf network measurement program).

As a side note, using a payload which is too small (such as 16 B) is never recommended with UDP, as it can deliver much lower performance.

This may be different when using other non-IP based protocol stacks (i.e. WSMP).

6.4 Traffic classes

This section presents a set of measurements related to the communication over the 4 different traffic classes which EDCA sets available to the user (i.e. AC_BK, AC_BE, AC_VI and AC_VO).

In particular, we measured the reachable throughput, the number of packets correctly received by each board and a parameter which is an indication of the connection stability when the two boards are contending the channel, both trying to send data using the same or even different traffic classes. In order to do so, we launched a couple of client and server on each board, with the client of one board connecting to server of the other board and viceversa.

On the other hand, we also measured the reachable throughput with one client and one server only, showing that there is actually an improvement over the maximum value when using higher priority AC, corresponding to faster AIFS times and smaller contention window sizes.

6.4.1 Network configuration and conditions

The network configuration is again similar to the previous cases. In particular, for what concerns the first set of measurements the configuration was the following one:

- Variable physical bitrate
- 5 dBi antenna (MIMO) connected to the boards
- 3 dBm txpower set in OpenWrt (so, this should be the transmitted power without the additional antenna gain of 5 dBi)
- Boards placed very near to each other, with 16 *cm* between the two enclosures
- Duration of each single test: 60 seconds
- iPerf port: 7000
- iPerf report interval: 2 seconds
- Layer 4 protocol: UDP
- Payload size: 1470 B (default iPerf value)

- UDP buffer size: 208 KB (default iPerf value when used with the APU boards)
- Channel: 178
- Theoretical offered traffic (iPerf “bandwidth”): 3 Mbit/s for 3 Mbit/s of physical rate, 10 Mbit/s for 6 Mbit/s and 12 Mbit/s for 12 Mbit/s (which is always more than what the network can really deliver, to try to get maximum throughput values)

Instead, for what concerns the second set, the conditions are the same except the fact of having now a single client+server running on the two boards.

6.4.2 Scripts and commands

The scripts we used for the first set of measurements are actually based on the “server6000client7000.sh” and “server7000client6000.sh” scripts presented in section 3.8.4, with the small different of removing the third command line parameter for “-b”, which is now set directly inside the script.

We wrote, then, an additional script, calling the client+server scripts mentioned before with different values of AC and physical rates, which should be set by the user before executing the script; this allowed us to automate the execution of the tests.

The script is unique for both the ALIX_102 and ALIX_103 boards, and the correct device is chosen thanks to the first command line parameter (“2” for APU_102 and “3” for APU_103).

The second parameter is instead the current physical data rate; this is used, however, only to properly name the logs that are generated.

The script is reported below:

PCEngines_measurements_AC.sh

```
1 #!/bin/bash
2
3 function waituntil {
4     while true; do
```

```

5  currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6  if [ $currsec -eq $1 ]; then
7      break
8  fi
9  done
10 }
11
12 if [ $# -ne 2 ]
13 then
14     echo "Error running tests. Expected three parameters."
15     echo "  1. Board number (2,3)"
16     echo "  2. Physical data rate (only for logging purposes)"
17     echo "Warning! No consistency check for the parameters: writing"
18     echo "  wrong parameters may result in an undefined behaviour."
19     exit 1
20 fi
21
22 if [ $1 -eq 2 ]
23 then
24     name=APU102 # server6000
25     connto=10.10.6.103 # IP address=10.10.6.102
26     AArray=( BK BK BK BK BE BE BE BE VI VI VI VI VO VO VO VO )
27 elif [ $1 -eq 3 ]
28 then
29     name=APU103 # client6000
30     connto=10.10.6.102 # IP address=10.10.6.103
31     AArray=( BK BE VI VO BK BE VI VO BK BE VI VO BK BE VI VO )
32 else
33     echo "Invalid board number. Valid numbers are 2,3 (APU)"
34     exit 1
35 fi
36
37 # Check if the Logs directory exists. If not, create it.
38 if [ ! -d "./Logs" ]; then

```

```
39  echo "Logs directory missing. It will be created now"
40  mkdir Logs
41  else
42      echo "Logs directory exists. Make a backup of its content before
        proceeding."
43  fi
44  read -p "Press enter to continue"
45
46  # Read physical rate from the user specified arguments
47  physrate=$2
48
49  # Read current seconds in time
50  currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
51
52  echo "Seconds in current time: $currsec"
53  echo "Current board: $name"
54  echo "Ready to start the tests"
55  read -p "Press enter to continue"
56
57  currwait=0
58  i=0
59
60  # parameters
61  len=1470
62
63  while [ $i -lt ${#ACarray[@]} ]; do
64      currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
65
66      echo "Synchronizing (actual sec: $currsec, target sec: $currwait)...
        "
67      # Synchronize (NTP time is required, otherwise this is meaningless)
68      waituntil $currwait
69      echo "-----"
70      echo "Starting test with board: $name..."
```

```

71  echo "Running test with -l $len -A ${ACarray[$i]} (phys. rate:
    $physrate Mbit/s)"
72  if [ "$name" = "APU102" ]; then
73      ./server6000client7000.sh $connto ${ACarray[$i]} | tee -a "Logs/
        APU102Log-$i-rate-$physrate-A-${ACarray[$i]}.txt"
74  else
75      ./server7000client6000.sh $connto ${ACarray[$i]} | tee -a "Logs/
        APU103Log-$i-rate-$physrate-A-${ACarray[$i]}.txt"
76  fi
77  echo "Test terminated."
78  echo "-----"
79
80  # Increment index and set currwait
81  i=$((i+1))
82  currwait=$(( (currwait+15)%60 ))
83 done

```

The different values of “-b” has been set, instead, by directly editing the “server6000-client7000.sh” and “server7000client6000.sh” scripts, depending on the current test, and the synchronization between the two boards happens, as usual, thanks to NTP and the “waituntil” bash function.

For the second set of measurements we used instead two scripts (one for the client, run on the APU_103 board and one for the server, run on the APU_102 board), which are more similar to the ones presented in previous sections, performing a single client+server test at each AC and for the three selectable values of physical bitrate, which, this time, are automatically set.

These scripts are reported below:

iperfserver7000ACb10M.sh

```

1  #!/bin/bash
2
3  function waituntil {

```



```

4  while true; do
5      currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6      if [ $currsec -eq $1 ]; then
7          break
8      fi
9      done
10 }
11
12 function iperfserverirA {
13     iperf -s -u -i 2 -p 7000 -t 64 2>&1 | tee -a "Logs/APU102Log-$1-rate
        -$2-A-$3.txt"
14 }
15
16 # Check if the Logs directory exists. If not, create it.
17 if [ ! -d "./Logs" ]; then
18     echo "Logs directory missing. It will be created now"
19     mkdir Logs
20 else
21     echo "Logs directory exists. Make a backup of its content before
        proceeding."
22 fi
23 read -p "Press enter to continue"
24
25 AArray=( BK BE VI V0 )
26 physrates=( 3 6 12 )
27
28 currwait=0
29 i=0
30 p=0
31 while [ $p -lt ${#physrates[@]} ]; do
32     wifiarate=$(( ${physrates[$p]} * 2 )
33     iw dev wlan0 set bitrates legacy-5 $wifiarate
34     echo "Rate set to ${physrates[$p]} (iw dev wlan0 set bitrates legacy
        -5 $wifiarate)"

```

```

35  sleep 1
36  while [ $i -lt ${#ACarray[@]} ]; do
37      echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":":
          " -f3 | cut -d" " -f1), waiting until second $currwait"
38      waituntil $currwait
39      echo "-----"
40      echo "Running test with -A ${ACarray[$i]} (iw dev wlan0 set
          bitrates ... ${physrates[$p]})"
41      iperfserverirA $i ${physrates[$p]} ${ACarray[$i]}
42      echo "Test with -A ${ACarray[$i]} terminated"
43      echo "-----"
44      i=$((i+1))
45      currwait=$(( (currwait+12)%60 ))
46  done
47  sleep 1
48  p=$((p+1))
49  i=0
50 done

```

iperfclient7000ACb10M.sh

```

1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":": -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfclientirA {

```

```

13   iperf -c 10.10.6.102 -u -i 2 -t 60 -A $3 -p 7000 -b 10M | tee -a "
      Logs/APU103ClientLog-$1-rate-$2-A-$3.txt"
14 }
15
16 # Check if the Logs directory exists. If not, create it.
17 if [ ! -d "./Logs" ]; then
18     echo "Logs directory missing. It will be created now"
19     mkdir Logs
20 else
21     echo "Logs directory exists. Make a backup of its content before
      proceeding."
22 fi
23 read -p "Press enter to continue"
24
25 AArray=( BK BE VI VO )
26 physrates=( 3 6 12 )
27
28 currwait=0
29 i=0
30 p=0
31 while [ $p -lt ${#physrates[@]} ]; do
32     wifirate=$(( ${physrates[$p]} * 2 ))
33     iw dev wlan0 set bitrates legacy-5 $wifirate
34     echo "Rate set to ${physrates[$p]} (iw dev wlan0 set bitrates legacy
      -5 $wifirate)"
35     sleep 1
36     while [ $i -lt ${#AArray[@]} ]; do
37         echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":
      " -f3 | cut -d" " -f1), waiting until second $currwait"
38         waituntil $currwait
39         echo "-----"
40         echo "Running test with -A ${AArray[$i]} (iw dev wlan0 set
      bitrates ... ${physrates[$p]})"
41         sleep 4

```

```

42     iperfclientirA $i ${physrates[$p]} ${ACarray[$i]}
43     echo "Test with -A ${ACarray[$i]} terminated"
44     echo "-----"
45     i=$((i+1))
46     currwait=$(( (currwait+12)%60 ))
47 done
48     sleep 1
49     p=$((p+1))
50     i=0
51 done

```

In order to extract data from the various logs, after the measurement sessions, and use them to both obtain the relevant information and all the useful plots, we wrote a log data extraction script, similar to the ones presented in Appendix G, which is able to parse all the useful data from a certain set of logs (each set of logs corresponds to a certain board and to a certain physical data rate, thus having 6 sets in total) and to output a corresponding *.csv* file, which can be then easily imported inside Matlab.

The script is also able to extract an additional useful information: for each measurement the maximum percentage variation of the measured throughput is extracted, as:

$$\%_{max} = 100 \cdot \frac{throughput_{max} - throughput_{min}}{throughput_{max}}$$

where the *throughput* values are the ones returned by iPerf at each report interval (i.e., in a normal case, every 2 s). This can be useful as a measure of the connection stability when using a certain AC against another AC used by another device.

As reference, it is reported below (line 100 is the one applying the formula mentioned before):

logExtractor.sh

```

1  #!/bin/bash
2
3  if [ $# -ne 2 ]; then
4      echo "Error using program. Expected one or two parameters: "

```

```

5  echo "1)  Rate (<3|6|12>)"
6  echo "2)  Board number (2 for APU_102, 3 for APU_103)"
7  exit 1
8  fi
9
10 if [ $2 -eq 2 ]
11 then
12     name=APU102 # server
13     ACarray=( BK BK BK BK BE BE BE BE VI VI VI VI VO VO VO VO )
14 elif [ $2 -eq 3 ]
15 then
16     name=APU103 # client
17     ACarray=( BK BE VI VO BK BE VI VO BK BE VI VO BK BE VI VO )
18 else
19     echo "Invalid board number. Valid numbers are 2,3 (APU)"
20     exit 1
21 fi
22
23 shopt -s lastpipe # In order to run the while read cycle in the
                    # current shell, otherwise the
24                    # values of "tputmin" and "tputmax" would not keep their
                    # values in the current shell;
25                    # with this, the last pipeline segment is always executed in
                    # the current shell and not
26                    # in a subshell
27
28 bwidth=10
29 length=1470
30 physrate=$1
31
32 i=0
33 echo "len,b,tput,loss,lostpkt,sentpkt,okpkt,maxvartputperc" > "$name-
    Rate$1_iperf_AC_test.csv"
34 while [ $i -lt ${#ACarray[@]} ]; do

```

```

35 filename=$(echo "$name"Log-$i-rate-$physrate-A-${ACarray[$i]}.txt)
36
37 echo $filename
38
39 lastline=$(cat $filename | head -n -2 | tail -1)
40 if [[ "$lastline" =~ "out-of-order" ]]; then
41     hcommand=-3
42 else
43     hcommand=-2
44 fi
45 #echo "lastline: $lastline"
46 echo "hcommand: $hcommand"
47
48 tput=$(cat $filename | head -n $hcommand | tail -1 | sed 's/\s\s*/ /
    g' | cut -d" " -f7)
49 umeas=$(cat $filename | head -n $hcommand | tail -1 | sed 's/\s\s*/
    /g' | cut -d" " -f8)
50 # Convert measurements in KBit/s to MBit/s
51 if [[ $umeas = *"K"* ]]; then
52     tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
53 fi
54 lostpkt=$(echo $(cat $filename | head -n $hcommand | tail -1 | cut
    -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" " -f1 | rev)
55 sentpkt=$(cat $filename | head -n $hcommand | tail -1 | cut -d"/" -
    f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
56 echo "$(cat $filename | head -n $hcommand | tail -1)"
57
58 okpkt=$((sentpkt-lostpkt))
59 loss=$(echo "$lostpkt $sentpkt" | awk '{printf("%.15f\n",$1/$2*100)
    }')
60 i=$((i+1))
61
62 lindex=0
63 tputmax=-1

```

```

64 tputmin=1000000
65 # Extract maximum % variation of throughput values
66 cat $filename | head -n $((hcommand-1)) | tail -n +9 | sed 's/\s\s*/
    /g' | while read line; do
67     if [[ "$line" =~ "out-of-order" ]]; then
68         echo "Skipping line..."
69     else
70         if [ $lindex -lt 4 ]; then
71             tputcurr=$(echo $line | cut -d" " -f8)
72             umeas=$(echo $line | cut -d" " -f9)
73         else
74             tputcurr=$(echo $line | cut -d" " -f7)
75             umeas=$(echo $line | cut -d" " -f8)
76         fi
77
78         if [[ $umeas = *"K"* ]]; then
79             tputcurr=$(echo "$tputcurr" | awk '{printf("%.5f\n",$1/1000)
                }')
80         fi
81
82         # Maximum found
83         if [ $(echo "$tputcurr > $tputmax" | bc -l) -eq 1 ]; then
84             tputmax=$tputcurr
85         fi
86
87         # Minimum found
88         if [ $(echo "$tputcurr < $tputmin" | bc -l) -eq 1 ]; then
89             tputmin=$tputcurr
90         fi
91
92         lindex=$((lindex+1))
93     fi
94 done
95 lindex=0

```

```

96
97 echo "Max throughput with -i 2: $tputmax"
98 echo "Min throughput with -i 2: $tputmin"
99
100 maxvarperc=$(echo "100*($tputmax-$tputmin)/$tputmax" | bc -l | sed '
      s/^\.\/0./')
101
102 # Print parsed information
103 echo "len=$length,b=$bwidth,tput=$tput,loss=$loss,lostpkt=$lostpkt,
      sentpkt=$sentpkt,okpkt=$okpkt,maxvarputperc=$maxvarperc"
104 # Write to .csv file
105 echo "$length,$bwidth,$tput,$loss,$lostpkt,$sentpkt,$okpkt,
      $maxvarperc" >> "$name-Rate$1_iperf_AC_test.csv"
106 done

```

This script shall always be put inside the same folder where the corresponding logs are placed.

6.4.3 Plots and results: first set

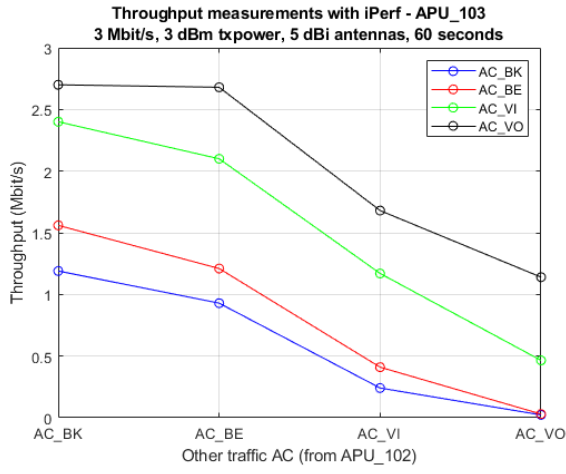
The results are reported inside the following plots, respectively showing, for each physical bitrate, the maximum throughput, the number of correctly received packets and the maximum percentage variation of the throughput when transmitting with a certain AC, against another transmission at a different AC coming from the other board (the “*Other traffic AC*”).

We decided to plot the number of packets which are correctly received instead of the packet loss, as the latter seems to depend on the iPerf behavior, at least when trying to measure the maximum throughput (offering a large amount of data), and as it is predominantly due to the kernel dropping packets for the aforementioned reason, before they can reach the physical medium. Thus, it would not be a measure of the number of packets which may actually be lost in the air.

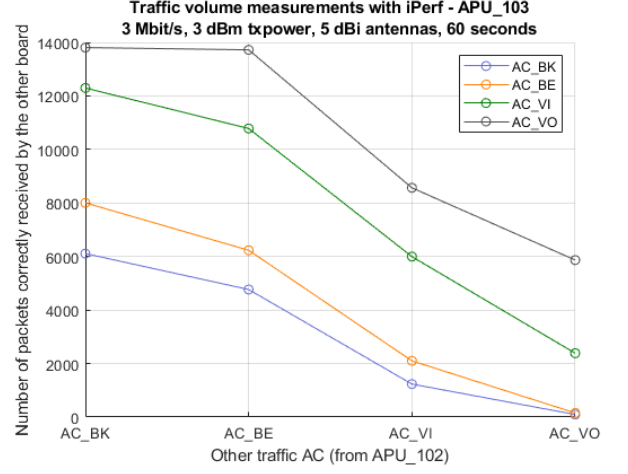
It is very important to take into account that iPerf requires unicast communication

for a certain measurement to be correctly performed. Thus, all these measurements are referred to packet transmission between two boards, using their respective IP and MAC address, not broadcast IP and MAC ones (i.e. 255.255.255.255 and FF:FF:FF:FF:FF:FF).

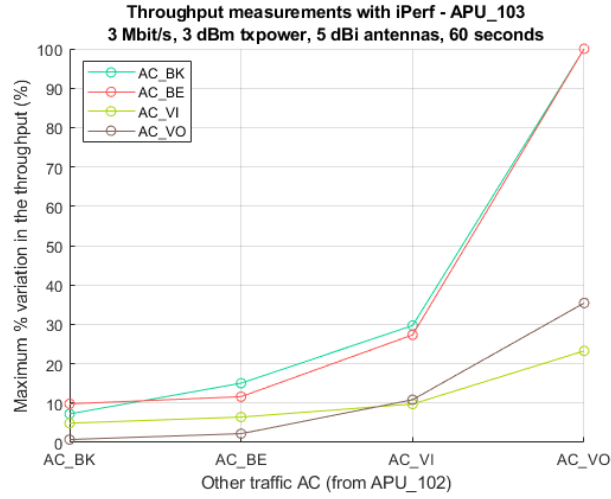
Broadcast communication still requires more investigation and, probably, some important patching work, as detailed at the end of this chapter.



(a) Reachable throughput (Mbit/s)

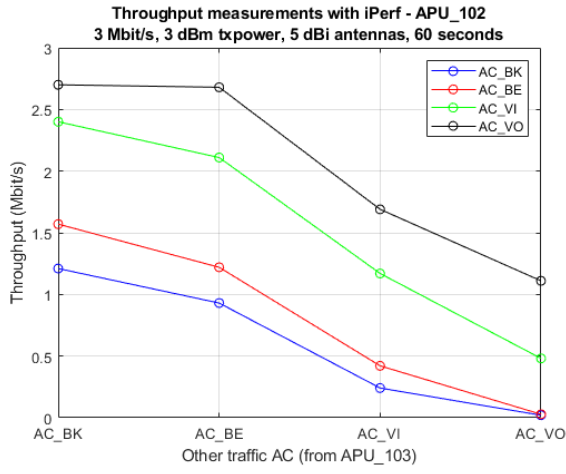


(b) Number of correctly received packets

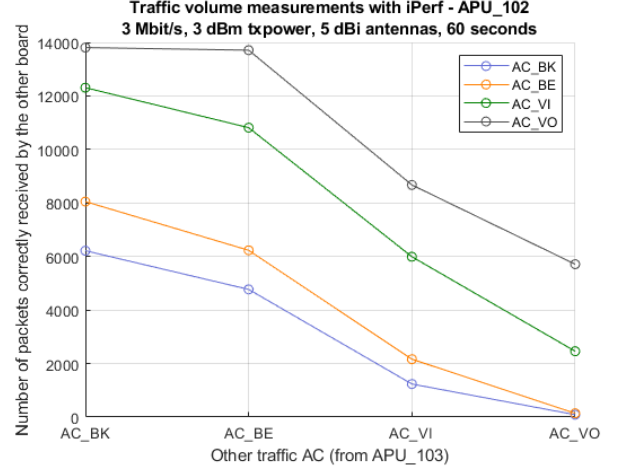


(c) Maximum % variation in the throughput during the test

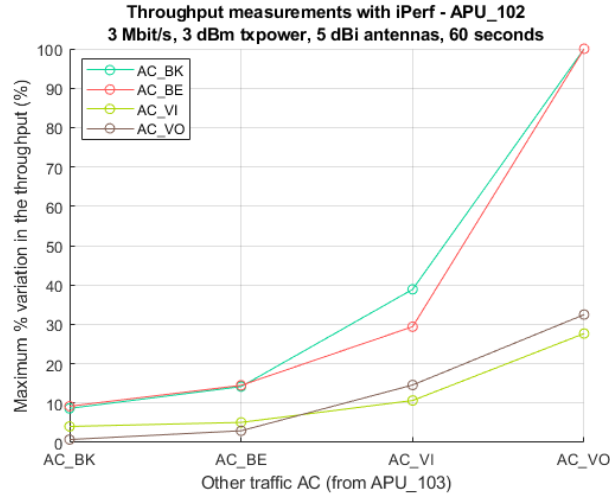
Figure 6.25: Measurements related to the APU_103 board, at 3 Mbit/s of physical data rate



(a) Reachable throughput (Mbit/s)

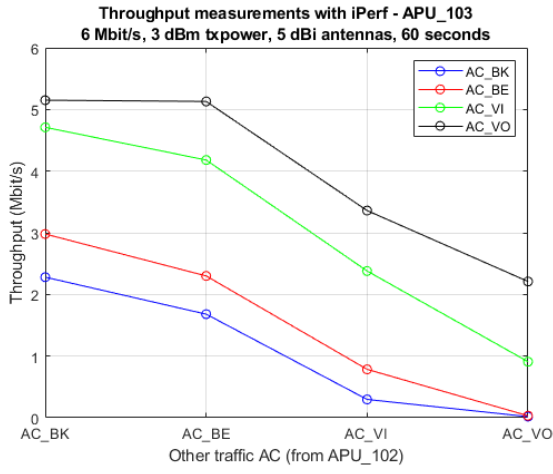


(b) Number of correctly received packets

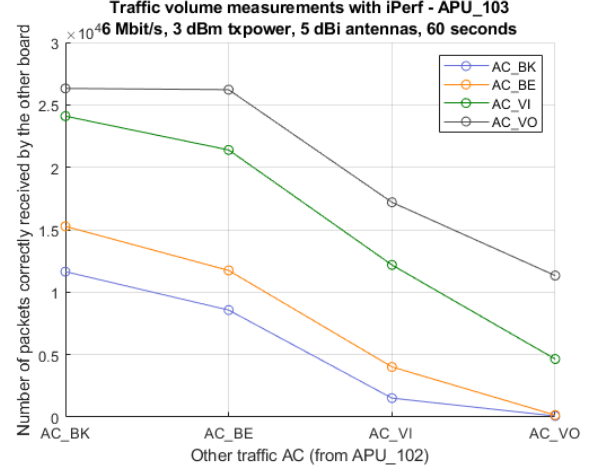


(c) Maximum % variation in the throughput during the test

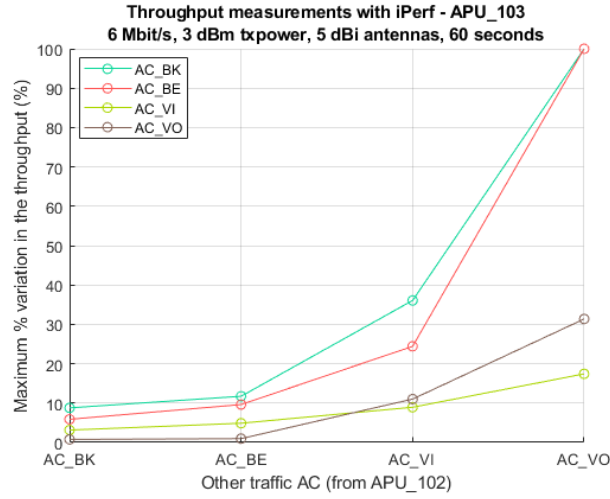
Figure 6.26: Measurements related to the APU_102 board, at 3 Mbit/s of physical data rate



(a) Reachable throughput (Mbit/s)

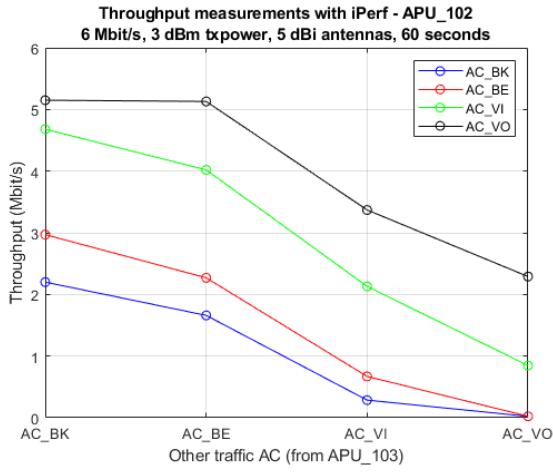


(b) Number of correctly received packets

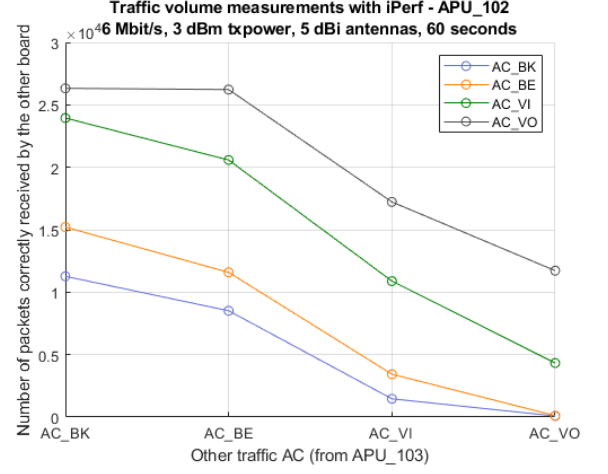


(c) Maximum % variation in the throughput during the test

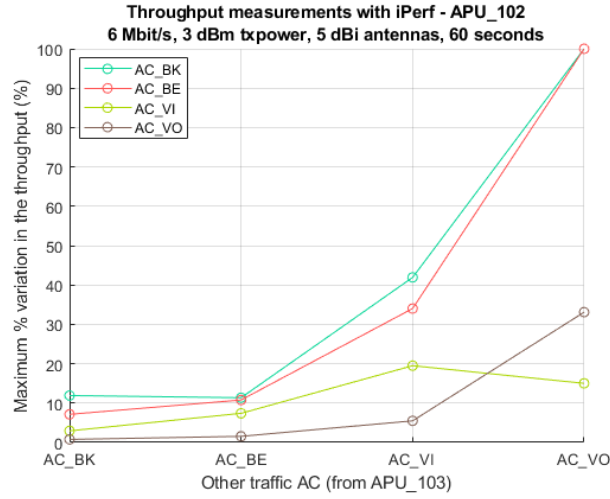
Figure 6.27: Measurements related to the APU_103 board, at 6 Mbit/s of physical data rate



(a) Reachable throughput (Mbit/s)



(b) Number of correctly received packets



(c) Maximum % variation in the throughput during the test

Figure 6.28: Measurements related to the APU_102 board, at 6 Mbit/s of physical data rate

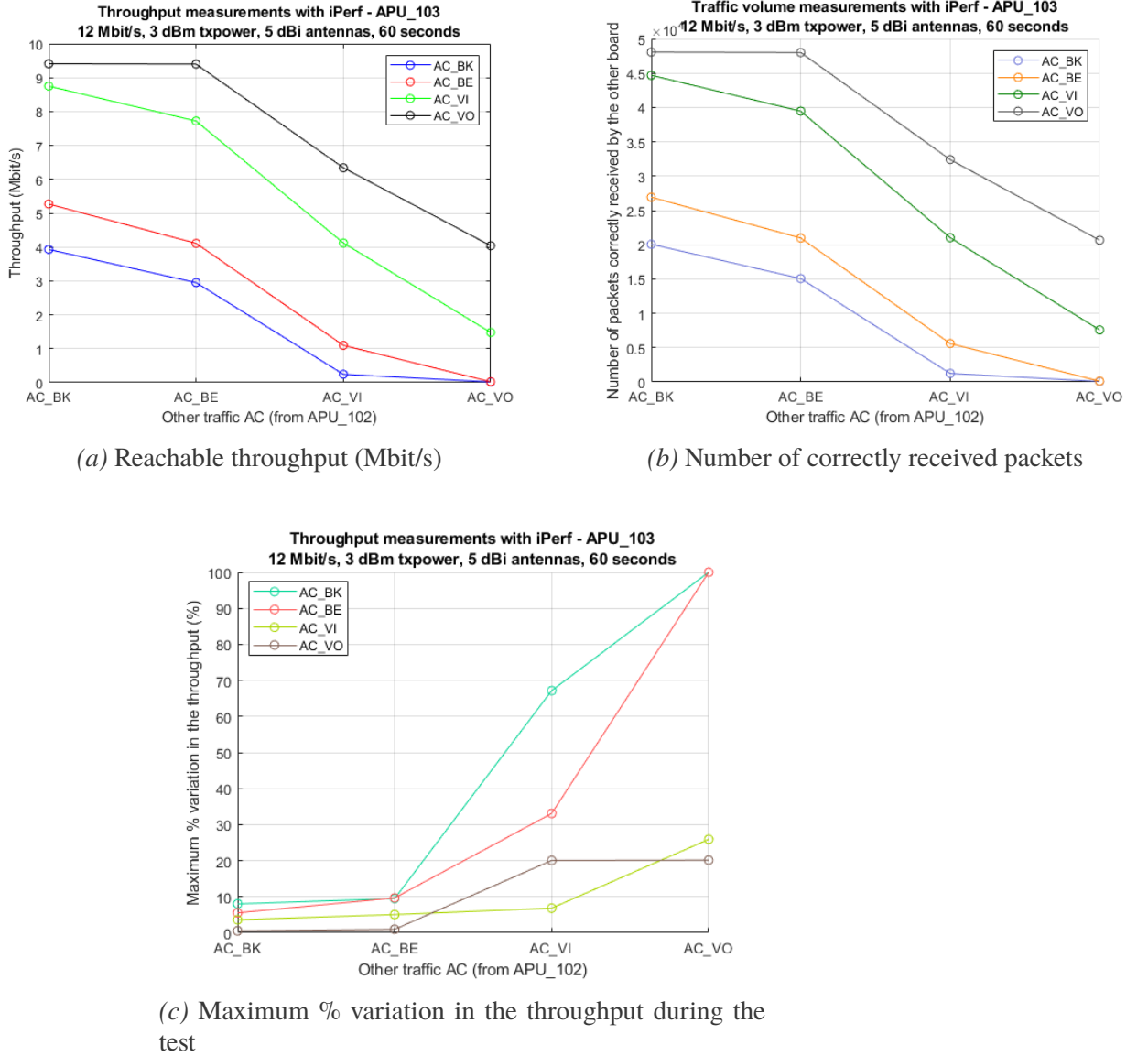


Figure 6.29: Measurements related to the APU_103 board, at 12 Mbit/s of physical data rate

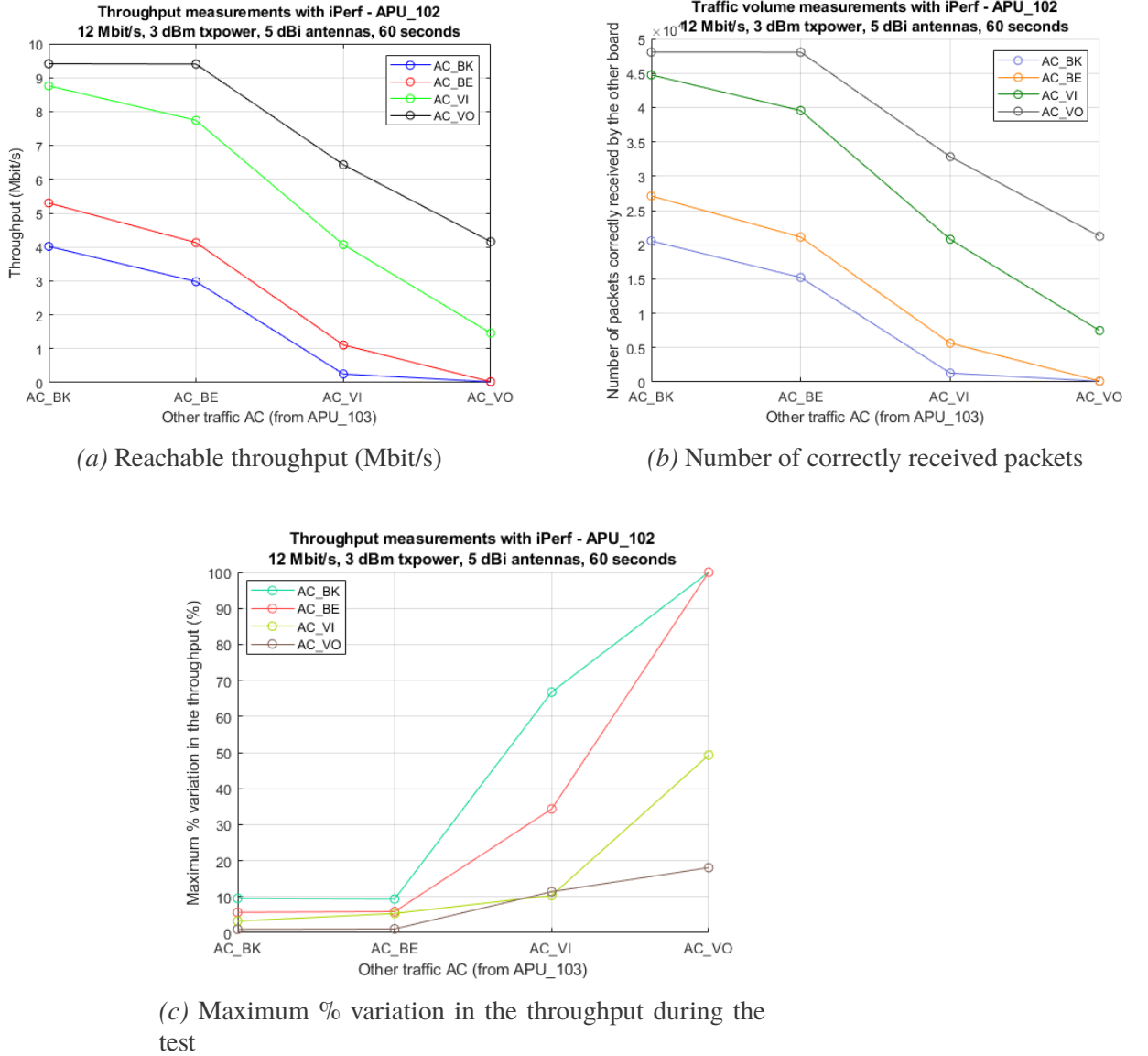


Figure 6.30: Measurements related to the APU_102 board, at 12 Mbit/s of physical data rate

Looking at the plots which are presented here, it is possible to notice that everything is working as expected and as the theory says.

In particular, when two boards are using the same traffic class to send their packets, there's a quite fair channel usage between them, with the throughput being quite well divided in two. This is shown by an horizontal line which can be ideally be traced at around 1.2 Mbit/s for 3 Mbit/s, 2.3 Mbit/s for 6 Mbit/s and 4.1 for 12 Mbit/s, of course with some oscillations around these values.

Then, when communicating with a higher priority AC, as expected, the boards are able to reach higher values of throughput even in presence of another traffic stream at a lower AC.

The stability of the connection, for what concerns throughput (as shown by all the “(c)” plots), has some non linearities with respect to the increasing priority AC, but more or less it is again showing how, using a higher priority AC, it is possible to achieve not only a better throughput, but also less oscillations in the measured values.

Using AC_VI and AC_VO, it also never happens that a board is unable to communicate over a full report interval (2 seconds - being unable to communicate for a full interval would result in a 100% maximum throughput variation), no matter the traffic class used by the “disturbing” traffic.

Moreover, looking at all the “(b)” plots, the trend of the number of correctly received packets is always the same with respect to the throughput, showing that probably hardly any packet is lost in the air in this case, but they are mostly lost in the kernel when trying to push 10 Mbit/s over a congested channel, which can reach a much lower throughput (mainly when using a lower priority AC).

