POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Telematica (Computer and Communication Networks Engineering)

Master Degree Thesis

Energy-Aware Emulation of Software Defined Networks

Relatore
prof. Chiasserini Carla Fabiana

Correlatori:
prof. Giaccone Paolo
prof. Casetti Claudio Ettore

Walter Laera
matricola: 208290

A.A. 2017 – 2018

Abstract

The aim of this thesis is the presentation and simulation of an algorithm that, through a southbound API, provides to the SDN controller the energy consumption of each network device.

Initially, we conducted an analysis of the state of the art of the SDN network and a research on the frameworks available for measuring/estimating the energy consumption of the different network devices.

At the end of the research, we have decided to use an analytical model that binds device's energy consumption with its ingress rate which can be calculated through the incoming bytes, of each switch's interface, provided by SNMP as southbound protocol. Then we created a virtual network to perform simulations of our algorithm for data collection. The differences between our tests' results and the energy consumptions measured in the paper [20] are less than 0.5 watts, which means that we have an error less than 0.1%.

In our opinion, the method considered in this study has points in favour and, at the same time, limitations that can be summarized, respectively, in the easy integration in the current controllers, since they already measure the rate of the different nodes, and in the fact that the SDN controller should have a database with all analytical models of each controlled device, this because at the same rate, each device has a different energy consumption that depends not only by the model but also by the producer.

From our study carried out, in conclusion, we can assert that this method can be a good alternative until the devices themselves become able to measure their energy consumption and provide it to southbound protocols.

Contents

List of Figures

List of Tables

Introduction

The use of IT networks has noticed an exponential growth in the last twenty years. This because, in a world in which devices that are connected to the network increase year by year, as well as increasing performances requested by users, therefore, Internet providers are amplifying and upgrading their networks to ensure a high-quality service. These conditions have made communication networks one of the biggest consumers of electricity in the world, placing research in front of a new challenge: reducing the consumption of IT networks.

In the last few years a new network architecture is being developed, the Software Defined Network (SDN), which separates the control plane from the data one, centralising it. In this way, it provides a wider view of the network, allowing better traffic management. It was therefore thought that a new decision parameter, for data routing, could be the energy consumption of the different network nodes in order to reduce as much as possible the total energy consumption of the network.

The aim of this thesis is the presentation and simulation of an algorithm that, through a southbound API, provides to the SDN controller the energy consumption of each network device.

Initially, we conducted an analysis of the state of the art of the SDN network and a research on the frameworks available for measuring/estimating the energy consumption of the different network devices. Then we created a virtual network to perform simulations of the framework chosen for data collection.

The thesis is divided into four chapters: in the next chapter, we show the state of the art of the SDN network architecture, with more attention on the southbound API and presenting the main protocols used. The second chapter presents the problems of world energy consumption regarding communication networks and the possible frameworks for real-time collection of energy consumption data from different network nodes. In the third chapter we present the tools and algorithms used for the simulations of an SDN network and its energy consumption collection algorithm. In the fourth and final chapter, we summarize the work done and comment the results obtained.

Chapter 1

Software Defined Network and Its Southbound Interfaces

In this chapter we will introduce a new network architecture SDN, focusing on southbound interfaces, in particular on SNMP (Simple Network Management Protocol).

1.1 Software-Defined Networking (SDN)

IT networks structures have a distributed control with network nodes run transport protocols. These structures allow that information reaches different places of the world. IP protocol based networks, even if widely utilized, are complicated and difficult to handle: to underline the required high-level network policies, network executives need to configure each network device individually using low-level languages and in many cases vendor-specific constraining languages. Also, network conditions have to endure the dynamics of faults and re-adjust to load variations. In today's IP networks, automated reconfiguration and response mechanisms are essentially unavailable.

To keep the increasing elasticity, innovation and growth of networking infrastructure the control plane (that determines how to manage network traffic) and the data plane (that delivers traffic in order to the decisions performed by the control plane) should not be bundled with the networking devices. Today's IP networks don't have this feature because they are vertically integrated.

Software-Defined Networking (SDN) is an emerging networking model [21] which improves the limitations of current network infrastructures. It will break the vertical integration dividing the network's control logic (the control plane) from the underlying data plane that forwards the traffic. The control logic is implemented in a centralized controller (or network operating system) to simplify the policy

requirement and network (re)configuration and development. Network switches, thanks to this division, will become simple forwarding devices.

