POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Analisi e implementazione di un sistema per comunicazioni intraveicolari a bassa latenza

Relatori
Prof. Enrico Masala
Prof. Guido Marchetto

Candidato
Alberto Gambuzza

Aprile 2019

Indice

1. Introduzione

Lo scopo di questa tesi è la caratterizzazione di un sistema di comunicazione multimediale, basato principalmente su traffico VoIP, da utilizzarsi in un ambiente rumoroso, in cui la normale comunicazione verbale può risultare difficile. Un'applicazione tipica di un sistema di questo tipo si può avere all'interno di un veicolo, in cui è necessario comunicare in modo efficiente e, quindi, con un buon livello di intelligibilità a destinazione. Il mittente, quindi, utilizzerà un microfono (o apparecchiatura simile) con cui parlerà, il flusso audio passerà attraverso il sistema di comunicazione dopo essere stato elaborato e arriverà a destinazione, dove il ricevente potrà ascoltare il messaggio con delle cuffie. Il sistema analizzato fa uso di sistemi embedded e di una distribuzione DDS (Data Distribution Service) open source, OpenDDS, che sfrutta il paradigma publish/subscriber in cui le parti sono controllate da un broker che le mette in contatto. Nel dettaglio, sono stati utilizzati un Microchip EVB-KSZ9477, ovvero un managed switch con un processore Atmel su cui è preinstallata una distribuzione Linux Embedded e una evaluation board i.MX SECO CQ7-A42 (SECO Q7-928 w/i.MX6 Quad @800Mhz – 4GB RAM – eMMC 8GB), fornita di microSD su cui è stata caricata la SDK utile per compilare l'ambiente OpenDDS per l'architettura specifica (ARM Cortex-A9). La versione di OpenDDS utilizzata è la 3.13. Inoltre, è stata utilizzata una macchina virtuale VirtualBox (Ubuntu 14.04 LTS x64) come ambiente di sviluppo, configurata ad hoc, su cui è stato compilato l'ambiente di sviluppo OpenDDS.

La prima parte della tesi riguarda lo studio delle caratteristiche e delle prestazioni dello switch Microchip, in particolare l'implementazione del RSTP (Rapid Spanning Tree Protocol) e le differenti tipologie di Quality of Service, utili per differenziare il traffico in base alle esigenze di priorità. Conclude la prima parte una panoramica sui principali comandi da terminale utili a configurare le varie caratteristiche dello switch e ad estrarre

delle statistiche di utilizzo, principalmente in termini di pacchetti trasmessi, ricevuti e persi.

Nella seconda parte, sono stati trattati alcuni tra i sistemi di text-to-speech e speech recognition disponibili gratuitamente e non, per poter essere eventualmente utilizzati nel contesto del sistema di comunicazione intraveicolare studiato, in modo da standardizzare la comunicazione e non utilizzare la voce naturale ma una prodotta da una macchina.

Nella terza parte, infine, è stato studiato il sistema OpenDDS, compilato sia sulla VM VirtualBox che sulla evaluation board Seco, attraverso il quale si rende la comunicazione topic-based, ovvero basata su argomenti che vengono ricevuti solo da chi è interessato e si è iscritto per ricevere messaggi relativi a quello specifico argomento, diminuendo sensibilmente il numero di pacchetti in transito per la rete locale e, di conseguenza, migliorando le prestazioni complessive del sistema in oggetto.

2. Studio su Microchip EVB-KSZ9477

Il dispositivo principale utilizzato per il sistema, per mettere in comunicazione in una rete locale i diversi utilizzatori, è un managed switch Microchip, fornito con una evaluation board con processore Atmel e una distribuzione Linux Embedded installata, compresi tutti i driver necessari alla comunicazione tra switch e processore Atmel. Interagendo da terminale Linux, è possibile settare la maggior parte delle impostazioni dello switch, con delle semplici modifiche a delle variabili spesso corrispondenti a dei registri specifici. Le caratteristiche di interesse che sono state studiate sono l'implementazione dello Spanning Tree Protocol (STP), nella sua evoluzione più recente, il Rapid Spanning Tree Protocol (RSTP) e le diverse tipologie di quality of service (QoS).

2.1 Rapid Spanning Tree Protocol (RSTP)

La prima caratteristica che si è deciso di analizzare è il supporto del Microchip al protocollo STP (Spanning Tree Protocol), in particolare la sua variante RSTP (Rapid STP). Questo protocollo di livello 2, utilizzato negli switch e nei bridge, è definito nello standard IEEE 802.1D. Il suo principale scopo è quello di assicurarsi di non creare cicli quando si hanno dei percorsi ridondanti nella rete, dal momento che i cicli sono potenzialmente dannosi per il traffico di rete. (1)

Il protocollo STP, a partire dalla topologia di rete, crea un albero, in cui è ammesso un solo percorso da qualunque host ad un altro e la cui radice è uno degli switch chiamato Root Bridge, eletto a questo ruolo tra tutti gli switch presenti nella topologia.

Successivamente, il protocollo, periodicamente, decide quale porta di ogni switch impostare in stato di forwarding o in stato di blocking. Il percorso da un host ad un altro, è scelto sulla base del costo attribuito ad esso, calcolato in riferimento alla distanza per raggiungere il Root Bridge.

Alla fine, dopo la convergenza del protocollo, ogni switch avrà tutte le sue porte in uno dei seguenti ruoli:

1) Root Port: porta attraverso cui è possibile raggiungere il Root Bridge, una per ogni switch non Root Bridge;
2) Designated Port: porta in stato di forwarding, tutte le porte del Root Bridge sono in questo stato;
3) Blocked Port: porta disattivata. (2)

La variante Rapid STP, migliora il tempo di convergenza dello Spanning Tree, spesso portato a circa 1 secondo, rendendo il suo utilizzo più diffuso. Variano i ruoli delle porte, infatti si hanno:

1) Root Port: identica funzione dello Spanning Tree tradizionale;
2) Designated Port: identica funzione dello Spanning Tree tradizionale;
3) Alternate Port: porta attraverso cui si ha un percorso alternativo verso il Root Bridge, è quella designata per sostituire la Root Port in caso di malfunzionamenti;
4) Backup Port: utilizzata come "riserva" per non isolare lo switch a cui appartiene da uno specifico link della LAN. (2)

Per testare l'effettivo supporto del Microchip a questo protocollo, e le sue prestazioni, in prima istanza si sono svolti dei test utilizzando due *Cisco SG300-10 10-Port Gigabit Managed Switch*, per familiarizzare con le caratteristiche del protocollo e verificarne il funzionamento con questi due dispositivi. Solo in seguito è stato aggiunto il Microchip alla topologia, eseguendo numerosi test per osservarne le peculiarità. Seguono i test effettuati con descrizione delle differenti topologie utilizzate e dei risultati sperimentali ottenuti. Tutti i test eseguiti si sono basati su un cambio di topologia (ad esempio disconnessione di un cavo o disconnessione di un intero dispositivo dalla rete), in modo

da attivare le operazioni proprie del protocollo RSTP e misurare i tempi di convergenza dello stesso, attraverso la generazione di traffico da un dispositivo ad un altro e la cattura di suddetto traffico con l'ausilio del tool packet sniffer *Wireshark.*

2.1.1 Test 1

Per questo primo test, sono stati collegati i due switch Cisco ad anello, uno dei quali collegato al PC tramite cavo Ethernet, in modo da poter catturare il traffico passante per la rete locale creata. La topologia è quella raffigurata in Figura 1.

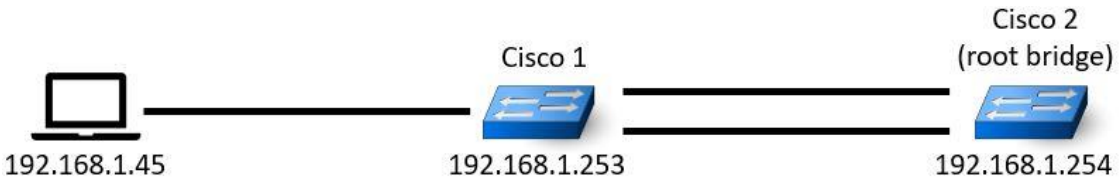
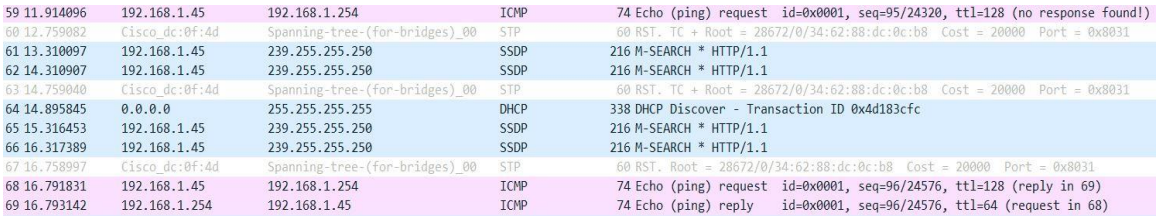


Figura 1 – Topologia di rete

Secondo il protocollo RSTP, lo switch sopra chiamato "Cisco 2" è stato designato come root bridge. Al fine di testare l'efficacia di questo protocollo, si è provveduto in prima istanza ad attivare il supporto a RSTP accedendo all'interfaccia web di configurazione dei due switch Cisco.

È stato generato del traffico *ping*, dal PC (192.168.1.45) al Cisco 2 (192.168.1.254), attraverso il prompt dei comandi di Windows. Dopo qualche secondo, è stato scollegato il cavo della porta di root del Cisco 1, ovvero è stato interrotto uno dei due path da Cisco 1 a Cisco 2, provocando in questo modo un cambio di topologia, con conseguente interruzione del trasferimento dei pacchetti ICMP in atto. **La trasmissione è tornata regolare dopo circa 4 secondi**, probabilmente a causa del parametro Hello msg time del

protocollo STP impostato a 2 secondi e al tempo di attesa del *ping reply* (settato al valore di default, che generalmente è proprio di 4 secondi). Questi due fattori hanno avuto con certezza un forte impatto sul tempo di convergenza del protocollo RSTP. Segue (Figura 2) un estratto della cattura del traffico durante il test, in cui si evidenzia il riconoscimento del cambio di topologia con conseguente perdita di un pacchetto *ping reply*, e il ritorno ad un normale flusso di *ping request* / *ping reply* in seguito.



59	11.914096	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=95/24320, ttl=128 (no response found!)
60	12.759082	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
61	13.310097	192.168.1.45	239.255.255.250	SSDP	216 M-SEARCH * HTTP/1.1
62	14.310907	192.168.1.45	239.255.255.250	SSDP	216 M-SEARCH * HTTP/1.1
63	14.759040	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
64	14.895845	0.0.0.0	255.255.255.255	DHCP	338 DHCP Discover - Transaction ID 0x4d183cfc
65	15.316453	192.168.1.45	239.255.255.250	SSDP	216 M-SEARCH * HTTP/1.1
66	16.317389	192.168.1.45	239.255.255.250	SSDP	216 M-SEARCH * HTTP/1.1
67	16.758997	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
68	16.791831	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=96/24576, ttl=128 (reply in 69)
69	16.793142	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=96/24576, ttl=64 (request in 68)

Figura 2 - Cattura

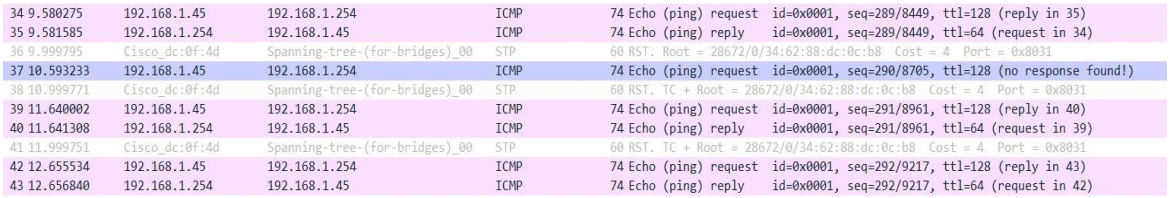
2.1.2 Test 2

In questo secondo test, è stata mantenuta la stessa topologia ed è stato generato del traffico *ping*, dal PC (192.168.1.45) al Cisco 2 (192.168.1.254). Dopo qualche secondo, a differenza del caso precedente, è stato scollegato il cavo connesso alla porta in stato "Alternate" del Cisco 1. Essendo una porta "di riserva", quindi non utilizzata per trasmettere del traffico, non si sono registrati cambi di topologia né tantomeno perdite di pacchetti.

2.1.3 Test 3

In questo test, per cercare di ridurre il tempo di convergenza del protocollo RSTP, si è cercato di risolvere le criticità riscontrate nel primo test. Per questo motivo, è stato deciso di ridurre il parametro Hello msg time dei due switch Cisco portandolo ad 1

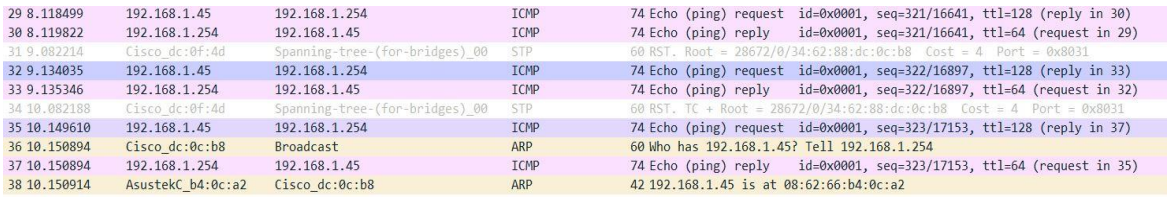
secondo. In questo modo, nella migliore delle ipotesi, un eventuale cambio di topologia viene riscontrato nella metà del tempo impiegato precedentemente (dal momento che il valore di default precedentemente impostato era di 2 secondi). Inoltre, è stato diminuito il tempo di attesa del *ping reply* da terminale, portandolo a 500 ms, per far sì che catturando il traffico in corso si potesse verificare la convergenza del protocollo RSTP in tempi più rapidi. Dopo aver effettuato queste due modifiche, è stato generato del traffico *ping*, dal PC (192.168.1.45) al Cisco 2 (192.168.1.254), disconnettendo, dopo qualche secondo, la porta di root del Cisco 1. **La trasmissione, questa volta, è tornata regolare dopo circa 1 secondo**, come è possibile verificare dal seguente estratto della cattura (Figura 3). Un solo pacchetto di *ping request* non ha ricevuto risposta.



34	9.580275	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=289/8449, ttl=128 (reply in 35)
35	9.581585	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=289/8449, ttl=64 (request in 34)
36	9.999795	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 4 Port = 0x8031
37	10.593233	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=290/8705, ttl=128 (no response found!)
38	10.999771	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 4 Port = 0x8031
39	11.640002	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=291/8961, ttl=128 (reply in 40)
40	11.641308	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=291/8961, ttl=64 (request in 39)
41	11.999751	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 4 Port = 0x8031
42	12.655534	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=292/9217, ttl=128 (reply in 43)
43	12.656840	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=292/9217, ttl=64 (request in 42)

Figura 3 – Estratto da cattura

Lo stesso test è stato ripetuto più volte, per cercare di ottenere un tempo di convergenza più breve, poiché se l'Hello msg che individua il cambio di topologia si presenta prima di una *ping request* (con un certo tempo tra i due pacchetti), è possibile rilevare la convergenza del protocollo in tempo utile per non far perdere la risposta. In una delle prove, tutti i pacchetti *ping request* hanno ricevuto risposta, nonostante il cambio di topologia riscontrato. Questo caso particolare è facilmente verificabile nell'estratto della cattura che segue (Figura 4).



29	8.118499	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=321/16641, ttl=128 (reply in 30)
30	8.119822	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=321/16641, ttl=64 (request in 29)
31	9.082214	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 4 Port = 0x8031
32	9.134035	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=322/16897, ttl=128 (reply in 33)
33	9.135346	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=322/16897, ttl=64 (request in 32)
34	10.082188	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 4 Port = 0x8031
35	10.149610	192.168.1.45	192.168.1.254	ICMP	74 Echo (ping) request id=0x0001, seq=323/17153, ttl=128 (reply in 37)
36	10.150894	Cisco_dc:0c:b8	Broadcast	ARP	60 Who has 192.168.1.45? Tell 192.168.1.254
37	10.150894	192.168.1.254	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=323/17153, ttl=64 (request in 35)
38	10.150914	AsustekC_b4:0c:a2	Cisco_dc:0c:b8	ARP	42 192.168.1.45 is at 08:62:66:b4:0c:a2

Figura 4 – Estratto da cattura

2.1.4 Introduzione del Microchip KSZ9477 nella rete

A questo punto, dopo aver testato il funzionamento e la convergenza del protocollo RSTP utilizzando solo due switch Cisco, è stato introdotto il Microchip nella topologia di rete, così come mostrato in Figura 5.