Looking at how the packets are received, by filtering data from a Wireshark capture, over a 20 seconds sample iPerf session with “-b 10M”, it is also possible to deduce that probably, when using high values of “-b 10M”, losses are mainly due kernel drops.

The time at which the packets are received by the sever is in fact quite well distributed around a certain value, with no big “jumps”, as it is possible too see from figure 6.31.

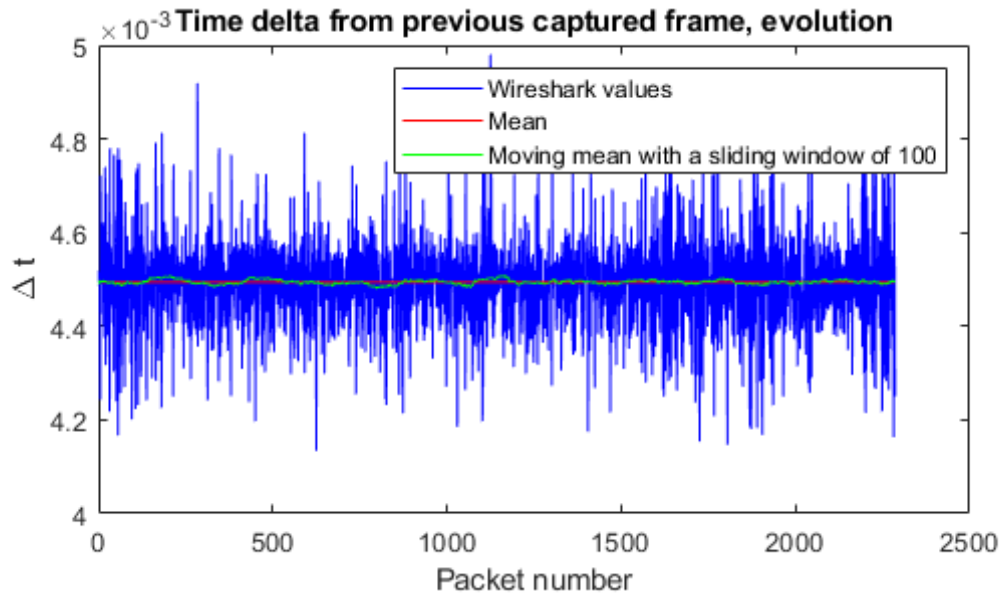


Figure 6.31: Time delta between frames, as seen by Wireshark, capturing frames from the board in which an iPerf server is running; mean value: 4.494 ms.

6.4.4 Plots and results: second set

The goal of these measurements was to verify the maximum reachable throughput and the connection stability, through the percentage variation mentioned before, when only one data stream is transmitted from one board to the other, at different AC and at different physical data rates.

The results are shown in the plots presented starting from the next page.

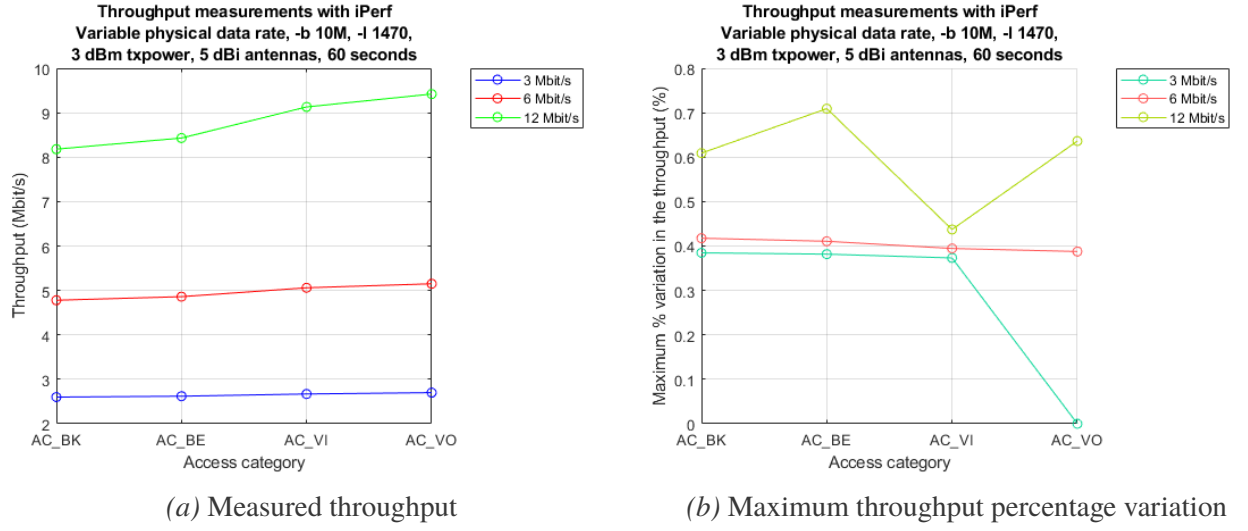


Figure 6.32: Single traffic flow measurements over 60s for each AC and selectable data rate

The measured values, which can be analyzed more in details, are reported in the following table:

Physical rate (Mbit/s)	AC	Measured throughput (Mbit/s)*	Maximum variation (%)*
3	AC_BK	2.60	0.3846%
3	AC_BE	2.62	0.3817%
3	AC_VI	2.67	0.3731%
3	AC_VO	2.70	0.0000%
6	AC_BK	4.78	0.4175%
6	AC_BE	4.86	0.4107%
6	AC_VI	5.06	0.3945%
6	AC_VO	5.15	0.3876%

12	AC_BK	8.18	0.6090%
12	AC_BE	8.43	0.7092%
12	AC_VI	9.13	0.4372%
12	AC_VO	9.42	0.6363%

*Over 60 seconds, with 2 seconds report intervals.

As it is possible to see, the throughput values increases as the priority is increased. This increase becomes more and more evident as higher physical data rates are used: this is actually to be expected, since higher priority AC actually define shorter AIFS (which are all different when working in OCB mode) and smaller contention window sizes; thus, the time between each packet transmission is typically lower for higher priority traffic classes and this effect becomes more important as packets are transmitted faster, i.e. as the physical data rate is increased.

The stability of the measured throughput values is instead always quite high, being it always less than 0.7092%; since only one traffic flow was present, this actually shown that the WNIC we used, together with the our platform, are able to maintain a quite stable throughput.

These values show an increasing behavior as the physical data rate is increased, with some bigger oscillation when using 12 Mbit/s, but they nevertheless remain less than 1%.

Finally, combining these measurements with all the previous ones, it is possible to conclude that AC_BE is the queue actually used when no explicit AC is specified.

6.5 Indoor received power and connectivity measurements

The aim of this final set of measurements is to provide us an idea on how far the boards can communicate, still maintaining a good enough data rate and measured throughput.

Since the measurements are still performed indoors, instead of measuring a distance we measured the received power, which is a more meaningful quantity: in fact the communication, when performed indoors, is affected by the various obstacles which are present and at the same distance different values of received power can be probably observed, as the boards are placed in different points of the house or of the laboratory.

6.5.1 Network configuration and conditions

The conditions are similar to the previous experiments. Now, however, the boards are placed in different points of the house.

In particular, the board running the iPerf client (APU_103) has been left on the desk, shown multiple times in the previous sections and chapters, and the board running the server (APU_102) has instead been placed on a chair, which was moved in different points of the house.

It was possible to power up and connect the board to the local cabled network (in order to enable access to the system via SSH) thanks to a small power strip with a long enough cable; the Ethernet connection was then provided thanks to Powerline, as mentioned in section 3.2.

- 6 Mbit/s physical bitrate
- 5 dBi antenna (MIMO) connected to the boards
- 10 dBm txpower set in OpenWrt (so, this should be the transmitted power without the additional antenna gain of 5 dBi)
- Client placed on the desk (APU_103), server moved around the house (APU_102)
- Duration of each single test: 60 seconds
- iPerf port: 7000
- iPerf report interval: 2 seconds

- Layer 4 protocol: UDP
- Payload size: variable
- UDP buffer size: 208 KB (default value)
- Channel: 178

We decided to set a 10 dBm transmit power since it was a value allowing us to easily move the board inside the house, without the necessity of maintaining a very short or long distance from the client in order to get meaningful values of received power and measured throughput.

6.5.2 Scripts and commands, meaning of the power values

In order to obtain the value of the received power, we decided to rely on the output of “iw dev wlan0 station dump”, constantly looking at the average received signal power, thanks to the following script, which was running on a separate PuTTY window:

watch_stationdump.sh

```
1  #!/bin/bash
2
3  while true; do
4      iw dev wlan0 station dump
5      ./millisleep 0.5
6      clear
7  done
```

This script relies again on the “millisleep” utility.

The first reported dBm value was continuously controlled, during the test, and the more frequent quantity was noted down in a Microsoft Excel spreadsheet. However, oscillations in the reported values of the order of 1 dBm were always present, so these values have to be always considered with a ± 1 dBm precision.

Moreover, no decimal digits are returned by “iw”, which is very probably obtaining these values directly from the driver, and this limits the precision of the observed values.

In particular, we observed a particular situation in which “iw dev wlan0 station dump” was returning the same average received signal level (-88 dBm) in two different, but very near to each other, positions, with almost the same ± 1 dBm oscillations. Probably, however, the true received power was a little higher in one point with respect the other, looking at how better results, in terms of throughput, were actually better.

In order to discriminate the two points and considering that probably the true power was a little higher in one of the two, we assigned to it a value of -87.5 dBm: that’s the meaning of this point that, when looking at the results, later on, could seem a bit odd.

The two scripts used for the measurements are then very similar to the ones presented in this chapter. However, for the sake of completeness, they are also reported here.

iperfserve_powertest2.sh

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 # pl -> $1
13 # pw -> $2
14 # b -> $3
15 # rate -> $4
16 # l -> $5
17 # no -> $6
18 function iperfserver-pl-pw-b-rate-l-no {
```

```

19  echo "# iperf -s -u -i 2 -p 7000 -l $5 -t 64 [txpower $2, place $1,
    no. $6, -b $3]" > "Logs/APU102-server-txpwr-$2-len-$5-place-$1-no
    -$6-rate-$4-b-$3.txt"
20  iperf -s -u -i 2 -p 7000 -l $5 -t 64 2>&1 | tee -a "Logs/APU102-
    server-txpwr-$2-len-$5-place-$1-no-$6-rate-$4-b-$3.txt"
21 }
22
23 if [ $# -ne 1 ]; then
24     echo "Error. One argument expected:"
25     echo "  1. Place number"
26     exit 1
27 fi
28
29 # Check if the Logs directory exists. If not, create it.
30 if [ ! -d "./Logs" ]; then
31     echo "Logs directory missing. It will be created now"
32     mkdir Logs
33 else
34     echo "Logs directory exists. Make a backup of its content before
        proceeding."
35 fi
36 read -p "Press enter to continue"
37
38 txpower=( 1000 )
39 len=( 104 208 500 700 1000 1200 1470 )
40
41 echo "Starting... tests will last aproximately $(( ${#txpower[@]}*${#
    len[@]}*60+${#txpower[@]}*${#len[@]}*15 )) seconds"
42 echo "Current place: $1"
43 sleep 1
44
45 # Parameters
46 b=10M
47 rate=6 # In Mbit/s

```

```

48
49 echo "Setting physical layer data rate to $rate Mbit/s..."
50 wifiarate=$((($rate*2))
51 iw dev wlan0 set bitrates legacy-5 $wifiarate
52 sleep 1
53
54 currwait=0
55 i=0
56 p=0
57 while [ $p -lt ${#txpower[@]} ]; do
58     echo "Setting txpower to txpower=${txpower[$p]} mBm..."
59     iw dev wlan0 set txpower fixed ${txpower[$p]}
60     sleep 1
61     echo "Current settings: " | tee -a "Logs/APU102-txpwr-${txpower[$p]}
        ]}-iw-dev-log.txt"
62     iw dev | tee -a "Logs/APU102-txpwr-${txpower[$p]}-iw-dev-log.txt"
63     sleep 1
64     while [ $i -lt ${#len[@]} ]; do
65         echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":
            " -f3 | cut -d" " -f1), waiting until second $currwait"
66         waituntil $currwait
67         echo "-----"
68         echo "Running test with txpower=${txpower[$p]} mBm, phys. rate=
            $rate Mbit/s,"
69         echo " -b $b, -l ${len[$i]}, -t 60, number=$i"
70         iperfserver-pl-pw-b-rate-l-no $1 ${txpower[$p]} $b $rate ${len[$i]}
            ]] $i
71         echo "Test with txpower=${txpower[$p]} mBm, -l ${len[$i]}, number=
            $i terminated"
72         echo "-----"
73         i=$((i+1))
74         currwait=$(( (currwait+25)%60 ))
75     done
76     sleep 1

```



```

77  p=$((p+1))
78  i=0
79  done

```

iperfclient_powertest2.sh

```

1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 # p1 -> $1
13 # pw -> $2
14 # b -> $3
15 # rate -> $4
16 # l -> $5
17 # no -> $6
18 function iperfclient-pl-pw-b-rate-l-no {
19     echo "# iperf -c 10.10.6.102 -u -i 2 -t 60 -b $3 -p 7000 -l $5 [
        txpower $2, place $1, no. $6]" > "Logs/APU103-client-txpwr-$2-len
        -$5-place-$1-no-$6-rate-$4-b-$3.txt"
20     iperf -c 10.10.6.102 -u -i 2 -t 60 -b $3 -p 7000 -l $5 2>&1 | tee -a
        "Logs/APU103-client-txpwr-$2-len-$5-place-$1-no-$6-rate-$4-b-$3.
        txt"
21 }
22
23 if [ $# -ne 1 ]; then
24     echo "Error. One argument expected:"

```

```

25  echo " 1. Place number"
26  exit 1
27  fi
28
29  # Check if the Logs directory exists. If not, create it.
30  if [ ! -d "./Logs" ]; then
31      echo "Logs directory missing. It will be created now"
32      mkdir Logs
33  else
34      echo "Logs directory exists. Make a backup of its content before
        proceeding."
35  fi
36  read -p "Press enter to continue"
37
38  txpower=( 1000 )
39  len=( 104 208 500 700 1000 1200 1470 )
40
41  echo "Starting... tests will last aproximately $(( ${#txpower[@]}*${#len[@]}*60+${#txpower[@]}*${#len[@]}*15 )) seconds"
42  echo "Current place: $1"
43  sleep 1
44
45  # Parameters
46  b=10M
47  rate=6 # In Mbit/s
48
49  echo "Setting physical layer data rate to $rate Mbit/s..."
50  wifiarate=$(( $rate*2 ))
51  iw dev wlan0 set bitrates legacy-5 $wifiarate
52  sleep 1
53
54  currwait=0
55  i=0
56  p=0

```

```

57 while [ $p -lt ${#txpower[@]} ]; do
58     echo "Setting txpower to txpower=${txpower[$p]} mBm..."
59     iw dev wlan0 set txpower fixed ${txpower[$p]}
60     sleep 1
61     echo "Current settings: " | tee -a "Logs/APU103-txpwr-${txpower[$p]}-iw-dev-log.txt"
62     iw dev | tee -a "Logs/APU103-txpwr-${txpower[$p]}-iw-dev-log.txt"
63     sleep 1
64     while [ $i -lt ${#len[@]} ]; do
65         echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":
66             " -f3 | cut -d" " -f1), waiting until second $currwait"
67         waituntil $currwait
68         echo "-----"
69         echo "Running test with txpower=${txpower[$p]} mBm, phys. rate=
70             $rate Mbit/s,"
71         echo " -b $b, -l ${len[$i]}, -t 60, number=$i"
72         sleep 2
73         iperfclient-pl-pw-b-rate-l-no $1 ${txpower[$p]} $b $rate ${len[$i]} $i
74         echo "Test with txpower=${txpower[$p]} mBm, -l ${len[$i]}, number=
75             $i terminated"
76         echo "-----"
77         i=$((i+1))
78         currwait=$(( (currwait+25)%60 ))
79     done
80     sleep 1
81     p=$((p+1))
82     i=0
83 done

```

It can be interesting to note that the logs are now generated with several parameters inside their names, in order to be able to manage them in an easier way, even in case other measurements of the same kind will be performed in the future.

The parameters are:

Parameter	Script name	Parameter number inside script function*
Place index**	pl (\$1)	1
<i>txpower</i>	pw (txpower[])	2
Offered traffic	b	3
Physical data rate	rate	4
Payload length	l (len[])	5
Current measurement loop index	no (i)	6

*The function this table refers at is “iperfclient-pl-pw-b-rate-l-no” or “iperfserver-pl-pw-b-rate-l-no”, depending on the script.

**An arbitrary index was given to each place where the board running the server was placed, in order to distinguish the various logs more easily.

After setting a certain *txpower* we also decided to log the output of “iw dev”, in order to check whether this parameter was correctly set or not.

This script was then launched inside every single place to perform a set of measurements for each position, with the logs being all saved inside the same folder (without any issue due to them having different names depending on the place).

The relevant data was then extracted thanks to a log extraction script (generating a .csv file), similar to all the other ones but adapted to work in this specific case, when placed, again, inside the same folder where all the logs are stored (after copying everything to the development PC: this script is not meant to be run directly on the boards).

logExtractor_power.sh

```
1 #!/bin/bash
2
```

```

3 # shopt is called in order to run the while read cycle in the current
  shell, otherwise the
4 # values of "tputmin" and "tputmax" would not keep their values in the
  current shell;
5 # with this, the last pipeline segment is always executed in the
  current shell and not
6 # in a subshell
7 shopt -s lastpipe
8
9 # parameters
10 bwidth=10 # M
11 physrate=6
12
13 # arrays
14 txpower=( 1000 )
15 len=( 104 208 500 700 1000 1200 1470 )
16 places=( 5 7 9 8 6 ) # In order of received power
17
18 i=0
19 p=0
20 pl=0
21 echo "txpower,len,place,b,tput,loss,lostpkt,sentpkt,okpkt,
    maxvarputperc,jittermax" > "APU102-power-tests-8090dBm.csv"
22 while [ $p -lt ${#txpower[@]} ]; do
23 while [ $i -lt ${#len[@]} ]; do
24 while [ $pl -lt ${#places[@]} ]; do
25 filename=$(echo APU102-server-txpwr-${txpower[$p]}-len-${len[$i]}-
    place-${places[$pl]}-no-$i-rate-$physrate-b-"$bwidth"M.txt)
26
27 echo $filename
28
29 lastline=$(cat $filename | tail -1)
30 if [[ "$lastline" =~ "out-of-order" ]]; then
31 hcommand=-1

```

```

32 else
33     hcommand=-0
34 fi
35 #echo "lastline: $lastline"
36 #echo "hcommand: $hcommand"
37
38 if [[ "$lastline" =~ "-----" ]]; then
39     tput=0
40     jittermax=60000 # 60 seconds as a very big value (it would be +inf
                     # theoretically)
41     lostpkt=0
42     sentpkt=0
43     okpkt=0
44     loss=100
45     maxvarperc=100
46     tputmax=0
47     tputmin=0
48
49 echo "No connection could be established in this case."
50 else
51     tput=$(cat $filename | head -n $hcommand | tail -1 | sed 's/\s\s*/ /
           g' | cut -d" " -f7)
52     umeas=$(cat $filename | head -n $hcommand | tail -1 | sed 's/\s\s*/
           /g' | cut -d" " -f8)
53     # Convert measurements in KBit/s to MBit/s
54     if [[ $umeas = *"K"* ]]; then
55         tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
56     fi
57     lostpkt=$(echo $(cat $filename | head -n $hcommand | tail -1 | cut
           -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" " -f1 | rev)
58     sentpkt=$(cat $filename | head -n $hcommand | tail -1 | cut -d"/" -
           f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
59
60     okpkt=$((sentpkt-lostpkt))

```

```

61  loss=$(echo "$lostpkt $sentpkt" | awk '{printf("%.15f\n",$1/$2*100)
      }')
62
63  lindex=0
64  tputmax=-1
65  tputmin=1000000
66  jittermax=-1
67  # Extract maximum % variation of throughput values
68  cat $filename | head -n $((hcommand-1)) | tail -n +9 | sed 's/\s\s*/
      /g' | while read line; do
69  if [ $lindex -eq 0 ]; then
70      echo "First line: $line"
71  fi
72
73  if [[ "$line" =~ "out-of-order" ]]; then
74      echo "Skipping line..."
75  else
76  if [ $lindex -lt 4 ]; then
77      tputcurr=$(echo $line | cut -d" " -f8)
78      umeas=$(echo $line | cut -d" " -f9)
79      jittercurr=$(echo $line | cut -d" " -f10)
80  else
81      tputcurr=$(echo $line | cut -d" " -f7)
82      umeas=$(echo $line | cut -d" " -f8)
83      jittercurr=$(echo $line | cut -d" " -f9)
84  fi
85
86  if [[ $umeas = *"K"* ]]; then
87      tputcurr=$(echo "$tputcurr" | awk '{printf("%.5f\n",$1/1000)}')
88  fi
89
90  # Maximum found
91  if [ $(echo "$tputcurr > $tputmax" | bc -l) -eq 1 ]; then
92      tputmax=$tputcurr

```

```

93  fi
94
95  # Minimum found
96  if [ $(echo "$tputcurr < $tputmin" | bc -l) -eq 1 ]; then
97      tputmin=$tputcurr
98  fi
99
100 # Maximum jitter in every 2s report
101 if [ $(echo "$jittercurr > $jittermax" | bc -l) -eq 1 ]; then
102     jittermax=$jittercurr
103 fi
104
105 lindex=$((lindex+1))
106 fi
107 done
108 lindex=0
109
110 maxvarperc=$(echo "100*($tputmax-$tputmin)/$tputmax" | bc -l | sed '
111     s/^\.\/0.\.\/')
112 fi
113 # Print parsed information
114 echo "txpower=${txpower[$p]},len=${len[$i]},place=${places[$pl]},b=
115     $bwidth,tput=$tput,loss=$loss,lostpkt=$lostpkt,sentpkt=$sentpkt,
116     okpkt=$okpkt,maxvarperc=$maxvarperc,jittermax=$jittermax,max-
117     tput=$tputmax Mbit/s,min-tput=$tputmin Mbit/s"
118 # Write to .csv file
119 echo "${txpower[$p]},${len[$i]},${places[$pl]},$bwidth,$tput,$loss,
120     $lostpkt,$sentpkt,$okpkt,$maxvarperc,$jittermax" >> "APU102-power-
121     tests-8090dBm.csv"
122 pl=$((pl+1))
123 done
124 i=$((i+1))
125 pl=0

```



```
121 done
122 p=$((p+1))
123 i=0
124 done
```

It is possible to notice, from line 16, that the data related to the different places (depending on the arbitrary index that we assigned them) are ordered from higher to lower received power, in order to extract the data from the logs in that order and obtain meaningful plots, with points first related to places where a higher received power was detected and then related to places with a lower received power.

6.5.3 Plots and results

The results are shown in the following plots:

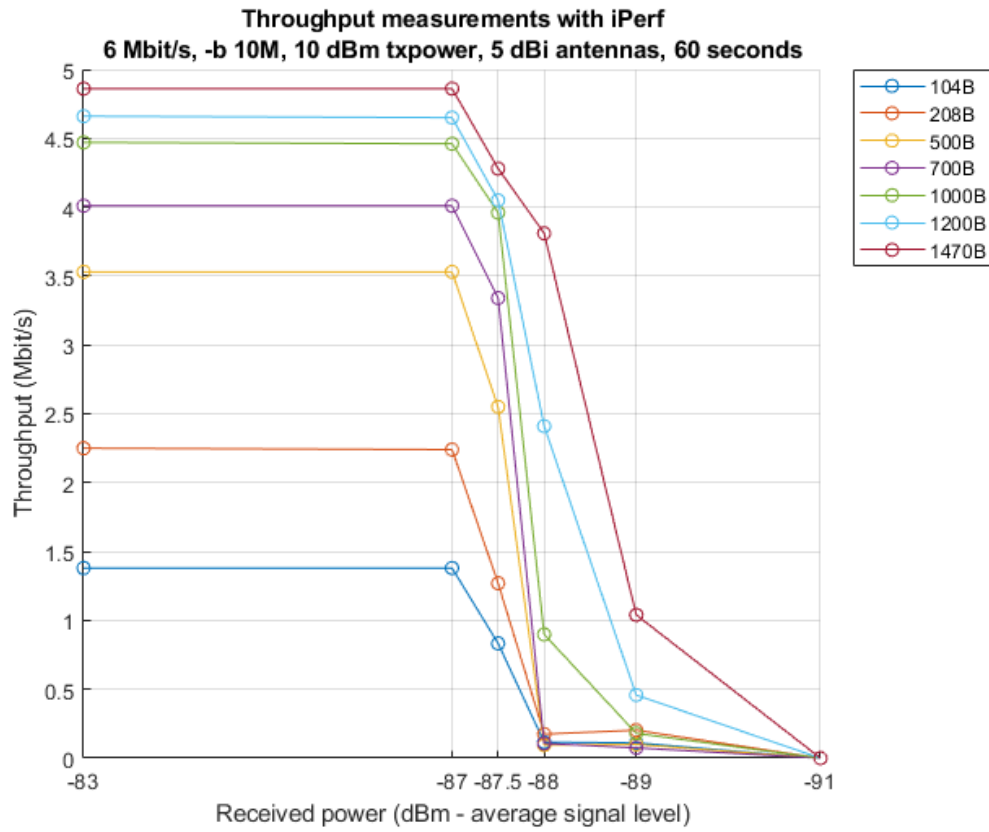


Figure 6.33: Throughput measurements for different values of received power

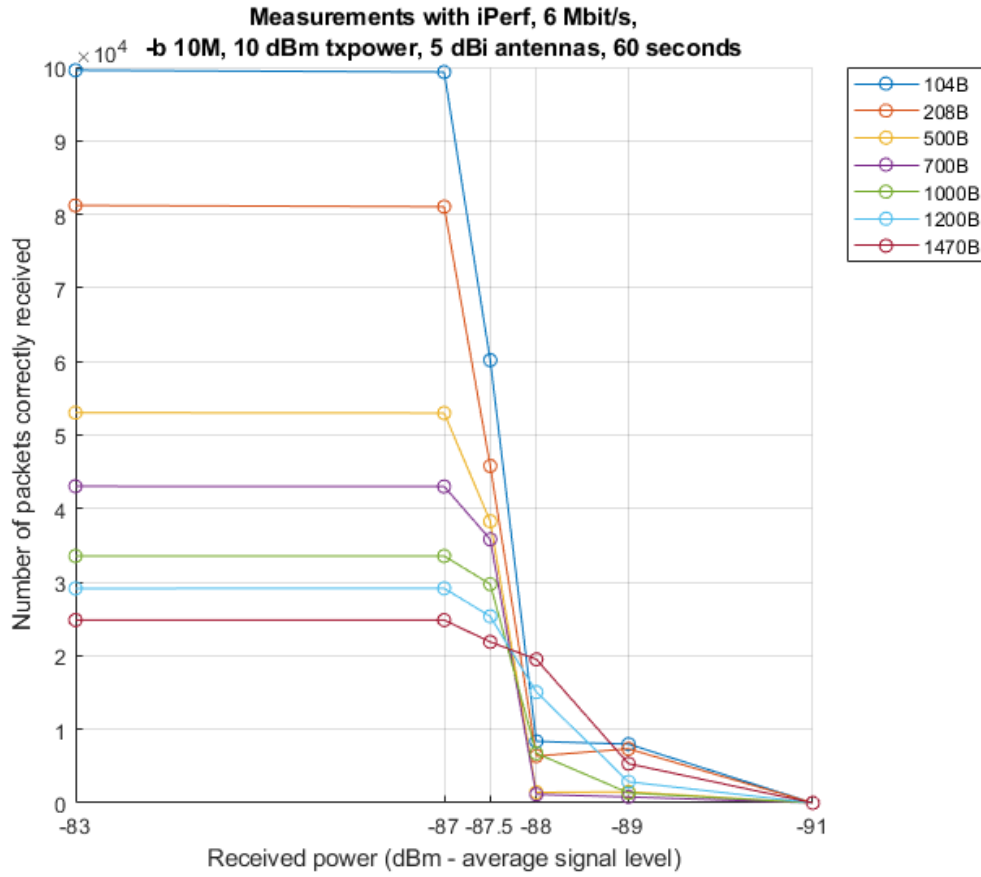


Figure 6.34: Received number of packets for different values of received power; it is possible to notice that as the payload is increase, the total number of transmitted packet is reduced. This is actually expected, as we are always trying to offer the same amount of traffic, which is very high, in order to assess the limits of our system.

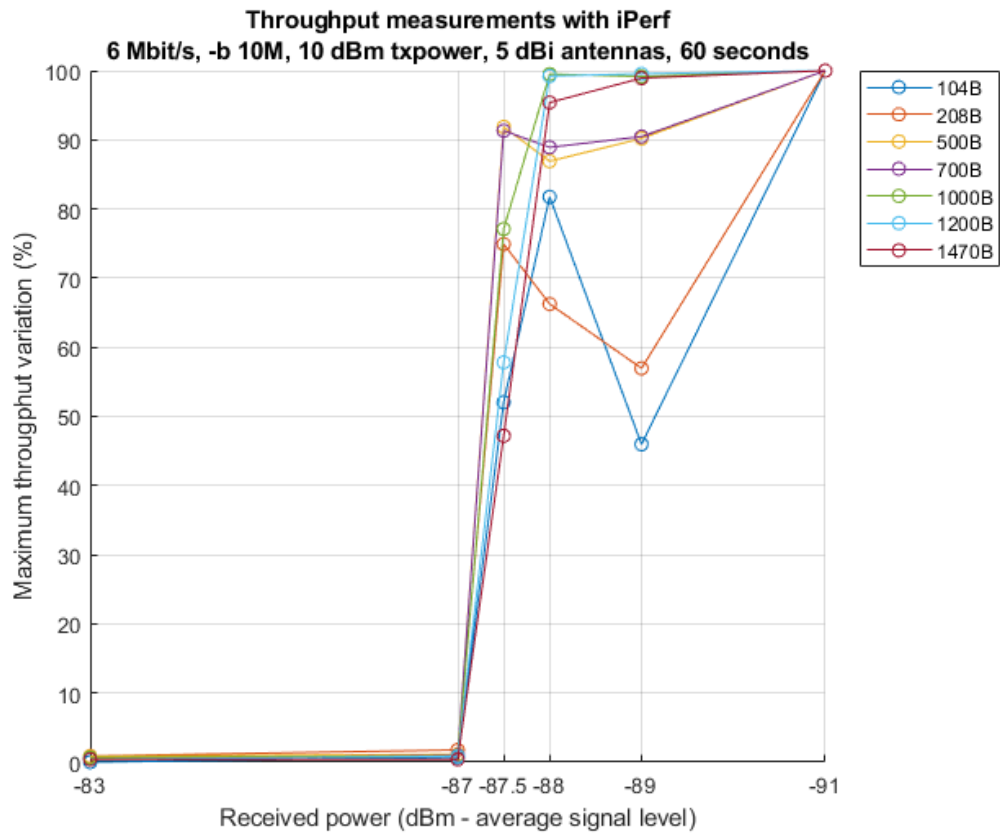


Figure 6.35: Maximum percentage variation of the throughput over 60s tests, with 2 seconds report intervals

As it is possible to see, the indoor connectivity remains stable, with good throughput values, until the received power goes below -87 dBm. Then, both the connection stability and the throughput heavily drop to much worse values, until no connectivity can happen anymore.

As expected from the previous measurements, the larger the payload size, the higher is the reachable throughput; at -89 dBm, when the connection is not very stable, this relationship is no more strictly respected and all the measurements with a payload length of 1000B or less give a throughput of maximum 203 Kbit/s, for “-1 208”, which is a quite low value. The highest throughput value remains, nevertheless, the one with a payload length of 1470 B.

We were never able to see a value of -91 dBm or less when the boards were communicating, even considering oscillations in these values, provided by the *station dump*. Thus, we considered this value as a received power at which a throughput of 0 Mbit/s can be reached, adding a corresponding point to each plot and for each payload size.

The related maximum variation value was set to 100% to show a completely unreliable connection (in fact, it seems that the connection cannot be established at all in these conditions), even through this is not a real 100% variation obtained from some iPerf reports (which could eventually return 0%, but in this case the test could not even start).

For what concerns instead the maximum % variation plot, it is possible to notice that oscillations and non linearities, with respect to the payload size, occur as soon as the received signal becomes weak.

It was also possible to notice that even small variations of the position of the receiving board could cause a non negligible variation in the received power; this was evident when the signal was weak, i.e. when moving around -87 to -89 dBm, for which even 20/25 cm were sufficient to vary the received power and the measured values.

By setting a *txpower* of 10 dBm (which is still far from the maximum of the UNEX DHXA-222 cards), in any case, the boards were able to communicate even through a closed door and in non-line-of-sight conditions.

6.5.4 Additional considerations

All the presented measurements are related to indoor tests, where obstacles, doors and walls are present.

In a future improvement of this work it could be important to make outdoor and on the field tests, using vehicles, to better assess the performance depending of the effective distance between the two boards.

6.6 Broadcast communication issue with Linux kernel version 4

To conclude this chapter, we want to present a problem that we discovered during the last days before the end of this work.

The problem is basically due to the fact that, when using recent versions of the Linux kernel (as in OpenWrt 18.06.1, with kernel 4.14.63), only one priority queue can be used as far as broadcast communication is concerned. This problem seems not to affect older kernel versions, such as version 3.18, on which OpenC2X was successfully tested.

When sending broadcasted data only AC_BE can be used, no matter the AC that the user is trying to set in his or her application; this seems to happen every time a broadcast destination MAC address is used (*FF:FF:FF:FF:FF:FF*), sending out packet which will not be acknowledged by any device, even though they are properly received.

Instead, when sending unicast data, everything works fine and all the queues can be used: it is the case of all the measurements presented in section 6.4, which require a precise destination MAC address to be set in each packet, otherwise iPerf cannot start any test.

It is possible to obtain statistics over the used queues and ACs thanks to *ath9k*, by querying a particular file: `"/sys/kernel/debug/ieee80211/phy0/ath9k/xmit"`.

A script periodically displaying this file's content has been written, in order to check the updated queues' status every 0.5 s (thanks to `"millisleep"`): as shown in figure 6.36,

only AC_BE can be used when sending broadcast data.

	BE	BK	VI	VO
MPDUs Queued:	0	0	0	0
MPDUs Completed:	16	0	0	0
MPDUs XRetried:	0	0	0	0
Aggregates:	0	0	0	0
AMPDUs Queued HW:	0	0	0	0
AMPDUs Completed:	0	0	0	0
AMPDUs Retried:	0	0	0	0
AMPDUs XRetried:	0	0	0	0
TXERR Filtered:	0	0	0	0
FIFO Underrun:	0	0	0	0
TXOP Exceeded:	0	0	0	0
TXTIMER Expiry:	0	0	0	0
DESC CFG Error:	0	0	0	0
DATA Underrun:	0	0	0	0
DELIM Underrun:	0	0	0	0
TX-Pkts-All:	16	0	0	0
TX-Bytes-All:	1560	0	0	0
HW-put-tx-buf:	16	0	0	0
HW-tx-start:	0	0	0	0
HW-tx-proc-desc:	16	0	0	0
TX-Failed:	0	0	0	0

Figure 6.36: Results of querying the `.../ath9k/xmit` file after running for a few seconds the C program for broadcast transmission described in section 3.8.5 with `“./broadcastSend 6000 255.255.255.255 4 1 test_payload”`. Even though AC_VE (user priority equal to 4) was selected, only AC_BE seemed to be used.

The script is the following one:

`watch_debugfs_ath9k_xmit.sh`