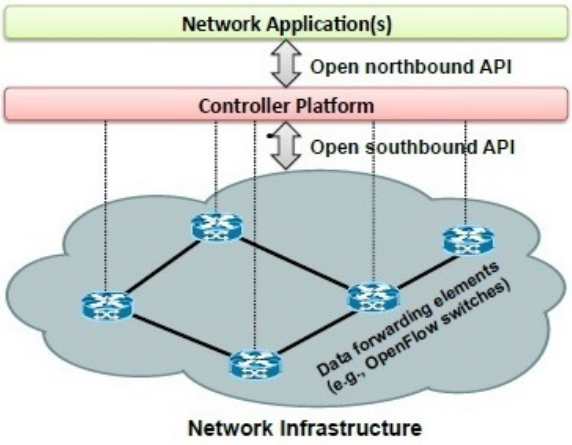

Figure 1.1. Simplified view of an SDN architecture. Reproduced from [21]

The SDN architecture (fig.1.1) is structured as follows:

- Forwarding device: hardware- or software-based data plane devices that execute a set of essential actions. These actions are defined by specific instruction sets (e.g. flow rules) and applied on the incoming packets (e.g. forward to particular ports, drop, rewrite some header, forward to the controller). The instruction sets are set by southbound interfaces and are installed in the forwarding devices by the SDN controllers implementing the southbound protocols.

- Southbound interface: they are used to formalize communication between the control and data plane elements. In the southbound interface, the APIs (Application Programming Interface) define an instruction set for the forwarding devices. These APIs are communication protocols between forwarding devices and control plane elements.

- Control plane: its components set up forwarding devices through southbound API. The control plane is nothing less than the network coordinator. It's composed by all control logic lays in the applications and controllers.

- Northbound interface: the network operating system (SDN controller) provides an API (the northbound interface) to develop the applications. Usually, this interface abstracts the low-level instruction sets used by southbound interfaces to process forwarding devices.

- Management plane: this plane is the set of applications that leverage the functions given by the northbound interface to apply network control and

operation logic. It involves applications such as routing, monitoring, load balancers, firewalls, and so on. It outlines the policies, which are finally translated to southbound-specific instructions, that define the function of the forwarding devices.

1.2 Southbound APIs for Monitoring

Southbound APIs are communication protocols that formalize a communication language between the control plane and the data plane. They are not only used to configure but also to monitor the forwarding devices. Gathering information is essential to manage the network, to make decisions and set the devices in the best way. Management applications need correct and timely statistics of the network resources with different aggregation levels. The network overhead for statistics collection should be minimal.

Various protocols (southbound API), used to monitor network devices, are controlled and examined in this thesis. The analysed protocols are the following:

- OpenFlow [11]: it is accepted as the actual interface between control and data planes. In order to give a global view of the network, Open-Flow provides different counters: per-flow, per-port, per-queue and per-group.

- SNMP (SimpleNetworkManagementProtocol) [29]: SNMP is a component of the Internet Protocol Suite as defined by the Internet Engineering Task Force (IETF). It aims to collect and organize information about controlled devices on IP networks (section 1.2.1).

- sFlow [16]: this standard tends to provide accurate network measurements without needs per-flow state of the switches. It employs time-based sampling for collecting traffic information and provides two forms of sampling: packet sampling and port counter sampling. sFlow is not generally utilized by the vendors.

- NetFlow [30]: created by Cisco, provides a procedure to obtain statistics about individual IP flows in data networks. NetFlow renders information such as source and destination IP address, port numbers, byte count, etc. It supports different technologies like multi-cast, IPSec, and MPLS. Although NetFlow is a proprietary protocol, NetFlow version 9 has been assumed to be a common and universal standard by the IP Flow Information Export (IPFIX) working group, so that non-Cisco devices can forward data to NetFlow collectors.

SNMP protocol is chosen because it's one of the most used protocols in the network and it can be deployed in a large set of heterogeneous devices. Furthermore, SNMP southbound is already present in OpenDayLight and ONOS controllers.

1.2.1 SNMP Simple Network Management Protocol

SNMP is a standard protocol that handles and organizes information concerning managed devices on IP networks. Currently, SNMP is mostly used for monitoring and managing the performance of network devices.

There are three versions of SNMP:

The first version (SNMPv1) was developed by IETF in the '80s and it still is the actual network-management protocol in the Internet community, although it is criticized for the poor security and its complicated code.

The second one (SNMPv2) updates version 1 and adds enhancements in performance, security, confidentiality and manager-to-manager communications.

The last version of the protocol (SNMPv3) adds cryptographic security and improves the problems related to the large-scale deployment of SNMP, accounting and fault management.

A SNMP management system contains:

- several nodes with a SNMP entity, called agents. They manage and collect information about the devices in which they are installed;
- at least one SNMP entity called manager, which sends commands to the agents and receives notifications from their;
- a management protocol used to transfer management information between the SNMP entities.

Agents control devices such as hosts, routers, terminal servers, etc. These devices are monitored and controlled through the access of their own management information.

Management information can be seen as a collection of managed objects organized in Management Information Base (MIB). Each MIB module collects similar objects. They are tree structures and each node of the tree is an object of the MIB which is identified by an Object IDentifier (OID). The OID consists of a dotted list of integers (fig.1.2).
The protocol provides three ways to access management information:

The first way of communication is request-response, in which the manager sends a request to an agent that responds. This interaction is used to retrieve or modify information regarding the managed device.