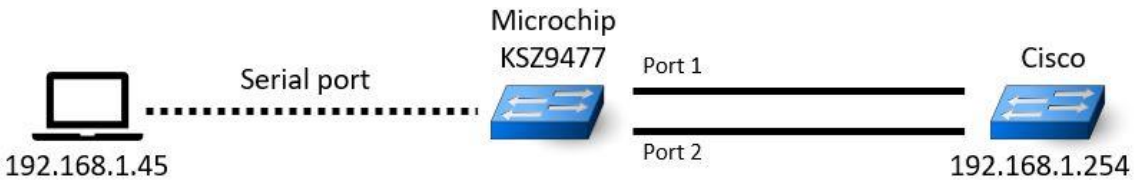


Figura 5 – Topologia di rete

Il Microchip è stato collegato ad anello ad uno switch Cisco e tramite porta seriale al PC, in modo da avere accesso all'interfaccia CLI di configurazione del dispositivo e, di conseguenza, visualizzare tutti i parametri utili per il caso. Come prima operazione, è stato attivato il supporto al RSTP, e successivamente sono stati settati tutti i parametri propri del protocollo, visualizzandoli complessivamente accedendo alla variabile stp_br_info del Microchip, che mostra una panoramica della configurazione del protocollo, con l'importante informazione legata al tempo dall'ultimo cambio di topologia (time since topology change), che ha permesso di avere una stima del tempo di convergenza del protocollo per il primo test eseguito con questa topologia.

2.1.4.1 Test 1

Con la topologia mostrata, la Port 1 del Microchip è stata individuata dal protocollo come porta di root. Disconnettendo il cavo e mostrando le informazioni sullo STP nell'istante immediatamente successivo, è stato individuato il cambio di topologia, con la

Port 2 nuova porta di root. La convergenza si ha in meno di un secondo, come è possibile vedere nello screenshot seguente (Figura 6) dal parametro “time since topology change”, che risulta 0 (quindi meno di 1 secondo), e dal parametro “topology change = yes”.

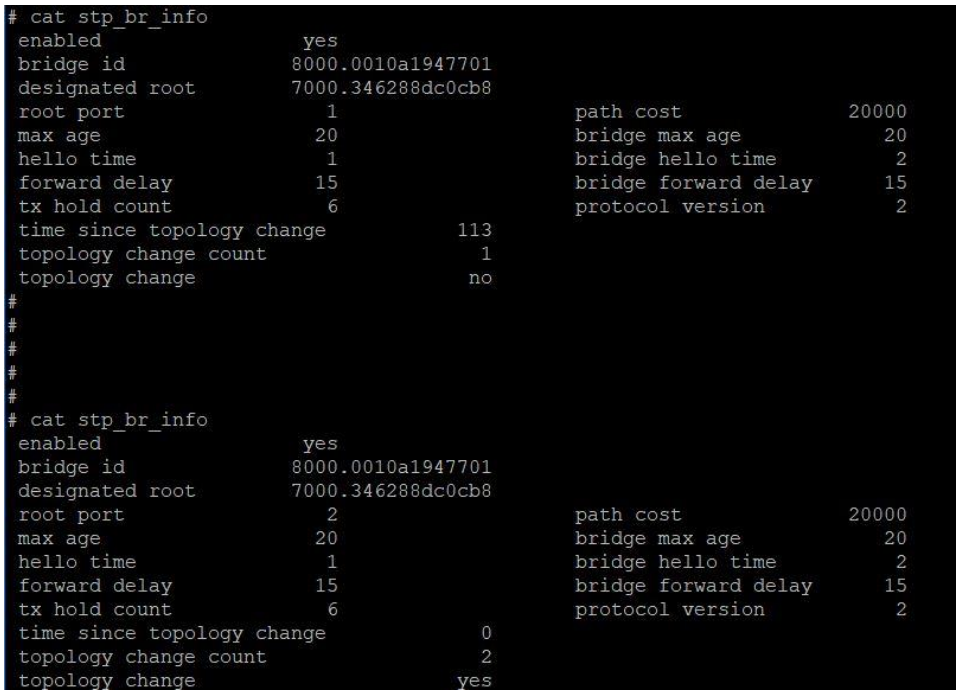


Figura 6 – Screenshot da terminale CLI del Microchip KSZ9477

2.1.4.2 Test 2

A questo punto si è passati ad una topologia più complessa, introducendo un secondo switch Cisco in una configurazione ad anello (Topologia 3 – Figura 7).

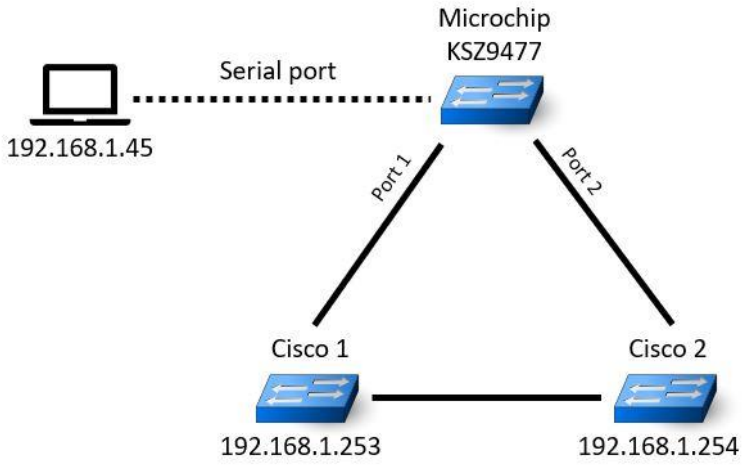


Figura 7 – Topologia di rete

Anche in questa configurazione, la Port 1 del Microchip è stata individuata come porta di root (informazione verificata accedendo alla variabile stp_br_info del Microchip). Disconnettendola e mostrando le informazioni sullo STP più volte negli istanti immediatamente successivi, risulta la Port 2 come porta di root dopo circa 5 secondi. Il tempo di convergenza è leggermente superiore rispetto ai casi precedenti a causa della maggiore complessità della topologia di rete, ma resta comunque un dato approssimativo, dal momento che il tempo è stato rilevato semplicemente accedendo alle informazioni da CLI. Nel test che seguirà, la convergenza sarà misurata in modo più accurato, facendo passare del traffico nella rete catturato con Wireshark.

2.1.4.3 Test 3

Per avere una misura più precisa ed accurata del tempo di convergenza del RSTP, è stato utilizzato un secondo PC collegato al Microchip, mantenendo una configurazione ad anello con gli altri due Cisco, come mostrato in Figura 8, e facendo passare del traffico da un PC all'altro.

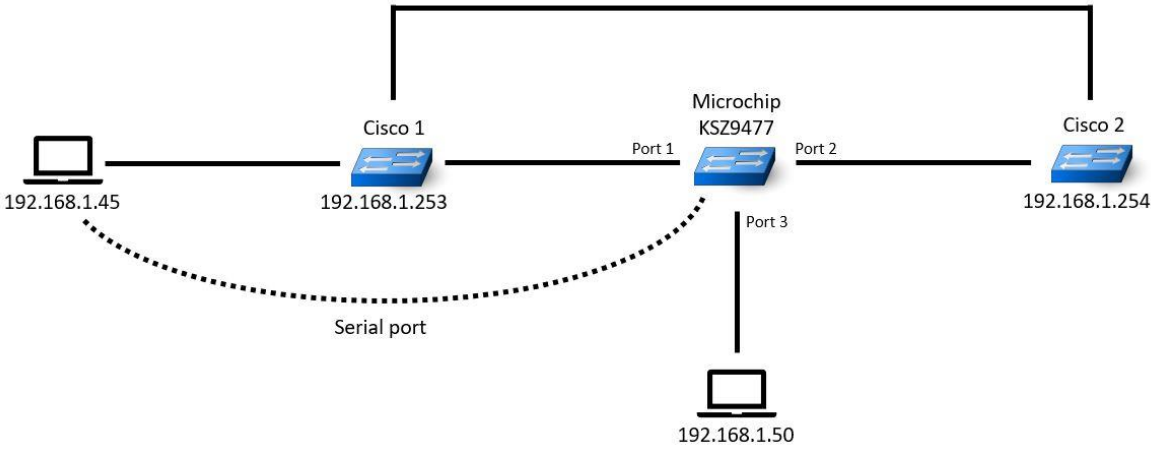


Figura 8 – Topologia di rete

Dopo aver configurato la topologia come in figura, è stato effettuato del traffico *ping* dal PC1 (192.168.1.45) al PC2 (192.168.1.50). Dopo qualche secondo, è stata disconnessa la root port del Microchip (Port 2). Si ha convergenza dopo circa 4 secondi, come visibile dalla cattura in Figura 9, probabilmente a causa dei tempi di attesa del *ping reply*, che è stato lasciato al valore di default.

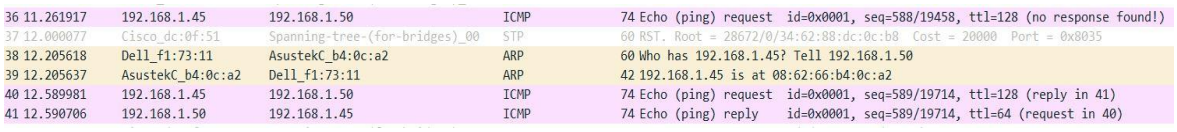
60	22.287711	192.168.1.45	192.168.1.50	ICMP	74	Echo (ping) request id=0x0001, seq=331/19201, ttl=128 (reply in 61)
61	22.288408	192.168.1.50	192.168.1.45	ICMP	74	Echo (ping) reply id=0x0001, seq=331/19201, ttl=64 (request in 60)
62	22.999466	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
63	23.303278	192.168.1.45	192.168.1.50	ICMP	74	Echo (ping) request id=0x0001, seq=332/19457, ttl=128 (no response found!)
64	23.999487	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
65	24.085166	0.0.0.0	255.255.255.255	DHCP	338	DHCP Discover - Transaction ID 0x3b3d1b63
66	24.999466	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
67	25.999445	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
68	26.999410	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
69	27.999402	Cisco_dc:0f:4d	Spanning-tree-(for-bridges)_00	STP	60	RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8031
70	28.224972	192.168.1.45	192.168.1.50	ICMP	74	Echo (ping) request id=0x0001, seq=333/19713, ttl=128 (reply in 71)
71	28.225630	192.168.1.50	192.168.1.45	ICMP	74	Echo (ping) reply id=0x0001, seq=333/19713, ttl=64 (request in 70)

Figura 9 – Estratto da cattura

2.1.4.4 Test 4

Con la stessa topologia del test precedente, è stato abbassato il tempo di attesa del *ping reply* portandolo a 100 ms, con lo stesso approccio utilizzato per i test senza

Microchip, in modo da far convergere il protocollo RSTP più rapidamente. Così come era prevedibile, il tempo di convergenza del protocollo è diminuito sensibilmente a circa 1,5 secondi (Figura 10). Un solo *ping request* non ha ricevuto risposta.

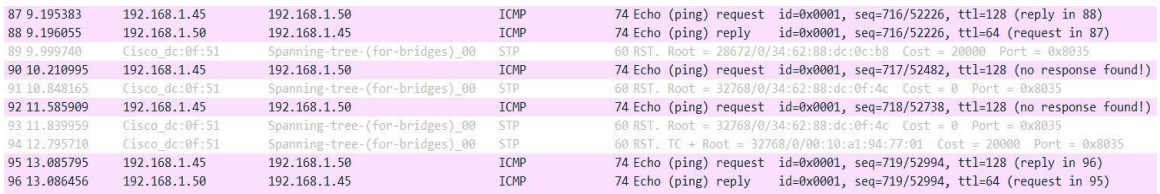


36	11.261917	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=588/19458, ttl=128 (no response found!)
37	12.000077	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8035
38	12.205618	Dell_f1:73:11	AsustekC_b4:0c:a2	ARP	60 Who has 192.168.1.45? Tell 192.168.1.50
39	12.205637	AsustekC_b4:0c:a2	Dell_f1:73:11	ARP	42 192.168.1.45 is at 08:62:66:b4:0c:a2
40	12.589981	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=589/19714, ttl=128 (reply in 41)
41	12.590706	192.168.1.50	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=589/19714, ttl=64 (request in 40)

Figura 10 – Estratto da cattura

2.1.4.5 Test 5

Mantenendo la topologia del test precedente, compreso il tempo di attesa del *ping reply* a 100 ms, si è deciso di provare ad isolare completamente uno dei due switch Cisco, per verificare il comportamento del Microchip di fronte ad una variazione consistente della topologia. Per questo motivo, dopo aver iniziato a trasmettere un traffico *ping* tra PC1 (192.168.1.45) e PC2 (192.168.1.50), è stata staccata l'alimentazione del dispositivo individuato come root bridge dal protocollo RSTP, ovvero quello che in Figura 7 è chiamato Cisco 2 (192.168.1.254). La trasmissione è tornata regolare dopo meno di 3 secondi e i pacchetti senza risposta sono stati 2, così come visibile nell'estratto della cattura effettuata (Figura 11).



87	9.195383	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=716/52226, ttl=128 (reply in 88)
88	9.196055	192.168.1.50	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=716/52226, ttl=64 (request in 87)
89	9.999740	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 28672/0/34:62:88:dc:0c:b8 Cost = 20000 Port = 0x8035
90	10.210995	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=717/52482, ttl=128 (no response found!)
91	10.848165	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/34:62:88:dc:0f:4c Cost = 0 Port = 0x8035
92	11.585909	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=718/52738, ttl=128 (no response found!)
93	11.839959	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. Root = 32768/0/34:62:88:dc:0f:4c Cost = 0 Port = 0x8035
94	12.795710	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 32768/0/00:10:a1:94:77:01 Cost = 20000 Port = 0x8035
95	13.085795	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=719/52994, ttl=128 (reply in 96)
96	13.086456	192.168.1.50	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=719/52994, ttl=64 (request in 95)

Figura 11 – Estratto da cattura

2.1.4.6 Test 6

Come ultimo test per il protocollo RSTP, è stata utilizzata una topologia leggermente diversa da quella dei casi precedenti:

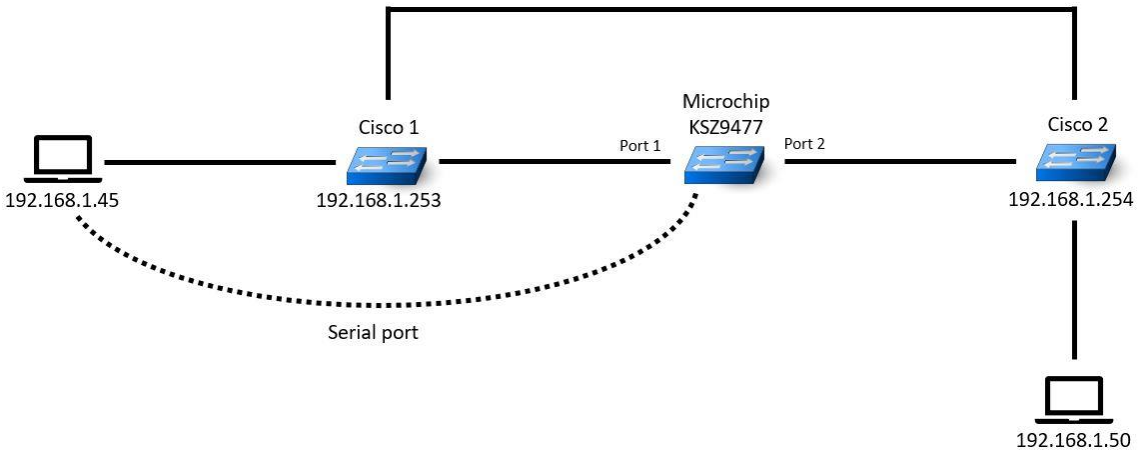


Figura 12 – Topologia di rete

in cui il secondo PC è stato connesso allo switch Cisco, anziché al Microchip. Anche in questo test è stato effettuato del traffico *ping* tra PC1 (192.168.1.45) e PC2 (192.168.1.50), disconnettendo la porta di root del Microchip (Port 1) dopo qualche secondo. Dalla cattura effettuata, il traffico torna ad essere trasmesso regolarmente dopo circa 4 secondi, a causa del valore di attesa del *ping reply* che è stato riportato al valore di default. Il tempo è, dunque, simile a quello misurato nei casi precedenti. Per completezza, si riporta l'estratto della cattura eseguita.