```

1  #!/bin/bash
2
3  while true; do
4      cat /sys/kernel/debug/ieee80211/phy0/ath9k/xmit
5      ./millisleep 0.5
6      clear
7  done

```

This problem, after debugging the kernel with a patched version of OpenWrt, in which

various calls to “`printk()`” were added, both inside *ath9k* and *mac80211*, seems to be caused by the introduction of the so-called *intermediate software queues* inside *mac80211*.

This feature was introduced to move the queuing implementation more towards the software side of the wireless subsystem, allowing the hardware to keep only short queues and enabling also more fairness between stations which are communicating, since these queues, when in sending unicast data, seems to be kept for each station (and for each Thread ID: “*tid*”).

The only documentation we were able to find about this patch comes from the “*mac80211.h*” header file, which reports:

“DOC: mac80211 software tx queueing

mac80211 provides an optional intermediate queueing implementation designed to allow the driver to keep hardware queues short and provide some fairness between different stations/interfaces.

In this model, the driver pulls data frames from the mac80211 queue instead of letting mac80211 push them via `drv_tx()`. Other frames (e.g. control or management) are still pushed using `drv_tx()`.

Drivers indicate that they use this model by implementing the `.wake_tx_queue` driver operation.

*Intermediate queues (struct `ieee80211_txq`) are kept per-sta per-tid, with **a single per-vif queue for multicast data frames**.*

The driver is expected to initialize its private per-queue data for stations and interfaces in the `.add_interface` and `.sta_add` ops.

*The driver can’t access the queue directly. To dequeue a frame, it calls `ieee80211_tx_dequeue()`. Whenever *mac80211* adds a new frame to a queue, it calls the `.wake_tx_queue` driver op.*

*For AP powersave TIM handling, the driver only needs to indicate if it has buffered packets in the driver specific data structures by calling `ieee80211_sta_set_buffered()`. For frames buffered in the `ieee80211_txq` struct, *mac80211* sets the appropriate TIM PVB bits*

and calls `.release_buffered_frames()`.

In that callback the driver is therefore expected to release its own buffered frames and afterwards also frames from the `ieee80211_txq` (obtained via the usual `ieee80211_tx_dequeue`).”

From this documentation, it seems to be clear that only one queue has been implemented for multicast (and broadcast) frames, not taking into account the necessity of sending prioritized broadcast data, which is a very peculiar characteristic of VANETs.

Also the *ath9k* driver has been patched to support these `mac80211` intermediate queues, with the result of queuing the broadcast data to `AC_BE` only.

In particular, the queuing and buffering inside the driver has been mostly removed, keeping only the retry queue, and the driver now gets data directly from *mac80211* when needed [75].

Exchanging few e-mails with one of the developers of the *ath9k* patch, Dr. Toke Høiland-Jørgensen, it is possible to speculate that the *mac80211* implementation (which was coded before he started working on the code for *ath9k*) could actually be due to the fact that normally, outside the context of VANETs, it is not beneficial to allow high priority broadcast traffic, since it may bring down the whole network performance, for instance when sending too much multicast data over `AC_VO`.

VANETs, however, are working at different frequencies than normal Wi-Fi and really require this feature to be implemented, making *mac80211* allocate and manage 4 queues for multicast, instead of one only. A good starting point, as suggested by Dr. Høiland-Jørgensen, could be the analysis of the “`struct ieee80211_vif`” structure in *mac80211*, managing the per-VIF data (as one multicast queue only is defined *per-VIF*).

This should be one of the most important improvements of this work, as it can be very useful to be able to build 802.11p systems based on updated embedded Linux versions.

Of course, after the implementation of such a feature and the coding of a proper patch (which requires working both on the *mac80211* and *ath9k* source files), it will be important to test this feature with measurement similar to the ones presented in section 6.4; this

will, however, require some patching work also on iPerf or, possibly, the use of a custom measurement tool, since broadcast measurements are not possible at the moment, at least as iPerf is concerned.

Chapter 7

Conclusions

7.1 APU boards startup

Before moving to the conclusions of this work, it can be useful to remind all the useful steps to bring the APU boards to an operating state, when using the platform described here, as we did every time we used them.

The suggested commands, after connecting the power supply to the main or to a suitable battery are:

1. Log in with “root” as user name; at the moment, to facilitate all the experiments, no password has been set. It may be important, however, if you use OpenWrt, to set up a strong enough password: this can be done using the “passwd” command or with the LuCi graphical interface, which can be accessed by opening any browser and writing the board’s Ethernet IP address.
2. Before doing anything else, always run “iw_startup” to launch the OCB mode and 802.11p startup script, in order to properly configure the boards.
3. If the *bash* shell is available, as in our case, run it by writing “bash”: since it is more advanced than the built-in *sh* shell and it is able to store the commands history, it is suggested to use it.

7.2 Channel switching

During this work, we also performed an alternating channel switching test, as required by the IEEE 1609.4 standard [30].

We verified that, due to time slots having a default duration of 50 ms, it is very difficult to rely only on “iw dev <interface> ocb leave” and “iw dev <interface> ocb join <new channel frequency in MHz> 10MHz”, since they seem to introduce too much delay, causing losses, even when launching a prototype of a possible channel switching daemon in both the boards and in a synchronized way (thanks to NTP and the “waituntil” function in the script responsible for starting it). The implementation of the IEEE 1609.4 features would also require implementing the proper queueing policy for the packets, for instance allowing only non IP traffic to be queued for transmission on CCH.

Moreover, since WLAN cards have reduced their overall cost lately, it could be more efficient to implement an ETSI compliant system, which foresees two WNIC listening to two different channels instead of looking into channel switching methods.

7.3 Improvements

This work leaves a lot of room for improvement and for the implementation of other V2X features.

Other than what it is written throughout all the previous chapters, the following list resumes the main improvements that can be done:

1. Implementation of 4 queues on *mac80211* in order to enable again the EDCA traffic prioritization for broadcast traffic, as detailed in section 6.6.
2. Analysis of why only 3 physical data rates are actually usable, and, if the cause is not in the hardware device, implementation of a patch to enable other intermediate rates and up to 27 Mbit/s; implementation of a patch making iw aware of 802.11p

rates, allowing to simply set the rate by writing the real value, not its double. This was mentioned before in section 5.8.

3. Measurements over true field tests, in real vehicles and outdoors, using batteries to power up the boards and proper antennas.
4. Integration of OpenC2X inside OpenWrt 18.06.1, as in OpenC2X embedded/LEDE.
5. Implementation of other IEEE 1609.x V2X features, such as security.

7.4 Conclusions

After integrating the patched OpenWrt platform inside the APU boards and performing all the measurements presented in chapter 6, we were able to confirm that the boards are actually capable of performing 802.11p based communications.

In particular, we were able to show that the EDCA queues are actually working and can prioritize the outgoing traffic, allowing higher priority traffic (AC_VI and AC_VO) to obtain a higher throughput and a better connection stability with respect to lower priority one (AC_BK and AC_BE), at least in the unicast case.

Unluckily broadcast traffic, as stated before, still cannot be properly prioritized in the new Linux kernel versions, but the unicast measurements presented in this thesis can be used as a base to show that EDCA is actually working as it should.

Also, the packet loss and throughput measurements were quite the expected ones; through them we were also able to better characterize how the system composed by an APU boards and a UNEX DHXA-222 WLAN card behaves.

The buffer characterization data is instead giving us few important information:

- When a programmer has to build an application, mainly when a lot of traffic is offered or when it will be used in congested situations, it has to take into account that there are buffers between his application and the physical medium. Some buffers, such as the UDP buffer we used, are managed by the kernel and their size, set for

instance by means of `setsockopt()` plays an important role both in the application performance and in the process of detecting possible packet losses.

- Packets may not only be dropped when the wireless medium is very congested or when the receiver detects an error, but they may also be dropped prior to the WNIC, when it cannot handle all the traffic the user is trying to send. This is also related to the socket send buffer mentioned before.
- All the packet loss measurements shown before, when trying to offer too much traffic with respect to what the UNEX cards can handle, are depending on the specific measurement application (i.e. iPerf) behavior, combined with the driver and WNIC one; a different application or a different driver (or even a version of mac80211 with new features inside) may influence these results, although they should maintain the trend we were able to obtain (i.e. low losses when offering a reasonable amount of traffic, higher losses as we offer more traffic with respect to the observed maximum). In any case, we were able to verify that hardly any packet got dropped in the air.

Considering this, it is possible to conclude that our platform can be a good and functional PHY and MAC layers skeleton, even though several improvements are still possible, for future development of more complex applications and 1609.x protocol features on top of the hardware and operating system.

7.5 Acknowledgements

For all the development of this work I want to thank Florian Klingler and the CCS Labs team, working on V2X solution in *University of Paderborn*.

They developed the OpenC2X, ETSI compliant, platform, together with the fundamental patches to OpenWrt and the Linux kernel to make everything work [36].

As mentioned before, these patches have been integrated in OpenWrt 18.06.1, together with some custom ones, and effectively used during all the measurement sessions.

Dr. rer. nat. Florian Klingler also kindly replied too all my e-mails, which I sent him when I had doubts about their platform or about the patched LEDE system.

I also want to thank all the users in the iPerf 2 and OpenWrt forums, which provided a valuable mean of getting in contact with other people using and developing networking systems, together with Dr. Toke Høiland-Jørgensen, from *Karlstad University*, who is one of the developers of the *ath9k* patch for the *mac80211* intermediate software queues, with whom we exchanged some e-mails related to this, giving me very useful advices.

Thanks a lot to Prof. Claudio Ettore Casetti and Dr. Marco Malinverno, who, as thesis advisors, were always very kind and helped me with all this work, discussing together the results, the possible issues and the next goals to try to achieve. It is also thanks to them that I discovered the world of VANETs, which I find very interesting and stimulating.

Thanks to my family and my brother, who supported me a lot, also economically, and always trusted me during these five years, which have been a very important part of my life.

Thanks to all my friends, who really supported and helped me during both the bachelor and master degrees, in *Politecnico di Torino*.

Appendices

Appendix A

C programs for broadcast transmissions

rawsock library

Rawsock_lib/rawsock.h

```
1 #ifndef RAWSOCK_H_INCLUDED
2 #define RAWSOCK_H_INCLUDED
3
4 #include <net/ethernet.h>
5 #include <linux/udp.h>
6 #include <linux/ip.h>
7 #include <arpa/inet.h>
8 #include <stdio.h>
9 #include <stdbool.h>
10 #define MAC_FILE_PATH_SIZE 23
11 #define MAC_ADDR_SIZE 6
12
13 // Errors
14 #define ERR_WLAN_NOIF 0 // No WLAN interfaces
15 #define ERR_WLAN_SOCKET -1 // socket() creation error
16 #define ERR_WLAN_GETIFADDRS -2 // getifaddrs() error
17 #define ERR_WLAN_INDEX -3 // Wrong index specified
18 #define ERR_WLAN_GETSRCMAC -4 // Unable to get source MAC address (if
    requested)
```

```
19 #define ERR_WLAN_GETIFINDEX -5 // Unable to get source interface index
    (if requested)
20
21 #define ERR_IPHEADP SOCK -10 // socket() creation error
22 #define ERR_IPHEAD_NOSRCADDR -11 // Unable to retrieve current device
    IP address
23
24 // Names
25 #define MAC_NULL 0x00
26 #define MAC_BROADCAST 0x01
27 #define MAC_UNICAST 0x02
28 #define MAC_MULTICAST 0x03
29
30 // Useful constants
31 // Additional EtherTypes
32 #define ETHERTYPE_GEONET 0x8947
33 #define ETHERTYPE_WSMP 0x88DC
34
35 // IP constants
36 #define BASIC_IHL 5
37 #define IPV4 4
38 #define BASIC_UDP_TTL 64 // From Wireshark captures
39
40 // UDP constant
41 #define UDPHEADERLEN 8
42
43 // Useful masks
44 #define FLAG_NOFRAG_MASK (1<<6)
45 #define FLAG_RESERVED_MASK (1<<7)
46 #define FLAG_MOREFRAG_MASK (1<<5)
47
48 // Checksum protocols, to be used inside the validateCsum() function
49 // (0x00->0x7F should be simple types, 0x80->0xFF should be combined
    types)
```

```
50 #define CSUM_IP 0x00
51 #define CSUM_UDP 0x01
52 #define CSUM_UDPIP 0x80
53
54 // Size definitions (macros)
55 #define UDP_PACKET_SIZE(data) sizeof(struct udphdr)+strlen(argv[5])
56 #define IP_UDP_PACKET_SIZE(data) sizeof(struct iphdr)+sizeof(struct
    udphdr)+strlen(argv[5])
57 #define ETH_IP_UDP_PACKET_SIZE(data) sizeof(struct iphdr)+sizeof(
    struct iphdr)+sizeof(struct udphdr)+strlen(argv[5])
58
59 typedef unsigned char * macaddr_t;
60 typedef unsigned short ethertype_t;
61 typedef unsigned char byte_t;
62 typedef int rawsockerr_t;
63 typedef unsigned char csumt_t;
64 typedef __sum16 csum16_t;
65 struct ipaddrs {
66     in_addr_t src;
67     in_addr_t dst;
68 };
69
70 // General utilities
71 rawsockerr_t wlanLookup(char *devname, int *ifindex, macaddr_t mac,
    unsigned int index);
72 macaddr_t prepareMacAddrT();
73 unsigned int macAddrTypeGet(macaddr_t mac);
74 void freeMacAddrT(macaddr_t mac);
75 void rs_prnterror(FILE *stream, rawsockerr_t code);
76 void display_packet(const char *text, byte_t *packet, unsigned int len);
77 void display_packetc(const char *text, byte_t *packet, unsigned int len)
    ;
78
79 // Ethernet level functions
```

```
80 void etherheadPopulateB(struct ether_header *etherHeader, macaddr_t
    mac, ethertype_t type);
81 void etherheadPopulate(struct ether_header *etherHeader, macaddr_t
    macsrc, macaddr_t macdst, ethertype_t type);
82 size_t etherEncapsulate(byte_t *packet, struct ether_header *header,
    byte_t *sdu, size_t sdusize);
83
84 // IP level functions
85 rawsockerr_t IP4headPopulateB(struct iphdr *IPhead, char *devname,
    unsigned char tos, unsigned short frag_offset, unsigned char ttl,
    unsigned char protocol, unsigned int flags, struct ipaddrs *addrs);
86 rawsockerr_t IP4headPopulate(struct iphdr *IPhead, char *devname, char
    *destIP, unsigned char tos, unsigned short frag_offset, unsigned
    char ttl, unsigned char protocol, unsigned int flags, struct ipaddrs
    *addrs);
87 void IP4headAddID(struct iphdr *IPhead, unsigned short id);
88 void IP4headAddTotLen(struct iphdr *IPhead, unsigned short len);
89 size_t IP4Encapsulate(byte_t *packet, struct iphdr *header, byte_t *sdu,
    size_t sdusize);
90
91 // UDP level functions
92 void UDPheadPopulate(struct udphdr *UDPhead, unsigned short sourceport
    , unsigned short destport);
93 size_t UDPencapsulate(byte_t *packet, struct udphdr *header, char *data,
    size_t payloadsize, struct ipaddrs addrs);
94
95 // Receiving device functions
96 byte_t *UDPgetpacketpointers(byte_t *pktbuf, struct ether_header **
    etherHeader, struct iphdr **IPheader, struct udphdr **UDPheader);
97 unsigned short UDPgetpayloadsize(struct udphdr *UDPheader);
98 bool validateEthCsum(byte_t *packet, csum16_t csum, csum16_t *
    combinedcsum, csumt_t type, void *args);
99
```

```
100 // Test functions, to inject errors inside packets - should never be
    used under normal circumstances
101 void test_injectIPChecksumError(byte_t *IPpacket);
102 void test_injectUDPCsumError(byte_t *UDPpacket);
103
104 #endif
```

Rawsock_lib/rawsock.c

```
1  #include "rawsock.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <ifaddrs.h>
5  #include <string.h>
6  #include <unistd.h>
7  #include <linux/wireless.h>
8  #include <sys/types.h>
9  #include <sys/ioctl.h>
10 #include <arpa/inet.h>
11 #include "ipcsun_alth.h"
12
13 /**
14  Fast UDP checksum calculation -
15  not an original work: coded by Andrea Righi in
16  the Minirighi IA-32 Operating System,
17  released under GNU GPL
18  Definition at line 29 of file udp.c
19 */
20 uint16_t udp_checksum(const void *buff, size_t len, in_addr_t src_addr
    , in_addr_t dest_addr) {
21     const uint16_t *buf=buff;
22     uint16_t *ip_src=(void *)&src_addr, *ip_dst=(void *)&dest_addr;
23     uint32_t sum;
24     size_t length=len;
25
26     sum = 0;
27     while (len > 1) {
28         sum += *buf++;
29         if (sum & 0x80000000)
30             sum = (sum & 0xFFFF) + (sum >> 16);
31         len -= 2;
```

```
32     }
33
34     if ( len & 1 )
35         sum += *((uint8_t *)buf);
36
37     sum += *(ip_src++);
38     sum += *ip_src;
39     sum += *(ip_dst++);
40     sum += *ip_dst;
41
42     sum += htons(IPPROTO_UDP);
43     sum += htons(length);
44
45     while (sum >> 16)
46         sum = (sum & 0xFFFF) + (sum >> 16);
47
48     return ( (uint16_t)(~sum) );
49 }
50
51 /**
52  * Prepare a macaddr_t variable -
53  * allocates a six element byte array
54  * to store any MAC address
55  */
56 macaddr_t prepareMacAddrT() {
57     macaddr_t mac;
58     int i;
59
60     mac=malloc(MAC_ADDR_SIZE*sizeof(unsigned char));
61
62     for(i=0;i<MAC_ADDR_SIZE;i++) {
63         mac[i]=0xFF;
64     }
65 }
```



```
66     return mac;
67 }
68
69 /**
70     Free a macaddr_t variable -
71     frees a previously allocated array
72 */
73 void freeMacAddrT(macaddr_t mac) {
74     free(mac);
75 }
76
77 /**
78     Get the MAC address type -
79     starting from a MAC address type
80     its type is returned (unicast, multicast, broadcast)
81 */
82 unsigned int macAddrTypeGet(macaddr_t mac) {
83     if(mac!=NULL) {
84         if(mac[0]==0x01) {
85             return MAC_MULTICAST;
86         } else if(mac[0]==0xFF && mac[1]==0xFF && mac[2]==0xFF && mac
87             [3]==0xFF && mac[4]==0xFF && mac[5]==0xFF) {
88             return MAC_BROADCAST;
89         } else {
90             return MAC_UNICAST;
91         }
92     } else {
93         return MAC_NULL;
94     }
95 }
96 /**
97     wlanLookup -
98     automatically look for available WLAN interfaces.
```

```
99  When only one interface is available and "0" is specified as index,
100  that interface name is returned inside "devname".
101  Then, if the other two arguments are not NULL, the interface index
    and
102  the corresponding source MAC address (if available) is returned.
103
104  If more than one interface is present, the number of available
    interfaces
105  is returned by the function and the index is used to point to a
    specific interface
106  (for instance index=1 can be used to point to a possible "wlan1"
    interface).
107
108  Return values: if ok: number of found interfaces
109  ERR_WLAN_NOIF -> no interfaces found
110  ERR_WLAN_SOCK -> cannot create socket to look for wireless
    interfaces
111  ERR_WLAN_GETIFADDRS -> error in calling getifaddrs()
112  ERR_WLAN_INDEX -> invalid index value
113  ERR_WLAN_GETSRCMAC -> unable to get source MAC address (if requested
    )
114  ERR_WLAN_GETIFINDEX -> unable to get source interface index (if
    requested)
115  **/
116  rawsockerr_t wlanLookup(char *devname, int *ifindex, macaddr_t mac,
    unsigned int index) {
117  // Variables for wlan interfaces detection
118  int sFd=-1;
119  // struct ifaddrs used to look for available interfaces and bind to
    a wireless interface
120  struct ifaddrs *ifaddr_head, *ifaddr_it;
121  // struct ifreq to check whether an interface is wireless or not.
    The ifr_name field is used to specify which device to affect.
122  struct ifreq wifireq;
```

```
123 // Pointers to manage the list containing all valid wireless
    interfaces
124 struct iflist *iflist_head=NULL; // Head
125 struct iflist *curr_ptr=NULL; // Current element
126 struct iflist *iflist_it=NULL; // To iterate the list
127 struct iflist *iflist_u=NULL; // To free the list
128 int ifno=0;
129 int return_value=1; // Return value: >0 ok - # of found interfaces,
    <=0 error
130 // (=0 for no WLAN interfaces, =-1 for socket error,
    =-2 for getifaddrs error)
131 // (= -3 for wrong index, =-4 unable to get MAC
    address, =-5 unable to get ifindex)
132
133 // Linked list nodes to store the WLAN interfaces
134 struct iflist {
135     struct iflist *next;
136     struct ifaddrs *ifaddr_ptr;
137 };
138
139 // Open socket (needed)
140 sFd=socket(AF_INET,SOCK_DGRAM,0); // Any socket should be fine (to
    be better investigated!)
141 if(sFd==-1) {
142     return_value=-1;
143     goto sock_error_occurred;
144 }
145
146 // Getting all interface addresses
147 if(getifaddrs(&ifaddr_head)==-1) {
148     return_value=-2;
149     goto getifaddrs_error_occurred;
150 }
151
```

```
152 // Looking for wlan interfaces
153 bzero(&wifireq,sizeof(wifireq));
154 // Iterating over the interfaces linked list
155 for(ifaddr_it=ifaddr_head;ifaddr_it!=NULL;ifaddr_it=ifaddr_it->
    ifa_next) {
156     if(ifaddr_it->ifa_addr!=NULL && ifaddr_it->ifa_addr->sa_family ==
        AF_PACKET) {
157         // fprintf(stdout,"Checking interface %s for use.\n",ifaddr_it->
            ifa_name);
158         // IFNAMSIZ is defined by system libraries and it "defines the
            maximum buffer size needed to hold an interface name,
159         // including its terminating zero byte"
160         // This is done because (from man7.org) "normally, the user
            specifies which device to affect by setting
161         // ifr_name to the name of the interface"
162         strncpy(wifireq.ifr_name,ifaddr_it->ifa_name,IFNAMSIZ);
163
164         // Trying to get the Wireless Extensions (a socket descriptor
            must be specified to ioctl())
165         if(ioctl(sFd,SIOCGIWNAME,&wifireq)!=-1) {
166             // Check if the interface is up
167             if(ioctl(sFd,SIOCGIFFLAGS,&wifireq)!=-1 && (wifireq.ifr_flags
                & IFF_UP)) {
168                 // If the interface is up, add it to the head of the "iflist
                    " (it is not added to the tail in order to
169                 // avoid defining an extra pointer)
170                 // fprintf(stdout,"Interface %s (%#d) is up. It may be used
                    .\n",ifaddr_it->ifa_name,ifno);
171                 ifno++;
172                 curr_ptr=malloc(sizeof(struct iflist));
173                 curr_ptr->ifaddr_ptr=ifaddr_it;
174                 if(iflist_head==NULL) {
175                     iflist_head=curr_ptr;
176                     iflist_head->next=NULL;
```

```
177         } else {
178             curr_ptr->next=iflist_head;
179             iflist_head=curr_ptr;
180         }
181     } // else {
182         // fprintf(stdout,"Interface is not up (or it is impossible
183             // to check that).\n");
184     } // else {
185         // fprintf(stdout,"Interface is not wireless.\n");
186     } // }
187 }
188 }
189
190 // No wireless interfaces found (the list is empty)
191 if(iflist_head==NULL) {
192     // fprintf(stderr,"No wireless interfaces found. The program will
193         // terminate now.\n");
194     return_value=0;
195     goto error_occurred;
196 } else if(ifno==1) {
197     // Only one wireless interface found -> it is possible to ignore '
198         // index'
199     strncpy(devname,iflist_head->ifaddr_ptr->ifa_name,IFNAMSIZ);
200     // fprintf(stdout,"Interface %s (#0) will be used.\n\n",devname);
201 } else {
202     // Multiple wireless interfaces found -> use the value of 'index'
203     // fprintf(stdout,"Please insert the interface # to be used: ");
204     // fflush(stdout);
205     // fscanf(stdin,"%d",&ifindex);
206     if(index>=ifno) {
207         //fprintf(stderr,"Invalid interface index. Aborting execution.\n
208             ");
209     }
210     return_value=-3;
```

```
207     goto error_occurred;
208 }
209 return_value=ifno; // Return the number of interfaces found
210 // Iterate the list until the chosen interface is reached
211 iflist_it=iflist_head;
212 while(index<=ifno-2) {
213     iflist_it=iflist_it->next;
214     index++;
215 }
216 strncpy(devname,iflist_it->ifaddr_ptr->ifa_name,IFNAMSIZ);
217 // fprintf(stdout,"Interface %s will be used.\n\n",devname);
218 }
219
220 // Get MAC address of the interface (if requested by the user with a
    non-NULL mac)
221 if(mac!=NULL) {
222     strncpy(wifireq.ifr_name,devname,IFNAMSIZ);
223     if(ioctl(sFd,SIOCGIFHWADDR,&wifireq)!=-1) {
224         memcpy(mac,wifireq.ifr_hwaddr.sa_data,MAC_ADDR_SIZE);
225     } else {
226         return_value=-4;
227         goto error_occurred;
228     }
229 }
230
231 // Get interface index of the interface (if requested by the user
    with a non-NULL ifindex)
232 if(ifindex!=NULL) {
233     strncpy(wifireq.ifr_name,devname,IFNAMSIZ);
234     if(ioctl(sFd,SIOCGIFINDEX,&wifireq)!=-1) {
235         *ifindex=wifireq.ifr_ifindex;
236     } else {
237         return_value=-5;
238         goto error_occurred;
```

```
239     }
240 }
241
242 error_occurred:
243 // iflist and the other list are no more useful -> free them
244 freeifaddrs(ifaddr_head);
245 for(iflist_it=iflist_head;iflist_it!=NULL;iflist_it=iflist_u) {
246     iflist_u=iflist_it->next;
247     free(iflist_it);
248 }
249
250 getifaddrs_error_occurred:
251 // Close socket
252 close(sFd);
253
254 sock_error_occurred:
255 return return_value;
256 }
257
258 /**
259  Print more detailed error messages -
260  given a stream (for ex. stdout or stderr) and
261  the error name (see rawsock.h)
262 */
263 void rs_prnterror(FILE *stream,rawsockerr_t code) {
264     switch(code) {
265         case ERR_WLAN_NOIF:
266             fprintf(stream,"wlanLookup: No WLAN interfaces found.\n");
267             break;
268
269         case ERR_WLAN_SOCK:
270             fprintf(stream,"wlanLookup: socket creation error.\n");
271             break;
272     }
```

```
273     case ERR_WLAN_GETIFADDRS:
274         fprintf(stream, "wlanLookup: getifaddrs() error.\n");
275     break;
276
277     case ERR_WLAN_INDEX:
278         fprintf(stream, "wlanLookup: wrong index specified.\n");
279     break;
280
281     case ERR_WLAN_GETSRCMAC:
282         fprintf(stream, "wlanLookup: unable to get source MAC address.\n"
283             );
284     break;
285
286     case ERR_WLAN_GETIFINDEX:
287         fprintf(stream, "wlanLookup: unable to get interface index.\n");
288     break;
289
290     case ERR_IPHEADP SOCK:
291         fprintf(stream, "IP4headPopulateB: socket creation error.\n");
292     break;
293
294     case ERR_IPHEAD_NOSRCADDR:
295         fprintf(stream, "IP4headPopulateB: unable to retrieve source IP
296             address.\n");
297     break;
298
299     default:
300         fprintf(stream, "Unknown error.\n");
301     }
302 }
303
304 /**
305  * Display packet in hexadecimal form -
306  * packet length is required
```



```
305 */
306 void display_packet(const char *text,byte_t *packet,unsigned int len)
307 {
308
309     fprintf(stdout,"%s -> ",text);
310     for(i=0;i<len;i++) {
311         fprintf(stdout,"%02x ",packet[i]);
312     }
313     fprintf(stdout,"\n");
314     fflush(stdout);
315 }
316
317 /**
318     Display packet in character form -
319     packet length is required
320 */
321 void display_packetc(const char *text,byte_t *packet,unsigned int len)
322 {
323
324     fprintf(stdout,"%s -> ",text);
325     for(i=0;i<len;i++) {
326         fprintf(stdout,"%c",packet[i]);
327     }
328     fprintf(stdout,"\n");
329     fflush(stdout);
330 }
331
332 /**
333     Populate broadcast Ethernet header -
334     the user shall specify an already existing ether_header structure,
335     the source MAC
```

```
335     and the Ethertype (either one type in net/ethernet.h or one type in
        rawsock.h)
336 */
337 void etherheadPopulateB(struct ether_header *etherHeader, macaddr_t
        mac, ethertype_t type) {
338     unsigned char broadcastMAC[ETHER_ADDR_LEN]={0xFF,0xFF,0xFF,0xFF,0xFF
        ,0xFF};
339
340     memcpy(etherHeader->ether_dhost,broadcastMAC,ETHER_ADDR_LEN);
341     memcpy(etherHeader->ether_shost,mac,ETHER_ADDR_LEN);
342     etherHeader->ether_type = htons(type);
343 }
344
345 /**
346     Populate standard Ethernet header -
347     the user shall specify an already existing ether_header structure,
        the source MAC
348     the destination MAC and the Ethertype (either one type in net/
        ethernet.h or one type in rawsock.h)
349 */
350 void etherheadPopulate(struct ether_header *etherHeader, macaddr_t
        macsrc, macaddr_t macdst, ethertype_t type) {
351     memcpy(etherHeader->ether_dhost,macdst,ETHER_ADDR_LEN);
352     memcpy(etherHeader->ether_shost,macsrc,ETHER_ADDR_LEN);
353     etherHeader->ether_type = htons(type);
354 }
355
356 /**
357     Combine Ethernet SDU and PCI -
358     the user shall specify a buffer in which the full packet will be put
        , the ethernet header, the SDU
359     (byte_t *sdu) and its size in bytes
360 */
```

```
361 size_t etherEncapsulate(byte_t *packet, struct ether_header *header,
    byte_t *sdu, size_t sdusize) {
362     size_t packetsize = sizeof(struct ether_header) + sdusize;
363
364     memcpy(packet, header, sizeof(struct ether_header));
365     memcpy(packet + sizeof(struct ether_header), sdu, packetsize);
366
367     return packetsize;
368 }
369
370 /**
371     Populate standard IP version 4 header -
372     the user shall specify an already existing IP header structure, the
        interface name (for instance "wlan0"),
373     the destination IP address, the TOS value, the fragment offset value
        , the TTL, the protocol, the flags (reserved, DF, MF).
374     The function, other than returning possible errors (ERR_IPHEADP SOCK
        or ERR_IPHEAD_NOSRCADDR), returns a structure
375     containing the source and destination IP addresses, if struct
        ipaddrs *addrs is not NULL.
376 */
377 rawsockerr_t IP4headPopulate(struct iphdr *IPhead, char *devname, char
    *destIP, unsigned char tos, unsigned short frag_offset, unsigned
    char ttl, unsigned char protocol, unsigned int flags, struct ipaddrs
    *addrs) {
378     struct in_addr destIPAddr;
379     int sFd; // To get the current IP address
380     struct ifreq wifireq;
381
382     IPhead->ihl = BASIC_IHL;
383     IPhead->version = IPV4;
384     IPhead->tos = (__u8) tos;
385     IPhead->frag_off = htons(frag_offset);
386     IPhead->frag_off = (IPhead->frag_off) | flags;
```

```
387 IPhead->ttl((__u8) ttl);
388 IPhead->protocol((__u8) protocol);
389 inet_pton(AF_INET,destIP,(struct in_addr *)&destIPAddr);
390 IPhead->daddr=destIPAddr.s_addr;
391
392 // Get own IP address
393 sFd=socket(AF_INET,SOCK_DGRAM,0);
394 if(sFd==-1) {
395     return -10;
396 }
397 strncpy(wifireq.ifr_name,devname,IFNAMSIZ);
398 wifireq.ifr_addr.sa_family = AF_INET;
399 if(ioctl(sFd,SIOCGIFADDR,&wifireq)!=0) {
400     close(sFd);
401     return -11;
402 }
403 close(sFd);
404 IPhead->saddr=((struct sockaddr_in*)&wifireq.ifr_addr)->sin_addr.
    s_addr;
405 if(addr!=NULL) {
406     addr->src=IPhead->saddr;
407     addr->dst=IPhead->daddr;
408 }
409
410 // Initialize checksum to 0
411 IPhead->check=0;
412
413 return 0;
414 }
415
416 /**
417  Populate broadcast IP version 4 header -
418  the user shall specify an already existing IP header structure, the
    interface name (for instance "wlan0"),
```

```
419 the TOS value, the fragment offset value, the TTL, the protocol, the
    flags (reserved, DF, MF).
420 The function, other than returning possible errors (ERR_IPHEADP SOCK
    or ERR_IPHEAD_NOSRCADDR), returns a structure
421 containing the source and destination IP addresses, if struct
    ipaddrs *addrs is not NULL.
422 **/
423 rawsockerr_t IP4headPopulateB(struct iphdr *IPhead, char *devname,
    unsigned char tos, unsigned short frag_offset, unsigned char ttl,
    unsigned char protocol, unsigned int flags, struct ipaddrs *addrs) {
424 struct in_addr broadIPAddr;
425 int sFd; // To get the current IP address
426 struct ifreq wifireq;
427
428 IPhead->ihl=BASIC_IHL;
429 IPhead->version=IPV4;
430 IPhead->tos((__u8) tos);
431 IPhead->frag_off=htons(frag_offset);
432 IPhead->frag_off=(IPhead->frag_off) | flags;
433 IPhead->ttl((__u8) ttl);
434 IPhead->protocol((__u8) protocol);
435 inet_pton(AF_INET, "255.255.255.255", (struct in_addr *)&broadIPAddr);
436 IPhead->daddr=broadIPAddr.s_addr;
437
438 // Get own IP address
439 sFd=socket(AF_INET, SOCK_DGRAM, 0);
440 if(sFd==-1) {
441     return -10;
442 }
443 strncpy(wifireq.ifr_name, devname, IFNAMSIZ);
444 wifireq.ifr_addr.sa_family = AF_INET;
445 if(ioctl(sFd, SIOCGIFADDR, &wifireq)!=0) {
446     close(sFd);
447     return -11;
```

```
448     }
449     close(sFd);
450     IPhead->saddr=((struct sockaddr_in*)&wifireq.ifr_addr)->sin_addr.
         s_addr;
451     if(addr != NULL) {
452         addr->src=IPhead->saddr;
453         addr->dst=IPhead->daddr;
454     }
455
456     // Initialize checksum to 0
457     IPhead->check=0;
458
459     return 0;
460 }
461
462 /**
463  * Add ID to a given (by the caller) IPv4 header
464  */
465 void IP4headAddID(struct iphdr *IPhead, unsigned short id) {
466     IPhead->id=htons(id);
467 }
468
469 /**
470  * Add Total Length to a given (by the caller) IPv4 header
471  */
472 void IP4headAddTotLen(struct iphdr *IPhead, unsigned short len) {
473     IPhead->tot_len=htons(len);
474 }
475
476 /**
477  * Combine IPv4 SDU and PCI -
478  * the user shall specify a buffer in which the full packet will be put
479  * , the ethernet header, the SDU
480  * (byte_t *sdu) and its size in bytes
```

```
480 /**/
481 size_t IP4Encapsulate(byte_t *packet, struct iphdr *header, byte_t *sdu,
482     size_t sdusize) {
483
484     header->tot_len=htons(packetsize);
485     header->check=0; // Reset to 0 in case of subsequent calls
486
487     header->check=ip_fast_csum((__u8 *)header, BASIC_IHL);
488
489     memcpy(packet, header, sizeof(struct iphdr));
490     memcpy(packet+sizeof(struct iphdr), sdu, packetsize);
491
492     return packetsize;
493 }
494
495 /**
496     Populate standard UDP 4 header -
497     the user shall specify an already existing UDP header structure, the
498     source port
499     and the destination port
500 */
501 void UDPheadPopulate(struct udphdr *UDPhead, unsigned short sourceport
502     , unsigned short destport) {
503
504     UDPhead->source=htons(sourceport);
505     UDPhead->dest=htons(destport);
506
507     // Initialize checksum to 0
508     UDPhead->check=0;
509 }
510
511 /**
512     Combine UDP payload and header -
```

```
510 the user shall specify a buffer in which the full packet will be put
    , the UDP header, the payload
511 (byte_t *payload), its size in bytes and a structure (see the
    definition in rawsock.h) containing
512 source and destion IP addresses, which are used to compute the
    checksum
513 */
514 size_t UDPencapsulate(byte_t *packet, struct udphdr *header, char *data,
    size_t payloadsize, struct ipaddrs addrs) {
515     size_t packetsize=sizeof(struct udphdr)+payloadsize;
516
517     header->len=htons(packetsize);
518     header->check=0; // Reset to 0 in case of subsequent calls
519
520     memcpy(packet, header, sizeof(struct udphdr));
521     memcpy(packet+sizeof(struct udphdr), data, packetsize);
522
523     header->check=udp_checksum(packet, packetsize, addrs.src, addrs.dst);
524
525     memcpy(packet, header, sizeof(struct udphdr));
526
527     return packetsize;
528 }
529
530 /**
531     Get pointers to headers and payload in UDP packet buffer -
532     obtain, given a certain buffer containing an UDP packet, the pointer
        to the headers
533     and payload sections
534     Example of call: payload=UDPgetpacketpointers(packet, &etherHeader, &
        IPheader, &udpHeader);
535     with:
536     struct ether_header* etherHeader;
537     struct iphdr *IPheader;
```



```
538 struct udphdr *udpHeader;
539 byte_t *payload;
540 -----
541 packet is specified by the user,
542 payload is returned by the function (requires an already allocated
    array!)
543 the ethernet header pointer is written by this function, as the IP
    and UDP header pointers
544 **/
545 byte_t *UDPgetpacketpointers(byte_t *pktbuf, struct ether_header **
    etherHeader, struct iphdr **IPheader, struct udphdr **UDPheader) {
546     byte_t *payload;
547
548     *etherHeader=(struct ether_header*) pktbuf;
549     *IPheader=(struct iphdr*)(pktbuf+sizeof(struct ether_header));
550     *UDPheader=(struct udphdr*)(pktbuf+sizeof(struct ether_header)+
        sizeof(struct iphdr));
551     payload=(pktbuf+sizeof(struct ether_header)+sizeof(struct iphdr)+
        sizeof(struct udphdr));
552
553     return payload;
554 }
555
556 /**
557     Get UDP payload size as unsigned short (int), given a UDP header
        pointer
558 **/
559 unsigned short UDPgetpayloadsize(struct udphdr *UDPheader) {
560     return (ntohs(UDPheader->len)-UDPHEADERLEN);
561 }
562
563 /**
564     Validate the checksum of a raw 'Ethernet' packet i.e.
565     of any packet containing a struct ether_header as first bytes -
```

```
566 Arguments:
567 . byte_t *packet -> pointer to the full packet buffer
568 . csum16_t csum -> checksum value to be checked against the newly
    computed value, from 'packet'
569 . csum16_t *combinedcsum -> should be NULL for non combined checksum
    types, otherwise it should contain
570 the value of the second checksum to be checked (the ordering between
    csum and *combinedcsum is the same
571 as the one in the checksum type constant - for instance: CSUM_UDPIP
    requires csum<-UDP and *combinedcsum<-IP)
572 If NULL is specified for any combined type, the function will always
    return 'false'.
573 . csumt_t type -> checksum protocol: various protocols can be
    specified within the same function, in order to keep
574 it easier to extend as newer protocols will be implemented inside
    the library.
575 The supported protocols are defined inside proper "Checksum
    protocols" constants in rawsock.h.
576 Both simple protocols and combined ones are supported: in the first
    case, only the checksum related to the
577 specified protocol is checked (csum16_t csum), in the second case,
    two checksums, contained inside the same
578 packet, are checked (csum16_t csum and csum16_t *combinedcsum) ->
    this can be useful, for instance, to
579 check both the UDP and IP checksums all at once; in case a combined
    mode is selected 'combinedcsum' shall
580 be non-NULL, otherwise the function will always return 'false'.
581 . void *args -> additional arguments: these are typically dependant
    on the specified protocol: for instance,
582 using IP, they are not needed and NULL can be passed; instead, when
    using UDP (or UDP+IP), they shall contain
583 the pointer to a single value which is the payload length (this may
    avoid computing it multiple times, outside
```

```
584     and inside this function, when a variable containing it is already
        available)
585 */
586 bool validateEthCsum(byte_t *packet, csum16_t csum, csum16_t *
        combinedcsum, csumt_t type, void *args) {
587     csum16_t currCsum;
588     bool returnVal=false;
589     void *headerPtr; // Generic header pointer
590     void *payloadPtr; // Generic payload/SDU pointer
591     size_t packetsize; // Used in UDP checksum calculation
592     __sum16 storedCsum; // To store the current value of checksum, read
        from 'packet'
593
594     // Directly return 'false' (as an error occurred) if a combined type
        is specified but combinedcsum is NULL
595     if(type>=0x80 && combinedcsum==NULL) {
596         return false;
597     }
598
599     // Discriminate the different protocols
600     switch(type) {
601         case CSUM_IP:
602             headerPtr=(struct iphdr*)(packet+sizeof(struct ether_header));
603
604             // Checksum should start with a value of 0x0000 in order to be
                correctly computed:
605             // set it to 0 and restore it, to avoid making a copy of the
                packet in memory
606             storedCsum=((struct iphdr *) headerPtr)->check;
607             ((struct iphdr *) headerPtr)->check=0;
608
609             currCsum=ip_fast_csum((__u8 *)headerPtr, ((struct iphdr *)
                headerPtr)->ihl);
610
```

```
611     ((struct iphdr *) headerPtr)->check=storedCsum;
612
613     returnVal=(currCsum==csum);
614     break;
615     case CSUM_UDP:
616     case CSUM_UDPIP:
617         // payloadsize should be specified, otherwise 'false' will be
           always returneds
618         if(args==NULL) {
619             returnVal=false;
620         } else {
621             // Get packetsize
622             packetsize=sizeof(struct udphdr)+*((size_t *) args);
623
624             headerPtr=(struct iphdr*)(packet+sizeof(struct ether_header));
625             payloadPtr=(struct udphdr*)(packet+sizeof(struct ether_header)
               +sizeof(struct iphdr));
626
627             storedCsum=((struct udphdr *) payloadPtr)->check;
628             ((struct udphdr *) payloadPtr)->check=0;
629
630             currCsum=udp_checksum(payloadPtr,packetsize,((struct iphdr *)
               headerPtr)->saddr,((struct iphdr *)headerPtr)->daddr);
631
632             ((struct udphdr *) payloadPtr)->check=storedCsum;
633
634             returnVal=(currCsum==csum);
635
636             if(type==CSUM_UDPIP) {
637                 if(combinedcsum==NULL) {
638                     returnVal=false;
639                 } else {
640                     // Compute IP checksum
641                     storedCsum=((struct iphdr *) headerPtr)->check;
```

```
642         ((struct iphdr *) headerPtr)->check=0;
643
644         currCsum=ip_fast_csum((__u8 *)headerPtr, ((struct iphdr *)
        headerPtr)->ihl);
645
646         ((struct iphdr *) headerPtr)->check=storedCsum;
647
648         returnVal=returnVal && (currCsum==(*combinedcsum));
649     }
650 }
651 }
652 break;
653 default:
654     returnVal=false;
655 }
656
657 return returnVal;
658 }
659
660 /**
661  Test function: inject a checksum error in an IP packet -
662  The pointer to the full IP packet shall be passed (IP header+payload
663  )
664  */
665 void test_injectIPCsumError(byte_t *IPpacket) {
666     // Get header pointer
667     struct iphdr *IPheader=(struct iphdr *) IPpacket;
668
669     if(IPpacket!=NULL) {
670         // Change checksum, avoiding possible overflow situations
671         if(IPheader->check!=0xFF) {
672             IPheader->check=IPheader->check+1;
673         } else {
674             IPheader->check=0x00;
```

```
674     }
675 }
676 }
677
678 /**
679  Test function: inject a checksum error in an UDP packet -
680  The pointer to the full UDP packet shall be passed (UDP header+
681  payload)
682  */
683 void test_injectUDPCsumError(byte_t *UDPpacket) {
684     // Get header pointer
685     struct udphdr *UDPheader=(struct udphdr *) UDPpacket;
686
687     if(UDPheader!=NULL) {
688         // Change checksum, avoiding possible overflow situations
689         if(UDPheader->check!=0xFF) {
690             UDPheader->check=UDPheader->check+1;
691         } else {
692             UDPheader->check=0x00;
693         }
694     }
695 }
```

Rawsock_lib/ipcsum_alth.h

```
1 #ifndef IPCSUM_ALTH_INCLUDED
2 #define IPCSUM_ALTH_INCLUDED
3
4 #include <linux/types.h>
5
6 // This is all taken from Linux kernel 4.19.1 (this is not original
   work)
7 __sum16 ip_fast_csum(const void *iph, unsigned int ihl);
8
9 #endif
```

Rawsock_lib/ipcsum_alth.c

```
1 #include "ipcsum_alth.h"
2
3 // This is all taken from Linux kernel 4.19.1 (this is not original
   work)
4 static inline unsigned short from64to16(unsigned long x)
5 {
6     /* Using extract instructions is a bit more efficient
7        than the original shift/bitmask version.  */
8
9     union {
10         unsigned long ul;
11         unsigned int  ui[2];
12         unsigned short us[4];
13     } in_v, tmp_v, out_v;
14
15     in_v.ul = x;
16     tmp_v.ul = (unsigned long) in_v.ui[0] + (unsigned long) in_v.ui[1];
17
18     /* Since the bits of tmp_v.sh[3] are going to always be zero,
19        we don't have to bother to add that in.  */
20     out_v.ul = (unsigned long) tmp_v.us[0] + (unsigned long) tmp_v.us[1]
21         + (unsigned long) tmp_v.us[2];
22
23     /* Similarly, out_v.us[2] is always zero for the final add.  */
24     return out_v.us[0] + out_v.us[1];
25 }
26
27 static inline unsigned long do_csum(const unsigned char * buff, int
   len) {
28     int odd, count;
29     unsigned long result = 0;
30
```



```
31  if (len <= 0)
32      goto out;
33  odd = 1 & (unsigned long) buff;
34  if (odd) {
35      result = *buff << 8;
36      len--;
37      buff++;
38  }
39  count = len >> 1;    /* nr of 16-bit words.. */
40  if (count) {
41      if (2 & (unsigned long) buff) {
42          result += *(unsigned short *) buff;
43          count--;
44          len -= 2;
45          buff += 2;
46      }
47      count >>= 1;    /* nr of 32-bit words.. */
48      if (count) {
49          if (4 & (unsigned long) buff) {
50              result += *(unsigned int *) buff;
51              count--;
52              len -= 4;
53              buff += 4;
54          }
55          count >>= 1; /* nr of 64-bit words.. */
56          if (count) {
57              unsigned long carry = 0;
58              do {
59                  unsigned long w = *(unsigned long *) buff;
60                  count--;
61                  buff += 8;
62                  result += carry;
63                  result += w;
64                  carry = (w > result);
```

```
65     } while (count);
66     result += carry;
67     result = (result & 0xffffffff) + (result >> 32);
68 }
69 if (len & 4) {
70     result += *(unsigned int *) buff;
71     buff += 4;
72 }
73 }
74 if (len & 2) {
75     result += *(unsigned short *) buff;
76     buff += 2;
77 }
78 }
79 if (len & 1)
80     result += *buff;
81 result = from64to16(result);
82 if (odd)
83     result = ((result >> 8) & 0xff) | ((result & 0xff) << 8);
84 out:
85     return result;
86 }
87
88 __sum16 ip_fast_csum(const void *iph, unsigned int ihl)
89 {
90     return (__sum16)~do_csum(iph, ihl*4);
91 }
```

Sender program

broadcastSend.c

```
1 #include <sys/socket.h>
2 #include <sys/timerfd.h>
3 #include <sys/ioctl.h>
4 #include <linux/wireless.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include <unistd.h>
9 #include <time.h>
10 #include <math.h>
11 #include <poll.h>
12 #include "Rawsock_lib/rawsock.h"
13 #include <linux/if_packet.h>
14
15 #define MAX_LEN 1470 // Maximum allowed payload length
16
17 #define SEC_TO_NANOSEC 1000000000
18 #define INDEFINITE_BLOCK -1
19 #define NO_FLAGS 0
20 #define SRCPORT 46772
21 #define START_ID 11349
22 #define INCR_ID 0
23
24 // After how many packets should the program inject an IP checksum
    error, in test mode?
25 #define ERRORFREQ 10
26
27 int main (int argc, char **argv) {
28     int sFd;
29     int broadPerm=1;
```

```
30  int up;
31  int broadPort;
32  size_t len;
33  int clockFd;
34
35  // Timer management variables
36  struct itimerspec new_value;
37  long nanosec;
38  double d_sec;
39  double d_time;
40  struct pollfd timerMon;
41
42  // Junk variable (needed to clear the timer event with read())
43  unsigned long long junk;
44
45  // wlanLookup() variables, for interface name, source MAC address
    and return value
46  char devname[IFNAMSIZ]={0}; // It will contain the used interface
    name
47  macaddr_t srcmacaddr=prepareMacAddrT();
48  int ret_wlanl_val;
49
50  // Ethernet header and packet container
51  struct ether_header etherHeader;
52  byte_t *ethernetpacket;
53
54  // IP header
55  struct iphdr ipHeader;
56  // IP address (src+dest) structure
57  struct ipaddrs ipaddrs;
58  // IP packet container
59  byte_t *ippacket;
60  // id to be inserted in the id field on the IP header
61  unsigned int id=START_ID;
```

```
62
63 // UDP header
64 struct udphdr udpHeader;
65 // UDP packet container
66 byte_t *udppacket;
67
68 // sockaddr_ll (device-independent physical-layer address)
69 struct sockaddr_ll addrll;
70 // Index of the interface which is used (and returned by wlanLookup
    ())
71 int ifindex;
72
73 // Final packet size
74 size_t finalpktsize;
75
76 // Check command line arguments
77 if(argc!=6) {
78     fprintf(stderr,"Error. Expected five parameters.\nCorrect usage:
        <%s> <broadcast port> <broadcast IP> <userpriority (0-7)> <
        time interval> <payload>.\n",argv[0]);
79     exit(EXIT_FAILURE);
80 }
81
82 // Get payload length
83 len=strlen(argv[5]);
84 if(len>MAX_LEN) {
85     fprintf(stderr,"Error. Payload string has length %zu, which is
        over the allowed maximum (%d)",len,MAX_LEN);
86     exit(EXIT_FAILURE);
87 }
88
89 // Open socket
90 sFd=socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL));
91 if(sFd==-1) {
```

```
92     perror("socket() error");
93     exit(EXIT_FAILURE);
94 }
95
96 // Look for available WLAN interfaces (wlan0 should be returned in
    the APU/ALIX boards)
97 ret_wlanl_val=wlanLookup(devname,&ifindex,srcmacaddr,0);
98 if(ret_wlanl_val<=0) {
99     fprintf(stderr,"wlanLookup() error.\n");
100    rs_pnterror(stderr,ret_wlanl_val);
101    close(sFd);
102    exit(EXIT_FAILURE);
103 }
104
105 // Check if wlanLookup() was able to properly write the source MAC
    address (which is normally initialized to broadcast)
106 if(macAddrTypeGet(srcmacaddr)==MAC_BROADCAST) {
107     fprintf(stderr,"Could not retrieve source MAC address.\n");
108     close(sFd);
109     exit(EXIT_FAILURE);
110 }
111
112 // Prepare sockaddr_ll structure
113 bzero(&addrll,sizeof(addrll));
114 addrll.sll_ifindex=ifindex;
115 addrll.sll_family=AF_PACKET;
116 addrll.sll_protocol=htons(ETH_P_ALL);
117
118 // Bind to the wireless interface
119 if(bind(sFd,(struct sockaddr *) &addrll,sizeof(addrll))<0) {
120     perror("Cannot bind to interface: bind() error");
121     close(sFd);
122     exit(EXIT_FAILURE);
123 }
```

```
124
125 // This works only with AF_INET sockets, so it should not be used
    here
126 // if(setsockopt(sFd,SOL_SOCKET,SO_BINDTODEVICE,devname,strlen(
    devname))== -1) {
127 // perror("setsockopt() for SO_BINDTODEVICE error");
128 // close(sFd);
129 // exit(EXIT_FAILURE);
130 // }
131
132 // Print AC corresponding to selected UP (and handle possible wrong
    UP)
133 up=atoi(argv[3]);
134 switch(up) {
135     case 0:
136     case 3:
137         fprintf(stdout,"AC_BE is selected, with UP: %d.\n",up);
138         break;
139
140     case 1:
141     case 2:
142         fprintf(stdout,"AC_BK is selected, with UP: %d.\n",up);
143         break;
144
145     case 4:
146     case 5:
147         fprintf(stdout,"AC_VI is selected, with UP: %d.\n",up);
148         break;
149
150     case 6:
151     case 7:
152         fprintf(stdout,"AC_VO is selected, with UP: %d.\n",up);
153         break;
154
```

```
155     default:
156         fprintf(stderr, "Wrong UP specified. AC_BE will be used.\n");
157         up=0;
158         break;
159     }
160
161     // Set user priority using socket layer options
162     if(setsockopt(sFd, SOL_SOCKET, SO_PRIORITY, &up, sizeof(up)) != 0) {
163         perror("setsockopt() for SO_PRIORITY error");
164         close(sFd);
165         exit(EXIT_FAILURE);
166     }
167
168     // Set broadcast permission using socket layer options (is this
169     // really needed here?)
170     if(setsockopt(sFd, SOL_SOCKET, SO_BROADCAST, (void *) &broadPerm, sizeof
171         (broadPerm)) != 0) {
172         perror("setsockopt() for SO_BROADCAST error");
173         close(sFd);
174         exit(EXIT_FAILURE);
175     }
176
177     broadPort=atoi(argv[1]); // Read port from command line
178
179     // Preparing headers (it can be done here since they won't change
180     // during the execution, in this specific case)
181     //unsigned char dstaddr[6]={0xD8,0x61,0x62,0x07,0x9D,0xA8}; <-
182     // example of use with non broadcast transmission
183     //etherheadPopulate(&etherHeader, srcmacaddr, dstaddr, ETHERTYPE_IP)
184     //; <- example of use with non broadcast transmission
185     etherheadPopulateB(&etherHeader, srcmacaddr, ETHERTYPE_IP);
186     IP4headPopulateB(&ipHeader, devname, 0, 0, BASIC_UDP_TTL,
187         IPPROTO_UDP, FLAG_NOFRAG_MASK, &ipaddrs);
```



```
182 //IP4headPopulate(&ipHeader, devname, "10.10.6.103", 0, 0,
    BASIC_UDP_TTL, IPPROTO_UDP, FLAG_NOFRAG_MASK, &ipaddrs); <-
    example of use with non broadcast transmission
183 UDPheadPopulate(&udpHeader, SRCPORT, broadPort);
184
185 // Allocating packet buffers
186 udppacket=malloc(UDP_PACKET_SIZE(argv[5]));
187 ippacket=malloc(IP_UDP_PACKET_SIZE(argv[5]));
188 ethernetpacket=malloc(ETH_IP_UDP_PACKET_SIZE(argv[5]));
189
190 // Create monotonic (increasing) timer
191 clockFd=timerfd_create(CLOCK_MONOTONIC,NO_FLAGS);
192 if(clockFd==-1) {
193     perror("timerfd_create() error");
194     close(sFd);
195     exit(EXIT_FAILURE);
196 }
197
198 // Get time from command line and convert it to sec and nanosec
199 d_time=strtod(argv[4],NULL);
200 if(d_time==0) {
201     perror("strtod() to get time interval returned an error");
202     close(sFd);
203     exit(EXIT_FAILURE);
204 }
205
206 nanosec=SEC_TO_NANOSEC*modf(d_time,&d_sec);
207 new_value.it_value.tv_nsec=nanosec;
208 new_value.it_value.tv_sec=(time_t)d_sec;
209 new_value.it_interval.tv_nsec=nanosec;
210 new_value.it_interval.tv_sec=(time_t)d_sec;
211 // time_t is long in my case, but being implementation dependant, I
    decided to print just d_sec as double with no decimal digits
```

```
212 fprintf(stdout,"Sending period set to %.0f second(s) and %.3f
    microsecond(s).\n",d_sec,nanosec/1000.0);
213
214 // Fill pollfd structure
215 timerMon.fd=clockFd;
216 timerMon.revents=0;
217 timerMon.events=POLLIN;
218
219 // Start timer
220 if(timerfd_settime(clockFd,NO_FLAGS,&new_value,NULL)==-1) {
221     perror("timerfd_settime() error");
222     close(sFd);
223     exit(EXIT_FAILURE);
224 } else {
225     fprintf(stdout,"Timer successfully started. Sending triggered.\n\n
        ");
226 }
227
228 while(1) {
229     // poll waiting for events happening on the timer descriptor (i.e.
        wait for timer expiration)
230     if(poll(&timerMon,1,INDEFINITE_BLOCK)>0) {
231         // "Clear the event" by performing a read() on a junk variable
232         read(clockFd,&junk,sizeof(junk));
233
234         // Prepare datagram
235         IP4headAddID(&ipHeader,(unsigned short) id);
236         id+=INCR_ID;
237         UDPencapsulate(udppacket,&udpHeader,argv[5],strlen(argv[5]),
            ipaddrs);
238         // 'IP4headAddTotLen' may also be skipped since IP4Encapsulate
            already takes care of filling the length field
239         // IP4headAddTotLen(&ipHeader, IP_UDP_PACKET_SIZE(argv[5]));
```

```
240     IP4Encapsulate(ippacket, &ipHeader, udppacket, UDP_PACKET_SIZE(  
        argv[5]));  
241     finalpktsize=etherEncapsulate(ethernetpacket, &etherHeader,  
        ippacket, IP_UDP_PACKET_SIZE(argv[5]));  
242  
243     // Send datagram  
244     if(sendto(sFd,ethernetpacket,finalpktsize,NO_FLAGS,(struct  
        sockaddr *)&addrll,sizeof(struct sockaddr_ll))!=finalpktsize)  
        {  
245         perror("sendto() for sending broadcasted data failed");  
246         fprintf(stderr,"The program will be terminated now");  
247         break;  
248     }  
249 }  
250 }  
251  
252 freeMacAddrT(srcmacaddr);  
253 free(udppacket);  
254 free(ethernetpacket);  
255 free(ippacket);  
256 close(sFd);  
257 close(clockFd);  
258  
259 return 0;  
260 }
```

Receiver program

broadcastReceive.c

```
1  #include <sys/socket.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <unistd.h>
6  #include <linux/wireless.h>
7  #include "Rawsock_lib/rawsock.h"
8  #include <linux/if_packet.h>
9
10 #define NO_FLAGS 0
11
12 int main (int argc, char **argv) {
13     int sFd;
14     ssize_t rcv_bytes;
15     byte_t packet[ETHERMTU]; // Packet buffer with size = Ethernet MTU
16
17     struct ether_header* etherHeader=NULL;
18     struct iphdr *IPheader=NULL;
19     struct udphdr *udpHeader=NULL;
20     byte_t *payload=NULL;
21
22     int ret_wlanl_val;
23     char devname[IFNAMSIZ]={0};
24     int ifindex;
25     struct sockaddr_ll addrll;
26
27     size_t payloadsize;
28
29     socklen_t AddrLen=sizeof(struct sockaddr);
30
```

```
31 // Already get the pointers to the headers, given the buffer which
    will contain the packet
32 payload=UDPgetpacketpointers(packet,&etherHeader,&IPheader,&
    udpHeader);
33
34 // Look for and bind to wireless interface, other than creating the
    socket
35 ret_wlanl_val=wlanLookup(devname,&ifindex,NULL,0);
36 if(ret_wlanl_val<=0) {
37     fprintf(stderr,"wlanLookup() error.\n");
38     rs_prnterror(stderr,ret_wlanl_val);
39     exit(EXIT_FAILURE);
40 }
41
42 sFd=socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
43 if(sFd==-1) {
44     perror("Cannot create socket: socket() error");
45     exit(EXIT_FAILURE);
46 }
47
48 fprintf(stdout,"Using interface: %s - index: 0x%02x - number of VIFs
    : %d\n",devname,ifindex,ret_wlanl_val);
49
50 // Prepare sockaddr_ll structure
51 bzero(&addrll,sizeof(addrll));
52 addrll.sll_ifindex=ifindex;
53 addrll.sll_family=AF_PACKET;
54 addrll.sll_protocol=htons(ETH_P_ALL);
55
56 // Bind to the wireless interface
57 if(bind(sFd,(struct sockaddr *) &addrll,sizeof(addrll))<0) {
58     perror("Cannot bind to interface: bind() error");
59     close(sFd);
60     exit(EXIT_FAILURE);
```

```
61  }
62
63  // This works only with AF_INET sockets, so it should not be used
    here
64  // if(setsockopt(sFd,SOL_SOCKET,SO_BINDTODEVICE,devname,strlen(
    devname))== -1) {
65  //  perror("setsockopt() for SO_BINDTODEVICE error");
66  //  close(sFd);
67  //  exit(EXIT_FAILURE);
68  // }
69
70  fprintf(stdout,"Ready to receive datagrams.\n\n");
71  while(1) {
72      // Receive datagram (blocking)
73      rcv_bytes = recvfrom(sFd,packet,ETHERMTU,NO_FLAGS,(struct sockaddr
        *)&addrll,&AddrLen);
74
75      // Go on only if it is a datagram of interest (in our case if it
        is UDP)
76      if (ntohs(etherHeader->ether_type)!=ETHERTYPE_IP) {
77          continue;
78      }
79      if (IPheader->protocol!=IPPROTO_UDP) {
80          continue;
81      }
82
83      // Validate checksum (combined mode: IP+UDP): if it is wrong,
        discard packet
84      payloadsize=UDPgetpayloadsize(udpHeader);
85      if(!validateEthCsum(packet, udpHeader->check, &(IPheader->check),
        CSUM_UDPIP, (void *) &payloadsize)) {
86          fprintf(stderr,"Wrong checksum! Packet will be discarded.\n");
87          continue;
88      }
```

```
89
90     if(rcv_bytes==-1){
91         perror("An error occurred in receive last message");
92         fprintf(stderr,"The execution will be terminated now.\n");
93         break;
94     }
95
96     // Print payload and source IP address
97     fprintf(stdout,"Received a new packet from %s\n",inet_ntoa(*(
98         struct in_addr*)&IPheader->saddr));
99     display_packetc("Received a new packet with payload:", payload,
100         payloadsize);
101 }
102
103 close(sFd);
104
105 return 0;
106 }
```

Receiver program (using AF_INET socket)

This receiver program was written to validate the raw socket sender program when using UDP.

It is using a standard AF_INET UDP datagram socket and it eventually supports a “sequential” mode that, when sending packets containing sequential numbers as payload, is able to provide, with a very basic mechanism, in which there is surely a lot of room for improvement, a packet loss count.

testprogram_broadcastReceive.c

```
1  #include <sys/socket.h>
2  #include <arpa/inet.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6  #include <string.h>
7  #include <unistd.h>
8  #include <errno.h>
9
10 #define MAX_LEN 1470
11 #define NO_FLAGS 0
12
13 int main (int argc, char **argv) {
14     int sFd;
15     int broadPort=3000; // Set to 3000 by default
16     struct sockaddr_in servAddr; // It contains properties to specify
        how the socket shall work
17     ssize_t rcv_bytes;
18     struct sockaddr_in srcAddr;
19     socklen_t srcAddrLen=sizeof(srcAddr);
20     char payload[MAX_LEN+1];
21     unsigned int sequential=0;
22     int curr_no,prev_no;
```



```
23 unsigned int lostCount=0;
24 unsigned int report_interval=0;
25
26 if(argc!=2) {
27     if(argc!=4 || strcmp(argv[2],"-seq")!=0) {
28         fprintf(stderr,"Error. Expected one mandatory (and two
29             optional) parameter.\nCorrect usage: <%s> <server port>
30             [-seq <report interval>].\n",argv[0]);
31         exit(EXIT_FAILURE);
32     } else {
33         sequential=1;
34         fprintf(stdout,"Using sequential mode.\n");
35     }
36 }
37
38 if(sequential) {
39     report_interval=atoi(argv[3]);
40     if(report_interval<1) {
41         fprintf(stderr,"Invalid report interval (in # of received frames
42             ) specified.\n");
43         exit(EXIT_FAILURE);
44     }
45 }
46
47 broadPort=atoi(argv[1]); // Read port from command line
48
49 // AF_INET (IPv4), SOCK_DGRAM (UDP for broadcast), 0 (socket is not
50 // raw)
51 sFd=socket(AF_INET,SOCK_DGRAM,0);
52 if(sFd==-1) {
53     perror("Cannot create socket: socket() error");
54     exit(EXIT_FAILURE);
55 }
```

```
53  bzero(&servAddr, sizeof(servAddr)); // Prepare servAddr struct (it
    helps initializing everything to 0 and catch possible bugs)
54  servAddr.sin_family=AF_INET; // Set address family (AF_INET for IPv4
    , AF_INET6 for IPv6)
55  servAddr.sin_addr.s_addr=htons(INADDR_ANY); // Listen to any address
56  servAddr.sin_port=htons(broadPort); // Convert port number to
    network byte order with htons (MSB first)
57
58  // Bind socket to port (accept connection based on properties
    defined in the struct)
59  if(bind(sFd, (struct sockaddr *)&servAddr, sizeof(servAddr))!=0) {
60      perror("Cannot bind socket: bind() error");
61      close(sFd);
62      exit(EXIT_FAILURE);
63  }
64
65  fprintf(stdout, "Ready to received datagrams on port %d.\n\n",
    broadPort);
66  while(1) {
67      // Use recvfrom to received datagrams on the specified port
68      rcv_bytes=recvfrom(sFd, payload, MAX_LEN, NO_FLAGS, (struct sockaddr
        *)&srcAddr, &srcAddrLen);
69      if(rcv_bytes==-1){
70          perror("An error occurred in receive last message");
71          fprintf(stderr, "The execution will be terminated now.\n");
72          break;
73      }
74      // Append final '\0' to convert the payload to a proper string
75      payload[rcv_bytes]='\0';
76      // Sequential mode: do not print the message
77      if(sequential) {
78          // Check for missing sequential packets
79          curr_no=atoi(payload);
80          if(curr_no!=0 && curr_no-prev_no>1) {
```

```
81     lostCount+=(curr_no-prev_no)-1;
82 }
83 if(curr_no%report_interval==0) {
84     fprintf(stdout,"%u/%d lost packets.\n",lostCount,
85             report_interval);
86     lostCount=0;
87 }
88 prev_no=curr_no;
89 } else {
90     fprintf(stdout,"Rx %d B from %s:%d: %s\n",(int)rcv_bytes,
91             inet_ntoa(srcAddr.sin_addr),srcAddr.sin_port,payload);
92 }
93 }
94
95 close(sFd);
96
97 return 0;
98 }
```

Compiler commands

In order to compile the sender and receiver programs, after putting our “rawsock” library inside “./Rawsock_lib”, we used the following commands:

```
x86_64-openwrt-linux-musl-gcc -I ./Rawsock_lib/ -o broadcastSend -  
    static broadcastSend.c Rawsock_lib/rawsock.h Rawsock_lib/rawsock.c  
    Rawsock_lib/ipcsum_alth.h Rawsock_lib/ipcsum_alth.c
```

and

```
x86_64-openwrt-linux-musl-gcc -I ./Rawsock_lib/ -o broadcastReceive -  
    static broadcastReceive.c Rawsock_lib/rawsock.h Rawsock_lib/rawsock  
    .c Rawsock_lib/ipcsum_alth.h Rawsock_lib/ipcsum_alth.c
```

These are the commands for the APU boards, for the ALIX ones the only modification should be in the name of the *gcc* compiler.

Appendix B

OpenWrt 18.06.1 patches

001-iperf-MAC_AC-patch.patch

In package/network/utils/iperf (custom)

```
1 --- a/include/Settings.hpp
2 +++ b/include/Settings.hpp
3 @@ -135,6 +135,7 @@ typedef struct thread_Settings {
4     // int's
5     int mThreads;           // -P
6     int mTOS;               // -S
7 +   int mMACUP;             // -A
8     #if WIN32
9         SOCKET mSock;
10    #else
11 --- a/include/version.h
12 +++ b/include/version.h
13 @@ -1,4 +1,4 @@
14 -#define IPERF_VERSION "2.0.12"
15 +#define IPERF_VERSION "2.0.12 OpenWrt ITS patch"
16 #define IPERF_VERSION_DATE "25 June 2018"
17 #define IPERF_VERSION_MAJORHEX 0x00020000
18 #define IPERF_VERSION_MINORHEX 0x000C0000
19 --- a/src/Locale.c
```

```
20 +++ b/src/Locale.c
21 @@ -103,6 +103,7 @@ Server specific:\n\
22     -s, --server                run in server mode\n\
23     -t, --time                  #          time in seconds to listen for new
        connections as well as to receive traffic (default not set)\n\
24     --udp-histogram #,#        enable UDP latency histogram(s) with bin
        width and count, e.g. 1,1000=1(ms),1000(bins)\n\
25 + -A, --accesscategory <AC> Forces a certain EDCA MAC access category
        to be used (BK, BE, VI, VO)\n\
26     -B, --bind <ip>[%<dev>]    bind to multicast address and optional
        device\n\
27     -H, --ssm-host <ip>        set the SSM source, use with -B for (S,G)
        \n\
28     -U, --single_udp           run in single threaded UDP mode\n\
29 @@ -125,6 +126,7 @@ Client specific:\n\
30     " -n, --num                #[kmgKMG]    number of bytes to transmit (instead
        of -t)\n\
31     -r, --tradeoff             Do a bidirectional test individually\n\
32     -t, --time                  #          time in seconds to transmit for (default
        10 secs)\n\
33 + -A, --accesscategory <AC> Forces a certain EDCA MAC access category
        to be used (BK, BE, VI, VO)\n\
34     -B, --bind [<ip> | <ip:port>] bind ip (and optional port) from
        which to source traffic\n\
35     -F, --fileinput <name>     input the data to be transmitted from a
        file\n\
36     -I, --stdin                input the data to be transmitted from
        stdin\n\
37 --- a/src/PerfSocket.cpp
38 +++ b/src/PerfSocket.cpp
39 @@ -155,6 +155,15 @@ void SetSocketOptions( thread_Settings *
40     }
41     #endif
42
```

```

43 +     // set MAC AC (access category) field, if specified only (i.e.
      if mMACUP != -1)
44 +     // AC is set starting from user priorities (UP)
45 + if ( inSettings->mMACUP >= 0 ) {
46 +     int up = inSettings->mMACUP;
47 +     Socklen_t len = sizeof(up);
48 +     int rc = setsockopt( inSettings->mSock, SOL_SOCKET, SO_PRIORITY, (
      char*) &up, len );
49 +     WARN_errno( rc == SOCKET_ERROR, "setsockopt SO_PRIORITY" );
50 + }
51 +
52 +     if ( !isUDP( inSettings ) ) {
53 +         // set the TCP maximum segment size
54 +         setsock_tcp_mss( inSettings->mSock, inSettings->mMSS );
55 --- a/src/Settings.cpp
56 +++ b/src/Settings.cpp
57 @@ -131,6 +131,7 @@ const struct option long_options[] =
58 {"realtime",          no_argument, NULL, 'z'},
59
60 // more esoteric options
61 +{"accesscategory",   required_argument, NULL, 'A'},
62 {"bind",             required_argument, NULL, 'B'},
63 {"compatibility",    no_argument, NULL, 'C'},
64 {"daemon",          no_argument, NULL, 'D'},
65 @@ -198,6 +199,7 @@ const struct option env_options[] =
66 {"IPERF_REPORTSTYLE",required_argument, NULL, 'y'},
67
68 // more esoteric options
69 +{"IPERF_MACAC",      required_argument, NULL, 'A'},
70 {"IPERF_BIND",       required_argument, NULL, 'B'},
71 {"IPERF_COMPAT",     no_argument, NULL, 'C'},
72 {"IPERF_DAEMON",     no_argument, NULL, 'D'},
73 @@ -218,7 +220,8 @@ const struct option env_options[] =
74

```

```
75  #define SHORT_OPTIONS()
76
77  -const char short_options[] = "1b:c:def:hi:l:mn:o:p:rst:uvw:x:y:zB:CDF
    :H:IL:M:NP:RS:T:UVWXZ:";
78  +// Edited to add the A: (A + 1 argument) short option
79  +const char short_options[] = "1b:c:def:hi:l:mn:o:p:rst:uvw:x:y:zA:B:
    CDF:IL:M:NP:RS:T:UVWZ:";
80
81  /* -----
82   * defaults
83  @@ -279,6 +282,7 @@ void Settings_Initialize( thread_Setting
84      //main->mThreads      = 0;          // -P,
85      //main->mRemoveService = false;      // -R,
86      //main->mTOS           = 0;          // -S, ie. don't set type
            of service
87  +   main->mMACUP           = -1;          // -A (set to an invalid
            number as default -> with -1 no setsockopt will be called for AC)
88      main->mTTL             = -1;          // -T, link-local TTL
89      //main->mDomain        = kMode_IPv4;  // -V,
90      //main->mSuggestWin    = false;       // -W, Suggest the window
            size.
91  @@ -692,6 +696,24 @@ void Settings_Interpret( char option, co
92      mExtSettings->mTOS = strtol( optarg, NULL, 0 );
93      break;
94
95  +   case 'A': // 802.11p/802.11e MAC layer access categories
96  +       // Mapping between UP (0 to 7) and AC (BK to VO)
97  +       if(strcmp(optarg,"BK") == 0) {
98  +           mExtSettings->mMACUP=1; // UP=1 (2) is AC_BK
99  +       } else if(strcmp(optarg,"BE") == 0) {
100  +           mExtSettings->mMACUP=0; // UP=0 (3) is AC_BE
101  +       } else if(strcmp(optarg,"VI") == 0) {
102  +           mExtSettings->mMACUP=4; // UP=4 (5) is AC_VI
103  +       } else if(strcmp(optarg,"VO") == 0) {
```



```
104 +         mExtSettings->mMACUP=6; // UP=6 (7) is AC_VO
105 +     } else {
106 +         // Leave to default (-1), i.e. no AC is set to socket
        , and print error
107 +         fprintf(stderr, "Invalid AC specified with -A\nValid
        ones are: BK, BE, VI, VO\nNo AC will be set\n");
108 +     }
109 +
110 +         //mExtSettings->mMACUP = strtol( optarg, NULL, 0 );
111 +         break;
112 +
113     case 'T': // time-to-live for both unicast and multicast
114         mExtSettings->mTTL = atoi( optarg );
115         break;
```

202-restore_ocb.patch

In package/network/utils/iw (custom)

```
1 --- a/Makefile
2 +++ b/Makefile
3 @@ -25,7 +25,7 @@ OBJS-$(HWSIM) += hwsim.o
4
5  OBJS += $(OBJS-y) $(OBJS-Y)
6
7 -OBJS_DISABLED = ocb offch cqm wowlan coalesce roc p2p ap
8 +OBJS_DISABLED = offch cqm wowlan coalesce roc p2p ap
9  OBJS:=$(filter-out $(patsubst %,%.o,$(OBJS_DISABLED)),$(OBJS))
10 ALL = iw
```

600-DE-openC2X-regdb.patch

In package/firmware/wireless-regdb (custom)

```
1 --- a/db.txt
2 +++ b/db.txt
3 @@ -368,6 +368,8 @@ country DE: DFS-ETSI
4     (5470 - 5725 @ 160), (500 mW), DFS
5     # short range devices (ETSI EN 300 440-1)
6     (5725 - 5875 @ 80), (25 mW)
7 + # ITS (802.11p)
8 + (5850 - 5925 @ 20), (33)
9     # 60 GHz band channels 1-4 (ETSI EN 302 567)
10    (57000 - 66000 @ 2160), (40)
```

601-IT-regdb.patch

In package/firmware/wireless-regdb (custom)