57	11.346788	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=566/13826, ttl=128 (no response found!)
58	13.000059	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 40000 Port = 0x8035
59	13.275382	Dell_f1:73:11	AsustekC_b4:0c:a2	ARP	60 Who has 192.168.1.45? Tell 192.168.1.50
60	13.275401	AsustekC_b4:0c:a2	Dell_f1:73:11	ARP	42 192.168.1.45 is at 08:62:66:b4:0c:a2
61	14.999746	Cisco_dc:0f:51	Spanning-tree-(for-bridges)_00	STP	60 RST. TC + Root = 28672/0/34:62:88:dc:0c:b8 Cost = 40000 Port = 0x8035
62	15.129522	0.0.0.0	255.255.255.255	DHCP	338 DHCP Discover - Transaction ID 0x68a0144a
63	16.080803	192.168.1.45	192.168.1.50	ICMP	74 Echo (ping) request id=0x0001, seq=567/14082, ttl=128 (reply in 64)
64	16.081326	192.168.1.50	192.168.1.45	ICMP	74 Echo (ping) reply id=0x0001, seq=567/14082, ttl=64 (request in 63)

Figura 13 – Estratto da cattura

2.2 Quality of Service (QoS)

Dopo aver osservato il comportamento e il funzionamento del Rapid Spanning Tree Protocol (RSTP), ed averne testato le caratteristiche principali, l'analisi del Microchip KSZ9477 si è focalizzata sulla Quality of Service (QoS) e sulle differenti modalità con cui esso permette di realizzarla. Il lavoro si è incentrato principalmente sulla verifica dell'effettivo supporto del Microchip alla Quality of Service attraverso test specifici con passaggio di traffico nella rete locale appositamente creata.

Sono state analizzate 3 differenti tipologie di QoS implementate sul Microchip KSZ9477:

1) Port-based Priority, ovvero priorità basata sulla porta. Permette di impostare un diverso livello di priorità per ogni singola porta dello switch, permettendo, in questo modo, di dare priorità al traffico da/per un certo dispositivo collegato ad una specifica porta del Microchip;

2) DiffServ Priority, meccanismo di QoS di livello 3, ma comunque disponibile per il Microchip KSZ9477. Permette di dare priorità ai diversi pacchetti in ingresso al dispositivo in base al valore del parametro DSCP (Differentiated Services Code Point) del pacchetto IP;

3) Rate limiting, ovvero la capacità di limitare la banda di trasferimento sia in ingresso che in uscita ad una data porta del dispositivo;

Per generare del traffico nella rete locale, per i test effettuati, è stato utilizzato iperf[3], un tool per la misura della banda massima nelle reti IP. Questo tool permette di impostare diversi parametri relativi al tempo, buffer e protocolli (TCP, UDP e SCTP con IPv4 e IPv6). Ad ogni esecuzione mostra la banda effettiva, le eventuali perdite e altri parametri. Si tratta di un tool cross-platform, in cui il server può gestire contemporaneamente più connessioni, senza la necessità di chiudersi dopo la terminazione di un solo test. Inoltre,

è possibile specificare il tempo di esecuzione, visualizzare delle statistiche intermedie su banda e perdite, ad intervalli specificati, durante l'esecuzione.

Un altro tool che è stato utilizzato per i test è bittwist[4], un semplice e potente generatore di frame Ethernet, ideato per essere complementare a tcpdump (tool di packet sniffing). Questo tool permette di rigenerare del traffico catturato in precedenza e rimetterlo nella rete. Il traffico si genera a partire da una file di una cattura in formato .pcap. Bittwist permette di modificare alcuni parametri del file, come ad esempio indirizzi IP e porte sorgente e destinazione. Anche questo è un tool multipiattaforma, in cui è possibile specificare il rate di trasmissione e che è in grado di correggere il checksum degli header dei pacchetti dopo le eventuali modifiche effettuate.

Seguono le analisi delle caratteristiche delle differenti tipologie di QoS, con le relative prove eseguite.

2.2.1 Port-based priority

Per questa tipologia di QoS, si è voluto testare se, dopo aver impostato una priorità elevata su una o più porte del Microchip, il traffico passante per quelle porte avesse realmente maggiore priorità sul traffico generico passante per le altre porte con priorità bassa. I test si sono basati sullo scambio di pacchetti tra due PC con il Microchip a fare da ponte, con l'inserimento di un secondo switch D-Link DES-1008D a 100 Mbps, per limitare la banda massima disponibile e quindi saturare in questo modo la rete, in caso contrario la banda sarebbe stata più elevata, e probabilmente non si sarebbe riusciti ad evidenziare la funzionalità *port-based priority*.

Pertanto, si è proceduto, in prima istanza, ad attivare la port-based priority sulle porte di interesse accedendo all'interfaccia CLI del Microchip, in particolare sovrascrivendo la variabile x_port_prio, con x numero della porta, impostando un valore compreso tra 1 e 7, con priorità crescente al crescere del valore.

La prima operazione effettuata è stata quella di misurare la banda disponibile con iperf tra PC1 e PC2, collegando in serie PC1, D-Link, Microchip e PC2, che, come previsto, è stata inferiore a 100 Mbps (come specificato sopra, a causa dello switch D-Link che ha svolto la funzione di limitatore), nel caso specifico 94.9 Mbps.

```
PS C:\Users\Alberto\Google Drive\DOCUMENTI POLITO - INGEGNERIA INFORMATICA (LM-32)\Tesi\iperf> .\iperf.exe -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.45 port 5001 connected with 192.168.1.50 port 51719
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-20.0 sec   227 MBytes  94.9 Mbits/sec
```

Figura 14 – Screenshot da terminale PowerShell

In seguito, sono stati eseguiti i test veri e propri.

2.2.1.1 Test 1

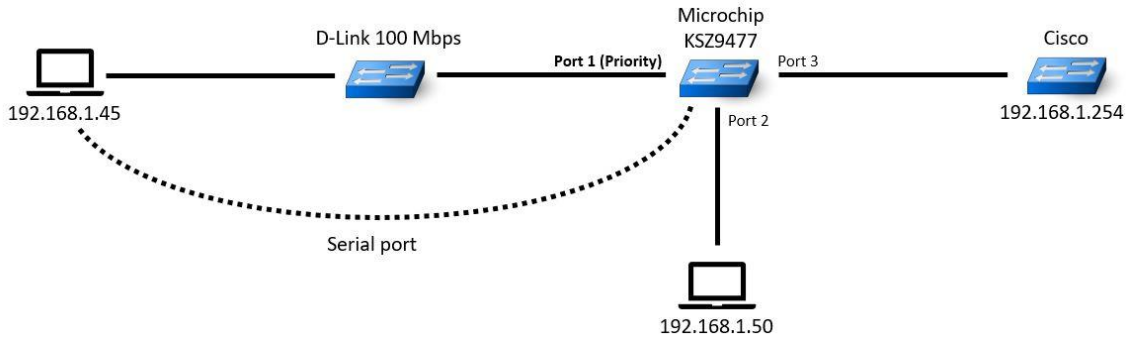


Figura 15 – Topologia di rete

Per questo primo test, con la topologia in figura, sono stati generati due differenti tipi di traffico:

1) traffico *ping* con pacchetti da 64K dal PC1 (indirizzo IP 192.168.1.45) al Microchip (192.168.1.201). Questo traffico è passante ESCLUSIVAMENTE per la porta 1 del Microchip, a priorità più elevata delle altre;

2) traffico ad 80 Mbps generato con iperf da PC1 (192.168.1.45) a PC2 (192.168.1.50) e viceversa, passante per la Port 2 a priorità più bassa e per quella a priorità alta.

Inizialmente è stato generato del traffico del primo tipo e, dopo qualche instante, è stato generato il secondo. Il traffico è stato catturato sul PC1 per verificare il comportamento e l'ordine di arrivo dei pacchetti. Il *ping reply* (diviso in più pacchetti da 1514 B), proveniente dalla porta 1 a priorità elevata, è stato ricevuto prima dei pacchetti del traffico proveniente dalla porta 2 (a bassa priorità), nonostante quest'ultimo passi comunque per la porta 1.

192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=51800, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=53280, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=54760, ID=4266) [Reassembled in #6437]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56933 [ACK] Seq=1 Ack=13914097 Win=170496 Len=0
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=56240, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=57720, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=59200, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=60680, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=62160, ID=4266) [Reassembled in #6437]
192.168.1.201	192.168.1.45	ICMP	402	Echo (ping) reply id=0x0001, seq=2270/56840, ttl=64 (request in 6357)

Figura 16 – Estratto da cattura

2.2.1.2 Test 2

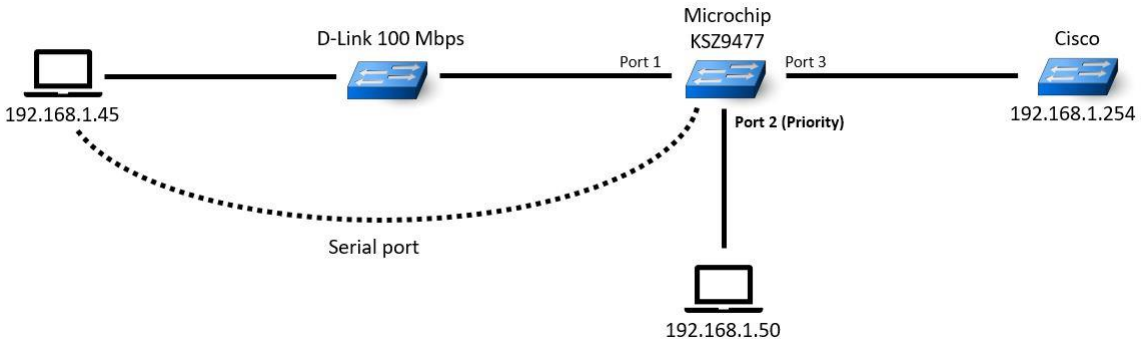


Figura 17 – Topologia di rete

Per questo secondo test, con la topologia in figura, sono stati generati due differenti tipi di traffico:

1) traffico ad 80 Mbps generato con iperf da PC1 (192.168.1.45) a PC2 (192.168.1.50) e viceversa, passante per una porta a priorità alta (Port 2).

2) traffico *ping* con pacchetti da 64K dal PC1 (indirizzo IP 192.168.1.45) al Microchip (192.168.1.201). Questo traffico è passante per la porta 1 del Microchip, a priorità bassa;

Inizialmente è stato generato del traffico del primo tipo e, dopo qualche instante, è stato generato il secondo. Il traffico è stato catturato sul PC1 per verificare il comportamento e l'ordine di arrivo dei pacchetti. Il *ping reply* (diviso in più pacchetti da 1514 B), proveniente dalla porta 1 a priorità bassa, non sono più ricevuti prima e tutti insieme, ma in alternanza ai pacchetti dell'altro tipo, prioritari perché passanti per la porta 2.

192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=50320, ID=635c) [Reassembled in #8671]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56008 [ACK] Seq=1 Ack=22085656 Win=233600 Len=0
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=51800, ID=635c) [Reassembled in #8671]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=53280, ID=635c) [Reassembled in #8671]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56008 [ACK] Seq=1 Ack=22088576 Win=233600 Len=0
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=54760, ID=635c) [Reassembled in #8671]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=56240, ID=635c) [Reassembled in #8671]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56008 [ACK] Seq=1 Ack=22091496 Win=233600 Len=0
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=57720, ID=635c) [Reassembled in #8671]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=59200, ID=635c) [Reassembled in #8671]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56008 [ACK] Seq=1 Ack=22094416 Win=233600 Len=0
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=60680, ID=635c) [Reassembled in #8671]
192.168.1.201	192.168.1.45	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=62160, ID=635c) [Reassembled in #8671]
192.168.1.50	192.168.1.45	TCP	60	5001 → 56008 [ACK] Seq=1 Ack=22097336 Win=233600 Len=0
192.168.1.201	192.168.1.45	ICMP	402	Echo (ping) reply id=0x0001, seq=2442/35337, ttl=64 (request in 8552)

Figura 18 – Estratto da cattura

2.2.1.3 Test 3

Per questo test, con la topologia precedente, si sono generati due differenti tipi di traffico:

1) traffico ad 80 Mbps generato con iperf da PC1 (192.168.1.45) a PC2 (192.168.1.50) e viceversa, passante per una porta a priorità alta (Port 2).

2) traffico *ping* con pacchetti da 64K dal PC1 (indirizzo IP 192.168.1.45) allo switch Cisco (192.168.1.254). Questo traffico è passante per le porte 1 e 3 del Microchip, a priorità bassa;

Inizialmente è stato generato del traffico del primo tipo e, dopo qualche instante, è stato generato il secondo. Il traffico è stato catturato sul PC1 per verificare il comportamento e l'ordine di arrivo dei pacchetti. I risultati sono analoghi a quelli del test precedente, ovvero i pacchetti del *ping reply* (più pacchetti da 1514 B), provenienti dalla porta 1 a priorità bassa, non sono più ricevuti prima e tutti insieme, ma in alternanza ai pacchetti dell'altro tipo, prioritari perché passanti per la porta 2.

2.2.2 Rate limiting

Il Microchip permette di limitare il rate su una specifica porta, sia in ingresso che in uscita, modificando delle variabili. Il Microchip dispone di differenti code di trasmissione/ricezione, ed è possibile specificare il rate per ogni singola coda di ogni porta. Per il rate in ricezione, le variabili sono nport_rx_p0_rate, nport_rx_p1_rate, nport_rx_p2_rate e così via, fino a nport_rx_p7_rate. Per il rate in trasmissione le variabili sono nport_tx_q1_rate, nport_tx_q2_rate e nport_tx_q3_rate.

Nel caso in cui si modifichi solo una delle variabili relative ad una porta (sia il valore di ricezione che di trasmissione), il Microchip assume quello come valore, altrimenti, secondo la documentazione, ne prende uno a caso.

Per verificare il funzionamento del rate limiting, è stata utilizzata la seguente topologia:

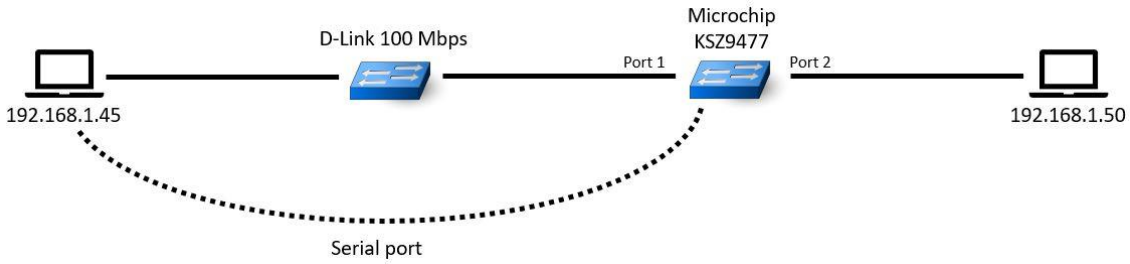


Figura 19 – Topologia di rete

ed è stata misurata la banda disponibile tra PC1 (192.168.1.45) e PC2 (192.168.1.50) con iperf sul PC1. Inizialmente è stata effettuata la misurazione lasciando le variabili ai valori di default. Si è misurata una banda di circa 100 Mbps. Successivamente si è iniziato col modificare i valori in vari modi e a misurare, ogni volta, la banda disponibile.