```
1 --- a/db.txt
2 +++ b/db.txt
3 @@ -616,6 +616,8 @@ country IT: DFS-ETSI
4     (5170 - 5250 @ 80), (20), AUTO-BW
5     (5250 - 5330 @ 80), (20), DFS, AUTO-BW
6     (5490 - 5710 @ 160), (27), DFS
7 + # ITS (802.11p)
8 + (5842 - 5932 @ 20), (27), DFS
9     # 60 GHz band channels 1-4, ref: Etsi En 302 567
10    (57000 - 66000 @ 2160), (40)
```

998-ath5k_ocb.patch

In package/kernel/mac80211 (OpenC2X - CCS Labs)

```

1  --- a/drivers/net/wireless/ath/ath5k/base.c
2  +++ b/drivers/net/wireless/ath/ath5k/base.c
3  @@ -287,6 +287,9 @@ static bool ath5k_is_standard_channel(sh
4      ((chan & 3) == 1 && chan >= 149 && chan <= 165) ||
5      /* 802.11j 5.030-5.080 GHz (20MHz) */
6      (chan == 8 || chan == 12 || chan == 16) ||
7  +  /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
8      Mar 30 16:32:13 2017 +0200 */
9  +  /* 802.11p (10MHz) */
10 +  (chan == 172) || (chan == 174) || (chan == 176) || (chan == 178)
11 +  || (chan == 180) || (chan == 182) || (chan == 184) ||
12 +  /* 802.11j 4.9GHz (20MHz) */
13 +  (chan == 184 || chan == 188 || chan == 192 || chan == 196));
14  }
15  @@ -1068,7 +1071,14 @@ ath5k_beaconq_config(struct ath5k_hw *ah
16      qi.tqi_aifs = 0;
17      qi.tqi_cw_min = 0;
18      qi.tqi_cw_max = 2 * AR5K_TUNE_CWMIN;
19  - }
20  + } else if (ah->opmode == NL80211_IFTYPE_OCB) {
21  +  /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
22      Mar 30 16:32:13 2017 +0200 */
23  +  /*
24  +   * OCB mode; backoff between 0 and (2 * cw_min).
25  +   */
26  +  qi.tqi_aifs = 0;
27  +  qi.tqi_cw_min = 0;
28  +  qi.tqi_cw_max = 2 * AR5K_TUNE_CWMIN; }
29
30  ATH5K_DBG(ah, ATH5K_DEBUG_BEACON,

```

```
28     "beacon queueprops tqi_aifs:%d tqi_cw_min:%d tqi_cw_max:%d\n",
29 @@ -1436,7 +1446,8 @@ ath5k_receive_frame(struct ath5k_hw *ah,
30     ewma_beacon_rssi_add(&ah->ah_beacon_rssi_avg, rs->rs_rssi);
31
32     /* check beacons in IBSS mode */
33 - if (ah->opmode == NL80211_IFTYPE_ADHOC)
34 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
35     Mar 30 16:32:13 2017 +0200 */
36 + if (ah->opmode == NL80211_IFTYPE_ADHOC || ah->opmode ==
37     NL80211_IFTYPE_OCB)
38     ath5k_check_ibss_tsf(ah, skb, rx);
39 }
40
41 @@ -1837,7 +1848,8 @@ ath5k_beacon_setup(struct ath5k_hw *ah,
42     antenna = ah->ah_tx_ant;
43
44     flags = AR5K_TXDESC_NOACK;
45 - if (ah->opmode == NL80211_IFTYPE_ADHOC && ath5k_hw_hasveol(ah)) {
46 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
47     Mar 30 16:32:13 2017 +0200 */
48 + if ((ah->opmode == NL80211_IFTYPE_ADHOC || ah->opmode ==
49     NL80211_IFTYPE_OCB) && ath5k_hw_hasveol(ah)) {
50     ds->ds_link = bf->daddr; /* self-linked */
51     flags |= AR5K_TXDESC_VEOL;
52 } else
53 @@ -2167,7 +2179,8 @@ ath5k_beacon_config(struct ath5k_hw *ah)
54
55     ah->imask |= AR5K_INT_SWBA;
56
57 - if (ah->opmode == NL80211_IFTYPE_ADHOC) {
58 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
59     Mar 30 16:32:13 2017 +0200 */
60 + if (ah->opmode == NL80211_IFTYPE_ADHOC || ah->opmode ==
61     NL80211_IFTYPE_OCB) {
```

```

56     if (ath5k_hw_hasveol(ah))
57         ath5k_beacon_send(ah);
58     } else
59 @@ -2193,7 +2206,8 @@ static void ath5k_tasklet_beacon(unsigne
60     * transmission time) in order to detect whether
61     * automatic TSF updates happened.
62     */
63 - if (ah->opmode == NL80211_IFTYPE_ADHOC) {
64 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
65     Mar 30 16:32:13 2017 +0200 */
66 + if (ah->opmode == NL80211_IFTYPE_ADHOC || ah->opmode ==
67     NL80211_IFTYPE_OCB) {
68     /* XXX: only if VEOL supported */
69     u64 tsf = ath5k_hw_get_tsf64(ah);
70     ah->nexttbtt += ah->bintval;
71 @@ -2552,7 +2566,8 @@ ath5k_init_ah(struct ath5k_hw *ah, const
72     BIT(NL80211_IFTYPE_AP) |
73     BIT(NL80211_IFTYPE_STATION) |
74     BIT(NL80211_IFTYPE_ADHOC) |
75 - BIT(NL80211_IFTYPE_MESH_POINT);
76 + BIT(NL80211_IFTYPE_MESH_POINT) |
77 + BIT(NL80211_IFTYPE_OCB); /* OCB patch by Florian Klingler <
78     klingler@ccs-labs.org>, date Thu Mar 30 16:32:13 2017 +0200 */
79
80     hw->wiphy->iface_combinations = &if_comb;
81     hw->wiphy->n_iface_combinations = 1;
82 --- a/drivers/net/wireless/ath/ath5k/mac80211-ops.c
83 +++ b/drivers/net/wireless/ath/ath5k/mac80211-ops.c
84 @@ -80,7 +80,9 @@ ath5k_add_interface(struct ieee80211_hw
85     mutex_lock(&ah->lock);
86
87     if ((vif->type == NL80211_IFTYPE_AP ||
88 -     vif->type == NL80211_IFTYPE_ADHOC)
89 +     vif->type == NL80211_IFTYPE_ADHOC ||

```

```
87 +      /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date
      Thu Mar 30 16:32:13 2017 +0200 */
88 +      vif->type == NL80211_IFTYPE_OCB)
89      && (ah->num_ap_vifs + ah->num_adhoc_vifs) >= ATH_BCBUF) {
90      ret = -ELNRNG;
91      goto end;
92 @@ -98,6 +100,7 @@ ath5k_add_interface(struct ieee80211_hw
93      case NL80211_IFTYPE_STATION:
94      case NL80211_IFTYPE_ADHOC:
95      case NL80211_IFTYPE_MESH_POINT:
96 + case NL80211_IFTYPE_OCB: /* OCB patch by Florian Klingler <
      klingler@ccs-labs.org>, date Thu Mar 30 16:32:13 2017 +0200 */
97      avf->opmode = vif->type;
98      break;
99      default:
100 @@ -111,7 +114,9 @@ ath5k_add_interface(struct ieee80211_hw
101      /* Assign the vap/adhoc to a beacon xmit slot. */
102      if ((avf->opmode == NL80211_IFTYPE_AP) ||
103          (avf->opmode == NL80211_IFTYPE_ADHOC) ||
104 -      (avf->opmode == NL80211_IFTYPE_MESH_POINT)) {
105 +      (avf->opmode == NL80211_IFTYPE_MESH_POINT) ||
106 +      /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date
      Thu Mar 30 16:32:13 2017 +0200 */
107 +      (avf->opmode == NL80211_IFTYPE_OCB)) {
108      int slot;
109
110      WARN_ON(list_empty(&ah->bcbuf));
111 @@ -134,6 +139,8 @@ ath5k_add_interface(struct ieee80211_hw
112      ah->num_adhoc_vifs++;
113      else if (avf->opmode == NL80211_IFTYPE_MESH_POINT)
114      ah->num_mesh_vifs++;
115 + else if (avf->opmode == NL80211_IFTYPE_OCB)
116 + ah->num_mesh_vifs++; /* OCB patch by Florian Klingler <
      klingler@ccs-labs.org>, date Thu Mar 30 16:32:13 2017 +0200 */
```



```

117     }
118
119     /* Any MAC address is fine, all others are included through the
120 @@ -177,6 +184,8 @@ ath5k_remove_interface(struct ieee80211_
121         ah->num_adhoc_vifs--;
122     else if (avf->opmode == NL80211_IFTYPE_MESH_POINT)
123         ah->num_mesh_vifs--;
124 + else if (avf->opmode == NL80211_IFTYPE_OCB)
125 +     ah->num_mesh_vifs--; /* OCB patch by Florian Klingler <
126         klingler@ccs-labs.org>, date Thu Mar 30 16:32:13 2017 +0200 */
127
128     ath5k_update_bssid_mask_and_opmode(ah, NULL);
129     mutex_unlock(&ah->lock);
130 @@ -426,6 +435,11 @@ ath5k_configure_filter(struct ieee80211_
131     rfilt |= AR5K_RX_FILTER_PROBEREQ |
132         AR5K_RX_FILTER_BEACON;
133     break;
134 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
135     Mar 30 16:32:13 2017 +0200 */
136 + case NL80211_IFTYPE_OCB:
137 +     rfilt |= AR5K_RX_FILTER_PROBEREQ |
138 +         AR5K_RX_FILTER_BEACON;
139 +     break;
140     case NL80211_IFTYPE_STATION:
141         if (ah->assoc)
142             rfilt |= AR5K_RX_FILTER_BEACON;
143 @@ -633,7 +647,8 @@ ath5k_reset_tsf(struct ieee80211_hw *hw,
144     * in IBSS mode we need to update the beacon timers too.
145     * this will also reset the TSF if we call it with 0
146     */
147 - if (ah->opmode == NL80211_IFTYPE_ADHOC)
148 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
149     Mar 30 16:32:13 2017 +0200 */

```

```
147 + if (ah->opmode == NL80211_IFTYPE_ADHOC || ah->opmode ==
    NL80211_IFTYPE_OCB)
148     ath5k_beacon_update_timers(ah, 0);
149 else
150     ath5k_hw_reset_tsf(ah);
151 --- a/drivers/net/wireless/ath/ath5k/pcu.c
152 +++ b/drivers/net/wireless/ath/ath5k/pcu.c
153 @@ -670,6 +670,9 @@ ath5k_hw_init_beacon_timers(struct ath5k
154     break;
155     case NL80211_IFTYPE_ADHOC:
156         AR5K_REG_ENABLE_BITS(ah, AR5K_TXCFG, AR5K_TXCFG_ADHOC_BCN_ATIM);
157 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
    Mar 30 16:32:13 2017 +0200 */
158 + case NL80211_IFTYPE_OCB:
159 +     AR5K_REG_ENABLE_BITS(ah, AR5K_TXCFG, AR5K_TXCFG_ADHOC_BCN_ATIM);
160     default:
161         /* On non-STA modes timer1 is used as next DMA
162          * beacon alert (DBA) timer and timer2 as next
163 @@ -893,6 +896,16 @@ ath5k_hw_set_opmode(struct ath5k_hw *ah,
164     pcu_reg |= AR5K_STA_ID1_ADHOC | AR5K_STA_ID1_KEYSRCH_MODE;
165     beacon_reg |= AR5K_BCR_ADHOC;
166     if (ah->ah_version == AR5K_AR5210)
167 +     pcu_reg |= AR5K_STA_ID1_NO_PSPOLL;
168 + else
169 +     AR5K_REG_ENABLE_BITS(ah, AR5K_CFG, AR5K_CFG_IBSS);
170 + break;
171 +
172 + /* OCB patch by Florian Klingler <klingler@ccs-labs.org>, date Thu
    Mar 30 16:32:13 2017 +0200 */
173 + case NL80211_IFTYPE_OCB:
174 +     pcu_reg |= AR5K_STA_ID1_ADHOC | AR5K_STA_ID1_KEYSRCH_MODE;
175 +     beacon_reg |= AR5K_BCR_ADHOC;
176 +     if (ah->ah_version == AR5K_AR5210)
177         pcu_reg |= AR5K_STA_ID1_NO_PSPOLL;
```

```
178     else
179         AR5K_REG_ENABLE_BITS(ah, AR5K_CFG, AR5K_CFG_IBSS);
```

998-ath9k_allow_11p.patch

In package/kernel/mac80211 (OpenC2X - CCS Labs)

```
1 --- a/drivers/net/wireless/ath/ath9k/common-init.c
2 +++ b/drivers/net/wireless/ath/ath9k/common-init.c
3 @@ -86,6 +86,28 @@ static const struct ieee80211_channel at
4     CHAN5G(5785, 35), /* Channel 157 */
5     CHAN5G(5805, 36), /* Channel 161 */
6     CHAN5G(5825, 37), /* Channel 165 */
7 +
8 + /* 802.11p patch by Florian Klingler <klingler@ccs-labs.org>, date
9     Wed Mar 15 17:50:09 2017 +0100 */
10 + /* ITS frequencies */
11 + CHAN5G(5850, 38), /* Channel 170 */
12 + /* ITA-G5B */
13 + CHAN5G(5855, 39), /* Channel 171 */
14 + CHAN5G(5860, 40), /* Channel 172 */
15 + CHAN5G(5865, 41), /* Channel 173 */
16 + CHAN5G(5870, 42), /* Channel 174 */
17 + /* ITS-G5A */
18 + CHAN5G(5875, 43), /* Channel 175 */
19 + CHAN5G(5880, 44), /* Channel 176 */
20 + CHAN5G(5885, 45), /* Channel 177 */
21 + CHAN5G(5890, 46), /* Channel 178 - IEEE CCH */
22 + CHAN5G(5895, 47), /* Channel 179 */
23 + CHAN5G(5900, 48), /* Channel 180 */
24 + CHAN5G(5905, 49), /* Channel 181 */
25 + /* ITS-G5D */
26 + CHAN5G(5910, 50), /* Channel 182 */
27 + CHAN5G(5915, 51), /* Channel 183 */
28 + CHAN5G(5920, 52), /* Channel 184 */
29 + CHAN5G(5925, 53), /* Channel 185 */
30 };
```

```

30
31  /* Atheros hardware rate code addition for short preamble */
32  --- a/drivers/net/wireless/ath/ath9k/hw.h
33  +++ b/drivers/net/wireless/ath/ath9k/hw.h
34  @@ -74,7 +74,8 @@
35
36  #define ATH9K_RSSI_BAD      -128
37
38  -#define ATH9K_NUM_CHANNELS 38
39  +/* 802.11p patch by Florian Klingler <klingler@ccs-labs.org>, date
36      Wed Mar 15 17:50:09 2017 +0100 */
40  +#define ATH9K_NUM_CHANNELS 54 /* Increased from 38 to 54, adding all
36      the ITS channels */
41
42  /* Register read/write primitives */
43  #define REG_WRITE(_ah, _reg, _val) \
44  --- a/drivers/net/wireless/ath/regd.c
45  +++ b/drivers/net/wireless/ath/regd.c
46  @@ -43,13 +43,15 @@ static struct reg_dmn_pair_mapping *ath_
47      NL80211_RRF_NO_IR | \
48      NL80211_RRF_NO_OFDM)
49
50  +/* Register read/write primitives */
51  +/* 802.11p patch by Florian Klingler <klingler@ccs-labs.org>, date
36      Wed Mar 15 17:50:09 2017 +0100 */
52  /* We allow IBSS on these on a case by case basis by regulatory
36      domain */
53  #define ATH9K_5GHZ_5150_5350 REG_RULE(5150-10, 5240+10, 80, 0, 30, 0)
36      ,\
54      REG_RULE(5260-10, 5350+10, 80, 0, 30,\
55      NL80211_RRF_NO_IR)
56  -#define ATH9K_5GHZ_5470_5850 REG_RULE(5470-10, 5850+10, 80, 0, 30,\
57  +#define ATH9K_5GHZ_5470_5925 REG_RULE(5470-10, 5925+10, 80, 0, 30,\
58      NL80211_RRF_NO_IR)

```

```
59 -#define ATH9K_5GHZ_5725_5850 REG_RULE(5725-10, 5850+10, 80, 0, 30,\n60 +#define ATH9K_5GHZ_5725_5925 REG_RULE(5725-10, 5925+10, 80, 0, 30,\n61     NL80211_RRF_NO_IR)\n62\n63 #define ATH9K_2GHZ_ALL    ATH9K_2GHZ_CH01_11, \n64 @@ -57,11 +59,11 @@ static struct reg_dmn_pair_mapping *ath_\n65     ATH9K_2GHZ_CH14\n66\n67 #define ATH9K_5GHZ_ALL    ATH9K_5GHZ_5150_5350, \n68 -     ATH9K_5GHZ_5470_5850\n69 +     ATH9K_5GHZ_5470_5925\n70\n71 /* This one skips what we call "mid band" */\n72 #define ATH9K_5GHZ_NO_MIDBAND    ATH9K_5GHZ_5150_5350, \n73 -     ATH9K_5GHZ_5725_5850\n74 +     ATH9K_5GHZ_5725_5925\n75\n76 #define REGD_RULES(...) \n77     .reg_rules = { __VA_ARGS__ }, \
```

999-Enable-queueing-in-all-4-ACs-BE-BK-VI-VO.patch

In package/kernel/mac80211 (OpenC2X - CCS Labs)

```
1 --- a/net/mac80211/wme.c
2 +++ b/net/mac80211/wme.c
3 @@ -215,9 +215,13 @@ u16 ieee80211_select_queue(struct ieee80
4
5     /* use the data classifier to determine what 802.1d tag the
6      * data frame has */
7 - qos_map = rcu_dereference(sdata->qos_map);
8 - skb->priority = cfg80211_classify8021d(skb, qos_map ?
9 -                                     &qos_map->qos_map : NULL);
10 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
11 +    paderborn.de>, date Wed, 27 Jul 2016 14:20:08 +0200 */
12 + // Commenting this out seems to disable, however, the IP ToS bit
13 + // Quick workaround to enable queueing in different AC (BE, BK, VO,
14 + // VI)
15 + // qos_map = rcu_dereference(sdata->qos_map);
16 + // skb->priority = cfg80211_classify8021d(skb, qos_map ?
17 + //                                     &qos_map->qos_map : NULL);
18
19     downgrade:
20     ret = ieee80211_downgrade_queue(sdata, sta, skb);
```

999-Get-hw-queue-pending-stats-from-ath9k-via-netlink.patch

In package/kernel/mac80211 (OpenC2X - CCS Labs)

```
1 --- a/drivers/net/wireless/ath/ath9k/ath9k.h
2 +++ b/drivers/net/wireless/ath/ath9k/ath9k.h
3 @@ -1017,6 +1017,10 @@ struct ath_softc {
4     struct survey_info *cur_survey;
5     struct survey_info survey[ATH9K_NUM_CHANNELS];
6
7 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
8     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
9 + struct flush_info hw_q_flush_info;
10 + u32 flush_hw_q_pending;
11 +
12     spinlock_t intr_lock;
13     struct tasklet_struct intr_tq;
14     struct tasklet_struct bcon_tasklet;
15 --- a/drivers/net/wireless/ath/ath9k/debug.c
16 +++ b/drivers/net/wireless/ath/ath9k/debug.c
17 @@ -593,7 +593,9 @@ static int read_file_xmit(struct seq_fil
18     struct ieee80211_hw *hw = dev_get_drvdata(file->private);
19     struct ath_softc *sc = hw->priv;
20
21 - seq_printf(file, "%30s %10s%10s%10s\n\n", "BE", "BK", "VI", "VO");
22 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
23     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
24 + // Changed %30s to %38s
25 + seq_printf(file, "%38s %10s%10s%10s\n\n", "BE", "BK", "VI", "VO");
26
27     PR("MPDUs Queued:      ", queued);
28     PR("MPDUs Completed: ", completed);
29 @@ -1666,5 +1668,9 @@ int ath9k_init_debug(struct ath_hw *ah)
30     debugfs_create_file("nf_override", S_IRUSR | S_IWUSR,
```



```

29         sc->debug.debugfs_phy, sc, &fops_nf_override);
30
31 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
        paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
32 + debugfs_create_u32("flush_hw_q_pending", S_IRUGO | S_IWUGO,
33 +         sc->debug.debugfs_phy, &sc->flush_hw_q_pending);
34 +
35     return 0;
36 }
37 --- a/drivers/net/wireless/ath/ath9k/debug.h
38 +++ b/drivers/net/wireless/ath/ath9k/debug.h
39 @@ -164,6 +164,8 @@ struct ath_interrupt_stats {
40     * @txstart:   Number of times hardware was told to start tx.
41     * @txprocdesc: Number of times tx descriptor was processed
42     * @txfailed:  Out-of-memory or other errors in xmit path.
43 + * @hw_flush_required: Number of times the hardware queue needs to be
        flushed.
44 + * @hw_flush_not_required: Number of times the hardware queue was
        found empty before pushing new packets to it.
45     */
46     struct ath_tx_stats {
47         u32 tx_pkts_all;
48 @@ -187,6 +189,9 @@ struct ath_tx_stats {
49         u32 txstart;
50         u32 txprocdesc;
51         u32 txfailed;
52 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
        paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
53 + u32 hw_flush_required;
54 + u32 hw_flush_not_required;
55     };
56
57     /*
58 --- a/drivers/net/wireless/ath/ath9k/main.c

```

```
59 +++ b/drivers/net/wireless/ath/ath9k/main.c
60 @@ -2002,6 +2002,28 @@ static int ath9k_get_survey(struct ieee8
61     return 0;
62 }
63
64 +/* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
65     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
66 +static int ath9k_get_flush_stats(struct ieee80211_hw *hw, int idx,
67     struct flush_info *survey) {
68 + printk(KERN_ALERT "%s:%d\n", __FILE__, __LINE__);
69 + struct ath_softc *sc = hw->priv;
70 + // No more necessary?
71 + //if (config_enabled(CONFIG_ATH9K_TX99))
72 + // return -EOPNOTSUPP;
73 + survey->be_flush_req = sc->debug.stats.txstats[ATH_TXQ_AC_BE].
74     hw_flush_required;
75 + survey->be_flush_not_req = sc->debug.stats.txstats[ATH_TXQ_AC_BE].
76     hw_flush_not_required;
77 +
78 + survey->bk_flush_req = sc->debug.stats.txstats[ATH_TXQ_AC_BK].
79     hw_flush_required;
80 + survey->bk_flush_not_req = sc->debug.stats.txstats[ATH_TXQ_AC_BK].
81     hw_flush_not_required;
82 +
83 + survey->vi_flush_req = sc->debug.stats.txstats[ATH_TXQ_AC_VI].
84     hw_flush_required;
85 + survey->vi_flush_not_req = sc->debug.stats.txstats[ATH_TXQ_AC_VI].
86     hw_flush_not_required;
87 +
88 + survey->vo_flush_req = sc->debug.stats.txstats[ATH_TXQ_AC_VO].
89     hw_flush_required;
90 + survey->vo_flush_not_req = sc->debug.stats.txstats[ATH_TXQ_AC_VO].
91     hw_flush_not_required;
92 +
```

```

83 + return 0;
84 +}
85 +
86 static void ath9k_enable_dynack(struct ath_softc *sc)
87 {
88 #ifdef CPTCFG_ATH9K_DYNACK
89 @@ -2687,6 +2709,8 @@ struct ieee80211_ops ath9k_ops = {
90     .reset_tsf          = ath9k_reset_tsf,
91     .ampdu_action       = ath9k_ampdu_action,
92     .get_survey         = ath9k_get_survey,
93 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
94     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
95 + .get_flush_stats     = ath9k_get_flush_stats,
96     .rfkill_poll        = ath9k_rfkill_poll_state,
97     .set_coverage_class = ath9k_set_coverage_class,
98     .flush              = ath9k_flush,
99 --- a/drivers/net/wireless/ath/ath9k/xmit.c
100 +++ b/drivers/net/wireless/ath/ath9k/xmit.c
101 @@ -2084,6 +2084,18 @@ static void ath_tx_txqaddbuf(struct ath_
102     if (list_empty(head))
103         return;
104 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
105     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
106 + if (sc->flush_hw_q_pending) {
107 +     if (ath9k_hw_numtxpending(ah, txq->axq_qnum)) {
108 +         // printk(KERN_ALERT "flush and pending!");
109 +         TX_STAT_INC(txq->axq_qnum, hw_flush_required);
110 +     } else {
111 +         // printk(KERN_ALERT "flush but nothing pending in queues!");
112 +         TX_STAT_INC(txq->axq_qnum, hw_flush_not_required);
113 +     }
114 + }

```

```
115 +
116     edma = !(ah->caps.hw_caps & ATH9K_HW_CAP_EDMA);
117     bf = list_first_entry(head, struct ath_buf, list);
118     bf_last = list_entry(head->prev, struct ath_buf, list);
119 @@ -2877,6 +2889,7 @@ int ath_tx_init(struct ath_softc *sc, in
120
121     if (sc->sc_ah->caps.hw_caps & ATH9K_HW_CAP_EDMA)
122         error = ath_tx_edma_init(sc);
123 + sc->flush_hw_q_pending = 0; /* MAC queues enabler patch by Gurjashan
124     S. Pannu */
125
126     return error;
127 }
128 --- a/include/net/cfg80211.h
129 +++ b/include/net/cfg80211.h
130 @@ -627,6 +627,30 @@ struct survey_info {
131     s8 noise;
132 };
133
134 +/* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
135     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
136 +/**
137 + * struct flush_info - stats for flushing pending packets in hardware
138     queues
139 + *
140 + * @be_flush_req: number of times when there was need to flush a
141     pending packet for AC_BE
142 + * @be_flush_not_req: number of times when there was no pending
143     packet in AC_BE
144 + * @bk_flush_req: number of times when there was need to flush a
145     pending packet for AC_BK
146 + * @bk_flush_not_req: number of times when there was no pending
147     packet in AC_BK
```

```
141 + * @vi_flush_req: number of times when there was need to flush a
    pending packet for AC_VI
142 + * @vi_flush_not_req: number of times when there was no pending
    packet in AC_VI
143 + * @vo_flush_req: number of times when there was need to flush a
    pending packet for AC_VO
144 + * @vo_flush_not_req: number of times when there was no pending
    packet in AC_VO
145 + */
146 +struct flush_info {
147 + u32 be_flush_req;
148 + u32 be_flush_not_req;
149 + u32 bk_flush_req;
150 + u32 bk_flush_not_req;
151 + u32 vi_flush_req;
152 + u32 vi_flush_not_req;
153 + u32 vo_flush_req;
154 + u32 vo_flush_not_req;
155 +};
156 +
157 #define CFG80211_MAX_WEP_KEYS 4
158
159 /**
160 @@ -3065,6 +3089,10 @@ struct cfg80211_ops {
161     int (*dump_survey)(struct wiphy *wiphy, struct net_device *netdev,
162         int idx, struct survey_info *info);
163
164 + /* MAC queues enabler patch by Gurjashan S. Pannu <gs pannu@mail.uni-
    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
165 + int (*dump_flush_stats)(struct wiphy *wiphy, struct net_device *
    netdev,
166 +     int idx, struct flush_info *info);
167 +
168     int (*set_pmksa)(struct wiphy *wiphy, struct net_device *netdev,
```

```
169         struct cfg80211_pmksa *pmksa);
170     int (*del_pmksa)(struct wiphy *wiphy, struct net_device *netdev,
171 --- a/include/net/mac80211.h
172 +++ b/include/net/mac80211.h
173 @@ -3619,6 +3619,9 @@ struct ieee80211_ops {
174         struct ieee80211_ampdu_params *params);
175     int (*get_survey)(struct ieee80211_hw *hw, int idx,
176         struct survey_info *survey);
177 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
178     + paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
179 + int (*get_flush_stats)(struct ieee80211_hw *hw, int idx,
180 +     struct flush_info *survey);
181     void (*rfkill_poll)(struct ieee80211_hw *hw);
182     void (*set_coverage_class)(struct ieee80211_hw *hw, s16
183         coverage_class);
184 #ifdef CPTCFG_NL80211_TESTMODE
185 --- a/include/uapi/linux/nl80211.h
186 +++ b/include/uapi/linux/nl80211.h
187 @@ -1198,6 +1198,10 @@ enum nl80211_commands {
188
189     NL80211_CMD_RELOAD_REGDB,
190
191 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
192     + paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
193 + NL80211_CMD_FLUSH_STATS, // Related to flushing hardware queues
194 + NL80211_CMD_NEW_FLUSH_STATS,
195 +
196     /* add new commands above here */
197
198     /* used to define NL80211_CMD_MAX below */
199 @@ -2584,6 +2588,9 @@ enum nl80211_attrs {
200
201     NL80211_ATTR_WIPHY_ANTENNA_GAIN,
```

```
200 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
201 + NL80211_ATTR_FLUSH_INFO,
202 +
203     /* add attributes here, update the policy in nl80211.c */
204
205     __NL80211_ATTR_AFTER_LAST,
206 @@ -3441,6 +3448,23 @@ enum nl80211_survey_info {
207     NL80211_SURVEY_INFO_MAX = __NL80211_SURVEY_INFO_AFTER_LAST - 1
208 };
209
210 +/* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
211 +enum nl80211_flush_info {
212 +     __NL80211_FLUSH_INFO_INVALID,
213 +     NL80211_FLUSH_REQ_BE,
214 +     NL80211_FLUSH_NOT_REQ_BE,
215 +     NL80211_FLUSH_REQ_BK,
216 +     NL80211_FLUSH_NOT_REQ_BK,
217 +     NL80211_FLUSH_REQ_VI,
218 +     NL80211_FLUSH_NOT_REQ_VI,
219 +     NL80211_FLUSH_REQ_VO,
220 +     NL80211_FLUSH_NOT_REQ_VO,
221 +
222 +     /* keep last */
223 +     __NL80211_FLUSH_INFO_AFTER_LAST,
224 +     NL80211_FLUSH_INFO_MAX = __NL80211_FLUSH_INFO_AFTER_LAST - 1
225 +};
226 +
227     /* keep old names for compatibility */
228     #define NL80211_SURVEY_INFO_CHANNEL_TIME    NL80211_SURVEY_INFO_TIME
229     #define NL80211_SURVEY_INFO_CHANNEL_TIME_BUSY
        NL80211_SURVEY_INFO_TIME_BUSY
230 --- a/net/mac80211/cfg.c
```

```
231 +++ b/net/mac80211/cfg.c
232 @@ -710,6 +710,16 @@ static int ieee80211_dump_survey(struct
233     return drv_get_survey(local, idx, survey);
234 }
235
236 +/* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
237     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
238 +static int ieee80211_dump_flush_stats(struct wiphy *wiphy, struct
239     net_device *dev,
240     int idx, struct flush_info *survey)
241 +{
242 + struct ieee80211_local *local = wdev_priv(dev->ieee80211_ptr);
243 +
244 + return drv_get_flush_stats(local, idx, survey);
245 +}
246 +
247 +static int ieee80211_get_station(struct wiphy *wiphy, struct
248     net_device *dev,
249     const u8 *mac, struct station_info *sinfo)
250 {
251 @@ -3700,6 +3710,7 @@ const struct cfg80211_ops mac80211_conf
252     .get_station = ieee80211_get_station,
253     .dump_station = ieee80211_dump_station,
254     .dump_survey = ieee80211_dump_survey,
255 + .dump_flush_stats = ieee80211_dump_flush_stats, /* MAC queues
256     enabler patch by Gurjashan S. Pannu */
257 #ifdef CPTCFG_MAC80211_MESH
258     .add_mpath = ieee80211_add_mpath,
259     .del_mpath = ieee80211_del_mpath,
260 --- a/net/mac80211/driver-ops.h
261 +++ b/net/mac80211/driver-ops.h
262 @@ -607,6 +607,16 @@ static inline int drv_get_survey(struct
263     return ret;
```



```
261 }
262
263 /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
264 +static inline int drv_get_flush_stats(struct ieee80211_local *local,
    int idx,
265 +    struct flush_info *survey)
266 +{
267 + int ret = -EOPNOTSUPP;
268 + if (local->ops->get_flush_stats)
269 +     ret = local->ops->get_flush_stats(&local->hw, idx, survey);
270 + return ret;
271 +}
272 +
273 static inline void drv_rfkill_poll(struct ieee80211_local *local)
274 {
275     might_sleep();
276 --- a/net/wireless/nl80211.c
277 +++ b/net/wireless/nl80211.c
278 @@ -8051,6 +8051,92 @@ static int nl80211_dump_survey(struct sk
279     return res;
280 }
281
282 /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
283 +static int nl80211_send_flush_stats(struct sk_buff *msg, u32 portid,
    u32 seq,
284 +    int flags, struct net_device *dev,
285 +    struct flush_info *survey)
286 +{
287 + void *hdr;
288 + struct nlattr *infoattr;
289 +
290 + hdr = nl80211hdr_put(msg, portid, seq, flags,
```

```
291 +     NL80211_CMD_NEW_FLUSH_STATS);
292 + if (!hdr)
293 +     return -ENOMEM;
294 +
295 + if (nla_put_u32(msg, NL80211_ATTR_IFINDEX, dev->ifindex))
296 +     goto nla_put_failure;
297 +
298 + infoattr = nla_nest_start(msg, NL80211_ATTR_FLUSH_INFO);
299 + if (!infoattr)
300 +     goto nla_put_failure;
301 +
302 + if (nla_put_u32(msg, NL80211_FLUSH_REQ_BE, survey->be_flush_req))
303 +     goto nla_put_failure;
304 + if (nla_put_u32(msg, NL80211_FLUSH_NOT_REQ_BE, survey->
305 +     be_flush_not_req))
306 +     goto nla_put_failure;
307 + if (nla_put_u32(msg, NL80211_FLUSH_REQ_BK, survey->bk_flush_req))
308 +     goto nla_put_failure;
309 + if (nla_put_u32(msg, NL80211_FLUSH_NOT_REQ_BK, survey->
310 +     bk_flush_not_req))
311 +     goto nla_put_failure;
312 + if (nla_put_u32(msg, NL80211_FLUSH_REQ_VI, survey->vi_flush_req))
313 +     goto nla_put_failure;
314 + if (nla_put_u32(msg, NL80211_FLUSH_NOT_REQ_VI, survey->
315 +     vi_flush_not_req))
316 +     goto nla_put_failure;
317 + if (nla_put_u32(msg, NL80211_FLUSH_REQ_VO, survey->vo_flush_req))
318 +     goto nla_put_failure;
319 + if (nla_put_u32(msg, NL80211_FLUSH_NOT_REQ_VO, survey->
320 +     vo_flush_not_req))
321 +     goto nla_put_failure;
322 +
323 + int ret = nla_nest_end(msg, infoattr);
```

```
321 + genlmsg_end(msg, hdr);
322 + return ret;
323 +
324 +nla_put_failure:
325 + genlmsg_cancel(msg, hdr);
326 + return -EMSGSIZE;
327 +
328 +}
329 +
330 +static int nl80211_dump_flush_stats(struct sk_buff *skb,
331 +    struct netlink_callback *cb)
332 +{
333 +    struct flush_info survey;
334 +    struct cfg80211_registered_device *rdev;
335 +    struct wireless_dev *wdev;
336 +    int survey_idx = cb->args[2];
337 +    int res;
338 +
339 +    // printk(KERN_ALERT "%s:%d\n", __FILE__, __LINE__);
340 +    res = nl80211_prepare_wdev_dump(skb, cb, &rdev, &wdev);
341 +    if (res)
342 +        return res;
343 +
344 +
345 +    if (!rdev->ops->dump_flush_stats) {
346 +        res = -EOPNOTSUPP;
347 +        printk(KERN_ALERT "Operation not supported: %s:%d\n", __FILE__,
348 +            __LINE__);
349 +        goto out_err;
350 +    }
351 +
352 +    res = rdev_dump_flush_stats(rdev, wdev->netdev, survey_idx, &survey)
353 +        ;
354 +}
```

```
353 + if (nl80211_send_flush_stats(skb,
354 +     NETLINK_CB(cb->skb).portid,
355 +     cb->nlh->nlmsg_seq, NLM_F_MULTI,
356 +     wdev->netdev, &survey)) {
357 +     goto out;
358 + }
359 + out:
360 + cb->args[2] = survey_idx; // <- needed ???
361 +     res = skb->len;
362 + out_err:
363 + rtnl_unlock(); // Changed to reflect latest updates to nl80211.c
364 + printk(KERN_ALERT "returning %s:%d", __FILE__, __LINE__);
365 + return 0;
366 +}
367 +
368 static bool nl80211_valid_wpa_versions(u32 wpa_versions)
369 {
370     return !(wpa_versions & ~(NL80211_WPA_VERSION_1 |
371 @@ -13362,6 +13448,12 @@ static __genl_const struct genl_ops nl80
372     .internal_flags = NL80211_FLAG_NEED_NETDEV_UP |
373         NL80211_FLAG_NEED_RTNL,
374 },
375 + /* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
376     paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
377 + {
378 +     .cmd = NL80211_CMD_FLUSH_STATS,
379 +     .policy = nl80211_policy,
380 +     .dumpit = nl80211_dump_flush_stats
381 + },
382 };
383
384 --- a/net/wireless/rdev-ops.h
385 +++ b/net/wireless/rdev-ops.h
```

```
386 @@ -644,6 +644,18 @@ static inline int rdev_dump_survey(struc
387     return ret;
388 }
389
390 +/* MAC queues enabler patch by Gurjashan S. Pannu <gspannu@mail.uni-
391 +    paderborn.de>, date Wed, 27 Jul 2016 14:23:38 +0200 */
392 + /* notification functions */
393 +static inline int rdev_dump_flush_stats(struct
394 +    cfg80211_registered_device *rdev,
395 +    struct net_device *netdev, int idx,
396 +    struct flush_info *info)
397 +{
398 +    int ret;
399 +    ret = rdev->ops->dump_flush_stats(&rdev->wiphy, netdev, idx, info);
400 +    return ret;
401 +}
402 +
403 +static inline int rdev_set_pmksa(struct cfg80211_registered_device *
404 +    rdev,
405 +    struct net_device *netdev,
406 +    struct cfg80211_pmksa *pmksa)
```

999-ITS-G5D-channels-fix.patch

In package/kernel/mac80211 (custom)

```
1 --- a/drivers/net/wireless/ath/ath5k/base.c
2 +++ b/drivers/net/wireless/ath/ath5k/base.c
3 @@ -322,6 +322,23 @@ ath5k_setup_channels(struct ath5k_hw *ah
4     for (ch = 1; ch <= size && count < max; ch++) {
5         freq = ieee80211_channel_to_frequency(ch, band);
6
7 +     /* ITS-G5D fix: added by Politecnico di Torino to enable channels
8 +        182
9 +        and 184 on UNEX DCMA 86P2 cards */
10 +    if(ch == 182 || ch == 184) {
11 +        switch(ch) {
12 +            case 182:
13 +                freq=5910;
14 +                break;
15 +            case 184:
16 +                freq=5920;
17 +                break;
18 +            default:
19 +                break;
20 +        }
21 +    } else {
22 +        freq = ieee80211_channel_to_frequency(ch, band);
23 +    }
24
25     if (freq == 0) /* mapping failed - not a standard channel */
26         continue;
```

.bashrc (on the development PC) - modified lines

```
# PATH environmental variable settings for OpenWrt toolchain
PATH=$PATH:/home/francesco/openwrt-18.06.1/staging_dir/toolchain-
    i386_pentium_gcc-7.3.0_musl/bin/
export PATH

# Set STAGING_DIR environment variable for the toolchain
STAGING_DIR=/home/francesco/openwrt-18.06.1/staging_dir/toolchain-
    i386_pentium_gcc-7.3.0_musl/bin/
export STAGING_DIR

# Set alias for writing 'compileboard' instead of the gcc full name (
    OpenWrt toolchain)
alias compileboard="i486-openwrt-linux-musl-gcc"

# Function to compile for the PC Engines boards
compileboardstatic() {
    i486-openwrt-linux-musl-gcc -o "$1" -static "$1.c"
}
```

The name “*francesco*” refers to the name of the account in the development PC: it should be substituted with the current user name.

Appendix C

OpenWrt 18.06.1 configuration file for the APU boards

Only the actually selected options are shown (otherwise over 4800 lines and 100 pages would have been generated), by invoking:

```
grep -v .config -e "is not set" >> config.selected
```

over the *.config* file, not printing all the options which are not set.

config.selected (grep -v over .config)

```
#
# Automatically generated file; DO NOT EDIT.
# OpenWrt Configuration
#
CONFIG_MODULES=y
CONFIG_HAVE_DOT_CONFIG=y
CONFIG_TARGET_x86=y
CONFIG_TARGET_x86_64=y
CONFIG_TARGET_x86_64_Generic=y
CONFIG_HAS_SUBTARGETS=y
CONFIG_TARGET_BOARD="x86"
CONFIG_TARGET_SUBTARGET="64"
CONFIG_TARGET_PROFILE="Generic"
CONFIG_TARGET_ARCH_PACKAGES="x86_64"
CONFIG_DEFAULT_TARGET_OPTIMIZATION="-Os -pipe"
```



```
CONFIG_CPU_TYPE=" "
CONFIG_LINUX_4_14=y
CONFIG_DEFAULT_base-files=y
CONFIG_DEFAULT_busybox=y
CONFIG_DEFAULT_dnsmasq=y
CONFIG_DEFAULT_dropbear=y
CONFIG_DEFAULT_e2fsprogs=y
CONFIG_DEFAULT_firewall=y
CONFIG_DEFAULT_fstools=y
CONFIG_DEFAULT_ip6tables=y
CONFIG_DEFAULT_iptables=y
CONFIG_DEFAULT_kmod-button-hotplug=y
CONFIG_DEFAULT_kmod-e1000=y
CONFIG_DEFAULT_kmod-e1000e=y
CONFIG_DEFAULT_kmod-igb=y
CONFIG_DEFAULT_kmod-ipt-offload=y
CONFIG_DEFAULT_kmod-r8169=y
CONFIG_DEFAULT_libc=y
CONFIG_DEFAULT_libgcc=y
CONFIG_DEFAULT_logd=y
CONFIG_DEFAULT_mkf2fs=y
CONFIG_DEFAULT_mtd=y
CONFIG_DEFAULT_netifd=y
CONFIG_DEFAULT_odhcp6c=y
CONFIG_DEFAULT_odhcpd-ipv6only=y
CONFIG_DEFAULT_opkg=y
CONFIG_DEFAULT_partx-utils=y
CONFIG_DEFAULT_ppp=y
CONFIG_DEFAULT_ppp-mod-pppoe=y
CONFIG_DEFAULT_uci=y
CONFIG_DEFAULT_uclient-fetch=y
CONFIG_HAS_FPU=y
CONFIG_AUDIO_SUPPORT=y
CONFIG_GPIO_SUPPORT=y
```

```
CONFIG_PCI_SUPPORT=y
CONFIG_PCIE_SUPPORT=y
CONFIG_PCMCIA_SUPPORT=y
CONFIG_USB_SUPPORT=y
CONFIG_RTC_SUPPORT=y
CONFIG_USES_SQUASHFS=y
CONFIG_USES_EXT4=y
CONFIG_USES_TARGZ=y
CONFIG_ARCH_64BIT=y
CONFIG_VIRTIO_SUPPORT=y
CONFIG_x86_64=y
CONFIG_ARCH="x86_64"

#
# Target Images
#
CONFIG_EXTERNAL_CPIO=""

#
# Root filesystem archives
#

#
# Root filesystem images
#
CONFIG_TARGET_ROOTFS_EXT4FS=y
CONFIG_TARGET_EXT4_RESERVED_PCT=0
CONFIG_TARGET_EXT4_BLOCKSIZE_4K=y
CONFIG_TARGET_EXT4_BLOCKSIZE=4096
CONFIG_TARGET_ROOTFS_SQUASHFS=y
CONFIG_TARGET_SQUASHFS_BLOCK_SIZE=256
CONFIG_TARGET_UBIFS_FREE_SPACE_FIXUP=y
CONFIG_TARGET_UBIFS_JOURNAL_SIZE=""
CONFIG_GRUB_IMAGES=y
```

```
CONFIG_GRUB_CONSOLE=y
CONFIG_GRUB_SERIAL="ttyS0"
CONFIG_GRUB_BAUDRATE=115200
CONFIG_GRUB_BOOTOPTS=""
CONFIG_GRUB_TIMEOUT="5"
CONFIG_TARGET_IMAGES_GZIP=y

#
# Image Options
#
CONFIG_TARGET_KERNEL_PARTSIZE=20
CONFIG_TARGET_ROOTFS_PARTSIZE=3960
CONFIG_TARGET_ROOTFS_PARTNAME=""

#
# Global build settings
#
CONFIG_SIGNED_PACKAGES=y

#
# General build options
#
CONFIG_DISPLAY_SUPPORT=y
CONFIG_SHADOW_PASSWORDS=y

#
# Kernel build options
#
CONFIG_KERNEL_BUILD_USER=""
CONFIG_KERNEL_BUILD_DOMAIN=""
CONFIG_KERNEL_PRINTK=y
CONFIG_KERNEL_SWAP=y
CONFIG_KERNEL_DEBUG_FS=y
CONFIG_KERNEL_KALLSYMS=y
```

```
CONFIG_KERNEL_DEBUG_KERNEL=y
CONFIG_KERNEL_DEBUG_INFO=y
CONFIG_KERNEL_MAGIC_SYSRQ=y
CONFIG_KERNEL_COREDUMP=y
CONFIG_KERNEL_ELF_CORE=y
CONFIG_KERNEL_PRINTK_TIME=y
CONFIG_KERNEL_KEXEC=y
CONFIG_KERNEL_PROC_VMCORE=y
CONFIG_KERNEL_CRASH_DUMP=y
CONFIG_KERNEL_IP_MROUTE=y
CONFIG_KERNEL_IPV6=y
CONFIG_KERNEL_IPV6_MULTIPLE_TABLES=y
CONFIG_KERNEL_IPV6_SUBTREES=y
CONFIG_KERNEL_IPV6_MROUTE=y

#
# Filesystem ACL and attr support options
#

#
# Package build options
#
CONFIG_IPV6=y

#
# Stripping options
#
CONFIG_USE_SSTRIP=y
CONFIG_USE_UCLIBCXX=y

#
# Hardening build options
#
CONFIG_PKG_CHECK_FORMAT_SECURITY=y
```

```
CONFIG_PKG_CC_STACKPROTECTOR_REGULAR=y
CONFIG_KERNEL_CC_STACKPROTECTOR_REGULAR=y
CONFIG_PKG_FORTIFY_SOURCE_1=y
CONFIG_PKG_RELRO_FULL=y
CONFIG_DEVEL=y
CONFIG_BINARY_FOLDER=""
CONFIG_DOWNLOAD_FOLDER=""
CONFIG_LOCALMIRROR=""
CONFIG_AUTOREBUILD=y
CONFIG_BUILD_SUFFIX=""
CONFIG_TARGET_ROOTFS_DIR=""
CONFIG_EXTERNAL_KERNEL_TREE=""
CONFIG_KERNEL_GIT_CLONE_URI=""
CONFIG_EXTRA_OPTIMIZATION="-fno-caller-saves -fno-plt"
CONFIG_TARGET_OPTIMIZATION="-Os -pipe"
CONFIG_NEED_TOOLCHAIN=y
CONFIG_EXTRA_BINUTILS_CONFIG_OPTIONS=""
CONFIG_EXTRA_GCC_CONFIG_OPTIONS=""
CONFIG_YASM=y
CONFIG_GDB=y
CONFIG_USE_MUSL=y
CONFIG_SSP_SUPPORT=y
CONFIG_BINUTILS_VERSION_2_29_1=y
CONFIG_BINUTILS_VERSION="2.29.1"
CONFIG_GCC_VERSION="7.3.0"
CONFIG_LIBC="musl"
CONFIG_TARGET_SUFFIX="musl"
CONFIG_TARGET_PREINIT_SUPPRESS_STDERR=y
CONFIG_TARGET_PREINIT_TIMEOUT=2
CONFIG_TARGET_PREINIT_IFNAME=""
CONFIG_TARGET_PREINIT_IP="192.168.1.1"
CONFIG_TARGET_PREINIT_NETMASK="255.255.255.0"
CONFIG_TARGET_PREINIT_BROADCAST="192.168.1.255"
CONFIG_TARGET_INIT_PATH="/usr/sbin:/usr/bin:/sbin:/bin"
```

```
CONFIG_TARGET_INIT_ENV=""
CONFIG_TARGET_INIT_CMD="/sbin/init"
CONFIG_TARGET_INIT_SUPPRESS_STDERR=y
CONFIG_PER_FEED_REPO=y
CONFIG_FEED_packages=y
CONFIG_FEED_luci=y
CONFIG_FEED_routing=y
CONFIG_FEED_telephony=y

#
# Base system
#
CONFIG_PACKAGE_base-files=y
CONFIG_PACKAGE_block-mount=m
CONFIG_PACKAGE_busybox=y
CONFIG_BUSYBOX_DEFAULT_HAVE_DOT_CONFIG=y
CONFIG_BUSYBOX_DEFAULT_INCLUDE_SUSv2=y
CONFIG_BUSYBOX_DEFAULT_LONG_OPTS=y
CONFIG_BUSYBOX_DEFAULT_SHOW_USAGE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VERBOSE_USAGE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_COMPRESS_USAGE=y
CONFIG_BUSYBOX_DEFAULT_LFS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DEVPTS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PIDFILE=y
CONFIG_BUSYBOX_DEFAULT_PID_FILE_PATH="/var/run"
CONFIG_BUSYBOX_DEFAULT_FEATURE_SUID=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PREFER_APPLETS=y
CONFIG_BUSYBOX_DEFAULT_BUSYBOX_EXEC_PATH="/proc/self/exe"
CONFIG_BUSYBOX_DEFAULT_FEATURE_SYSLOG=y
CONFIG_BUSYBOX_DEFAULT_PLATFORM_LINUX=y
CONFIG_BUSYBOX_DEFAULT_CROSS_COMPILER_PREFIX=""
CONFIG_BUSYBOX_DEFAULT_SYSROOT=""
CONFIG_BUSYBOX_DEFAULT_EXTRA_CFLAGS=""
CONFIG_BUSYBOX_DEFAULT_EXTRA_LDFLAGS=""
```

```
CONFIG_BUSYBOX_DEFAULT_EXTRA_LDLIBS=""
CONFIG_BUSYBOX_DEFAULT_INSTALL_APPLET_SYMLINKS=y
CONFIG_BUSYBOX_DEFAULT_PREFIX="./_install"
CONFIG_BUSYBOX_DEFAULT_NO_DEBUG_LIB=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_BUFFERS_GO_ON_STACK=y
CONFIG_BUSYBOX_DEFAULT_PASSWORD_MINLEN=6
CONFIG_BUSYBOX_DEFAULT_MD5_SMALL=1
CONFIG_BUSYBOX_DEFAULT_SHA3_SMALL=1
CONFIG_BUSYBOX_DEFAULT_FEATURE_FAST_TOP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_EDITING=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_EDITING_MAX_LEN=512
CONFIG_BUSYBOX_DEFAULT_FEATURE_EDITING_HISTORY=256
CONFIG_BUSYBOX_DEFAULT_FEATURE_TAB_COMPLETION=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_EDITING_FANCY_PROMPT=y
CONFIG_BUSYBOX_DEFAULT_SUBST_WCHAR=0
CONFIG_BUSYBOX_DEFAULT_LAST_SUPPORTED_WCHAR=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_NON_POSIX_CP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_USE_SENDFILE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_COPYBUF_KB=4
CONFIG_BUSYBOX_DEFAULT_IOCTL_HEX2STR_ERROR=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_SEAMLESS_GZ=y
CONFIG_BUSYBOX_DEFAULT_GUNZIP=y
CONFIG_BUSYBOX_DEFAULT_ZCAT=y
CONFIG_BUSYBOX_DEFAULT_BUNZIP2=y
CONFIG_BUSYBOX_DEFAULT_BZCAT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_BZIP2_DECOMPRESS=y
CONFIG_BUSYBOX_DEFAULT_GZIP=y
CONFIG_BUSYBOX_DEFAULT_GZIP_FAST=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_GZIP_DECOMPRESS=y
CONFIG_BUSYBOX_DEFAULT_TAR=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TAR_CREATE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TAR_FROM=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TAR_GNU_EXTENSIONS=y
CONFIG_BUSYBOX_DEFAULT_BASENAME=y
```

```
CONFIG_BUSYBOX_DEFAULT_CAT=y
CONFIG_BUSYBOX_DEFAULT_CHGRP=y
CONFIG_BUSYBOX_DEFAULT_CHMOD=y
CONFIG_BUSYBOX_DEFAULT_CHOWN=y
CONFIG_BUSYBOX_DEFAULT_CHROOT=y
CONFIG_BUSYBOX_DEFAULT_CP=y
CONFIG_BUSYBOX_DEFAULT_CUT=y
CONFIG_BUSYBOX_DEFAULT_DATE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DATE_ISOFORMAT=y
CONFIG_BUSYBOX_DEFAULT_DD=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DD_SIGNAL_HANDLING=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DD_IBS_OBS=y
CONFIG_BUSYBOX_DEFAULT_DF=y
CONFIG_BUSYBOX_DEFAULT_DIRNAME=y
CONFIG_BUSYBOX_DEFAULT_DU=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DU_DEFAULT_BLOCKSIZE_1K=y
CONFIG_BUSYBOX_DEFAULT_ECHO=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FANCY_ECHO=y
CONFIG_BUSYBOX_DEFAULT_ENV=y
CONFIG_BUSYBOX_DEFAULT_EXPR=y
CONFIG_BUSYBOX_DEFAULT_EXPR_MATH_SUPPORT_64=y
CONFIG_BUSYBOX_DEFAULT_FALSE=y
CONFIG_BUSYBOX_DEFAULT_FSYNC=y
CONFIG_BUSYBOX_DEFAULT_HEAD=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FANCY_HEAD=y
CONFIG_BUSYBOX_DEFAULT_ID=y
CONFIG_BUSYBOX_DEFAULT_LN=y
CONFIG_BUSYBOX_DEFAULT_LS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_FILETYPES=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_FOLLOWLINKS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_RECURSIVE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_WIDTH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_SORTFILES=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_TIMESTAMPS=y
```



```
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_USERNAME=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_COLOR=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LS_COLOR_IS_DEFAULT=y
CONFIG_BUSYBOX_DEFAULT_MD5SUM=y
CONFIG_BUSYBOX_DEFAULT_SHA256SUM=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MD5_SHA1_SUM_CHECK=y
CONFIG_BUSYBOX_DEFAULT_MKDIR=y
CONFIG_BUSYBOX_DEFAULT_MKFIFO=y
CONFIG_BUSYBOX_DEFAULT_MKNOD=y
CONFIG_BUSYBOX_DEFAULT_MKTEMP=y
CONFIG_BUSYBOX_DEFAULT_MV=y
CONFIG_BUSYBOX_DEFAULT_NICE=y
CONFIG_BUSYBOX_DEFAULT_PRINTF=y
CONFIG_BUSYBOX_DEFAULT_PWD=y
CONFIG_BUSYBOX_DEFAULT_READLINK=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_READLINK_FOLLOW=y
CONFIG_BUSYBOX_DEFAULT_RM=y
CONFIG_BUSYBOX_DEFAULT_RMDIR=y
CONFIG_BUSYBOX_DEFAULT_SEQ=y
CONFIG_BUSYBOX_DEFAULT_SLEEP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FANCY_SLEEP=y
CONFIG_BUSYBOX_DEFAULT_SORT=y
CONFIG_BUSYBOX_DEFAULT_SYNC=y
CONFIG_BUSYBOX_DEFAULT_TAIL=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FANCY_TAIL=y
CONFIG_BUSYBOX_DEFAULT_TEE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TEE_USE_BLOCK_IO=y
CONFIG_BUSYBOX_DEFAULT_TEST=y
CONFIG_BUSYBOX_DEFAULT_TEST1=y
CONFIG_BUSYBOX_DEFAULT_TEST2=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TEST_64=y
CONFIG_BUSYBOX_DEFAULT_TOUCH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TOUCH_SUSV3=y
CONFIG_BUSYBOX_DEFAULT_TR=y
```

```
CONFIG_BUSYBOX_DEFAULT_TRUE=y
CONFIG_BUSYBOX_DEFAULT_UNAME=y
CONFIG_BUSYBOX_DEFAULT_UNAME_OSNAME="GNU/Linux"
CONFIG_BUSYBOX_DEFAULT_UNIQ=y
CONFIG_BUSYBOX_DEFAULT_WC=y
CONFIG_BUSYBOX_DEFAULT_YES=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PRESERVE_HARDLINKS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_HUMAN_READABLE=y
CONFIG_BUSYBOX_DEFAULT_CLEAR=y
CONFIG_BUSYBOX_DEFAULT_DEFAULT_SETFONT_DIR=""
CONFIG_BUSYBOX_DEFAULT_RESET=y
CONFIG_BUSYBOX_DEFAULT_START_STOP_DAEMON=y
CONFIG_BUSYBOX_DEFAULT_WHICH=y
CONFIG_BUSYBOX_DEFAULT_AWK=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_AWK_LIBM=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_AWK_GNU_EXTENSIONS=y
CONFIG_BUSYBOX_DEFAULT_CMP=y
CONFIG_BUSYBOX_DEFAULT_SED=y
CONFIG_BUSYBOX_DEFAULT_VI=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_MAX_LEN=1024
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_COLON=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_YANKMARK=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_SEARCH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_USE_SIGNALS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_DOT_CMD=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_READONLY=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_SETOPTS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_SET=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_WIN_RESIZE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_ASK_TERMINAL=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_VI_UNDO_QUEUE_MAX=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_ALLOW_EXEC=y
CONFIG_BUSYBOX_DEFAULT_FIND=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_PRINTO=y
```

```
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_MTIME=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_PERM=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_TYPE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_XDEV=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_MAXDEPTH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_NEWER=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_EXEC=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_USER=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_GROUP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_NOT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_DEPTH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_PAREN=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_SIZE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_PRUNE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_PATH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FIND_REGEX=y
CONFIG_BUSYBOX_DEFAULT_GREP=y
CONFIG_BUSYBOX_DEFAULT_EGREP=y
CONFIG_BUSYBOX_DEFAULT_FGREP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_GREP_CONTEXT=y
CONFIG_BUSYBOX_DEFAULT_XARGS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_XARGS_SUPPORT_CONFIRMATION=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_XARGS_SUPPORT_QUOTES=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_XARGS_SUPPORT_TERMOPT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_XARGS_SUPPORT_ZERO_TERM=y
CONFIG_BUSYBOX_DEFAULT_HALT=y
CONFIG_BUSYBOX_DEFAULT_POWEROFF=y
CONFIG_BUSYBOX_DEFAULT_REBOOT=y
CONFIG_BUSYBOX_DEFAULT_TELINIT_PATH=""
CONFIG_BUSYBOX_DEFAULT_FEATURE_KILL_DELAY=0
CONFIG_BUSYBOX_DEFAULT_INIT_TERMINAL_TYPE=""
CONFIG_BUSYBOX_DEFAULT_FEATURE_SHADOWPASSWDS=y
CONFIG_BUSYBOX_DEFAULT_LAST_ID=0
CONFIG_BUSYBOX_DEFAULT_FIRST_SYSTEM_ID=0
```

```
CONFIG_BUSYBOX_DEFAULT_LAST_SYSTEM_ID=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_DEFAULT_PASSWD_ALGO="md5"
CONFIG_BUSYBOX_DEFAULT_LOGIN=y
CONFIG_BUSYBOX_DEFAULT_LOGIN_SESSION_AS_CHILD=y
CONFIG_BUSYBOX_DEFAULT_PASSWD=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PASSWD_WEAK_CHECK=y
CONFIG_BUSYBOX_DEFAULT_DEFAULT_MODULES_DIR=""
CONFIG_BUSYBOX_DEFAULT_DEFAULT_DEPMOD_FILE=""
CONFIG_BUSYBOX_DEFAULT_DMESG=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_DMESG_PRETTY=y
CONFIG_BUSYBOX_DEFAULT_FLOCK=y
CONFIG_BUSYBOX_DEFAULT_HEXDUMP=y
CONFIG_BUSYBOX_DEFAULT_HWCLOCK=y
CONFIG_BUSYBOX_DEFAULT_MKSWAP=y
CONFIG_BUSYBOX_DEFAULT_MOUNT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MOUNT_HELPERS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MOUNT_CIFS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MOUNT_FLAGS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MOUNT_FSTAB=y
CONFIG_BUSYBOX_DEFAULT_PIVOT_ROOT=y
CONFIG_BUSYBOX_DEFAULT_SWITCH_ROOT=y
CONFIG_BUSYBOX_DEFAULT_UMOUNT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_UMOUNT_ALL=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_MOUNT_LOOP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_BEEP_FREQ=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_BEEP_LENGTH_MS=0
CONFIG_BUSYBOX_DEFAULT_CROND=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_CROND_DIR="/etc"
CONFIG_BUSYBOX_DEFAULT_CRONTAB=y
CONFIG_BUSYBOX_DEFAULT_LESS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_LESS_MAXLINES=9999999
CONFIG_BUSYBOX_DEFAULT_LOCK=y
CONFIG_BUSYBOX_DEFAULT_STRINGS=y
CONFIG_BUSYBOX_DEFAULT_TIME=y
```

```
CONFIG_BUSYBOX_DEFAULT_FEATURE_IPV6=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PREFER_IPV4_ADDRESS=y
CONFIG_BUSYBOX_DEFAULT_VERBOSE_RESOLUTION_ERRORS=y
CONFIG_BUSYBOX_DEFAULT_BRCTL=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_BRCTL_FANCY=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_BRCTL_SHOW=y
CONFIG_BUSYBOX_DEFAULT_IFCONFIG=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IFCONFIG_STATUS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IFCONFIG_HW=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IFCONFIG_BROADCAST_PLUS=y
CONFIG_BUSYBOX_DEFAULT_IFUPDOWN_IFSTATE_PATH=""
CONFIG_BUSYBOX_DEFAULT_IP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_ADDRESS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_LINK=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_ROUTE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_ROUTE_DIR="/etc/iproute2"
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_RULE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_IP_NEIGH=y
CONFIG_BUSYBOX_DEFAULT_NC=y
CONFIG_BUSYBOX_DEFAULT_NETMSG=y
CONFIG_BUSYBOX_DEFAULT_NETSTAT=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_NETSTAT_WIDE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_NETSTAT_PRG=y
CONFIG_BUSYBOX_DEFAULT_NSLOOKUP_OPENWRT=y
CONFIG_BUSYBOX_DEFAULT_NTPD=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_NTPD_SERVER=y
CONFIG_BUSYBOX_DEFAULT_PING=y
CONFIG_BUSYBOX_DEFAULT_PING6=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_FANCY_PING=y
CONFIG_BUSYBOX_DEFAULT_ROUTE=y
CONFIG_BUSYBOX_DEFAULT_TRACEROUTE=y
CONFIG_BUSYBOX_DEFAULT_TRACEROUTE6=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TRACEROUTE_VERBOSE=y
CONFIG_BUSYBOX_DEFAULT_DHCPD_LEASES_FILE=""
```

```
CONFIG_BUSYBOX_DEFAULT_UDHCPC=y
CONFIG_BUSYBOX_DEFAULT_UDHCPC_DEFAULT_SCRIPT="/usr/share/udhcp/
    default.script"
CONFIG_BUSYBOX_DEFAULT_UDHCP_DEBUG=0
CONFIG_BUSYBOX_DEFAULT_UDHCPC_SLACK_FOR_BUGGY_SERVERS=80
CONFIG_BUSYBOX_DEFAULT_FEATURE_UDHCP_RFC3397=y
CONFIG_BUSYBOX_DEFAULT_IFUPDOWN_UDHCPC_CMD_OPTIONS=""
CONFIG_BUSYBOX_DEFAULT_FEATURE_MIME_CHARSET=""
CONFIG_BUSYBOX_DEFAULT_FREE=y
CONFIG_BUSYBOX_DEFAULT_KILL=y
CONFIG_BUSYBOX_DEFAULT_KILLALL=y
CONFIG_BUSYBOX_DEFAULT_PGREP=y
CONFIG_BUSYBOX_DEFAULT_PIDOF=y
CONFIG_BUSYBOX_DEFAULT_PS=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_PS_WIDE=y
CONFIG_BUSYBOX_DEFAULT_BB_SYSCTL=y
CONFIG_BUSYBOX_DEFAULT_TOP=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TOP_CPU_USAGE_PERCENTAGE=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_TOP_CPU_GLOBAL_PERCENTS=y
CONFIG_BUSYBOX_DEFAULT_UPTIME=y
CONFIG_BUSYBOX_DEFAULT_SV_DEFAULT_SERVICE_DIR=""
CONFIG_BUSYBOX_DEFAULT_SH_IS_ASH=y
CONFIG_BUSYBOX_DEFAULT_BASH_IS_NONE=y
CONFIG_BUSYBOX_DEFAULT_ASH=y
CONFIG_BUSYBOX_DEFAULT_ASH_INTERNAL_GLOB=y
CONFIG_BUSYBOX_DEFAULT_ASH_BASH_COMPAT=y
CONFIG_BUSYBOX_DEFAULT_ASH_JOB_CONTROL=y
CONFIG_BUSYBOX_DEFAULT_ASH_ALIAS=y
CONFIG_BUSYBOX_DEFAULT_ASH_EXPAND_PRMT=y
CONFIG_BUSYBOX_DEFAULT_ASH_ECHO=y
CONFIG_BUSYBOX_DEFAULT_ASH_PRINTF=y
CONFIG_BUSYBOX_DEFAULT_ASH_TEST=y
CONFIG_BUSYBOX_DEFAULT_ASH_GETOPTS=y
CONFIG_BUSYBOX_DEFAULT_ASH_CMDCMD=y
```

```
CONFIG_BUSYBOX_DEFAULT_FEATURE_SH_MATH=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_SH_MATH_64=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_SH_NOFORK=y
CONFIG_BUSYBOX_DEFAULT_LOGGER=y
CONFIG_BUSYBOX_DEFAULT_FEATURE_SYSLOGD_READ_BUFFER_SIZE=0
CONFIG_BUSYBOX_DEFAULT_FEATURE_IPC_SYSLOG_BUFFER_SIZE=0
CONFIG_PACKAGE_dnsmasq=y
CONFIG_PACKAGE_dropbear=y

#
# Configuration
#
CONFIG_DROPBEAR_CURVE25519=y
CONFIG_PACKAGE_ead=m
CONFIG_PACKAGE_firewall=y
CONFIG_PACKAGE_fstools=y
CONFIG_PACKAGE_fwtool=y
CONFIG_PACKAGE_jsonfilter=y
CONFIG_PACKAGE_libc=y
CONFIG_PACKAGE_libgcc=y
CONFIG_PACKAGE_libpthread=y
CONFIG_PACKAGE_librt=y
CONFIG_PACKAGE_libstdcpp=m
CONFIG_PACKAGE_logd=y
CONFIG_PACKAGE_mtd=y
CONFIG_PACKAGE_netifd=y
CONFIG_PACKAGE_om-watchdog=m
CONFIG_PACKAGE_openwrt-keyring=y
CONFIG_PACKAGE_opkg=y
CONFIG_PACKAGE_procd=y

#
# Configuration
#
```

```
CONFIG_PACKAGE_rpcd=y
CONFIG_PACKAGE_ubox=y
CONFIG_PACKAGE_ubus=y
CONFIG_PACKAGE_ubusd=y
CONFIG_PACKAGE_uci=y
CONFIG_PACKAGE_usign=y

#
# Administration
#

#
# openwisp
#

#
# zabbix
#

#
# Boot Loaders
#
CONFIG_PACKAGE_grub2=y

#
# Development
#

#
# Libraries
#
CONFIG_PACKAGE_ar=y
CONFIG_PACKAGE_binutils=y
CONFIG_PACKAGE_objdump=y
```



```
#
# Extra packages
#

#
# Firmware
#

#
# ath10k IPQ4019 Boarddata
#
CONFIG_PACKAGE_ath9k-htc-firmware=y
CONFIG_PACKAGE_r8169-firmware=y
CONFIG_PACKAGE_wireless-regdb=y

#
# Fonts
#

#
# DeJaVu
#

#
# Kernel modules
#

#
# Block Devices
#
CONFIG_PACKAGE_kmod-ata-core=y
CONFIG_PACKAGE_kmod-ata-ahci=y
CONFIG_PACKAGE_kmod-scsi-core=y
```

```
#
# CAN Support
#

#
# Cryptographic API modules
#
CONFIG_PACKAGE_kmod-crypto-acompress=m
CONFIG_PACKAGE_kmod-crypto-aead=y
CONFIG_PACKAGE_kmod-crypto-cbc=y
CONFIG_PACKAGE_kmod-crypto-crc32c=y
CONFIG_PACKAGE_kmod-crypto-des=m
CONFIG_PACKAGE_kmod-crypto-ecb=y
CONFIG_PACKAGE_kmod-crypto-fcrypt=m
CONFIG_PACKAGE_kmod-crypto-hash=y
CONFIG_PACKAGE_kmod-crypto-hmac=m
CONFIG_PACKAGE_kmod-crypto-hw-geode=y
CONFIG_PACKAGE_kmod-crypto-manager=y
CONFIG_PACKAGE_kmod-crypto-md4=m
CONFIG_PACKAGE_kmod-crypto-md5=m
CONFIG_PACKAGE_kmod-crypto-null=y
CONFIG_PACKAGE_kmod-crypto-pcbc=m
CONFIG_PACKAGE_kmod-crypto-pcompress=y
CONFIG_PACKAGE_kmod-crypto-sha1=y
CONFIG_PACKAGE_kmod-crypto-sha256=m

#
# Filesystems
#
CONFIG_PACKAGE_kmod-fs-afs=m
CONFIG_PACKAGE_kmod-fs-autofs4=m
CONFIG_PACKAGE_kmod-fs-btrfs=m
CONFIG_PACKAGE_kmod-fs-cifs=m
```

```
CONFIG_PACKAGE_kmod-fs-configfs=m
CONFIG_PACKAGE_kmod-fs-cramfs=m
CONFIG_PACKAGE_kmod-fs-exportfs=m
CONFIG_PACKAGE_kmod-fs-ext4=y
CONFIG_PACKAGE_kmod-fs-fscache=m
CONFIG_PACKAGE_kmod-fs-jfs=m
CONFIG_PACKAGE_kmod-fs-minix=m
CONFIG_PACKAGE_kmod-fs-msdos=m
CONFIG_PACKAGE_kmod-fs-nfs=m
CONFIG_PACKAGE_kmod-fs-nfs-common=m
CONFIG_PACKAGE_kmod-fs-nfs-v3=m
CONFIG_PACKAGE_kmod-fs-nfs-v4=m
CONFIG_PACKAGE_kmod-fs-nfsd=m
CONFIG_PACKAGE_kmod-fs-ntfs=m
CONFIG_PACKAGE_kmod-fs-reiserfs=m
CONFIG_PACKAGE_kmod-fs-udf=m
CONFIG_PACKAGE_kmod-fs-vfat=m
CONFIG_PACKAGE_kmod-fs-xfs=m
CONFIG_PACKAGE_kmod-fuse=m

#
# FireWire support
#

#
# Hardware Monitoring Support
#
CONFIG_PACKAGE_kmod-hwmon-core=y
CONFIG_PACKAGE_kmod-hwmon-lm90=y

#
# I2C support
#
CONFIG_PACKAGE_kmod-i2c-core=y
```

```
CONFIG_PACKAGE_kmod-i2c-algo-bit=y
CONFIG_PACKAGE_kmod-i2c-piix4=y

#
# Industrial I/O Modules
#

#
# Input modules
#
CONFIG_PACKAGE_kmod-hid=y
CONFIG_PACKAGE_kmod-hid-generic=y
CONFIG_PACKAGE_kmod-input-core=y
CONFIG_PACKAGE_kmod-input-evdev=y

#
# LED modules
#
CONFIG_PACKAGE_kmod-leds-gpio=y
CONFIG_PACKAGE_kmod-ledtrig-gpio=y
CONFIG_PACKAGE_kmod-ledtrig-heartbeat=y
CONFIG_PACKAGE_kmod-ledtrig-netdev=y

#
# Libraries
#
CONFIG_PACKAGE_kmod-lib-crc-ccitt=y
CONFIG_PACKAGE_kmod-lib-crc-itu-t=m
CONFIG_PACKAGE_kmod-lib-crc16=y
CONFIG_PACKAGE_kmod-lib-crc32c=m
CONFIG_PACKAGE_kmod-lib-lzo=m
CONFIG_PACKAGE_kmod-lib-raid6=m
CONFIG_PACKAGE_kmod-lib-xor=m
CONFIG_PACKAGE_kmod-lib-zlib-deflate=m
```

```
CONFIG_PACKAGE_kmod-lib-zlib-inflate=m
CONFIG_PACKAGE_kmod-lib-zstd=m

#
# Native Language Support
#
CONFIG_PACKAGE_kmod-nls-base=y
CONFIG_PACKAGE_kmod-nls-cp437=m
CONFIG_PACKAGE_kmod-nls-iso8859-1=m
CONFIG_PACKAGE_kmod-nls-utf8=m

#
# Netfilter Extensions
#
CONFIG_PACKAGE_kmod-ip6tables=y
CONFIG_PACKAGE_kmod-ipt-contrack=y
CONFIG_PACKAGE_kmod-ipt-core=y
CONFIG_PACKAGE_kmod-ipt-nat=y
CONFIG_PACKAGE_kmod-ipt-offload=y
CONFIG_PACKAGE_kmod-ipt-tee=y
CONFIG_PACKAGE_kmod-nf-contrack=y
CONFIG_PACKAGE_kmod-nf-contrack6=y
CONFIG_PACKAGE_kmod-nf-flow=y
CONFIG_PACKAGE_kmod-nf-ipt=y
CONFIG_PACKAGE_kmod-nf-ipt6=y
CONFIG_PACKAGE_kmod-nf-nat=y
CONFIG_PACKAGE_kmod-nf-reject=y
CONFIG_PACKAGE_kmod-nf-reject6=y