2.2.2.1 Test sulla banda

Di seguito i test con relativa misurazione della banda disponibile:

1) È stato limitato il rate in ingresso alla porta 1 a 10 Kbps, mantenendo inalterati il rate in uscita e i rate in ingresso/uscita della porta 2 e in seguito è stato generato il traffico con iperf. Si è rilevato un abbassamento della banda disponibile a **2.26 Mbps**, valore superiore al limite in ingresso a 10 Kbps, dovuto probabilmente all'elevata quantità di dati generati dal tool iperf, che produce pacchetti fino a saturare la rete.

```
[  3] local 192.168.1.45 port 58722 connected with 192.168.1.50 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-20.4 sec  5.50 MBytes  2.26 Mbits/sec
```

Figura 20 – Screenshot da terminale PowerShell

2) È stato limitato il rate in ingresso e in uscita della porta 1 a 10 Kbps, mantenendo inalterati i rate in ingresso/uscita della porta 2. Non dovrebbero esserci grossi cambiamenti sulla banda misurata, poiché non si sta inviando traffico in uscita dalla porta 1, a meno di pacchetti di ACK che incidono poco sul totale dei byte trasmessi. Effettivamente si è registrata una banda di **2.17 Mbps**, di poco inferiore alla precedente.

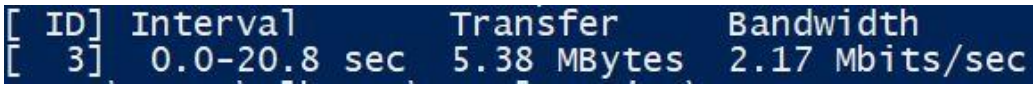


Figura 21 – Screenshot da terminale PowerShell

3) È stato limitato il rate in ingresso della porta 1 e in uscita della porta 2 a 10 Kbps. Con questi settaggi, si ottiene una banda di **7.39 Mbps** superiore a quella misurata in precedenza, probabilmente a causa delle impostazioni interne al Microchip che utilizza diverse code per la trasmissione del traffico, una sola delle quali è limitata in banda.

Figura 22 – Screenshot da terminale PowerShell

4) Sono stati limitati tutti i rate in ingresso/uscita delle porte 1 e 2 a 10 Kbps. Si è registrata una banda di **7.42 Mbps** simile a quella misurata nel precedente test.

Figura 23 – Screenshot da terminale PowerShell

5) È stato limitato solo il rate in uscita dalla porta 2 a 10 Kbps. Sono stati reimpostati ai valori di default gli altri rate delle porte 1 e 2 (ovvero senza limite). Si è misurata una banda leggermente superiore, ovvero **9.49 Mbps**.

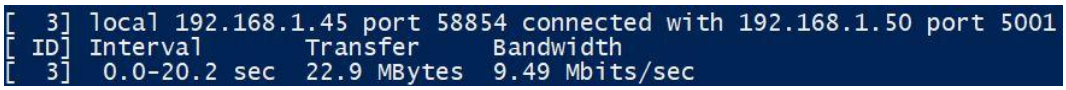


Figura 24 – Screenshot da terminale PowerShell

6) Sono stati limitati i rate in trasmissione di tutte le code della porta 2 a 10 Kbps. Come ci si aspettava, non si sono avute variazioni nella banda, che è risultata essere identica a quella del precedente test.

7) Come ultima prova, è stata invertita la direzione del traffico, ovvero da PC2 (192.168.1.50) a PC1 (192.168.1.45), mantenendo come unica limitazione il rate in trasmissione dalla porta 2 a 10000 bps. Non essendoci limiti al rate in ingresso alla porta 2 e al rate in uscita dalla porta 1, la banda misurata è effettivamente quella massima disponibile, ovvero circa 100 Mbps (**94.9 Mbps**).

Figura 25 – Screenshot da terminale PowerShell

2.2.3 Differentiated Service (DiffServ)

La terza, ed ultima, tipologia di QoS analizzata è una tecnologia di livello 3, basata sull'inserimento del parametro DSCP nei pacchetti IP per differenziare il traffico. I valori possibili da assegnare al parametro (i corrispondenti valori di Type of Service), sono i seguenti:

ToS Value	ToS String Format
0x20	CS1 - Priority
0x28	CS1 - Priority
0x30	CS1 - Priority
0x38	CS1 - Priority
0x40	CS2 - Immediate
0x48	CS2 - Immediate
0x50	CS2 - Immediate
0x58	CS2 - Immediate
0x60	CS3 - Flash
0x68	CS3 - Flash
0x70	CS3 - Flash
0x78	CS3 - Flash
0x80	CS4 -FlashOverride
0x88	CS4 -FlashOverride
0x90	CS4 -FlashOverride
0x98	CS4 -FlashOverride
0XA0	CS5 - Critical
0XB8	CS5 - Critical
0XC0	CS6 – Internetwork
0XE0	CS7 – Network control

Tabella 1 – Valori Type of Service (ToS values)

Per questo scopo si è fatto uso del tool iperf per generare del traffico con uno specifico valore DSCP, costruendo questa topologia:

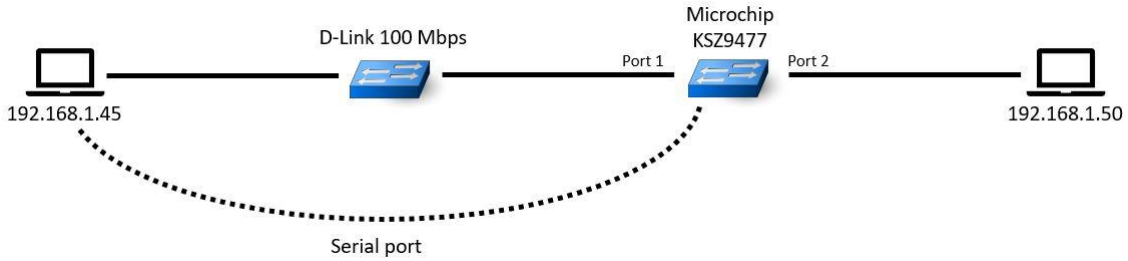


Figura 26 – Topologia di rete

È stato generato traffico da PC2 (192.168.1.50) a PC1 (192.168.1.45). Seguono i vari test effettuati, con relative schermate del terminale su cui è stato eseguito iperf.

2.2.3.1 Test 1

Per il primo test è stato **attivato** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0xB8 (ToS 5 – Critical) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x04 (ToS 0 – Routine) sulla stessa direzione. Da ciò che viene mostrato dal terminale, risulta una banda di **77.5 Mbps** per il traffico a priorità più alta e una banda di **18.3 Mbps** per quello a priorità più bassa.

```
[  4]   0.0- 5.3 sec  11.5 MBytes  18.3 Mbits/sec
[  5]   0.0- 5.5 sec  50.9 MBytes  77.5 Mbits/sec
[SUM]   0.0- 5.5 sec  62.4 MBytes  95.0 Mbits/sec
```

Figura 27 – Cattura da terminale

2.2.3.2 Test 2

Per il secondo test è stato mantenuto **attivo** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0x40 (ToS 2 – Immediate) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x04 (ToS 0 – Routine) sulla stessa direzione. È stata ottenuta una banda di **73.3 Mbps** per il traffico a priorità più alta e una banda di **22.0 Mbps** per quello a priorità più bassa.

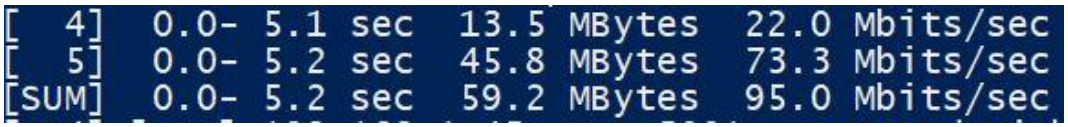


Figura 28 – Cattura da terminale

2.2.3.3 Test 3

Per il terzo test è stato mantenuto **attivo** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0x60 (ToS 3 – Flash) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x04 (ToS 0 – Routine) sulla stessa direzione. È stata ottenuta una banda di **81.1 Mbps** per il traffico a priorità più alta e una banda di **13.8 Mbps** per quello a priorità più bassa.

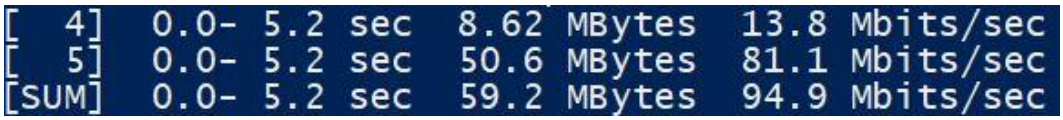


Figura 29 – Cattura da terminale

2.2.3.4 Test 4

Per il quarto test è stato mantenuto **attivo** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0x80 (ToS 4 – FlashOverride) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x04 (ToS 0 – Routine) sulla stessa direzione. È stata ottenuta una banda di **61.4 Mbps** per il traffico a priorità più alta e una banda di **34.5 Mbps** per quello a priorità più bassa.

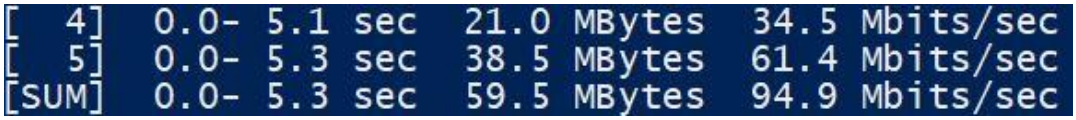


Figura 30 – Cattura da terminale

2.2.3.5 Test 5

Il quinto test è stato il primo in cui è stato **disattivato** il supporto a DiffServ sul Microchip, sulle porte 1 e 2, per osservare in che modo vengono trasmessi i pacchetti a differente priorità senza l'attivazione del protocollo. È stato generato del traffico con valore DSCP di 0x40 (ToS 2 – Immediate) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x20 (ToS 1 – Priority) sulla stessa direzione. È stata ottenuta una banda di **46.2 Mbps** per il traffico a priorità più alta e una banda di **52.2 Mbps** per quello a priorità più bassa, quindi una banda circa uguale, con leggero vantaggio a favore del traffico a bassa priorità, ovvero il comportamento opposto a quello mostrato nei test precedenti.

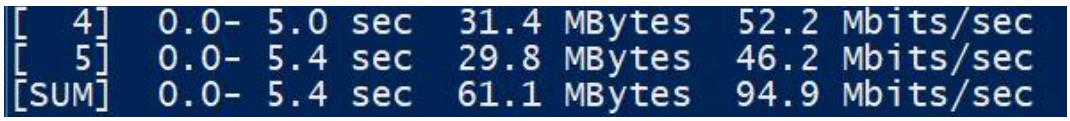


Figura 31 – Cattura da terminale

2.2.3.6 Test 6

Per il sesto test è stato mantenuto **disattivo** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0x60 (ToS 3 – Flash) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x20 (ToS 1 – Priority) sulla stessa direzione. È stata ottenuta una banda di **43.7 Mbps** per il traffico a priorità più alta e una banda di **54.2 Mbps** per quello a priorità più bassa. Anche in questo caso è stato rilevato un leggero vantaggio a favore del traffico a bassa priorità.

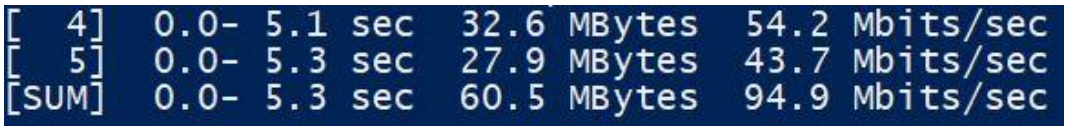


Figura 32 – Cattura da terminale

2.2.3.7 Test 7

Per quest'ultimo test è stato mantenuto **disattivo** il supporto a DiffServ sul Microchip, sulle porte 1 e 2. È stato generato del traffico con valore DSCP di 0x60 (ToS 4 – FlashOverride) da PC2 a PC1 e contemporaneamente traffico con DSCP 0x20 (ToS 1 – Priority) sulla stessa direzione. È stata ottenuta una banda di **41.9 Mbps** per il traffico a priorità più alta e una banda di **57.0 Mbps** per quello a priorità più bassa. Anche in questo caso è stato rilevato un leggero vantaggio a favore del traffico a bassa priorità.

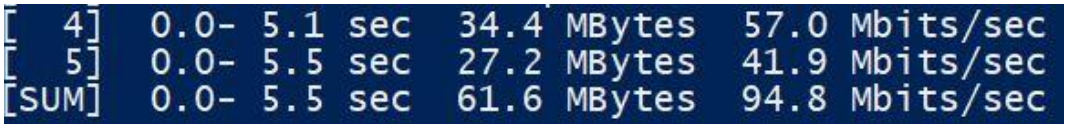


Figura 33 – Cattura da terminale

2.3 Specifiche e configurazione del Microchip KSZ9477

Dalla lettura del datasheet dello switch Microchip KSZ9477 e attraverso dei test sperimentali, sono state estratte utili informazioni per ciò che riguarda alcune delle principali peculiarità dello switch. Inoltre, sono stati sintetizzati in una tabella i principali comandi da terminale Linux per la configurazione, con i registri dello switch associati a ciascun comando. Per alcuni di questi comandi, lo switch non dispone di registri che, se modificati, permettono la configurazione con gli stessi effetti prodotti dal comando da terminale, per cui l'unica modalità di configurazione per le caratteristiche che non prevedono registri specifici è quella da terminale.

2.3.1 Tempo di boot

Il tempo necessario allo switch per regolarizzare la trasmissione del traffico è di circa 35 secondi. Il tempo è stato ottenuto sperimentalmente, generando del traffico TCP (attraverso il tool iperf) tra due host collegati tramite il Microchip. Durante la trasmissione del traffico, il Microchip è stato spento e riacceso, per simulare una caduta di tensione (con conseguente riavvio del dispositivo) o una vera e propria interruzione di corrente.

2.3.2 Tempo di reboot

Il tempo necessario al Microchip per un riavvio completo è di circa 10 secondi. Il tempo è stato misurato lanciando da terminale il comando reboot.

2.3.3 Carico effettivo supportato

La quantità di traffico che il Microchip è in grado di supportare, a regime e senza eventuali impostazioni di rate limiting, è leggermente inferiore a 1 Gbps. Il valore è quindi compatibile con i requisiti dell'applicazione in esame.

2.3.4 Latenza

Da una lettura del datasheet (par. 4.4.15 pag. 45), emerge che la latenza minima è proporzionale alla dimensione del pacchetto in oggetto. Esiste la possibilità di attivare la modalità “cut-through”, che permette una riduzione della latenza inoltrando il pacchetto sulla porta d’uscita senza attendere la completa ricezione dello stesso. Con questa modalità, l’inoltro inizia dopo aver ricevuto in ingresso i primi 64 byte del pacchetto, con una latenza di circa 900 ns per un traffico a 1 Gbps, indipendente dalla dimensione del pacchetto.

2.3.5 Consumi

Nel datasheet (par. 6.1 e 6.2, pag. 235) sono contenute informazioni sui valori massimi di tensione da fornire e quella in ingresso/uscita, oltre alla temperatura massima supportata. Sono disponibili anche i valori consigliati. Non ci sono informazioni specifiche sui consumi effettivi.

2.3.6 Principali configurazioni dello switch

Servendosi del terminale Linux, è possibile configurare la maggior parte delle impostazioni dello switch. Le stesse configurazioni possono essere eseguite sovrascrivendo i corrispondenti registri del Microchip. L’evaluation board fornisce un bootloader, U-Boot, che permette di settare diverse variabili d’ambiente per altre configurazioni per cui non sono previsti comandi o registri associati. Si accede ad U-Boot, premendo un tasto qualsiasi sul terminale durante la fase di boot. Alcune configurazioni dello switch Microchip KSZ9477, come ad esempio l’attivazione dello STP, possono essere effettuate, per questo specifico modello, esclusivamente accedendo ad U-Boot e modificando una variabile d’ambiente. Nella tabella che segue sono riassunte le principali impostazioni, con i relativi comandi Linux e gli eventuali registri associati.

Opzione	Descrizione	Valori	Comandi	Registri
U-BOOT				
printenv	Da terminale U-Boot, stampa a video tutte le variabili d'ambiente del Microchip.	///	=>printenv	//
multi_dev	Variabile d'ambiente visualizzabile e modificabile da terminale U-Boot, che permette di visualizzare, da terminale Linux, tutte le interfacce con le relative configurazioni	multi_dev = 0 mostra una sola interfaccia, eth0; multi_dev = 1 visualizza un'interfaccia per ogni porta dello switch (eth0, eth1, ecc…); multi_dev = 2 mostra una sola interfaccia, eth0, e una VLAN virtuale per ogni porta;	=>setenv multi_dev 1 =>saveenv =>boot oppure =>setenv multi_dev 0 =>saveenv =>boot	//
stp	Variabile d'ambiente visualizzabile e modificabile da terminale U-Boot, attraverso cui è possibile abilitare lo switch al supporto dello STP (RSTP).	0 (off) 1 (on)	=>setenv stp 1 =>saveenv =>boot oppure =>setenv stp 0 =>saveenv =>boot	//
ALTRE CONFIGURAZIONI DA TERMINALE LINUX				
interfaces	Visualizzazione e modifica delle impostazioni delle interfacce dello switch (indirizzo IP, indirizzo MAC, subnet mask, indirizzo broadcast ecc…).	///	per visualizzare: #cd etc/network #cat interfaces per modificare: #cd etc/network #vi interfaces	caratteristica propria della development board
mib	Variabile che mostra delle statistiche riguardo pacchetti totali ricevuti/trasmessi, numero di pacchetti broadcast, multicast, scartati e altre statistiche generali, per tutto il dispositivo.	///	per visualizzare: #cd sys/class/net/eth0/sw #cat mib per resettare: #cd sys/class/net/eth0/sw #echo 0 > mib	caratteristica propria della development board
N_mib	Variabile che mostra le stesse statistiche della precedente, ma per ogni singola porta N		per visualizzare: #cd sys/class/net/eth0/sw0 #cat 0_mib per resettare: #cd sys/class/net/eth0/sw0 #echo 0 > 0_mib	caratteristica propria della development board

info	Variabile che, se settata a 1, mostra un commento ai valori delle variabili che si visualizzano, per aiutare l'utente a comprendere il valore visualizzato	0 (off)
		1 (on)
			per visualizzare il valore:
			`#cd sys/class/net/eth0/sw`
			`#cat info`
			
			per modificare:
			`#cd sys/class/net/eth0/sw`
			`#echo 1 > info`
				caratteristica propria della development board

stp_br_on	Variabile che indica se il protocollo STP è attivo o meno	0 (off)
		1 (on)
			per visualizzare il valore:
			`#cd sys/class/net/eth0/sw`
			`#cat stp_br_on`
			
			per modificare:
			`#cd sys/class/net/eth0/sw`
			`#echo 1 > stp_br_on`
				nessun registro per questa opzione. Il protocollo va attivato settando la variabile d'ambiente in fase di boot.

stp_br_info	Variabile attraverso cui è possibile visualizzare le impostazioni correnti dello STP, per ogni porta dello switch.	///	`#cd sys/class/net/eth0` `#cat stp_br_info`	caratteristica propria della development board

N_diffserv	Variabile attraverso cui è possibile attivare o disattivare il protocollo DiffServ per la QoS. N è il numero della porta dello switch.	0 (off)
		1 (on)
			per visualizzare:
			`#cd sys/class/net/eth0/sw0`
			`#cat 0_diffserv`
			
			per modificare:
			`#cd sys/class/net/eth0/sw0`
			`#echo 1 > 0_diffserv`
				Port Priority Control Register
				
				0xN801
				bit 1
				(par. 5.2.8.2 p.189)

N_port_prio	Variabile attraverso cui è possibile settare un valore di priorità per una data porta N, utilizzata per la Port Priority QoS.	da 0 a 7, con priorità crescente	per visualizzare:
			`#cd sys/class/net/eth0/sw0`
			`#cat 0_port_prio`
			
			per modificare:
			`#cd sys/class/net/eth0/sw0`
			`#echo 5 > 0_port_prio`
				Port Ingress MAC Control Register
				
				0xN802
				bit 0:2
				(par. 5.2.8.3 p.189)

N_rx_prio_rate	Variabile utilizzata per abilitare o disabilitare il rate limiting per una data porta N dello switch	0 (off)
		1 (on)
			per visualizzare:
			`#cd sys/class/net/eth0/sw0`
			`#cat 0_rx_prio_rate`
			
			per modificare:
			`#cd sys/class/net/eth0/sw0`
			`#echo 1 > 0_rx_prio_rate`
				Port Ingress Rate Limit Control Register
				
				0xN403
				(par. 5.2.5.3 p. 176)

N_rx_p0_rate	Variabile utilizzata per limitare la banda in ingresso alla porta N, per i pacchetti con priorità px. Il nome della variabile per ogni priorità varia solo nell'indice (p0, p1, ecc…)	da 0 (full rate) a 0x73, con differenti valori di rate (vedi doc Microchip).	per visualizzare:
			`#cd sys/class/net/eth0/sw0`
			`#cat 0_rx_p0_rate`
			
			per modificare:
			`#cd sys/class/net/eth0/sw0`
			`#echo 0x66 > 0_rx_p0_rate`
				Port Priority N Ingress Limit Control Register
				
				0xN410 - 0xN417
				bit 0:6
				(par. 5.2.5.4 – 5.2.5.11, p. 177-179)

N_tx_q0_rate	Variabile utilizzata per limitare la banda in uscita alla porta N. Esistono 4 code in trasmissione, che è possibile settare individualmente.	da 0 (full rate) a 0x73 (in esadecimale), con differenti valori di rate (vedi doc Microchip).	per visualizzare:
			`#cd sys/class/net/eth0/sw0`
			`#cat 0_tx_q0_rate`
			
			per modificare:
			`#cd sys/class/net/eth0/sw0`
			`#echo 0x66 > 0_tx_q0_rate`
				Port Queue N Egress Limit Control Register
				
				0xN420 - 0xN423
				bit 0:6
				(par. 5.2.5.12 – 5.2.5.15, p. 180-181)

Tabella 2 – Comandi e configurazioni dello switch Microchip KSZ9477

3. Sintetizzazione audio e intelligibilità

Dopo aver studiato le caratteristiche del Microchip KSZ9477, l'attenzione si è spostata su come poter rendere efficiente la comunicazione. In un ambiente rumoroso, così come è stato pensato il contesto di applicazione del sistema, le parole pronunciate potrebbero subire una forte distorsione, provocata, appunto, dal rumore ambientale. Di conseguenza, anche in assenza di disturbi o problemi nella rete locale, il destinatario potrebbe non comprendere ciò che è stato pronunciato dal mittente, rendendo inefficace il sistema stesso. Per questo motivo, si è pensato di utilizzare la voce sintetizzata da un computer al posto di quella umana. I messaggi vocali possono essere prodotti da un software text-to-speech e inviati nella rete locale, annullando gli effetti negativi sopra citati. Con questa modalità, in base alle esigenze, possono anche essere creati dei messaggi predefiniti salvati in locale per poi essere inviati al bisogno, senza la necessità di dover pronunciare più volte la stessa sequenza di parole in momenti diversi.

Un software text-to-speech è in grado di generare una voce simile a quella umana, a partire da un testo. Si può, dunque, utilizzare un sistema di questo tipo per generare i messaggi da inviare nel sistema di comunicazione. Allo stesso modo, a destinazione, può essere utilizzato un altro software dal comportamento opposto, ovvero uno speech-to-text, che prende ciò che è stato ricevuto e lo converte automaticamente in testo. Resta inteso che le due operazioni sono indipendenti, che possono essere utilizzate entrambe, solo una o nessuna. In questo modo si può anche avere un indice di intelligibilità, valutando il tasso di errore a destinazione e dare una valutazione al sistema.

Ciò che è stato studiato è la disponibilità dei suddetti software in modalità open source, da scaricare e utilizzare gratuitamente. Sono state anche analizzate delle versioni a pagamento, con i relativi vantaggi e svantaggi rispetto a quelle gratuite. In seguito, sono stati cercati in rete dei tools utili per calcolare un indice di intelligibilità audio in ricezione,

e valutare, quindi, la qualità della voce dopo che la stessa è passata attraverso il sistema in esame.

3.1 Sistemi text-to-speech

In rete è possibile trovare molti sistemi text-to-speech, molti dei quali disponibili gratuitamente e con un buon livello di funzionalità. Assume un'importanza fondamentale il fatto che la voce sintetizzata sia parecchio simile a quella umana e non eccessivamente "robotica". Per poter utilizzare una voce di questo tipo è chiaro che debba essere comprensibile a chi la ascolta, ancora di più nel caso di un ambiente rumoroso.

Molti dei tools disponibili online, dispongono di esempi che permettono di verificare la qualità della voce generata, in modo da dare un'idea all'utente di quello che sarà il prodotto finale della conversione da testo a voce. In genere, la qualità finale è molto buona, inoltre alcuni dei software permettono la scelta tra voce di sesso maschile e femminile, con buoni risultati in entrambi i casi. Per quasi tutti quelli analizzati, sono disponibili diverse lingue tra cui scegliere (ovviamente l'Inglese è la lingua sempre presente). Tra questi, sono stati selezionati quelli che risultano essere, allo stato attuale, i migliori per quanto riguarda la semplicità di utilizzo e l'intelligibilità della voce prodotta, e quelli che supportano la lingua italiana.

Segue una breve descrizione dei migliori tools presenti nella rete, e una tabella finale riassuntiva con le principali caratteristiche di interesse, tra le quali costo, lingue disponibili ed esempi online.

3.1.1 Microsoft Elsa Desktop

Questo primo sintetizzatore vocale è integrato nel sistema operativo Windows 10, e permette di convertire testo in voce in modo molto semplice. Il suo principale scopo, per

cui è stato inserito nel sistema operativo, è quello di fornire un supporto agli utenti con problemi di accessibilità. La voce, come è facilmente intuibile dal nome dato dagli sviluppatori, è femminile. Il risultato finale è ottimo, infatti la voce è molto simile a quella umana. Essendo integrato nel sistema operativo, non ha alcun costo. Supporta la lingua italiana, oltre a quella inglese. Esiste un altro sintetizzatore vocale, Microsoft Zira, che non supportando l'Italiano, non è stato approfondito.

3.1.2 eSpeak

Un buon tool text-to-speech è eSpeak, disponibile sia su sistemi Windows che Linux. Con l'interfaccia a riga di comando è possibile leggere da tastiera o da file (disponibile in entrambi i S.O.). Su Windows, invece, si dispone di un'interfaccia grafica semplice e intuitiva, attraverso cui è possibile scrivere il testo che si vuole convertire ed eventualmente riprodurlo con gli altoparlanti o salvare il risultato finale in un file audio in formato WAV. Integra il sintetizzatore Microsoft Elsa Desktop precedentemente descritto, oltre al proprio motore di sintesi.

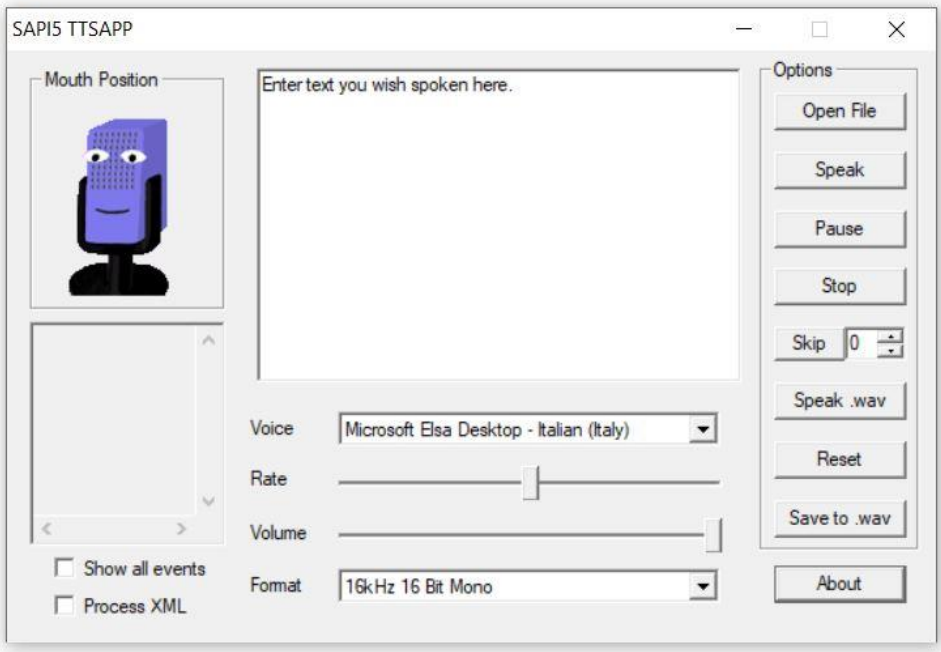


Figura 34 – eSpeak TTS App (Windows)

Quest'ultimo ha una qualità inferiore al precedente, infatti, sebbene la voce sia comprensibile, è certamente meno somigliante alla voce umana, evidenziando dei suoni robotici. Si consiglia quindi di utilizzare il sintetizzatore Microsoft e di sfruttare l'interfaccia grafica dell'applicazione. Supporta l'Inglese e l'Italiano, occupa pochissimo spazio in memoria ed è stato sviluppato in linguaggio C. Questo tool è disponibile anche per altri sistemi operativi, quali Android e Mac OSX. Per maggiori informazioni è possibile visitare il sito web degli sviluppatori [6].

3.1.3 iSpeech

Un altro sintetizzatore vocale è iSpeech, un tool utilizzabile online. L'esempio presente sul sito è molto chiaro, per utilizzarlo basta inserire il testo nella casella indicata, scegliere la lingua da utilizzare, e premere il tasto "Create". Una volta sintetizzato il testo, verrà riprodotto immediatamente, con la possibilità di riprodurlo altre volte premendo il pulsantino verde.

Figura 35 – iSpeech (online)

Si può ascoltare gratuitamente il risultato dell'elaborazione. Per scaricare il file audio, invece, è necessario acquistare dei pacchetti di parole dal prezzo variabile in base al numero. La voce sintetizzata è simile a quella umana, sono presenti molte lingue europee ed extraeuropee tra cui l'Italiano, inoltre è possibile scegliere, per la maggior parte delle

lingue, la versione maschile o femminile. Per maggiori informazioni è possibile visitare il sito web [7].

3.1.4 MaryTTS

MaryTTS è un altro tool online di sintesi vocale, open source, utilizzabile mediante un'interfaccia web essenziale o scaricando e installando il software. MaryTTS è stato scritto in Java, per cui è necessario che nel PC in cui deve essere utilizzato sia presente una versione di Java Runtime. Si è testata la versione online, che dispone di molte lingue diverse, anche qui con la possibilità di scelta tra voce maschile e femminile. Non richiede alcun pagamento sia per la riproduzione che per il download del file audio generato, scaricabile in diversi formati (tra cui WAV).

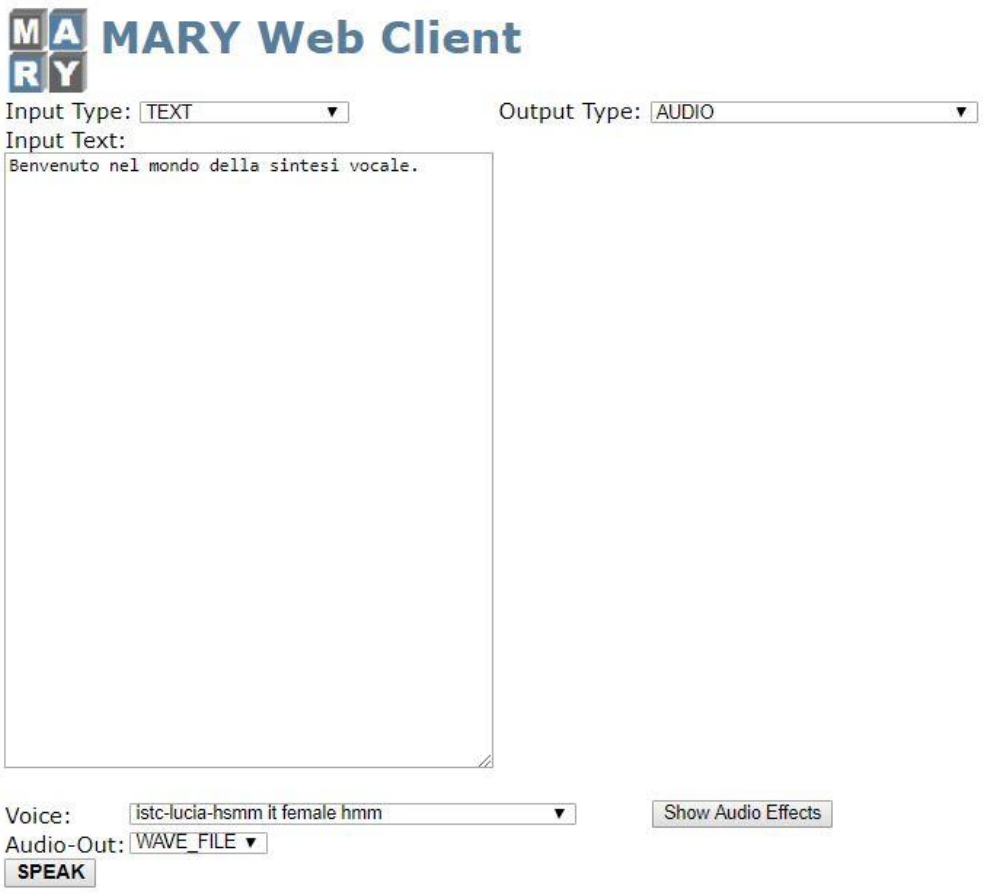


Figura 36 – MaryTTS (online)

La voce prodotta è di livello medio, abbastanza comprensibile. La sua semplicità di utilizzo e la sua disponibilità sia online che offline lo rendono un buon prodotto. Per maggiori informazioni è possibile visitare il sito web degli sviluppatori [8].