#
# Network Devices
#
CONFIG_PACKAGE_kmod-8139cp=y
CONFIG_PACKAGE_kmod-e1000e=y
```

```
CONFIG_PACKAGE_kmod-igb=y
CONFIG_PACKAGE_kmod-mii=y
CONFIG_PACKAGE_kmod-r8169=y
CONFIG_PACKAGE_kmod-solos-pci=y
CONFIG_PACKAGE_kmod-via-rhine=y

#
# Network Support
#
CONFIG_PACKAGE_kmod-atm=y
CONFIG_PACKAGE_kmod-dnsresolver=m
CONFIG_PACKAGE_kmod-ppp=y
CONFIG_PACKAGE_kmod-mppe=y
CONFIG_PACKAGE_kmod-pppoe=y
CONFIG_PACKAGE_kmod-pppox=y
CONFIG_PACKAGE_kmod-rxrpc=m
CONFIG_PACKAGE_kmod-sched-core=y
CONFIG_PACKAGE_kmod-slhc=y

#
# Other modules
#
CONFIG_PACKAGE_kmod-button-hotplug=y
CONFIG_PACKAGE_kmod-pps=y
CONFIG_PACKAGE_kmod-pty=y
CONFIG_PACKAGE_kmod-sp5100_tco=y

#
# PCMCIA support
#

#
# SPI Support
```

```
#

#
# Sound Support
#

#
# USB Support
#
CONFIG_PACKAGE_kmod-usb-acm=y
CONFIG_PACKAGE_kmod-usb-core=y
CONFIG_PACKAGE_kmod-usb-ehci=y
CONFIG_PACKAGE_kmod-usb-hid=y
CONFIG_PACKAGE_kmod-usb-ohci=y
CONFIG_PACKAGE_kmod-usb-ohci-pci=y
CONFIG_PACKAGE_kmod-usb-serial=y
CONFIG_PACKAGE_kmod-usb-serial-cp210x=y
CONFIG_PACKAGE_kmod-usb-serial-ftdi=y
CONFIG_PACKAGE_kmod-usb-serial-pl2303=y
CONFIG_PACKAGE_kmod-usb-storage=y
CONFIG_PACKAGE_kmod-usb-storage-extras=y
CONFIG_PACKAGE_kmod-usb-uhci=y
CONFIG_PACKAGE_kmod-usb2=y
CONFIG_PACKAGE_kmod-usb2-pci=y
CONFIG_PACKAGE_kmod-usb3=y

#
# Video Support
#

#
# Virtualization
#
```

```
#
# Voice over IP
#

#
# W1 support
#

#
# WPAN 802.15.4 Support
#

#
# Wireless Drivers
#
CONFIG_PACKAGE_kmod-ath=y
CONFIG_ATH_USER_REGD=y
CONFIG_PACKAGE_ATH_DEBUG=y
CONFIG_PACKAGE_ATH_DFS=y
CONFIG_PACKAGE_kmod-ath5k=y
CONFIG_PACKAGE_kmod-ath9k=y
CONFIG_ATH9K_SUPPORT_PCOEM=y
CONFIG_PACKAGE_kmod-ath9k-common=y
CONFIG_PACKAGE_kmod-cfg80211=y
CONFIG_PACKAGE_kmod-mac80211=y
CONFIG_PACKAGE_MAC80211_DEBUGFS=y
CONFIG_PACKAGE_MAC80211_MESH=y

#
# Languages
#

#
# Erlang
```



```
#

#
# Go
#

#
# Java
#

#
# Lua
#
CONFIG_PACKAGE_libiwininfo-lua=y
CONFIG_PACKAGE_lua=y

#
# Node.js
#

#
# Module Selection
#

#
# PHP
#

#
# Perl
#

#
# Python
```

```
#  
  
#  
# Configuration  
#  
  
#  
# Configuration  
#  
  
#  
# Ruby  
#  
  
#  
# Tcl  
#  
  
#  
# Libraries  
#  
  
#  
# Compression  
#  
  
#  
# Filesystem  
#  
CONFIG_PACKAGE_libsysfs=y  
  
#  
# Firewall  
#
```

```
CONFIG_PACKAGE_libip4tc=y
CONFIG_PACKAGE_libip6tc=y
CONFIG_PACKAGE_libxtables=y

#
# Instant Messaging
#

#
# IoT
#

#
# Languages
#

#
# Networking
#

#
# SSL
#
CONFIG_PACKAGE_libopenssl=y
CONFIG_OPENSSL_WITH_EC=y
CONFIG_OPENSSL_WITH_DEPRECATED=y
CONFIG_OPENSSL_WITH_NPN=y
CONFIG_OPENSSL_WITH_PSK=y
CONFIG_OPENSSL_WITH_SRP=y

#
# Sound
#
```

```
#
# Telephony
#

#
# database
#

#
# libelektra
#
CONFIG_PACKAGE_libbfd=y
CONFIG_PACKAGE_libblkid=y
CONFIG_PACKAGE_libblobmsg-json=y
CONFIG_PACKAGE_libcap=y
CONFIG_PACKAGE_libcomerr=y
CONFIG_PACKAGE_libext2fs=y
CONFIG_PACKAGE_libf2fs=y
CONFIG_PACKAGE_libftdi1=y
CONFIG_PACKAGE_libgps=y
CONFIG_PACKAGE_libiwinio=y
CONFIG_PACKAGE_libjson-c=y
CONFIG_PACKAGE_libkmod=y
CONFIG_PACKAGE_liblua=y
CONFIG_PACKAGE_liblucihttp=y
CONFIG_PACKAGE_liblucihttp-lua=y
CONFIG_PACKAGE_libncurses=y
CONFIG_PACKAGE_libnl-tiny=y
CONFIG_PACKAGE_libopcodes=y
CONFIG_PACKAGE_libpcap=y

#
# Configuration
#
```

```
CONFIG_PACKAGE_libreadline=y
CONFIG_PACKAGE_libsmartcols=y
CONFIG_PACKAGE_libss=y
CONFIG_PACKAGE_libubox=y
CONFIG_PACKAGE_libubus=y
CONFIG_PACKAGE_libubus-lua=y
CONFIG_PACKAGE_libuci=y
CONFIG_PACKAGE_libuclient=y
CONFIG_PACKAGE_libusb-1.0=y
CONFIG_PACKAGE_libusb-compat=y
CONFIG_PACKAGE_libuuid=y
CONFIG_PACKAGE_linux-atm=y
CONFIG_PACKAGE_rpcd-mod-rrdns=y
CONFIG_PACKAGE_terminfo=y
CONFIG_PACKAGE_uclibcxx=y
CONFIG_PACKAGE_zlib=y

#
# Configuration
#

#

#
# LuCI
#

#

# 1. Collections
#
CONFIG_PACKAGE_luci=y

#

# 2. Modules
#
CONFIG_PACKAGE_luci-base=y
```

```
#
# Translations
#
CONFIG_PACKAGE_luci-mod-admin-full=y

#
# 3. Applications
#
CONFIG_PACKAGE_luci-app-firewall=y

#
# 4. Themes
#
CONFIG_PACKAGE_luci-theme-bootstrap=y

#
# 5. Protocols
#
CONFIG_PACKAGE_luci-proto-ipv6=y
CONFIG_PACKAGE_luci-proto-ppp=y

#
# 6. Libraries
#
CONFIG_PACKAGE_luci-lib-ip=y
CONFIG_PACKAGE_luci-lib-jsonc=y
CONFIG_PACKAGE_luci-lib-nixio=y

#
# 9. Freifunk
#
#
```

```
# Mail
#
#
# Select postfix build options
#
CONFIG_POSTFIX_TLS=y
CONFIG_POSTFIX_SASL=y
CONFIG_POSTFIX_LDAP=y
CONFIG_POSTFIX_CDB=y
CONFIG_POSTFIX_SQLITE=y
CONFIG_POSTFIX_PCRE=y
#
# Multimedia
#
#
# Streaming
#
#
# Network
#
#
# BitTorrent
#
#
# Captive Portals
#
#
```

```
# Download Manager
#
#
# File Transfer
#
#
# Filesystem
#
#
# Firewall
#
CONFIG_PACKAGE_ip6tables=y
CONFIG_PACKAGE_iptables=y
CONFIG_PACKAGE_iptables-mod-tee=y
#
# Firewall Tunnel
#
#
# FreeRADIUS (version 3)
#
#
# IP Addresses and Names
#
#
# Instant Messaging
#
```



```
#
# Linux ATM tools
#
CONFIG_PACKAGE_br2684ctl=y

#
# NMAP Suite
#

#
# NTRIP
#

#
# OLSR.org network framework
#

#
# Open vSwitch
#

#
# Printing
#

#
# Routing and Redirection
#
CONFIG_PACKAGE_tc=y

#
# SSH
#
```

```
#  
# THC-IPv6 attack and analyzing toolkit  
#  
  
#  
# Telephony  
#  
  
#  
# Telephony Lantiq  
#  
  
#  
# Time Synchronization  
#  
CONFIG_PACKAGE_chrony=y  
  
#  
# VPN  
#  
  
#  
# Version Control Systems  
#  
  
#  
# WWAN  
#  
  
#  
# Web Servers/Proxies  
#  
CONFIG_PACKAGE_uhttpd=y
```

```
#
# dial-in/up
#

#
# tcprelay
#

#
# wireless
#
CONFIG_PACKAGE_hostapd-common=y
CONFIG_PACKAGE_iperf=y
CONFIG_PACKAGE_iperf3=y
CONFIG_PACKAGE_iw=y
CONFIG_PACKAGE_odhcp6c=y
CONFIG_PACKAGE_odhcp6c_ext_cer_id=0
CONFIG_PACKAGE_odhcpd-ipv6only=y

#
# Configuration
#
CONFIG_PACKAGE_odhcpd-ipv6only_ext_cer_id=0
CONFIG_PACKAGE_ppp=y
CONFIG_PACKAGE_ppp-mod-pppoe=y
CONFIG_PACKAGE_ppp-mod-pppoe=y
CONFIG_PACKAGE_pppdump=y
CONFIG_PACKAGE_pppstats=y
CONFIG_PACKAGE_soloscli=y
CONFIG_PACKAGE_tcpdump=y
CONFIG_PACKAGE_uclient-fetch=y
CONFIG_WPA_MSG_MIN_PRIORITY=3
CONFIG_DRIVER_11N_SUPPORT=y
CONFIG_DRIVER_11W_SUPPORT=y
```

```
CONFIG_PACKAGE_wpad-mini=y

#
# Sound
#

#
# Utilities
#

#
# Boot Loaders
#

#
# Compression
#

#
# Disc
#
CONFIG_PACKAGE_partx-utils=y

#
# Editors
#

#
# Encryption
#
CONFIG_PACKAGE_px5g-standalone=m

#
# Filesystem
```

```
#
CONFIG_PACKAGE_e2fsprogs=y
CONFIG_PACKAGE_mkfs=y

#
# Image Manipulation
#

#
# Microcontroller programming
#

#
# RTKLIB Suite
#

#
# Shells
#
CONFIG_PACKAGE_bash=y

#
# Telephony
#

#
# Terminal
#
CONFIG_PACKAGE_script-utils=y
CONFIG_PACKAGE_setterm=y
CONFIG_PACKAGE_wall=y

#
# Virtualization
```

```
#  
  
#  
# Zoneinfo  
#  
  
#  
# database  
#  
CONFIG_PACKAGE_dmidecode=y  
CONFIG_PACKAGE_flashrom=y  
CONFIG_PACKAGE_gpsd=y  
CONFIG_PACKAGE_hwclock=y  
CONFIG_PACKAGE_iwinfo=y  
CONFIG_PACKAGE_jshn=y  
CONFIG_PACKAGE_libjson-script=y  
CONFIG_PACKAGE_pciutils=y  
CONFIG_PACKAGE_strace=y  
  
#  
# Xorg  
#  
  
#  
# font-utils  
#
```

Appendix D

APU boards network configuration files and iw_startup

These files are related to configuring the Ethernet and WLAN interfaces of the APU boards, as reported in section 5.5.

APU_102

/etc/config/network

```
1
2 config interface 'loopback'
3     option ifname 'lo'
4     option proto 'static'
5     option ipaddr '127.0.0.1'
6     option netmask '255.0.0.0'
7
8 config globals 'globals'
9     option ula_prefix 'fdbb:5452:15c1::/48'
10
11 config interface 'lan'
12 # option type 'bridge'
13     option ifname 'eth2'
14     option proto 'static'
15     option ipaddr '192.168.1.182'
```

```
16  option netmask '255.255.255.0'
17  option gateway '192.168.1.1'
18  # option ip6assign '60'
19
20  config interface 'wan'
21      option ifname 'eth0'
22      option proto 'dhcp'
23
24  config interface 'wan6'
25      option ifname 'eth0'
26      option proto 'dhcpv6'
```


/etc/config/wireless

```
1
2 config wifi-device 'radio0'
3     option type 'mac80211'
4     option channel '36'
5     option hwmode '11a'
6     option path 'pci0000:00/0000:00:15.0/0000:05:00.0'
7     option htmode 'HT20'
8     option disabled '0'
9     option txpower '3'
10
11 config wifi-iface 'default_radio0'
12     option device 'radio0'
13     option network 'lan'
14     option mode 'ap'
15     option ssid 'OpenWrt'
16     option encryption 'none'
```

/root/iw_startup

```
1 #!/bin/sh
2
3 echo "Waiting 5 seconds..."
4 sleep 5
5 echo "Pacthed Italian Frequency Register Set"
6 iw reg set IT
7 sleep 1
8 echo "Turn off wlan0"
9 ifconfig wlan0 down
10 sleep 1
11 echo "Set Mode OCB"
12 iw dev wlan0 set type ocb
13 echo "Turn on wlan0"
14 ifconfig wlan0 up
15 sleep 1
16 echo "Leave possibile OCB..."
17 iw dev wlan0 ocb leave
18 echo "Set Frequency 5890 MHz (CCH) and channel width 10MHz (802.11p)"
19 iw dev wlan0 ocb join 5890 10MHz
20 echo "IP address set to 10.10.6.102"
21 ifconfig wlan0 10.10.6.102 netmask 255.255.0.0
22 echo "Set Rate 3M and txpower 15 dBm, using iw"
23 iw dev wlan0 set bitrates legacy-5 6
24 iw dev wlan0 set txpower fixed 1500
25 echo "Completing configuration, waiting 8 seconds..."
26 sleep 8
27 echo "Configured as: "
28 iw dev
```

APU_103

/etc/config/network

```
1
2 config interface 'loopback'
3     option ifname 'lo'
4     option proto 'static'
5     option ipaddr '127.0.0.1'
6     option netmask '255.0.0.0'
7
8 config globals 'globals'
9     option ula_prefix 'fd36:b1bb:9cb5::/48'
10
11 config interface 'lan'
12 # option type 'bridge'
13     option ifname 'eth2'
14     option proto 'static'
15     option ipaddr '192.168.1.183'
16     option netmask '255.255.255.0'
17     option gateway '192.168.1.1'
18 # option ip6assign '60'
19
20 config interface 'wan'
21     option ifname 'eth0'
22     option proto 'dhcp'
23
24 config interface 'wan6'
25     option ifname 'eth0'
26     option proto 'dhcpv6'
```

/etc/config/wireless

```
1
2 config wifi-device 'radio0'
3     option type 'mac80211'
4     option channel '36'
5     option hwmode '11a'
6     option path 'pci0000:00/0000:00:15.0/0000:05:00.0'
7     option htmode 'HT20'
8     option disabled '0'
9     option txpower '3'
10
11 config wifi-iface 'default_radio0'
12     option device 'radio0'
13     option network 'lan'
14     option mode 'ap'
15     option ssid 'OpenWrt'
16     option encryption 'none'
```

/root/iw_startup

```
1 #!/bin/sh
2
3 echo "Waiting 5 seconds..."
4 sleep 5
5 echo "Pacthed Italian Frequency Register Set"
6 iw reg set IT
7 sleep 1
8 echo "Turn off wlan0"
9 ifconfig wlan0 down
10 sleep 1
11 echo "Set Mode OCB"
12 iw dev wlan0 set type ocb
13 echo "Turn on wlan0"
14 ifconfig wlan0 up
15 sleep 1
16 echo "Leave possibile OCB..."
17 iw dev wlan0 ocb leave
18 echo "Set Frequency 5890 MHz (CCH) and channel width 10MHz (802.11p)"
19 iw dev wlan0 ocb join 5890 10MHz
20 echo "IP address set to 10.10.6.103"
21 ifconfig wlan0 10.10.6.103 netmask 255.255.0.0
22 echo "Set Rate 3M and txpower 15 dBm, using iw"
23 iw dev wlan0 set bitrates legacy-5 6
24 iw dev wlan0 set txpower fixed 1500
25 echo "Completing configuration, waiting 8 seconds..."
26 sleep 8
27 echo "Configured as: "
28 iw dev
```

Appendix E

chrony and system configuration files

These files are all related to setting up the APU (and ALIX) boards for NTP synchronization, as reported in section 5.6.

/etc/config/chrony

```
1 config pool
2     option hostname 'ntp1.inrim.it'
3     option maxpoll '12'
4     option iburst 'yes'
5
6 config dhcp_ntp_server
7     option iburst 'yes'
8
9 config allow
10    option interface 'lan'
11
12 config makestep
13    option threshold '1.0'
14    option limit '3'
```

/etc/config/system

```
1
2 config system
3     option hostname 'OpenWrt'
4     option timezone 'UTC'
5     option ttylogin '0'
6     option log_size '64'
7     option urandom_seed '0'
8
9 config timeserver 'ntp'
10    option enabled '0'
11    option enable_server '0'
12    list server '0.openwrt.pool.ntp.org'
13    list server '1.openwrt.pool.ntp.org'
14    list server '2.openwrt.pool.ntp.org'
15    list server '3.openwrt.pool.ntp.org'
16
17 config led 'led_wan'
18    option name 'WAN'
19    option sysfs 'apu2:green:led3'
20    option trigger 'netdev'
21    option mode 'link tx rx'
22    option dev 'eth0'
23
24 config led 'led_lan'
25    option name 'LAN'
26    option sysfs 'apu2:green:led2'
27    option trigger 'netdev'
28    option mode 'link tx rx'
29    option dev 'br-lan'
30
31 config led 'led_diag'
32    option name 'DIAG'
```

```
33 option sysfs 'apu2:green:power'
34 option default '1'
```


/etc/chrony/chrony.conf

```
1 # This file is included from config file generated from /etc/config/
   chrony
2
3 # Log clock errors above 0.5 seconds
4 logchange 0.5
5
6 # Don't log client accesses
7 noclientlog
8
9 # set the system clock else the kernel will always stay in UNSYNC
   state
10 rtcsync
```

Appendix F

millisleep utility and iPerf debug modifications

millisleep

The millisleep utility, used in section 6.2, is a very simple custom implementation of `sleep` to wait for fraction of seconds (which is not currently supported by the OpenWrt's `sleep`).

It is based on Linux timers. In particular:

- A monotonic increasing file descriptor-based timer is created using “`timerfd_create()`”.
- The waiting time (in seconds) is parsed from the command line.
- Since the execution time of this utility has still not been measured in details (even though the “`time`” Linux utility seems to prove that it is working properly), any wait less than 10 ms is forbidden, since accuracy may be no more guaranteed (also based on the broadcast transmission C programs, using Linux timers).
- A proper “`struct itimerspec`” is filled in with the wait time in seconds and nanoseconds.
- A proper “`struct pollfd`” is filled for the subsequent calls to “`poll`”, which waits for certain events to happen on a given file descriptor.

In our case the file descriptor is the timer one: “timerMon.fd=clockFd;”

Then, we set “timerMon.events=POLLIN;” to wait until “there is data to read” on the file descriptor, i.e. until the timer expires.

- The timer is started with “timerfd_settime()”.
- A while loop continuously runs until the timer expires and “poll()” returns a positive value.

This program is meant to be run on the target boards, thus the need of cross compiling it, rather than generating the binaries by direct compilation.

millisleep.c

```
1  #include <sys/timerfd.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5  #include <poll.h>
6  #include <math.h>
7  #include <unistd.h>
8
9  #define SEC_TO_NANOSEC 1000000000
10 #define INDEFINITE_BLOCK -1
11 #define NO_FLAGS 0
12
13 int main (int argc, char **argv) {
14     int clockFd;
15     struct itimerspec new_value;
16     long nanosec;
17     double d_sec;
18     double d_time;
19     struct pollfd timerMon;
20
21     if(argc!=2) {
```

```
22     exit(EXIT_FAILURE);
23 }
24
25 // Create monotonic (increasing) timer
26 clockFd=timerfd_create(CLOCK_MONOTONIC,NO_FLAGS);
27 if(clockFd==-1) {
28     perror("timerfd_create() error");
29     exit(EXIT_FAILURE);
30 }
31
32 // Get time from command line and convert it to sec and nanosec
33 d_time=strtod(argv[1],NULL);
34 if(d_time==0) {
35     perror("strtod() to get time interval returned an error");
36     close(clockFd);
37     exit(EXIT_FAILURE);
38 }
39
40 if(d_time<0.01) {
41     fprintf(stderr,"Cannot wait for less than 10 ms.\n");
42     close(clockFd);
43     exit(EXIT_FAILURE);
44 }
45
46 nanosec=SEC_TO_NANOSEC*modf(d_time,&d_sec);
47 new_value.it_value.tv_nsec=nanosec;
48 new_value.it_value.tv_sec=(time_t)d_sec;
49 new_value.it_interval.tv_nsec=nanosec;
50 new_value.it_interval.tv_sec=(time_t)d_sec;
51
52 // Fill pollfd structure
53 timerMon.fd=clockFd;
54 timerMon.revents=0;
55 timerMon.events=POLLIN;
```

```
56
57 // Start timer
58 if(timerfd_settime(clockFd,NO_FLAGS,&new_value,NULL)==-1) {
59     perror("timerfd_settime() error");
60     close(clockFd);
61     exit(EXIT_FAILURE);
62 }
63
64 while(poll(&timerMon,1,INDEFINITE_BLOCK)<=0);
65
66 close(clockFd);
67
68 return 0;
69 }
```

iPerf debug modifications

This section reports the modified iPerf client code, used in section 6.2.

As mentioned in chapter 6 the modifications are related to the UDP send loop, which has been modified in order to output otherwise unavailable (or more difficult to obtain) information, such as the running delay and the UDP transmission buffer evolutions.

Two C++ files have been modified, both related to the client, *./src/Client.cpp*, for what concerns the UDP send loop only, and its header file, *./src/Client.hpp*.

The relevant sections are presented below, followed by a brief comment on each modification.

./include/Client.hpp

```
1 /*-----
2  * Copyright (c) 1999,2000,2001,2002,2003
3  * The Board of Trustees of the University of Illinois
4  * All Rights Reserved.
5  *-----
```

```
6 * Permission is hereby granted, free of charge, to any person
7 * obtaining a copy of this software (Iperf) and associated
8 * documentation files (the "Software"), to deal in the Software
9 * without restriction, including without limitation the
10 * rights to use, copy, modify, merge, publish, distribute,
11 * sublicense, and/or sell copies of the Software, and to permit
12 * persons to whom the Software is furnished to do
13 * so, subject to the following conditions:
14 *
15 *
16 * Redistributions of source code must retain the above
17 * copyright notice, this list of conditions and
18 * the following disclaimers.
19 *
20 *
21 * Redistributions in binary form must reproduce the above
22 * copyright notice, this list of conditions and the following
23 * disclaimers in the documentation and/or other materials
24 * provided with the distribution.
25 *
26 *
27 * Neither the names of the University of Illinois, NCSA,
28 * nor the names of its contributors may be used to endorse
29 * or promote products derived from this Software without
30 * specific prior written permission.
31 *
32 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
33 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES
34 * OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
35 * NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT
36 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
37 * WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
38 * ARISING FROM, OUT OF OR IN CONNECTION WITH THE
39 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
40  * -----
41  * National Laboratory for Applied Network Research
42  * National Center for Supercomputing Applications
43  * University of Illinois at Urbana-Champaign
44  * http://www.ncsa.uiuc.edu
45  * -----
46  * Client.hpp
47  * by Mark Gates <mgates@nlanr.net>
48  * -----
49  * A client thread initiates a connect to the server and handles
50  * sending and receiving data, then closes the socket.
51  * -----
52  */
53
54 #ifndef CLIENT_H
55 #define CLIENT_H
56
57 #include "Settings.hpp"
58 #include "Timestamp.hpp"
59
60 /* -----
61
62 */
61 class Client {
62 public:
63     // stores server hostname, port, UDP/TCP mode, and UDP rate
64     Client( thread_Settings *inSettings );
65
66     // destroy the client object
67     ~Client();
68
69     // Set up the traffic thread and invokes
70     // appropriate traffic loop per the protocol
71     // and type of traffic
72     void Run( void );
```

```
73
74     // For things like dual tests a server needs to be started by the
       client,
75     // The code in src/launch.cpp will invoke this
76     void InitiateServer();
77
78 private:
79     void WritePacketID(void);
80     void InitTrafficLoop(void);
81     void FinishTrafficActions(void);
82     void FinalUDPHandshake(void);
83     void write_UDP_FIN(void);
84     bool InProgress(void);
85
86     ReportStruct *reportstruct;
87     double delay_lower_bounds;
88     max_size_t totLen;
89
90     // TCP plain
91     void RunTCP( void );
92     // TCP version which supports rate limiting per -b
93     void RunRateLimitedTCP( void );
94     // UDP traffic with isochronous and vbr support
95     void RunUDPIsochronous( void );
96     // UDP traffic with threads synced to transmit at near the same
       time
97     void RunUDPTxSync( void );
98     // UDP plain
99     void RunUDP( void );
100    // client connect
101    void Connect( );
102    void HdrXchange(int flags);
103
104    thread_Settings *mSettings;
```



```
105     char* mBuf;
106     Timestamp mEndTime;
107     Timestamp lastPacketTime;
108     Timestamp now;
109     char* readAt;
110     Timestamp syncTime;
111
112     double *dlyarray;
113     unsigned int *bufarray;
114
115 }; // end class Client
116
117 #endif // CLIENT_H
```

- **Lines 112-113:** two pointer definitions have been added inside the “Client” class: these will be used to allocate two arrays containing all the data related to the evolution of the running delay (`double *dlyarray`) and the length of the UDP transmission queue (`unsigned int *bufarray`)

./src/Client.cpp (UDP send loop only)

```
464  /*
465   * UDP send loop
466   */
467  void Client::RunUDP( void ) {
468      struct UDP_datagram* mBuf_UDP = (struct UDP_datagram*) mBuf;
469      int currLen;
470
471      double delay_target = 0;
472      double delay = 0;
473      double adjust = 0;
474
475      const int arraysize=100000;
476
477      unsigned int bufsize;
478
479      fprintf(stdout, "Entering UDP send loop!\n");
480
481      // Allocating dlyarray
482      dlyarray=(double *)malloc(arraysize*sizeof(double));
483      if(!dlyarray) {
484          fprintf(stderr, "Unable to allocate dlyarray. Panic exit.\n");
485          exit(EXIT_FAILURE);
486      }
487
488      // Allocating bufarray
489      bufarray=(unsigned int *)malloc(arraysize*sizeof(unsigned int));
490      if(!bufarray) {
491          fprintf(stderr, "Unable to allocate bufarray. Panic exit.\n");
492          free(dlyarray);
493          exit(EXIT_FAILURE);
494      }
495
```

```
496     int arridx=0;
497     bufarray[0]=0;
498
499     // compute delay target in units of nanoseconds
500     if (mSettings->mUDPRateUnits == kRate_BW) {
501         // compute delay for bandwidth restriction, constrained to [0,1]
502         // seconds
503         delay_target = (double) ( mSettings->mBufLen * ((kSecs_to_nsecs *
504             kBytes_to_Bits)
505             / mSettings->mUDPRate) );
506     } else {
507         delay_target = 1e9 / mSettings->mUDPRate;
508     }
509     if ( delay_target < 0 ||
510         delay_target > 1.0 * kSecs_to_nsecs ) {
511         fprintf( stderr, warn_delay_large, delay_target / kSecs_to_nsecs );
512         delay_target = 1.0 * kSecs_to_nsecs;
513     }
514
515     fprintf(stdout,"Target delay with -b %d: %f ms\n",(int) mSettings
516         ->mUDPRate,delay_target/1e6);
517     fprintf(stdout,"Delay lower bounds: %f ms\n",delay_lower_bounds/1
518         e6);
519
520     // Set this to > 0 so first loop iteration will delay the IPG
521     currLen = 1;
522     double variance = mSettings->mVariance;
523
524     while (InProgress()) {
525         // Test case: drop 17 packets and send 2 out-of-order:
526         // sequence 51, 52, 70, 53, 54, 71, 72
527         //switch( datagramID ) {
528         // case 53: datagramID = 70; break;
529         // case 71: datagramID = 53; break;
```

```

526         // case 55: datagramID = 71; break;
527         // default: break;
528         //}
529     now.setnow();
530     reportstruct->packetTime.tv_sec = now.getSecs();
531     reportstruct->packetTime.tv_usec = now.getUsecs();
532     if (isVaryLoad(mSettings) && mSettings->mUDPRateUnits ==
        kRate_BW) {
533         static Timestamp time3;
534         if (now.subSec(time3) >= VARYLOAD_PERIOD) {
535             int var_rate = lognormal(mSettings->mUDPRate, variance);
536             if (var_rate < 0)
537                 var_rate = 0;
538
539             delay_target = (double) ( mSettings->mBufLen * ((kSecs_to_nsecs *
                kBytes_to_Bits)
540                 / var_rate) );
541             time3 = now;
542         }
543     }
544
545     // store datagram ID into buffer
546     WritePacketID();
547     mBuf_UDP->tv_sec = htonl(reportstruct->packetTime.tv_sec);
548     mBuf_UDP->tv_usec = htonl(reportstruct->packetTime.tv_usec);
549
550     if (!isSeqNo64b(mSettings) && (reportstruct->packetID & 0x80000000L)
        ) {
551         // seqno wrapped
552         fprintf(stderr, "%s", warn_seqno_wrap);
553         break;
554     }
555     // Adjustment for the running delay
556     // o measure how long the last loop iteration took

```

```
557 // o calculate the delay adjust
558 //   - If write succeeded, adjust = target IPG - the loop time
559 //   - If write failed, adjust = the loop time
560 // o then adjust the overall running delay
561 // Note: adjust units are nanoseconds,
562 //       packet timestamps are microseconds
563 if (currLen > 0)
564     adjust = delay_target + \
565     (1000.0 * lastPacketTime.subUsec( reportstruct->packetTime ));
566 else
567     adjust = 1000.0 * lastPacketTime.subUsec( reportstruct->
568         packetTime );
569 lastPacketTime.set( reportstruct->packetTime.tv_sec,
570     reportstruct->packetTime.tv_usec );
571 // Since linux nanosleep/busyloop can exceed delay
572 // there are two possible equilibriums
573 // 1) Try to perserve inter packet gap
574 // 2) Try to perserve requested transmit rate
575 // The latter seems preferred, hence use a running delay
576 // that spans the life of the thread and constantly adjust.
577 // A negative delay means the iperf app is behind.
578 delay += adjust;
579 dlyarray[arridx]=delay;
580 arridx=(arridx+1)%arraysize;
581
582 // Don't let delay grow unbounded
583 if (delay < delay_lower_bounds) {
584     delay = delay_target;
585 }
586
587 reportstruct->errwrite = 0;
588 reportstruct->emptyreport = 0;
589
```

```

590 // Log UDP buffer status before sending the next packet
591 ioctl(mSettings->mSock,TIOCOUTQ,&bufsize);
592 bufarray[arridx]=bufsize;
593
594 // perform write
595 currLen = write( mSettings->mSock, mBuf, mSettings->mBufLen );
596 if ( currLen < 0 ) {
597     reportstruct->packetID--;
598     reportstruct->errwrite = 1;
599     reportstruct->emptyreport = 1;
600     currLen = 0;
601     if (
602 #ifdef WIN32
603         (errno = WSAGetLastError()) != WSAETIMEDOUT &&
604         errno != WSAECONNREFUSED
605 #else
606         errno != EAGAIN && errno != EWOULDBLOCK &&
607         errno != EINTR && errno != ECONNREFUSED &&
608         errno != ENOBUFFS
609 #endif
610     ) {
611         WARN_errno( 1, "write" );
612         break;
613     }
614 }
615
616 // report packets
617 reportstruct->packetLen = (unsigned long) currLen;
618 ReportPacket( mSettings->reporthdr, reportstruct );
619 // Insert delay here only if the running delay is greater than 1
620 // usec,
621 // otherwise don't delay and immediately continue with the next tx.
622 if ( delay >= 1000 ) {
623     // Convert from nanoseconds to microseconds

```

```

623     // and invoke the microsecond delay
624     delay_loop((unsigned long) (delay / 1000));
625 }
626 if (!isModeTime(mSettings)) {
627     /* mAmount may be unsigned, so don't let it underflow! */
628     if( mSettings->mAmount >= (unsigned long) currLen ) {
629         mSettings->mAmount -= (unsigned long) currLen;
630     } else {
631         mSettings->mAmount = 0;
632     }
633 }
634
635 }
636
637 // Print to stderr all the running delays and then all the buffer
638 // status data
639 for(int aidx=0;aidx<arridx;aidx++) {
640     fprintf(stderr,"%f\n",dlyarray[aidx]);
641 }
642
643 fprintf(stderr,"-----\n");
644
645 for(int aidx=0;aidx<arridx;aidx++) {
646     fprintf(stderr,"%u\n",bufarray[aidx]);
647 }
648
649 fprintf(stdout,"Exiting UDP send loop!\n");
650
651 FinishTrafficActions();
652 }

```

- **General note:** as reported in chapter 6, “the data about the evolution of the buffer and of the running delay is stored inside two arrays, which are printed on stderr

only at the end of the UDP send loop execution, to avoid continuous write to files or on stdout/stderr which could potentially affect performance in an evident way”.

- **Lines 475-477:** The size of the additional arrays (i.e. the maximum number of historical points to be collected) is defined as a constant “arraysize” and a second unsigned integer variable is declared: it will be used to store the current number of bytes inside the UDP transmission queue.
- **Lines 479:** An additional debug *printf* on standard output has been inserted, telling that the UDP loop has been correctly started.
- **Lines 481-494:** The two arrays mentioned before are dynamically allocated. In case of error, the program is immediately terminated.
- **Lines 496-497:** An additional index is defined, in order to scan the arrays and print all the data inside them when the UDP loop execution is going to terminate. The first element of the buffer size array is then initialized to 0, due to the mechanism of memorizing the buffer size at one index more with respect to the running delay (otherwise there is the risk of having uninitialized random data inside this first element).
- **Lines 513-514:** The target delay and the lower bound are printed, in ms.
- **Lines 579-580:** The current value of the running delay (variable “delay”) is saved inside the current element of the delay array and the index is (cyclically, storing at most “arraysize” values) incremented.
- **Lines 590-592:** The status of the UDP buffer is stored inside “bufsize”, using the TIOCOUTQ request within the “ioctl” system call. It is then stored inside the proper element of the “bufarray”, at one index more with respect to the running delay stored in “dlyarray”, as detailed in section 6.2.

- **Lines 637-646:** After the UDP send loop terminates the regular transmissions, the two arrays are printed, sequentially, on *stderr* (first, the delay array, then, the buffer size array). This choice allows the user to log them to a separate file by just redirecting *stderr* with “2>”, without affecting *stdout*.
- **Line 648:** A final debug *printf* is inserted to tell the user that the UDP send loop has terminated its execution, except for the final handshake.

In order to cross-compile this larger project for the APU boards, using GNU configure and GNU make, we used the following commands, after configuring the toolchain and moving to the iPerf directory (with “cd”):

```
./configure --build=x86_64-unknown-linux-gnu --host=x86_64-openwrt-  
linux-musl  
make CC=x86_64-openwrt-linux-musl-gcc LD=x86_64-openwrt-linux-musl-ld
```

After “build” the user should write the output of the command “./config.guess”, while, after “host”, the proper OpenWrt architecture, where the compiled binaries will be run, should be specified (in our case “x86_64-openwrt-linux-musl”).

More details are available inside the OpenWrt documentation¹, where, however, “uClibc” (if it is still present, as at the time of writing this) should be substituted by “musl”, in all the most recent OpenWrt versions.

¹<https://openwrt.org/docs/guide-developer/crosscompile>

Appendix G

Scripts for systematic measurements of throughput and packet loss

These are the scripts used for the measurements reported in section 6.3.

They use *bash* arrays, so they require the *bash* shell to be installed on the target boards.

Small modifications can be noticed when looking at the scripts used with 3 Mbit/s of physical rate with respect to the ones used with 6 and 12 Mbit/s: they are little enhancements that have been introduced to improve the measurements after the first data set has been collected.

3 Mbit/s

iperfclient7000L3.sh (client)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
```

```
10 }
11
12 function iperfclientbl_log {
13     echo "# iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2" >> "
        Logs/ClientLog_b_$1_l_$2B.txt"
14     iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2 2>&1 | tee -a
        "Logs/ClientLog_b_$1_l_$2B.txt"
15 }
16
17 # Check if the Logs directory exists. If not, create it.
18 if [ ! -d "./Logs" ]; then
19     echo "Logs directory missing. It will be created now"
20     mkdir Logs
21 else
22     echo "Logs directory exists. Make a backup of its content before
        proceeding."
23 fi
24 read -p "Press enter to continue"
25
26 echo "Setting rate to 3 MBit/s..."
27 iw dev wlan0 set bitrates legacy-5 6
28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 2.2M 2.4M 2.6M 2.8M 3M 5M 7M 9M 10M 12M )
31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[
        @]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $1 -lt ${#len[@]} ]; do
```

```

40 while [ $i -lt ${#rates[@]} ]; do
41     echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":":
         " -f3 | cut -d" " -f1), waiting until second $currwait"
42     waituntil $currwait
43     echo "-----"
44     echo "Running test with -b ${rates[$i]} -l ${len[$1]}"
45     iperfclientbl_log ${rates[$i]} ${len[$1]}
46     echo "Test with -b ${rates[$i]} -l ${len[$1]} terminated"
47     echo "-----"
48     i=$((i+1))
49     currwait=$(( (currwait+10)%60 ))
50 done
51 sleep 1
52 l=$((l+1))
53 i=0
54 done