3.1.5 Google Translate

Il principale motore di ricerca fornisce un servizio di traduzione supportando tantissime lingue. Integrato in questo sistema, esiste la possibilità di far "leggere" nella lingua selezionata ciò che è scritto nella casella di testo, premendo il pulsante su cui è disegnato un megafono, svolgendo quindi proprio il lavoro di un text-to-speech. La qualità della voce sintetizzata è molto buona.

3.1.6 Altre soluzioni

Segue una tabella riassuntiva dei tool appena descritti, con le caratteristiche più significative e l'aggiunta di due ulteriori soluzioni, più complesse e non adatte al caso specifico di utilizzo.

URL	Esempi	API online o software	Costo	Lingue	Note
Microsoft Elsa Desktop	Nessun esempio	Integrato in Windows 10	Gratuito	Italiano	Non open source. Qualità della voce sintetizzata molto buona
eSpeak espeak.sourceforge.net	L'esempio online in Inglese è poco chiaro, lingua italiana ben riconoscibile con Microsoft Elsa Desktop	Software	Gratuito	Italiano Inglese altre	open-source, scritto in C, disponibile interfaccia a riga di comando per leggere da file o da tastiera.

iSpeech www.ispeech.org	Esempio online abbastanza chiaro	Online	Ascolto gratuito, download a pagamento	Italiano Inglese altre	fornisce una demo per speech-to-text
MaryTTS mary.dfki.de	Esempio online abbastanza chiaro	Software (+ demo online)	Gratuito	Italiano Inglese altre	open-source, scritto in Java
Google Translate translate.google.it	Nessun esempio, testabile direttamente con del testo	Online	Gratuito	Italiano Inglese altre	Non open source. Qualità della voce sintetizzata molto buona
Tacotron github.com/keithito/tacotron	Nessun esempio disponibile online	Git repository	Gratuito	Inglese	soluzione basata sul Machine Learning (TensorFlow) e Google Tacotron speech synthesis
DSpeech dimio.altervista.org/ita/#DSpeech	Esempio online di una conversazione tra diversi soggetti, abbastanza chiaro, voci leggermente robotiche	Software	Gratuito	Italiano Inglese altre	non si installa, possibilità di creare dialoghi interattivi, in grado di "doppiare" i film sincronizzando i sottotitoli (funziona con VLC e MediaPlayer)

Tabella 3 – Sistemi text-to-speech

3.2 Sistemi speech-to-text

A questo punto, dopo aver analizzato i principali software open source di sintesi vocale a partire dal testo, sono stati studiati quelli che svolgono il compito opposto, ovvero il riconoscimento vocale e la conseguente scrittura in testo. Esistono molti tools che svolgono un lavoro del genere, alcuni di essi sono integrati nei dispositivi stessi, basti pensare agli smartphone, in cui l'Assistente Google per Android e Siri per iOS svolgono proprio questo compito, sia online per effettuare ricerche sul browser o offline per inviare

messaggi o impostare promemoria. Nel caso studiato in questa tesi, un tool del genere può essere sfruttato a destinazione per convertire in testo un messaggio proveniente da un altro utente della rete locale, nel caso in cui sia necessaria una operazione di questo tipo.

Ciò che risulta essere importante è che il numero di parole convertito correttamente sia il maggiore possibile, o quantomeno vicino al 100%. La conversione risulta complicata dalla variabilità della voce umana, per questo motivo utilizzando della voce sintetizzata in ingresso al sistema di speech recognition, si può ridurre sensibilmente il numero degli errori.

Ciò che si nota sui principali tools di questo genere, è che si basano quasi tutti su soluzioni implementate da Google, adattate o con aggiunta di altre funzionalità e con diverse interfacce utente.

Segue una breve descrizione dei migliori tools presenti nella rete, e una tabella finale riassuntiva con le principali caratteristiche di interesse, tra le quali costo, lingue disponibili e la valutazione delle prestazioni con voce sintetizzata.

3.2.1 Google Cloud Speech-to-Text

Il sistema di speech recognition di Google è basato sul machine learning e sull'applicazione di modelli di reti neurali in una API semplice e intuitiva, per audio di breve e lunga durata [(9)]. Il sistema è, quindi, moderno, efficiente e in continuo aggiornamento. Permette di riconoscere automaticamente la lingua parlata e include tantissime altre funzionalità quali la resistenza al rumore, caratteristica significativa per il caso in esame in questa tesi.

Il servizio ha un costo, infatti è gratuito solo per i primi 60 minuti, in seguito il costo è di 0.006$ ogni 15 minuti. Sono possibili, inoltre, ulteriori costi nel caso in cui si utilizzino altri servizi di Google Cloud Platform. Questo è, certamente, uno dei migliori servizi disponibili allo stato attuale.

3.2.2 Google Translate

Il servizio di traduzione di Google include, oltre al text-to-speech visto nel paragrafo precedente, anche uno speech-to-text, accessibile cliccando sull'icona su cui è disegnato un microfono. Probabilmente è utilizzato a questo scopo il servizio Google Cloud Speech-to-Text precedentemente analizzato.

3.2.3 Dictation.io

Questo secondo tool, non open source, sfrutta il motore Google Speech Recognition, di conseguenza i risultati sono ottimi, sia con voce naturale che con voce sintetizzata. Il servizio è online, gratuito, supporta la lingua italiana, inglese e altre 50 circa. Si è testato facendo leggere a Google Translate un testo casuale e il tool ha riconosciuto tutte le parole pronunciate. Permette, inoltre, di dare dei comandi particolari, quali, ad esempio, apertura/chiusura di parentesi, nuova riga e vari caratteri speciali. L'interfaccia è lineare e comprensibile.

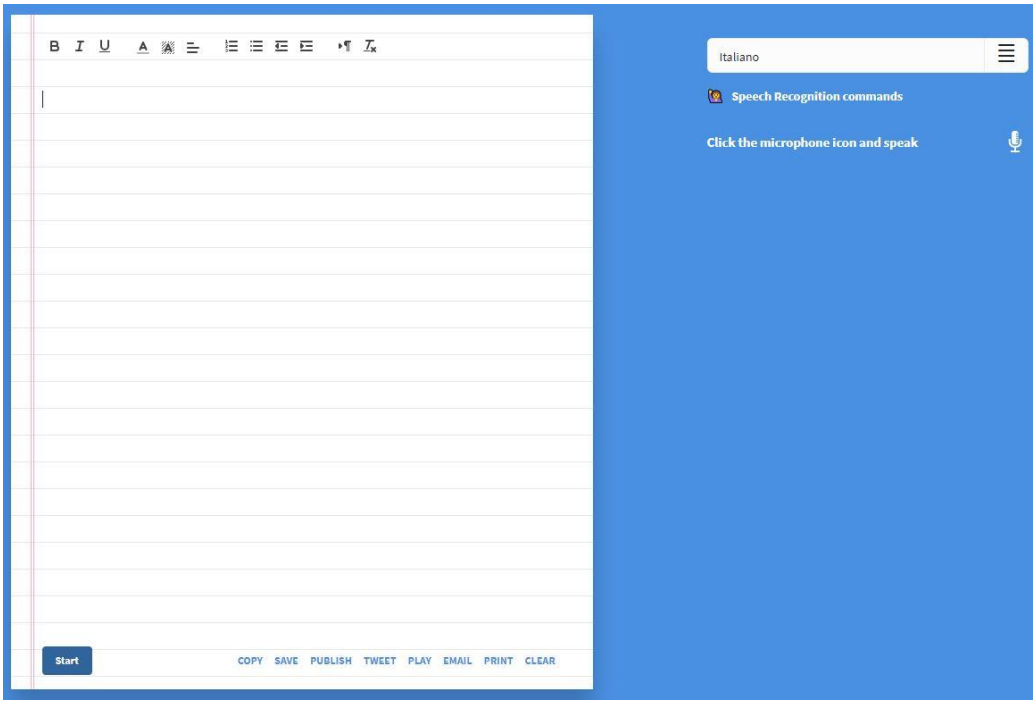


Figura 37 – Dictation.io

3.2.4 Mozilla DeepSpeech

Uscendo dal panorama dei servizi Google per lo speech recognition, esiste un progetto implementato dalla comunità Mozilla, chiamato DeepSpeech. Anche questa soluzione è basata su tecniche di machine learning, così come intuibile dal nome dato al progetto. Resta un legame con Google, dal momento che questo progetto utilizza la libreria TensorFlow sviluppata dal team Google Brain, che si occupa di intelligenza artificiale. In ogni caso, questa soluzione è distribuita con licenza open source e tutto il codice è disponibile gratuitamente su GitHub e gira sia su piattaforme Linux che Windows.

Il motore DeepSpeech permette, una volta installate tutte le dipendenze, di convertire in testo brevi campioni audio in formato WAV, attraverso una interfaccia a riga di comando. Si è testato il suo funzionamento con un campione audio di una voce sintetizzata con Google Translate lungo qualche secondo, su macchina virtuale Linux. Il risultato è stato soddisfacente, infatti in pochi secondi il motore è riuscito a convertire in testo correttamente ciò che veniva pronunciato nel campione sottomesso.

3.2.5 Altre soluzioni

Segue una tabella riassuntiva dei tool appena descritti, con le caratteristiche più significative e l'aggiunta di altre soluzioni, tra cui gli assistenti vocali per dispositivi mobili e alcune più complesse e non adatte al caso specifico di utilizzo.

Nome	URL	API online o software	Costo	Lingue	Funziona con testo sintetizzato	Note
Google Cloud speech-to-text	cloud.google.com/speech-to-text/	Online	Gratuito fino a 60 minuti di audio	100+	Sì	versione non controllabile, poiché servizio cloud.
Google Translate	translate.google.it	Online	Gratuito	Italiano Inglese altre	Sì	Buone prestazioni

Dictation.io	dictation.io	Online	Gratuito	Italiano Inglese 50+	Sì, testato con Google Translate	Non open-source, utilizza il motore Google Speech Recognition
Mozilla DeepSpeech	github.com/mozilla/DeepSpeech	Github	Gratuito	Inglese	Sì, testato con Google Translate	Buone prestazioni, testato su macchina virtuale Linux
Assistente Google		Integrato in dispositivi mobili Android	Gratuito	Italiano Inglese altre		Assistente vocale per dispositivi Android, continuamente aggiornato
Siri (iOS)		Integrato in dispositivi mobili iOS	Gratuito	Italiano Inglese altre		Assistente vocale per dispositivi iOS, continuamente aggiornato
Cortana (Windows)		Integrato in dispositivi Microsoft	Gratuito	Italiano Inglese altre		Assistente vocale per dispositivi Microsoft, continuamente aggiornato
Kaldi	kaldi-asr.org	Github	Gratuito	Inglese		Molto complesso, utilizzato dai ricercatori in speech recognition

Tabella 4 – Sistemi speech-to-text

3.3 Valutazione dell'intelligibilità audio

Una delle principali questioni che coinvolgono la progettazione del sistema di comunicazione è la valutazione del livello di comprensione dell'audio vocale che si ha dopo che il campione è passato attraverso il sistema stesso. Per effettuare una misurazione di questo tipo è necessario utilizzare un metro di valutazione standard, che tenga conto delle caratteristiche del segnale audio vocale e di come il sistema di comunicazione è in grado di mantenerle. Questo approccio è l'unico modo per avere una stima dell'intelligibilità della voce da utilizzare per valutare la qualità del sistema.

È stato preso in considerazione l'indice STI (Speech Transmission Index), nella sua variante STIPA (STI for Public Address Systems), che risulta essere una buona unità di misura per l'intelligibilità della voce.

3.3.1 STIPA

L'indice STI è stato introdotto per la prima volta da Houtgast e Steeneken in una pubblicazione del 1971[10], e sviluppato successivamente dagli stessi due autori lavorando in un team dell'Organizzazione olandese per la ricerca scientifica applicata (TNO). Questo indice misura alcune delle caratteristiche fisiche di un canale di comunicazione, come ad esempio una stanza, una linea telefonica o un attrezzatura acustica, ed esprime l'abilità del canale in oggetto di mantenere le specifiche della voce. Alla fine, l'indice sarà un numero compreso tra 0 e 1, dove più è alto il valore e migliore è l'intelligibilità della voce.

STI value	Quality according to IEC 60268-16	Intelligibility of syllables in %	Intelligibility of words in %	Intelligibility of sentences in %
0 – 0.3	bad	0 – 34	0 – 67	0 – 89
0.3 – 0.45	poor	34 – 48	67 – 78	89 – 92
0.45 – 0.6	fair	48 – 67	78 – 87	92 – 95
0.6 – 0.75	good	67 – 90	87 – 94	95 – 96
0.75 – 1	excellent	90 – 96	94 – 96	96 – 100

Tabella 5 - Valori di riferimento per STI

Negli anni 2000, è stato introdotto un nuovo indice, costruito ad hoc per la valutazione dell'intelligibilità della voce nei luoghi pubblici (ad esempio stazioni o aeroporti) chiamato STIPA (STI for Public Address Systems). La differenza con il suo predecessore è che invece delle 14 frequenze di modulazione applicate in successione alle 7 bande di ottave, vengono applicate solo 2 frequenze di modulazione simultaneamente ad ognuna delle 7 bande. È stato stimato che il tempo necessario al calcolo dell'indice sia di circa 15-20 secondi[11].

STIPA utilizza un segnale di test che, sebbene all'ascolto appaia come un suono rumoroso per nulla simile alla voce umana, il suo contenuto in termini di frequenze lo è. Per il calcolo dello STIPA è necessario utilizzare esclusivamente il segnale di test fornito.

3.3.2 Misuratori indici di intelligibilità

In commercio esistono molteplici misuratori per l'indice STI o STIPA. Ciò che viene venduto è un dispositivo in grado di calcolare l'indice acquisendo in input il segnale di test prodotto all'interno del contesto in cui si vuole effettuare la misurazione (alcuni rivenditori propongono anche l'acquisto di un generatore del segnale di test). Spesso, i misuratori sono venduti con un software ad hoc che permette di aggiungere o eliminare rumore e di effettuare vari settaggi che possono risultare convenienti in base alle necessità dell'utilizzatore. In media, il costo da sostenere per l'acquisto di uno STI/STIPA meter di discreta qualità supera i 1000 $, un prezzo non indifferente.

Una soluzione alternativa potrebbe essere quella di utilizzare il microfono del proprio smartphone come misuratore. I dispositivi moderni dispongono, ormai, di microfoni di buona qualità, certamente non pensati per questo scopo ma pur sempre una valida alternativa a basso costo. Per dispositivi mobili con sistema operativo iOS, è disponibile sull'App Store un'applicazione, distribuita da Studio Six Digital, che svolge proprio questo compito, dal nome STIPA [12]. Si tratta di un'app dal costo contenuto (50 $ per la versione base, 150 $ per l'upgrade a STIPA Pro) che sfrutta, appunto, il microfono dello smartphone come STIPA meter, implementa l'algoritmo di calcolo dell'indice e dopo aver ricevuto in input il segnale di test di cui si vuole conoscere l'intelligibilità, mostra il corrispondente valore sullo schermo. Non esiste, al momento, un'alternativa simile per dispositivi Android.

4. Data Distribution Service (DDS)

Dopo aver completato lo studio sulle varie possibilità di sintetizzazione della voce e aver visto una panoramica di software e tools open source o a pagamento utili a questo scopo, la fase successiva si è incentrata sulla ricerca di una metodologia di trasferimento dei messaggi da un utente ad un altro che fosse veloce e al tempo stesso performante.

Una buona soluzione è certamente un sistema basato sullo standard DDS (Data Distribution Service for Real-Time Systems), ovvero uno standard della Object Management Group (OMG) che consiste in un middleware basato sul paradigma publish/subscriber per il trasferimento di dati in real-time. Questo standard permette alle applicazioni di condividere informazioni semplicemente leggendo e scrivendo oggetti dato indirizzati attraverso un application-defined name, il topic [13].

4.1 Componenti e funzionamento del middleware

La comunicazione, in DDS, avviene in uno spazio chiamato Global Data Space (GDS) che rappresenta il principale componente di DDS, suddiviso in più domini distinti da un nome che li caratterizza. In ogni dominio sono presenti i topic, che devono necessariamente avere nomi diversi (possono avere nomi uguali solo se presenti in domini diversi). Ogni dominio è indipendente da un altro. DDS è di tipo data-centric, che assicura il fatto che i messaggi scambiati contengano tutte le informazioni necessarie alle applicazioni per comprendere a pieno ciò che si sta ricevendo. Al tempo stesso, la gestione del codice per inviare i messaggi non è più lasciata al programmatore, così come avviene per i middleware message-centric, ma è completamente gestita da DDS, il che rappresenta un ulteriore punto a vantaggio di questo standard. Si accede al GDS attraverso un componente chiamato DomainParticipant, che ha la funzione di factory per la creazione dei topic e dei publisher e subscriber.

Il topic è quello spazio del dominio su cui i publisher e i subscriber rispettivamente scrivono e leggono i dati, sono identificati da un nome univoco (nel dominio) e da un tipo che dà un'indicazione su ciò che il dato rappresenta. È necessario specificare i tipi dei topic preliminarmente, utilizzando il linguaggio IDL (Interface Description Language). Ogni volta che si modifica il file di definizione dei tipi dei topic (che per quanto detto avrà estensione .idl), è necessario ricompilare la soluzione per rendere effettive le modifiche. (14)

Il publisher è colui che pubblica i dati, o meglio, li distribuisce sul dominio. La prima operazione che viene effettuata è creare uno o più DataWriter (da parte del publisher) associati ad un topic specifico. Ogni publisher avrà, quindi, associato un DataWriter per ogni topic. (14)

Il subscriber è colui che riceve i dati che sono stati pubblicati sul topic a cui è interessato. Anche in questo caso, vengono creati uno o più DataReader (da parte del subscriber) per ogni topic. Il subscriber avrà il compito di gestire i dati che sono ricevuti da tutti i DataReader ad esso associati. (14)

4.2 OpenDDS

Il sistema DDS utilizzato nel caso specifico è la distribuzione OpenDDS, in particolare l'ultima versione attualmente disponibile, la 3.13. OpenDDS è un'implementazione dello standard DDS open source in linguaggio C++, utilizzabile anche dalle applicazioni Java per mezzo della Java Native Interface. Questa implementazione è stata sviluppata da Object Computing. È possibile utilizzarla in molte architetture diverse e con differenti sistemi operativi come, ad esempio, su macchine Linux, Windows, su sistemi embedded, Raspberry Pi o ARM (15). OpenDDS fa uso di alcune soluzioni open source, di seguito elencate, utili alla sua compilazione ed esecuzione. Il fatto che siano anch'esse open source è un altro vantaggio considerevole per questa soluzione, a garanzia della sua compatibilità con sistemi di vario tipo.

OpenDDS utilizza ACE (Adaptive Communication Environment), un framework open source per lo sviluppo, che consiste in componenti e pattern per sistemi embedded e real-time distribuiti. Fornisce astrazioni per le principali primitive di sincronizzazione e per la comune programmazione di rete (principalmente socket e thread). Il framework è scritto in linguaggio C++.

Insieme ad ACE, OpenDDS utilizza TAO (la cui sigla completa è ACE ORB TAO), un'implementazione in C++ dello standard CORBA basato, appunto, su ACE. È indicato per chi è interessato ad avere alte prestazioni nelle proprie applicazioni, come nel caso in esame in questa tesi. Lo standard CORBA (Common Object Request Broker Architecture) è stato definito da Object Management Group, e ha il compito di gestire e facilitare la comunicazione fra entità in modo slegato dal linguaggio di programmazione con cui esse sono sviluppate. È la soluzione utilizzata per far comunicare publisher e subscriber, tramite un oggetto con la funzione di broker che, appunto, mette in relazione i componenti in base alle specifiche del DDS.

L'ambiente OpenDDS è stato compilato sulla macchina virtuale Ubuntu creata ad hoc come ambiente di sviluppo, e le istruzioni dettagliate per la compilazione in ambiente Linux sono riportate nell'Appendice A. Successivamente si è effettuata la cross-compilazione anche sulla evaluation board Seco (i.MX).

4.2.1 Esecuzione dell'ambiente su i.MX

La fase di studio successiva è stata quella di cross-compilare l'ambiente OpenDDS per essere eseguito sulla evaluation board Seco CQ7-A42, il cui processore interno è un ARM Cortex-A9. Al fine di consentire una corretta compilazione per la giusta architettura, è stato utilizzato l'SDK Poky della Yocto Project, nella sua versione specifica per la scheda.

La procedura di compilazione dell'ambiente sulla evaluation board Seco CQ7-A42, ha richiesto particolare attenzione per quanto riguarda i vari settaggi delle variabili

d’ambiente necessarie, nonché dei parametri da passare ai comandi da terminale Linux sulla macchina virtuale. Fondamentale per una corretta compilazione senza errori è utilizzare i compilatori giusti per ogni singolo sorgente. La soluzione OpenDDS, come già specificato, dispone di altre distribuzioni open source esterne, come ACE+TAO, che non possono essere compilate utilizzando Poky, ma necessitano del compilatore g++ di default, ovvero quello preinstallato sul sistema operativo Ubuntu. Forzando ad utilizzare Poky per tutta la soluzione OpenDDS si va incontro, ovviamente, a diversi errori che impediscono la corretta conclusione del processo di compilazione. Per le istruzioni complete sulla compilazione di OpenDDS per architettura ARM, si rimanda all’Appendice B, mentre per quanto riguarda l’esecuzione di un esempio dell’ambiente, nel caso specifico quello dal nome IntroductionToOpenDDS, si rimanda all’Appendice C.

5. Conclusioni

L'idea alla base di questa tesi è stata quella di implementare un sistema di comunicazione a bassa latenza, in grado di trasmettere in modo efficiente all'interno di un veicolo in condizioni ambientali particolarmente complesse, come, ad esempio, un ambiente rumoroso. Sono stati utilizzati strumenti e attrezzature dal costo contenuto, oltre a del software open source (e, quindi, completamente gratuito) per la sintetizzazione audio da testo a voce e viceversa. Infine, l'ambiente OpenDDS, anch'esso open source, è risultato molto efficace e rispondente ai bisogni di questa tesi, grazie alla versatilità e alla facilità con cui è possibile impostare e modificare i messaggi da scambiare e far comunicare efficientemente mittente e destinatario. Si è riusciti, previa attenta analisi dei sistemi utilizzati, a far interagire l'ambiente OpenDDS con le due diverse architetture in esame, ovvero ARM e sistema Linux, e a permettere, infine, lo scambio ordinato dei messaggi tra i due ambienti.

I componenti hardware utilizzati si sono dimostrati validi per le caratteristiche intrinseche e la loro corrispondente gestione. Innanzitutto, la gestione della Quality of Service sullo switch Microchip KSZ9477 è stata fondamentale. Il dispositivo, infatti, permette di utilizzare diverse modalità di QoS e il relativo settaggio è molto semplice. A seconda delle esigenze del momento, durante l'utilizzo del sistema implementato, è dunque possibile utilizzare a propria scelta una delle modalità disponibili (banalmente attivandone una e disattivando le altre).

I possibili scenari futuri di questo sistema di comunicazione sono molteplici, infatti il suo utilizzo non è assolutamente vincolato al contesto preso in esame, ma può trovare utilizzo in tutte quelle situazioni in cui si rende necessario avere delle buone prestazioni in termini di bassa latenza, intelligibilità e priorità del traffico. Tutti i componenti del sistema sono facilmente implementabili in differenti ambienti fisici, non presentando alcun problema di ingombro. Il sistema può sicuramente essere esteso o modificato, ad

esempio utilizzando altri sistemi embedded o architetture general-purpose, magari a costi ancora più ridotti, il tutto facilitato dalla ottima versatilità di OpenDDS.

Appendice A

Compilazione di OpenDDS in ambiente Linux

Dopo aver correttamente scaricato dal sito ufficiale (opendds.org) la distribuzione di OpenDDS, è necessario estrarre il pacchetto .tar.gz attraverso il comando da terminale:

```
> tar -xvzf OpenDDS-<version>.tar.gz
```

verrà creata una cartella nella directory corrente con tutti i file estratti. In seguito, sarà necessario seguire, esattamente nell'ordine con cui sono elencate, le seguenti operazioni:

1. Entrare nella directory di OpenDDS.

2. Lanciare da terminale il comando:

   ```
   > ./configure
   ```

   Verrà scaricata automaticamente l'ultima versione di ACE+TAO e verranno settati tutti i parametri necessari alla compilazione per architettura Linux, compresa la generazione dei Makefile che saranno utilizzati nel passo successivo.

3. Al termine delle operazioni precedenti, lanciare il comando:

   ```
   > make
   ```

il processo durerà molto tempo (probabilmente più di un'ora, in base alle impostazioni di sistema della macchina virtuale), al termine del quale la distribuzione di OpenDDS sarà compilata e pronta ad essere eseguita sulla macchina stessa. Per informazioni aggiuntive, come, ad esempio, abilitare il Java bindings, è possibile consultare la guida ufficiale di OpenDDS: http://opendds.org/quickstart/GettingStartedLinux.html

Appendice B

Cross-compilazione di OpenDDS per ARM

A differenza della semplice compilazione della distribuzione per Linux, nel caso della cross-compilazione per un'architettura differente, nel caso specifico per ARM, bisogna seguire degli accorgimenti ulteriori, e prestare particolare attenzione alle operazioni da eseguire. Dopo aver correttamente scaricato dal sito ufficiale (opendds.org) la distribuzione di OpenDDS, è necessario estrarre il pacchetto .tar.gz attraverso il comando da terminale:

```
> tar -xvzf OpenDDS-<version>.tar.gz
```

verrà creata una cartella nella directory corrente con tutti i file estratti. In seguito, sarà necessario seguire esattamente le seguenti operazioni:

1. Aggiungere alla directory @INC per perl della SDK poky (<root-directory>/ sysroots/x86_64-pokysdk-linux/usr/lib/perl/5.20.0 i file .pm mancanti e richiesti per la cross-compilazione, reperibili su Ubuntu tramite il comando da terminale:

   ```
   > cpan -D nome_file.pm
   ```

   che restituisce il percorso in cui reperire i suddetti file.

   I file mancanti sono:

   - FindBin.pm
   - Driver.pm
   - mpc_debug.pm

- Options.pm
- DirectoryManager.pm
- StringProcessor.pm
- ProjectCreator.pm
- FileHandle.pm,
- Creator.pm
- Parser.pm
- OutputManager.pm
- TemplateInputReader.pm
- TemplateParser.pm
- WinVersionTranslator.pm
- FeatureParser.pm
- CommandHelper.pm
- Version.pm
- ConfigParser.pm

2. Esiste, nella directory di installazione di Poky, un file di settaggio delle variabili d'ambiente per la cross-compilazione. Non utilizzarlo, poiché forzerebbe ad utilizzare il compilatore Poky per tutta la distribuzione.

3. Entrare nella directory "OpenDDS-<version>"

4. Lanciare da terminale il comando:

```
> ./configure --target=linux-cross –target-compiler=”/opt/
poky/1.7/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-
linux-gnueabi/arm-poky-linux-gnueabi-g++”
```

dove `/opt/poky/1.7` è la directory di default su cui è installata l'SDK per la cross-compilazione. Verrà scaricata automaticamente l'ultima versione di ACE+TAO e verranno settati tutti i parametri necessari alla cross-compilazione, compresa la generazione dei Makefile.

5. Al termine delle operazioni precedenti, lanciare il comando:

   ```
   > CXXFLAGS=”-02 -pipe -g -feliminate-unused-debug-types -mfloat-abi=hard” CPPFLAGS=”-mfloat-abi=hard” make
   ```

   il processo durerà molto tempo (probabilmente più di un’ora, in base alle impostazioni di sistema della macchina virtuale), al termine del quale la distribuzione di OpenDDS sarà compilata e pronta ad essere caricata sulla scheda i.MX. Tutto ciò che è possibile far girare sulla scheda si trova sotto la directory `<root>/build/target`, dove sono stati compilati tutti i sorgenti e gli esempi che è possibile utilizzare e modificare in base alle esigenze.

6. Dopo aver collegato la scheda i.MX al PC attraverso un cavo di rete, ed aver settato le impostazioni di rete della VM con un indirizzo IPv4 compreso nella sottorete della i.MX (del tipo 192.168.1.X, con subnet mask 255.255.255.0) è possibile accedere alla scheda tramite SSH.

7. Creare un archivio .tar.gz della distribuzione di OpenDDS cross-compilata lanciando da terminale il comando:

   ```
   > tar czf OpenDDS.tar.gz OpenDDS-<version>
   ```

8. Copiare l’archivio appena creato sulla scheda, lanciando il comando:

   ```
   > scp OpenDDS.tar.gz USER@ADDRESS:<destination_path>
   ```

   dove USER@ADDRESS, nel caso specifico, è root@192.168.1.10, per accedere alla scheda da root. Non inserendo il path di destinazione, l’archivio sarà copiato nella directory /home/root della scheda.

9. Da terminale, accedere alla i.MX tramite SSH, con il comando:

```
> ssh root@192.168.1.10
```

10. Spostarsi sulla directory in cui è stata copiata la distribuzione ed estrarre l'archivio lanciando il comando:

```
> tar xzf OpenDDS.tar.gz
```

Spostandosi sulla directory estratta, sarà possibile utilizzare OpenDDS sulla i.MX.

Appendice C

Esecuzione di IntroductionToOpenDDS

È stato testato l'esempio IntroductionToOpenDDS, opportunamente modificato per permettere l'esecuzione delle varie entità (broker, publisher e subscriber) non solo sulla scheda i.MX ma anche su diverse macchine e, quindi, su differenti indirizzi IP. Nel caso specifico gli host coinvolti sono la i.MX (architettura ARM) raggiungibile all'indirizzo IPv4 locale 192.168.1.10 e la virtual machine Ubuntu (Linux) la cui scheda di rete virtuale è stata impostata con indirizzo IPv4 192.168.1.50. L'esempio IntroductionToOpenDDS è stato scritto per permettere lo scambio di quotazioni in borsa, oltre che ad eventi di scambio significativi, come le negoziazioni aperte, chiuse, sospese o riprese. La struttura dei topic è presente nel file StockQuoter.idl, nel caso specifico sono presenti due topic: uno per la quotazione (con tutti gli attributi necessari) e uno per l'evento di trading.