```

iperfserver7000L3.sh (server)

```

1 #!/bin/bash
2
3 function waituntil {
4     while true; do
5         currsec=$(date | cut -d$'\t' -f5 | cut -d":": -f3 | cut -d" " -f1)
6         if [ $currsec -eq $1 ]; then
7             break
8         fi
9     done
10 }
11
12 function iperfserverl_log {
13     echo "# iperf -s -u -i 2 -p 7000 -l $2 -t 62" >> "Logs/
        ServerLog_b_$1_l_$2B.txt"

```

```
14  iperf -s -u -i 2 -p 7000 -l $2 -t 62 2>&1 | tee -a "Logs/
    ServerLog_b_$1_l_$2B.txt"
15 }
16
17 # Check if the Logs directory exists. If not, create it.
18 if [ ! -d "./Logs" ]; then
19     echo "Logs directory missing. It will be created now"
20     mkdir Logs
21 else
22     echo "Logs directory exists. Make a backup of its content before
        proceeding."
23 fi
24 read -p "Press enter to continue"
25
26 echo "Setting rate to 3 MBit/s..."
27 iw dev wlan0 set bitrates legacy-5 6
28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 2.2M 2.4M 2.6M 2.8M 3M 5M 7M 9M 10M 12M )
31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[
        @]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $l -lt ${#len[@]} ]; do
40     while [ $i -lt ${#rates[@]} ]; do
41         echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":
            " -f3 | cut -d" " -f1), waiting until second $currwait"
42         waituntil $currwait
43         echo "-----"
```

```
44     echo "Running test with -b ${rates[$i]} -l ${len[$1]}"
45     iperfserverl_log ${rates[$i]} ${len[$1]}
46     echo "Test with -b ${rates[$i]} -l ${len[$1]} terminated"
47     echo "-----"
48     i=$((i+1))
49     currwait=$(( (currwait+10)%60 ))
50 done
51 sleep 1
52 l=$((l+1))
53 i=0
54 done
```

Development PC data log data extraction script - 3 Mbit/s

This script should be placed inside the same folder where the 3 Mbit/s logs are placed.

logExtractor.sh (3 Mbit/s)

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]; then
4      if [ $# -ne 2 -o ! $2 = "-M" ]; then
5          echo "Error using program. Expected one or two parameters: "
6          echo "1) Rate (<3|6|12>)"
7          echo "2) [-M] for log files with an extra 'M' after 'b'"
8          exit 1
9      fi
10 fi
11
12 # Defining rates (-b) and len (-l) arrays to correctly extract data
    from the logs
13 rates=( 0.5M 1M 1.5M 2M 2.2M 2.4M 2.6M 2.8M 3M 5M 7M 9M 10M 12M )
14 len=( 16 48 104 208 500 700 1000 1200 1470 )
15
16 i=0
17 l=0
18 echo "len,b,tput,loss,lostpkt,sentpkt,okpkt" > "Rate$1_iperf_bl_test.
    csv"
19 while [ $l -lt ${#len[@]} ]; do
20     while [ $i -lt ${#rates[@]} ]; do
21         b=$(echo ${rates[$i]} | sed 's/.$//')
22         if [ ! -z "$2" -a "$2" = "-M" ]; then
23             tput=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail -1
                | sed 's/\s\s*/ /g' | cut -d" " -f7)
24             umeas=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail
                -1 | sed 's/\s\s*/ /g' | cut -d" " -f8)
25             # Convert measurements in KBit/s to MBit/s
```

```

26     if [[ $umeas = *"K"* ]]; then
27         tput=$(echo "$tput" | awk '{printf("%.3f\n", $1/1000)}')
28     fi
29     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
30     sentpkt=$(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
31     echo "$(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt | tail
        -1)"
32 else
33     tput=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail -1
        | sed 's/\s\s\s*/ /g' | cut -d" " -f7)
34     umeas=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail -1
        | sed 's/\s\s\s*/ /g' | cut -d" " -f8)
35     # Convert measurements in KBit/s to MBit/s
36     if [[ $umeas = *"K"* ]]; then
37         tput=$(echo "$tput" | awk '{printf("%.3f\n", $1/1000)}')
38     fi
39     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
40     sentpkt=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
41     echo "$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail
        -1)"
42 fi
43 okpkt=$((sentpkt-lostpkt))
44 loss=$(echo "$lostpkt $sentpkt" | awk '{printf("%.15f\n", $1/$2
        *100)}')
45 i=$((i+1))
46
47 # Print parsed information

```



```
48     echo "len=${len[$1]},b=$b,tput=$tput,loss=$loss,lostpkt=$lostpkt,
        sentpkt=$sentpkt,okpkt=$okpkt"
49     # Write to .csv file
50     echo "${len[$1]},$b,$tput,$loss,$lostpkt,$sentpkt,$okpkt" >> "
        Rate$1_iperf_b1_test.csv"
51 done
52 l=$((l+1))
53 i=0
54 done
```

6 Mbit/s

iperfclient7000L6.sh (client)

```

1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfclientbl_log {
13     echo "# iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2" >> "
14         Logs/ClientLog_b_$1_l_$2B.txt"
15     iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2 2>&1 | tee -a
16         "Logs/ClientLog_b_$1_l_$2B.txt"
17 }
18
19 # Check if the Logs directory exists. If not, create it.
20 if [ ! -d "./Logs" ]; then
21     echo "Logs directory missing. It will be created now"
22     mkdir Logs
23 else
24     echo "Logs directory exists. Make a backup of its content before
25         proceeding."
26 fi
27 read -p "Press enter to continue"
28
29 echo "Setting rate to 6 MBit/s..."
30 iw dev wlan0 set bitrates legacy-5 12

```

```
28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 4M 4.2M 4.4M 4.6M 4.8M 5M 7M 9M 10M 12M 14M 16
        M 18M 20M )
31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[
        @]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $l -lt ${#len[@]} ]; do
40     while [ $i -lt ${#rates[@]} ]; do
41         echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":
            " -f3 | cut -d" " -f1), waiting until second $currwait"
42         waituntil $currwait
43         echo "-----"
44         echo "Running test with -b ${rates[$i]} -l ${len[$l]}"
45         sleep 2
46         iperfclientbl_log ${rates[$i]} ${len[$l]}
47         echo "Test with -b ${rates[$i]} -l ${len[$l]} terminated"
48         echo "-----"
49         i=$((i+1))
50         currwait=$(( (currwait+10)%60 ))
51     done
52     sleep 1
53     l=$((l+1))
54     i=0
55 done
```

iperfserver7000L6.sh (server)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfserverl_log {
13     echo "# iperf -s -u -i 2 -p 7000 -l $2 -t 64" >> "Logs/
        ServerLog_b_$1_l_$2B.txt"
14     iperf -s -u -i 2 -p 7000 -l $2 -t 64 2>&1 | tee -a "Logs/
        ServerLog_b_$1_l_$2B.txt"
15 }
16
17 # Check if the Logs directory exists. If not, create it.
18 if [ ! -d "./Logs" ]; then
19     echo "Logs directory missing. It will be created now"
20     mkdir Logs
21 else
22     echo "Logs directory exists. Make a backup of its content before
        proceeding."
23 fi
24 read -p "Press enter to continue"
25
26 echo "Setting rate to 6 MBit/s..."
27 iw dev wlan0 set bitrates legacy-5 12
28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 4M 4.2M 4.4M 4.6M 4.8M 5M 7M 9M 10M 12M 14M 16
        M 18M 20M )
```

```
31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[@]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $l -lt ${#len[@]} ]; do
40     while [ $i -lt ${#rates[@]} ]; do
41         echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":
42             " -f3 | cut -d" " -f1), waiting until second $currwait"
43         waituntil $currwait
44         echo "-----"
45         echo "Running test with -b ${rates[$i]} -l ${len[$l]}"
46         iperfserverl_log ${rates[$i]} ${len[$l]}
47         echo "Test with -b ${rates[$i]} -l ${len[$l]} terminated"
48         echo "-----"
49         i=$((i+1))
50         currwait=$(( (currwait+10)%60 ))
51     done
52     sleep 1
53     l=$((l+1))
54     i=0
55 done
```

Development PC data log data extraction script - 6 Mbit/s

This script should be placed inside the same folder where the 6 Mbit/s logs are placed.

logExtractor.sh (3 Mbit/s)

```

1  #!/bin/bash
2
3  if [ $# -ne 1 ]; then
4      if [ $# -ne 2 -o ! $2 = "-M" ]; then
5          echo "Error using program. Expected one or two parameters: "
6          echo "1) Rate (<3|6|12>)"
7          echo "2) [-M] for log files with an extra 'M' after 'b'"
8          exit 1
9      fi
10 fi
11
12 # Defining rates (-b) and len (-l) arrays to correctly extract data
    from the logs
13 rates=( 0.5M 1M 1.5M 2M 4M 4.2M 4.4M 4.6M 4.8M 5M 7M 9M 10M 12M 14M 16
    M 18M 20M )
14 len=( 16 48 104 208 500 700 1000 1200 1470 )
15
16 i=0
17 l=0
18 echo "len,b,tput,loss,lostpkt,sentpkt,okpkt" > "Rate$1_iperf_bl_test.
    csv"
19 while [ $l -lt ${#len[@]} ]; do
20     while [ $i -lt ${#rates[@]} ]; do
21         b=$(echo ${rates[$i]} | sed 's/.$//')
22         if [ ! -z "$2" -a "$2" = "-M" ]; then
23             tput=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail -1
                | sed 's/\s\s*/ /g' | cut -d" " -f7)
24             umeas=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail
                -1 | sed 's/\s\s*/ /g' | cut -d" " -f8)

```

```

25     # Convert measurements in KBit/s to MBit/s
26     if [[ $umeas = *"K"* ]]; then
27         tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
28     fi
29     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}M_1_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
30     sentpkt=$(cat ServerLog_b_${rates[$i]}M_1_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
31     echo "$(cat ServerLog_b_${rates[$i]}M_1_${len[$1]}B.txt | tail
        -1)"
32 else
33     tput=$(cat ServerLog_b_${rates[$i]}_1_${len[$1]}B.txt | tail -1
        | sed 's/\s\s*/ /g' | cut -d" " -f7)
34     umeas=$(cat ServerLog_b_${rates[$i]}_1_${len[$1]}B.txt | tail -1
        | sed 's/\s\s*/ /g' | cut -d" " -f8)
35     # Convert measurements in KBit/s to MBit/s
36     if [[ $umeas = *"K"* ]]; then
37         tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
38     fi
39     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}_1_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
40     sentpkt=$(cat ServerLog_b_${rates[$i]}_1_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
41     echo "$(cat ServerLog_b_${rates[$i]}_1_${len[$1]}B.txt | tail
        -1)"
42 fi
43 okpkt=$((sentpkt-lostpkt))
44 loss=$(echo "$lostpkt $sentpkt" | awk '{printf("%.15f\n",$1/$2
        *100)}')
45 i=$((i+1))
46
47 # Print parsed information

```

```
48     echo "len=${len[$1]},b=$b,tput=$tput,loss=$loss,lostpkt=$lostpkt,
        sentpkt=$sentpkt,okpkt=$okpkt"
49     # Write to .csv file
50     echo "${len[$1]},$b,$tput,$loss,$lostpkt,$sentpkt,$okpkt" >> "
        Rate$1_iperf_b1_test.csv"
51 done
52 l=$((l+1))
53 i=0
54 done
```


12 Mbit/s

iperfclient7000L12.sh (client)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfclientbl_log {
13     echo "# iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2 (rate
14         12M)" >> "Logs/ClientLog_b_$1_l_$2B.txt"
15     iperf -c 10.10.6.102 -u -i 2 -t 60 -b $1 -p 7000 -l $2 2>&1 | tee -a
16         "Logs/ClientLog_b_$1_l_$2B.txt"
17 }
18
19 # Check if the Logs directory exists. If not, create it.
20 if [ ! -d "./Logs" ]; then
21     echo "Logs directory missing. It will be created now"
22     mkdir Logs
23 else
24     echo "Logs directory exists. Make a backup of its content before
25         proceeding."
26 fi
27 read -p "Press enter to continue"
28
29 echo "Setting rate to 12 MBit/s..."
30 iw dev wlan0 set bitrates legacy-5 24
```

```

28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 3M 5M 7M 8M 8.1M 8.2M 8.3M 8.4M 8.5M 10M 13M
        16M 19M 22M )
31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[
        @]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $l -lt ${#len[@]} ]; do
40     while [ $i -lt ${#rates[@]} ]; do
41         echo "Client started at second $(date | cut -d$'\t' -f5 | cut -d":
            " -f3 | cut -d" " -f1), waiting until second $currwait"
42         waituntil $currwait
43         echo "-----"
44         echo "Running test with -b ${rates[$i]} -l ${len[$l]}"
45         sleep 2
46         iperfclientbl_log ${rates[$i]} ${len[$l]}
47         echo "Test with -b ${rates[$i]} -l ${len[$l]} terminated"
48         echo "-----"
49         i=$((i+1))
50         currwait=$(( (currwait+10)%60 ))
51     done
52     sleep 1
53     l=$((l+1))
54     i=0
55 done

```

iperfserver7000L12.sh (server)

```
1  #!/bin/bash
2
3  function waituntil {
4      while true; do
5          currsec=$(date | cut -d$'\t' -f5 | cut -d":" -f3 | cut -d" " -f1)
6          if [ $currsec -eq $1 ]; then
7              break
8          fi
9      done
10 }
11
12 function iperfserverl_log {
13     echo "# iperf -s -u -i 2 -p 7000 -l $2 -t 64 (rate 12M)" >> "Logs/
        ServerLog_b_$1_l_$2B.txt"
14     iperf -s -u -i 2 -p 7000 -l $2 -t 64 2>&1 | tee -a "Logs/
        ServerLog_b_$1_l_$2B.txt"
15 }
16
17 # Check if the Logs directory exists. If not, create it.
18 if [ ! -d "./Logs" ]; then
19     echo "Logs directory missing. It will be created now"
20     mkdir Logs
21 else
22     echo "Logs directory exists. Make a backup of its content before
        proceeding."
23 fi
24 read -p "Press enter to continue"
25
26 echo "Setting rate to 12 MBit/s..."
27 iw dev wlan0 set bitrates legacy-5 24
28 sleep 1
29
30 rates=( 0.5M 1M 1.5M 2M 3M 5M 7M 8M 8.1M 8.2M 8.3M 8.4M 8.5M 10M 13M
        16M 19M 22M )
```

```

31 len=( 16 48 104 208 500 700 1000 1200 1470 )
32
33 echo "Starting... tests will last aproximately $(( ${#rates[@]} * ${#len[
    @]} + ${#rates[@]} * ${#len[@]} * 10 / 60 )) minutes"
34 sleep 1
35
36 currwait=0
37 i=0
38 l=0
39 while [ $l -lt ${#len[@]} ]; do
40     while [ $i -lt ${#rates[@]} ]; do
41         echo "Server started at second $(date | cut -d$'\t' -f5 | cut -d":
            " -f3 | cut -d" " -f1), waiting until second $currwait"
42         waituntil $currwait
43         echo "-----"
44         echo "Running test with -b ${rates[$i]} -l ${len[$l]}"
45         iperfserverl_log ${rates[$i]} ${len[$l]}
46         echo "Test with -b ${rates[$i]} -l ${len[$l]} terminated"
47         echo "-----"
48         i=$((i+1))
49         currwait=$(( (currwait+10)%60 ))
50     done
51     sleep 1
52     l=$((l+1))
53     i=0
54 done

```

Development PC data log data extraction script - 12 Mbit/s

This script should be placed inside the same folder where the 12 Mbit/s logs are placed.

logExtractor.sh (12 Mbit/s)

```
1  #!/bin/bash
2
3  if [ $# -ne 1 ]; then
4      if [ $# -ne 2 -o ! $2 = "-M" ]; then
5          echo "Error using program. Expected one or two parameters: "
6          echo "1) Rate (<3|6|12>)"
7          echo "2) [-M] for log files with an extra 'M' after 'b'"
8          exit 1
9      fi
10 fi
11
12 # Defining rates (-b) and len (-l) arrays to correctly extract data
    from the logs
13 rates=( 0.5M 1M 1.5M 2M 3M 5M 7M 8M 8.1M 8.2M 8.3M 8.4M 8.5M 10M 13M
    16M 19M 22M )
14 len=( 16 48 104 208 500 700 1000 1200 1470 )
15
16 i=0
17 l=0
18 echo "len,b,tput,loss,lostpkt,sentpkt,okpkt" > "Rate$1_iperf_bl_test.
    csv"
19 while [ $l -lt ${#len[@]} ]; do
20     while [ $i -lt ${#rates[@]} ]; do
21         b=$(echo ${rates[$i]} | sed 's/.$//')
22         if [ ! -z "$2" -a "$2" = "-M" ]; then
23             tput=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail -1
                | sed 's/\s\s*/ /g' | cut -d" " -f7)
24             umeas=$(cat ServerLog_b_${rates[$i]}M_l_${len[$l]}B.txt | tail
                -1 | sed 's/\s\s*/ /g' | cut -d" " -f8)
```

```

25     # Convert measurements in KBit/s to MBit/s
26     if [[ $umeas = *"K"* ]]; then
27         tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
28     fi
29     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
30     sentpkt=$(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
31     echo "$(cat ServerLog_b_${rates[$i]}M_l_${len[$1]}B.txt | tail
        -1)"
32 else
33     tput=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail -1
        | sed 's/\s\s*/ /g' | cut -d" " -f7)
34     umeas=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail -1
        | sed 's/\s\s*/ /g' | cut -d" " -f8)
35     # Convert measurements in KBit/s to MBit/s
36     if [[ $umeas = *"K"* ]]; then
37         tput=$(echo "$tput" | awk '{printf("%.3f\n",$1/1000)}')
38     fi
39     lostpkt=$(echo $(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt
        | tail -1 | cut -d"/" -f2 | cut -d"/" -f1) | rev | cut -d" "
        -f1 | rev)
40     sentpkt=$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail
        -1 | cut -d"/" -f3 | sed 's/^[ \t]*//' | cut -d" " -f1)
41     echo "$(cat ServerLog_b_${rates[$i]}_l_${len[$1]}B.txt | tail
        -1)"
42 fi
43 okpkt=$((sentpkt-lostpkt))
44 loss=$(echo "$lostpkt $sentpkt" | awk '{printf("%.15f\n",$1/$2
        *100)}')
45 i=$((i+1))
46
47 # Print parsed information

```

```
48     echo "len=${len[$1]},b=$b,tput=$tput,loss=$loss,lostpkt=$lostpkt,
        sentpkt=$sentpkt,okpkt=$okpkt"
49     # Write to .csv file
50     echo "${len[$1]},$b,$tput,$loss,$lostpkt,$sentpkt,$okpkt" >> "
        Rate$1_iperf_b1_test.csv"
51 done
52 l=$((l+1))
53 i=0
54 done
```

Bibliography

- [1] PC Engines home page, PC Engines, [Online]. Available: <https://www.pcengines.ch/> (visited on 08/11/2018).
- [2] N. Agafonovs, G. Strazdins, and M. Greitans, «Accessible, Customizable, High-Performance, IEEE 802.11p Vehicular Communication Solution,» *2012 Vehicular Communications and Applications Workshop*, pp. 128–129, 2012.
- [3] «Directive 2010/40/EU of the European Parliament and of the Council of 7 July 2010 on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport,» *Official journal of the European Union, L 207/1*, pp. 1–13, Jul. 7, 2010. [Online]. Available: <https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:207:0001:0013:EN:PDF>.
- [4] Vehicle-to-Vehicle Wireless Communications for Virtual Traffic Lights, Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni, [Online]. Available: <http://www.wcsg.ieiit.cnr.it/people/bazzi/VTL.html> (visited on 08/11/2018).
- [5] G. Avino, M. Malinverno, C. Casetti, C. F. Chiasserini, F. Malandrino, M. Rapelli, and G. Zennaro, «Support of Safety Services through Vehicular Communications: The Intersection Collision Avoidance Use Case,» *IEEE AUTOMOTIVE 2018*, pp. 1–6, 2018. [Online]. Available: https://iris.polito.it/retrieve/handle/11583/2707415/199413/AUTOMOTIVE_2018_Preprint.pdf.

- [6] L. G. Nisi, «Sperimentazione di comunicazioni veicolari IEEE 802.11p per l'offloading di reti cellulari,» Università di Bologna, 2014.
- [7] O. K. Tonguz, N. Wisitpongphan, and F. Bai, «DV-CAST: a distributed vehicular broadcast protocol for vehicular ad hoc network,» *IEEE Wireless Communications*, pp. 47–57, Apr. 2010.
- [8] B. Bloessl, M. Segata, C. Sommer, and F. Dressler, «Performance Assessment of IEEE 802.11p with an Open Source SDR-Based Prototype,» *IEEE transactions on mobile computing*, vol. 17, no. 5, p. 1, May 2018.
- [9] Intelligent Roadside Unit, NXP, [Online]. Available: <https://www.nxp.com/applications/solutions/automotive/connectivity/intelligent-roadside-unit:INTELLIGENTRSU> (visited on 08/13/2018).
- [10] Z. Iqbal, T. Saeed, and N. A. Zafar, «Effective Formal Unicast Routing for VANETs,» *2017 Fifth International Conference on Aerospace Science & Engineering (ICASE)*, Nov. 14, 2017.
- [11] S. He, J. Li, and T. Z. Qiu, «Vehicle-to-Pedestrian Communication Modeling and Collision Avoiding Method in Connected Vehicle Environment,» *Transportation Research Record Journal of the Transportation Research Board*, vol. 2621, pp. 21–30, Jan. 2017. [Online]. Available: https://www.researchgate.net/publication/309034893_Vehicle_to_Pedestrian_Communication_Modeling_and_Collision_Avoiding_Methodology_in_Connected_Vehicle_Environment (visited on 09/14/2018).
- [12] M. Malinverno, G. Avino, C. Casetti, C. F. Chiasserini, F. Malandrino, and S. Scarpina, «Performance Analysis of C-V2I-based Automotive Collision Avoidance,» *IEEE WoWMoM*, Mar. 24, 2018. arXiv: 1803.08798v1 [quant-ph].
- [13] J. A. Sanguesa, J. Barrachina, M. Fogue, P. Garrido, F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, «Sensing Traffic Density Combining V2V and V2I

- Wireless Communications,» *ResearchGate Sensors*, Dec. 2015. [Online]. Available: https://www.researchgate.net/publication/287406124_Sensing_Traffic_Density_Combining_V2V_and_V2I_Wireless_Communications?_sg=ocHlImjCYgTQ8P__homizl4Fx461b08rSJhJ1bx9X14ihS_m5YwpmTSZM0iQeg68BHw-b-Qzw.
- [14] X. Xiang, X. Wang, and Z. Zhou, «Self-Adaptive On-Demand Geographic Routing for Mobile Ad Hoc Networks,» *IEEE transactions on mobile computing*, vol. 11, no. 9, pp. 1572–1586, Aug. 18, 2011.
- [15] P.-H. Lee and T.-C. Huang, «An Improved Distance-Based Scheme for Broadcast Storm Suppression in VANETs,» *2014 9th IEEE International Conference on Networking, Architecture, and Storage*, pp. 200–206, Aug. 6, 2014.
- [16] J. Heinovski, F. Klingler, F. Dressler, and C. Sommer, «Performance Comparison of IEEE 802.11p and ARIB STD-T109,» *2016 IEEE Vehicular Networking Conference (VNC)*, Dec. 8, 2016.
- [17] *IEEE Std 802.11-2016 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11™-2016, IEEE Computer Society, 2016.
- [18] *Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band*, Draft ETSI EN 302 663 V1.2.0 (2012-11), 2012. [Online]. Available: https://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.02.00_20/en_302663v010200a.pdf.
- [19] *Intelligent Transport Systems (ITS); Communications Architecture*, ETSI EN 302 665 V1.1.1 (2010-09), 2010. [Online]. Available: https://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf.

- [20] *Intelligent Transport Systems (ITS); Decentralized Congestion Control Mechanisms for Intelligent Transport Systems operating in the 5 GHz range; Access layer part*, ETSI TS 102 687 V1.1.1 (2011-07), 2011. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/102600_102699/102687/01.01.01_60/ts_102687v010101p.pdf.
- [21] (Mar. 8, 2012). SAE J2735 DSRC Message Dictionary Overview. Hosted on the Car 2 Car Consortium website, CAMP - VSC3 Consortium, [Online]. Available: https://www.car-2-car.org/fileadmin/user_upload/OEM_Workshop_WOB/Message_Dictionary_Overview.pdf (visited on 08/14/2018).
- [22] Coexistence Interoperability of ETSI ITS-G5 & DSRC, ETSI, [Online]. Available: <https://www.etsi.org/about/what-we-do/plugtests/calendar-of-events/coexistence-interoperability-of-etsi-its-g5-dsrc> (visited on 08/14/2018).
- [23] SDK, Cohda Wireless, [Online]. Available: <http://www.cohdawireless.com/solutions/sdk/> (visited on 08/15/2018).
- [24] B. Bloessl, F. Klingler, F. Missbrenner, and C. Sommer, «A Systematic Study on the Impact of Noise and OFDM Interference on IEEE 802.11p,» *2017 IEEE Vehicular Networking Conference (VNC)*, Nov. 27, 2017.
- [25] J. de Jongh, J. van de Sluis, D. Heuven, A. Voronov, and I. Passchier, *IEEE 802.11p [CTU-IIIG] on PCEngines APUID running Voyage*, v0.7, Feb. 11, 2016.
- [26] XCVR2450, Ettus Research, [Online]. Available: <https://kb.ettus.com/XCVR2450> (visited on 08/19/2018).
- [27] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Networking Services*, IEEE Std 1609.3™-2016, IEEE Vehicular Technology Society, 2016.
- [28] *IEEE Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages*, IEEE Std 1609.2™-2016, IEEE Vehicular Technology Society, 2016.

- [29] *IEEE Guide for Wireless Access in Vehicular Environments (WAVE) Architecture*, IEEE Std 1609.0™-2013, 2013.
- [30] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-Channel Operation*, IEEE Std 1609.4™-2016, IEEE Vehicular Technology Society, 2016.
- [31] *IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Identifier Allocations*, IEEE Std 1609.12™-2016, 2016.
- [32] A. Al-Jzari and K. Iviva, «Cyclic Prefix Length Determination for Orthogonal Frequency Division Multiplexing System over Different Wireless Channel Models Based on the Maximum Excess Delay Spread,» *American Journal of Engineering and Applied Sciences*, vol. 2015, 8 (1), pp. 82–93, Apr. 10, 2015. [Online]. Available: <http://thescipub.com/pdf/10.3844/ajeassp.2015.82.93> (visited on 08/20/2018).
- [33] Y. Li, «An overview of the DSRC/WAVE technology,» *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Jan. 2012.
- [34] (May 2007). AMD Geode™ CS5536 Companion Device Data Book, AMD, [Online]. Available: https://support.amd.com/TechDocs/33238G_cs5536_db.pdf (visited on 09/16/2018).
- [35] seagreen. (Apr. 7, 2009). What's the difference between "dB", "dBm", and "dBi". DSLReports.com, Ed., [Online]. Available: <http://www.dslreports.com/faq/14091> (visited on 10/02/2018).
- [36] S. Laux, G. S. Pannu, S. Schneider, J. Tiemann, F. Klingler, C. Sommer, and F. Dressler, «OpenC2X - An Open Source Experimental and Prototyping Platform Supporting ETSI ITS-G5,» in *8th IEEE Vehicular Networking Conference (VNC 2016), Demo Session*, Columbus, OH: IEEE, Dec. 2016, pp. 152–153. doi: 10.1109/VNC.2016.7835955.

- [37] OpenWrt. (2018). Table of Hardware, [Online]. Available: <https://openwrt.org/toh/start> (visited on 09/17/2018).
- [38] F. Klingler. CCS Labs OpenC2X (Paderborn University), [Online]. Available: <http://www.ccs-labs.org/software/openc2x/> (visited on 09/17/2018).
- [39] OpenWrt. (Jun. 23, 2018). Supported Devices, [Online]. Available: https://openwrt.org/supported_devices (visited on 09/17/2018).
- [40] —, (Aug. 8, 2018). OpenWrt Version History, [Online]. Available: <https://openwrt.org/about/history> (visited on 09/17/2018).
- [41] —, (Aug. 3, 2018). About the OpenWrt/LEDE project, [Online]. Available: <https://openwrt.org/about> (visited on 09/17/2018).
- [42] —, (Aug. 23, 2016). OpenWrt's build system – About, [Online]. Available: <https://wiki.openwrt.org/about/toolchain> (visited on 09/17/2018).
- [43] G. Kroah-Hartman, «The Kernel Configuration and Build Process,» *Linux Journal (online)*, May 1, 2003. [Online]. Available: <https://www.linuxjournal.com/article/6568> (visited on 09/17/2018).
- [44] A. Abunei, C.-R. Comşa, and I. Bogdan, «Implementation of a Cost-effective V2X hardware and software platform,» *IEEE 2016 International Conference on Communications (COMM)*, pp. 367–370, Jun. 9, 2016. doi: 10.1109/iccomm.2016.7528312.
- [45] R. Lisový, M. Sojka, and Z. Hanzálek, «IEEE 802.11p Linux Kernel Implementation,» *Technical report, Czech Technical University in Prague*, Dec. 10, 2014. [Online]. Available: https://rttime.felk.cvut.cz/publications/public/ieee80211p_linux_2014_final_report.pdf (visited on 09/17/2018).
- [46] F. Chou. (Mar. 15, 2015). Linux Wireless Networking: a short walk. L. (blog), Ed., [Online]. Available: <https://www.linux.com/blog/linux-wireless-networking-short-walk> (visited on 10/01/2018).

- [47] iPerf. iPerf Home Page, [Online]. Available: <https://iperf.fr/> (visited on 10/02/2018).
- [48] CTU-IIG 802.11p-linux project, Czech Technical University, Industrial Informatics Research Center, [Online]. Available: <https://github.com/CTU-IIG/802.11p-linux> (visited on 08/11/2018).
- [49] OpenWrt. (Sep. 19, 2018). Build system – Installation, [Online]. Available: <https://openwrt.org/docs/guide-developer/build-system/install-buildsystem> (visited on 11/08/2018).
- [50] —, (Aug. 29, 2018). Build system – Usage, [Online]. Available: [https://openwrt.org/docs/guide-developer/build-system/use-buildsystem?s\[\]=make&s\[\]=defconfig](https://openwrt.org/docs/guide-developer/build-system/use-buildsystem?s[]=make&s[]=defconfig) (visited on 10/03/2018).
- [51] —, (Mar. 4, 2018). Cross Compile, [Online]. Available: <https://openwrt.org/docs/guide-developer/crosscompile> (visited on 10/04/2018).
- [52] —, (Jun. 7, 2018). Filesystems, [Online]. Available: <https://openwrt.org/docs/techref/filesystems>.
- [53] —, (Feb. 17, 2018). Working with patches in the build system, [Online]. Available: [https://openwrt.org/docs/guide-developer/build-system/use-patches-with-buildsystem?s\[\]=quilt](https://openwrt.org/docs/guide-developer/build-system/use-patches-with-buildsystem?s[]=quilt) (visited on 10/04/2018).
- [54] TCPDUMP. (2018). Manpage of TCPDUMP, [Online]. Available: <http://www.tcpdump.org/manpages/tcpdump.1.html> (visited on 10/05/2018).
- [55] (Jun. 25, 2015). iptables-extensions man, [Online]. Available: <http://ipset.netfilter.org/iptables-extensions.man.html> (visited on 10/05/2018).
- [56] RadioTap. Defined Fields, [Online]. Available: <http://www.radiotap.org/fields/defined> (visited on 10/07/2018).

- [57] (Jan. 26, 2015). Linux Wireless: mac80211 support for radiotap, [Online]. Available: https://wireless.wiki.kernel.org/en/developers/documentation/radiotap#mac80211_support_for_radiotap (visited on 10/07/2018).
- [58] (Aug. 24, 2018). The Linux Kernel Archives: Active kernel releases, [Online]. Available: <https://www.kernel.org/category/releases.html> (visited on 10/05/2018).
- [59] (Sep. 29, 2018). OpenWrt Forums: What is the purpose of "200-reduce_size.patch" in iw-4.14? [Online]. Available: <https://forum.openwrt.org/t/what-is-the-purpose-of-200-reduce-size-patch-in-iw-4-14/22267> (visited on 10/05/2018).
- [60] OpenWrt. (Sep. 3, 2018). 4/32 warning, [Online]. Available: https://openwrt.org/supported_devices/432_warning (visited on 10/05/2018).
- [61] K. Technologies, Ed., *Keysight Technologies: Spectrum Analysis Basics, Application Note 150*, Nov. 2, 2016. [Online]. Available: <http://literature.cdn.keysight.com/litweb/pdf/5952-0292.pdf> (visited on 09/24/2018).
- [62] J. Crane. (Jul. 4, 2017). Wi-Spy Data Sheet. MetaGeek, Ed., [Online]. Available: <https://support.metageek.com/hc/en-us/articles/203802010-Wi-Spy-Data-Sheet> (visited on 09/24/2018).
- [63] ———, (2018). Chanalyzer + Wi-Spy User Guide. MetaGeek, Ed., [Online]. Available: https://support.metageek.com/hc/en-us/articles/201872824-Chanalyzer-Wi-Spy-User-Guide?utm_campaign=Software&utm_medium=Chanalyzer%205&utm_source=Help%20Menu (visited on 09/24/2018).
- [64] Kismet. Kismet Spectrum-Tools (Home page), [Online]. Available: <https://www.kismetwireless.net/spectools/> (visited on 09/25/2018).
- [65] Ubuntu. Man page for spectool_raw. Canonical, Ed., [Online]. Available: http://manpages.ubuntu.com/manpages/trusty/man1/spectool_raw.1.html (visited on 09/25/2018).

- [66] PC Engines apu1d product file, PC Engines, [Online]. Available: <https://www.pcengines.ch/apu1d.htm> (visited on 10/12/2018).
- [67] OpenWrt. (May 2, 2018). PC Engines APU, [Online]. Available: <https://openwrt.org/toh/pcengines/apu> (visited on 10/12/2018).
- [68] UNEX DHXA-222, Unex Technology Corporation, [Online]. Available: <https://unex.com.tw/products/wi-fi/interfaces/pcie-wifi/80211n-bluetooth/detail/dhxa-222> (visited on 10/12/2018).
- [69] *PC Engines apu2 series system board*, PC Engines, Dec. 10, 2017. [Online]. Available: <https://www.pcengines.ch/pdf/apu2.pdf> (visited on 10/12/2018).
- [70] —, (Jun. 2, 2018). NTP client / NTP server, [Online]. Available: [https://openwrt.org/docs/guide-user/services/ntp/client-server?s\[\]=ntp](https://openwrt.org/docs/guide-user/services/ntp/client-server?s[]=ntp) (visited on 10/21/2018).
- [71] (Sep. 26, 2017). chrony - Introduction, chrony, [Online]. Available: <https://chrony.tuxfamily.org/> (visited on 10/21/2018).
- [72] (2018). NTP (Network Time Protocol), INRiM, [Online]. Available: <http://rime.inrim.it/labtf/ntp/> (visited on 10/21/2018).
- [73] (Aug. 28, 2018). chronyc(1) Manual Page, chrony, [Online]. Available: <https://chrony.tuxfamily.org/doc/3.4/chronyc.html> (visited on 10/21/2018).
- [74] (Nov. 11, 2018). OpenWrt Forums: OpenWrt, Linux kernel and sockets: how much time a write() call is blocked in front of a full buffer? [Online]. Available: <https://forum.openwrt.org/t/openwrt-linux-kernel-and-sockets-how-much-time-a-write-call-is-blocked-in-front-of-a-full-buffer/24803/6> (visited on 11/13/2018).
- [75] (Sep. 2, 2016). Patchwork: [v5] ath9k: Switch to using mac80211 intermediate software queues., [Online]. Available: <https://patchwork.kernel.org/patch/9311037/> (visited on 11/14/2018).