StockQuoter.idl

```
1 // ********************************************************************
2 //
3 // (c) Copyright 2006, Object Computing, Inc.
4 // All Rights Reserved.
5 //
6 // ********************************************************************
7
8 #include "orbsvcs/TimeBase.idl"
9
10 module StockQuoter
11 {
12 #pragma DCPS_DATA_TYPE "StockQuoter::Quote"
13 #pragma DCPS_DATA_KEY "StockQuoter::Quote ticker"
14
15   struct Quote {
16     string ticker;
17     string exchange;
18     string full_name;
19     double value;
20     TimeBase::TimeT timestamp;
21   };
22
23 #pragma DCPS_DATA_TYPE "StockQuoter::ExchangeEvent"
24 #pragma DCPS_DATA_KEY "StockQuoter::ExchangeEvent exchange"
25
26   enum ExchangeEventType { TRADING_OPENED,
27                            TRADING_CLOSED,
28                            TRADING_SUSPENDED,
29                            TRADING_RESUMED };
30   struct ExchangeEvent {
31     string exchange;
32     ExchangeEventType event;
33     TimeBase::TimeT timestamp;
34   };
35 };
```

Figura 38 – StockQuoter.idl

In questo esempio basilare, il publisher è configurato per generare diversi messaggi associati al topic "Quote" e all'evento "ExchangeEventType" per qualche secondo, mentre il subscriber, quando lanciato, inizia a ricevere i messaggi. L'esempio può essere opportunamente modificato, in modo molto semplice, cambiando nomi e attributi ai topic (o aggiungendone degli altri) per adattarlo alle proprie esigenze. Per farlo, è necessario modificare il file con estensione .idl, e, successivamente, compilarlo. Il comando da eseguire, dopo essere entrati nella directory in cui si trova il file, è il seguente:

```
> $DDS_ROOT/bin/opendds_idl StockQuoter.idl
```

in cui viene utilizzato il compilatore per le interfacce IDL, mentre se vengono modificate le classi C++ (ad esempio per modificare il comportamento generale dei publisher o subscriber), è necessario lanciare il seguente comando:

```
> $ACE_ROOT/ bin /mwc. pl –type gnuace StockQuoter.mwc &&
make
```

Questo esempio, è presente, nella scheda i.MX, nella directory `<root_dir>/build/target/examples/DCPS/IntroductionToOpenDDS`, nella soluzione compilata per la VM Ubuntu, invece, si trova all'interno della directory `/examples/DCPS/IntroductionToOpenDDS`. Il funzionamento è stato testato inizialmente lanciando le tre entità sulla i.MX, e successivamente su i.MX e macchina virtuale, per provare l'effettiva comunicazione. Per quanto riguarda il primo caso, dopo aver fatto l'accesso alla i.MX tramite SSH, è necessario innanzitutto lanciare un broker, che metterà in comunicazione i vari publisher e subscriber. Per fare questo, è necessario preliminarmente creare i riferimenti alle librerie necessarie a OpenDDS per la sua corretta esecuzione, nonché settare i path corretti in cui trovare le distribuzioni di ACE e TAO. È possibile creare uno script dal nome `setenv.sh` nella root directory di OpenDDS, costituito dalle seguenti istruzioni:

```
export DDS_ROOT=/home/root/OpenDDS-3.13/build/target
export ACE_ROOT=$DDS_ROOT/ACE_wrappers
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$ACE_ROOT/lib:$DDS_ROOT/lib
export PATH=${PATH}:$ACE_ROOT/bin:$DDS_ROOT/bin
export CIAO_ROOT=unused
export MPC_ROOT=/home/root/OpenDDS-3.13/ACE_wrappers/MPC
export TAO_ROOT=/home/root/OpenDDS-3.13/ACE_wrappers/TAO
export DANCE_ROOT=unused
```

dove `/home/root` deve essere sostituito con la directory in cui è presente la distribuzione, nel caso sia diversa. Dopo essere entrati nella directory di OpenDDS, va lanciato lo script precedente, con il comando:

```
> source ./setenv.sh
```

Importante: questo comando va lanciato su tutti i terminali che verranno utilizzati.

Per attivare il broker, e conseguentemente indicare un dominio specifico, lanciare il comando:

```
> $DDS_ROOT/bin/DCPSInfoRepo -ORBListenEndpoints
iiop://192.168.1.10:12345 -d 1066
```

dove `192.168.1.10` è l'indirizzo del broker (in questo caso la scheda i.MX stessa), `12345` è la porta su cui è in ascolto il broker e il flag `-d` indica il dominio. In seguito, aprendo altri due terminali e collegandosi sempre alla i.MX tramite SSH, dopo aver lanciato lo script `setenv.sh` su entrambi i terminali, lanciare un publisher e un subscriber con i seguenti comandi:

Publisher:

```
> cd $DDS_ROOT/examples/DCPS/IntroductionToOpenDDS/

> ./publisher -DCPSConfigFile ./dds_tcp_conf.ini
```

Subscriber:

```
> cd $DDS_ROOT/examples/DCPS/IntroductionToOpenDDS/

> ./subscriber -DCPSConfigFile ./dds_tcp_conf.ini
```

dove il file dds_tcp_conf.ini è il file di configurazione in cui sono presenti le istruzioni necessarie ad individuare il broker (DCPSInfoRepo) e il protocollo per il trasporto dei dati da utilizzare (in questo caso TCP). La figura che segue, mostra il file dds_tcp_conf.ini così come è stato modificato per la macchina virtuale Ubuntu. È possibile, infatti, vedere come sia presente una riga in cui appare l'indirizzo della macchina virtuale (DCPSBitTransportIPAddress). Questo file di configurazione deve essere utilizzato quando il broker è attivo sulla scheda i.MX, altrimenti devono essere modificate la riga 10 e la riga 35, sostituendo l'indirizzo presente con quello corrente su cui il broker è in ascolto.

dds_tcp_conf.ini

```
1 # This "common" section configures the data in Service_Participant.
2 [common]
3
4 # Debug Level
5 DCPSDebugLevel=0
6
7 DCPSBitTransportIPAddress=192.168.1.50
8 DCPSBitTransportPort=11111
9 # IOR of DCPSInfoRepo process.
10 DCPSInfoRepo=corbaloc::192.168.1.10:12345/DCPSInfoRepo
11 DomainId=1066
12
13 # Sets the global transport configuration (used by default in the
14 # process to config1, defined below
15 DCPSGlobalTransportConfig=config1
16
17
18 # Transport configuration named config1, contains a single transport
19 # instance named tcp1 (defined below)
20 [config/config1]
21 transports=tcp1
22
23
24 # Transport instance named tcp1, of type "tcp".  Uses defaults for
25 # all configuration paramaters.
26 [transport/tcp1]
27 transport_type=tcp
28 local_address=192.168.1.50
29
30 [domain/1]
31 DiscoveryConfig=DiscoveryConfig1
32
33 [repository/DiscoveryConfig1]
34 #RepositoryIor=192.168.1.50:12345
35 RepositoryIor=192.168.1.10:12345
```

Figura 39 – dds_tcp_conf.ini

L'indirizzo del broker, come visibile dal file di configurazione, è settato dalla riga 10 (`DCPSInfoRepo=corbaloc::192.168.1.10:12345/DCPSInfoRepo`), in seguito è anche specificato il dominio di interesse, `DomainId=1066`. Quando tutti i componenti di OpenDDS girano sulla i.MX, attivando in sequenza il broker, il subscriber e il publisher, i tre terminali si presentano in questo modo:

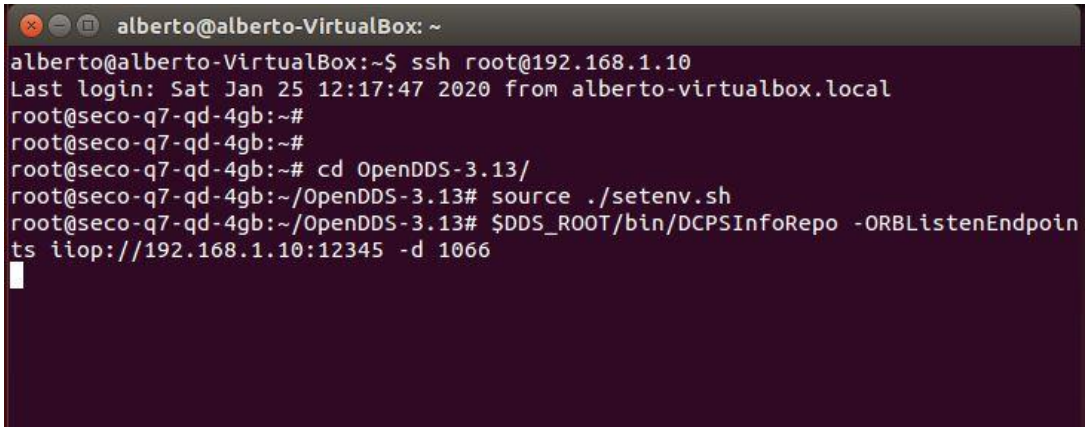


Figura 40 – Terminale Linux (broker)

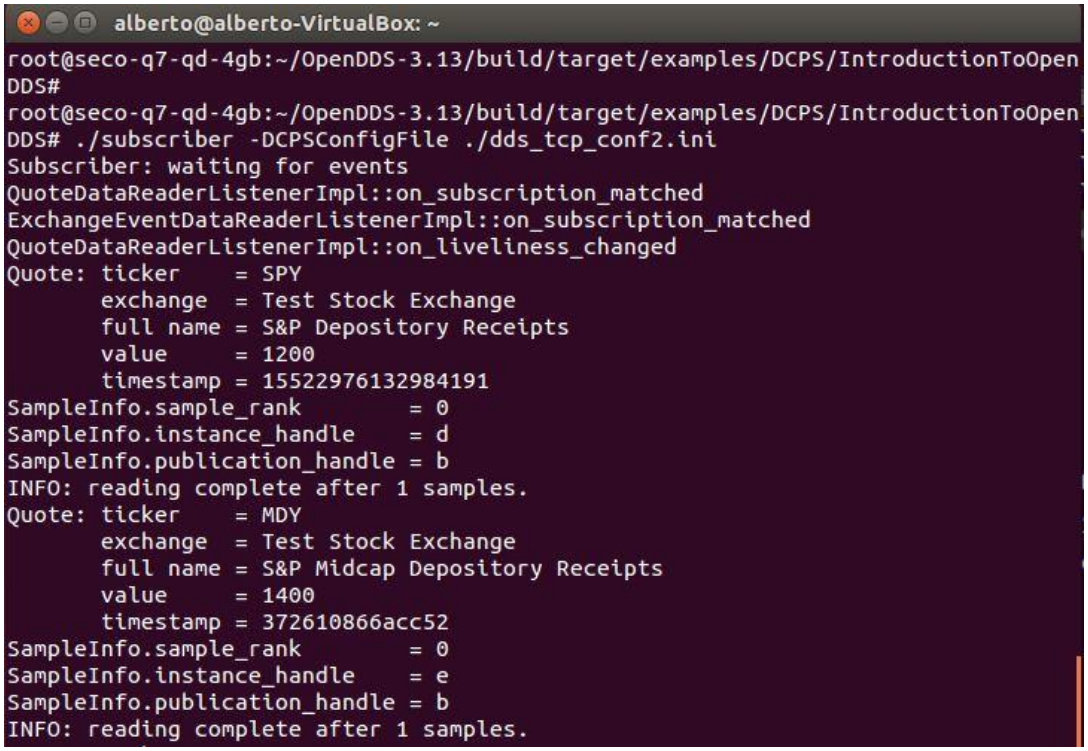


Figura 41 – Terminale Linux (subscriber)

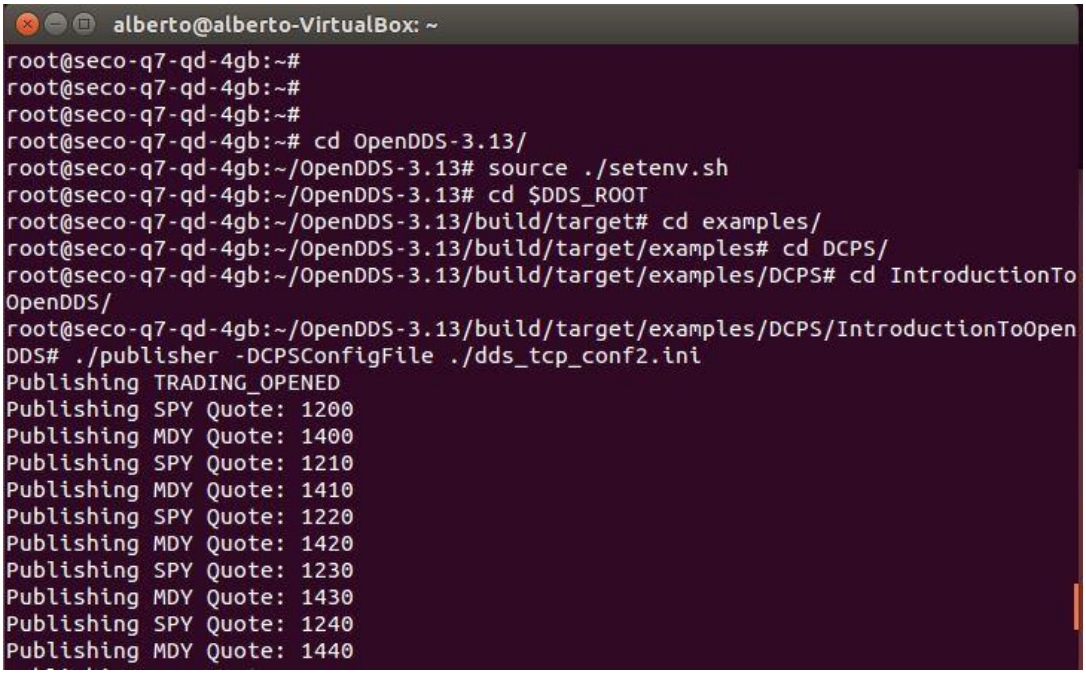


Figura 42 – Terminale Linux (publisher)

Sono state provate anche altre configurazioni, in cui i vari componenti sono stati eseguiti sia su scheda che su macchina virtuale. In particolare, complessivamente, sono stati testati i seguenti casi:

- broker, publisher e subscriber su i.MX
- broker, publisher e subscriber su VM Ubuntu
- broker su i.MX, publisher e subscriber su VM Ubuntu
- broker e subscriber su i.MX, publisher su VM Ubuntu
- broker su VM Ubuntu, publisher e subscriber su i.MX
- broker e publisher su VM Ubuntu, subscriber su i.MX

Bibliografia

[1] Spanning Tree Protocol (Cisco sito web). URL: https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree protocol/5234-5.html

[2] Spanning Tree Protocol, Rapid Spanning Tree Protocol, Prof. Fulvio Risso - Politecnico di Torino – Progetto di reti locali, syllabus (slides). URL: https://prl.frisso.net/syllabus

[3] The ultimate speed test tool for TCP, UDP and SCTP. URL: iperf.fr

[4] Libpcap-based Ethernet packet generator. URL: bittwist.sourceforge.net

[5] DSCP and ToS Values. URL: https://www.bytesolutions.com/dscp-tos-cospresidence-conversion-chart/

[6] eSpeak text-to-speech, URL: http://espeak.sourceforge.net/

[7] iSpeech, URL: https://www.ispeech.org/create.text.to.speech.audio/

[8] MaryTTS, URL: http://mary.dfki.de/

[9] Google Cloud Speech-to-Text, URL: https://cloud.google.com/speech-to-text/

[10] Houtgast, Steeneken (Dicembre 1971) - "Evaluation of Speech Transmission Channels by Using Artificial Signals", Acustica Vol. 25, n.6, pag. 355–367

[11] BSI Standard Publication – Sound System Equipment, Part 16: Objective rating of speech intelligibility by speech transmission index (BS EN 60268-16:2011), Annex B, pag. 47

[12] STIPA, Speech Intelligibility Tester, Apple App Store - URL: https://itunes.apple.com/us/app/stipa/id582112259?mt=8

[13] Object Management Group (OMG) – Data Distribution Service (DDS) – URL: https://www.omg.org/omg-dds-portal/

[14] DDS, The Proven Data Connectivity Standard for the IoT – URL: www.omgwiki.org/dds/what-is-dds-3/

[15] OpenDDS (Object Computing) – URL: opendds.org

Ringraziamenti

Un sentito ringraziamento va, innanzitutto, ai miei relatori, il Chiar.mo Prof. Enrico Masala e il Chiar.mo Prof. Guido Marchetto, per avermi supportato ed essere stati sempre disponibili durante tutto il lavoro svolto per la stesura di questa tesi. Un particolare ringraziamento va a tutti coloro che sono stati fondamentali durante il mio percorso accademico, ovvero alla mia famiglia, ad amici e conoscenti di vecchia data, ai coinquilini e ai nuovi amici e colleghi conosciuti a Torino.