The second way of communication is also a request-response, but between two managers. This type of interaction is used to exchange information from an SNMP

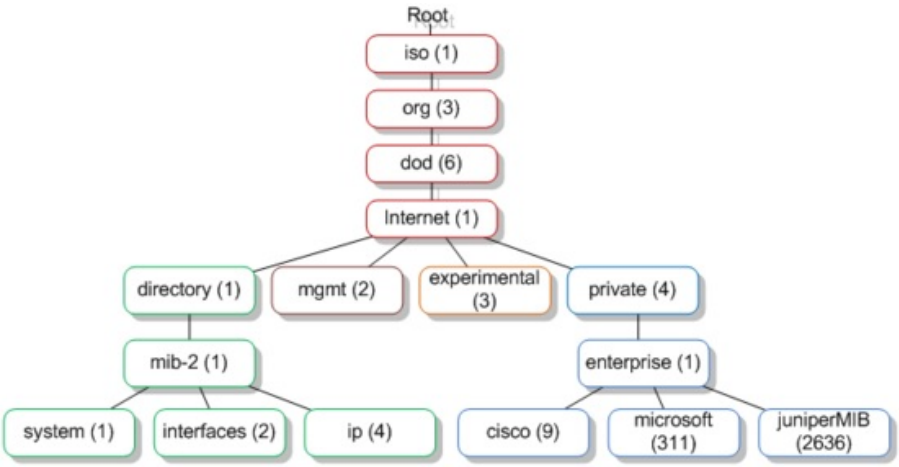

Figure 1.2. OID tree example.

Reproduced from:
"www.networkmanagementsoftware.com/snmp-tutorial-part-2-rounding-out-the-basics/"

manager to another.

The last way of communication is an unsolicited message from an agent that has an unexpected situation caused by modifications to management information related to the device. This interaction can be with or without the acknowledgement from the manager.

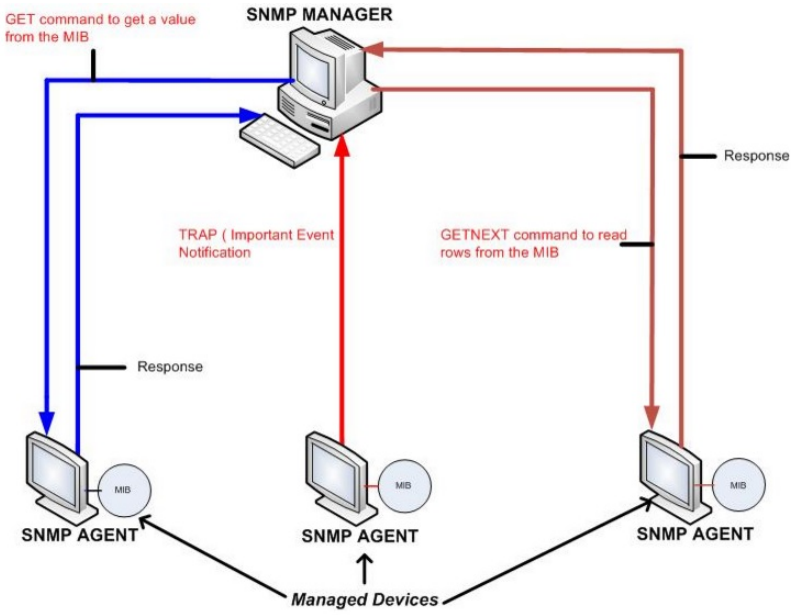

Figure 1.3. SNMP Communication.

Reproduced from: "www.protoconvert.com/TechnicalResourses/Tutorials/SNMP.aspx"

The commands used to communicate between SNMP entities (e.g. fig.1.3) are:

- GET: this command is sent by a manager to an agent to retrieve one data value from a MIB.
- GETBULK: this operation is used to recover lots of data from MIBs. The manager sends a GETBULK command when requests to the agent more than one OID, in this way the SNMP traffic is reduced.
- GETNEXT: this command is like GET. The difference is that in this operation it collects the value of the next OID in the MIB tree.
- SET: the manager uses this command to modify or assign a value to an OID. If the operation is successful the agent sends back a GET-RESPONSE message, in the other case the agent sends an error indication with the motivation of the failure.
- TRAPS: differently from all previous commands, the TRAP message is sent from agent to manager. It is automatically generated to inform the manager about an important event.
- INFORM: this message is similar to the TRAP, the difference is that when an agent sends an INFORM message, the SNMP manager responds with an acknowledge message.
- RESPONSE: this command is used to respond to the different GET messages (GET, GETBULK, GETNEXT) or to the SET message.

Chapter 2

Energy Management Relevance and Energy Frameworks

In this chapter, we will underline the importance of energy consumption of IT networks, and will present different frameworks that collect data regarding energy consumption.

As seen in fig. 2.1 global telecommunication networks have a high energy consumption and this is the reason why the information technology (IT) industry is focusing on the energy performance of the network, in order to lower capital and recurring costs.

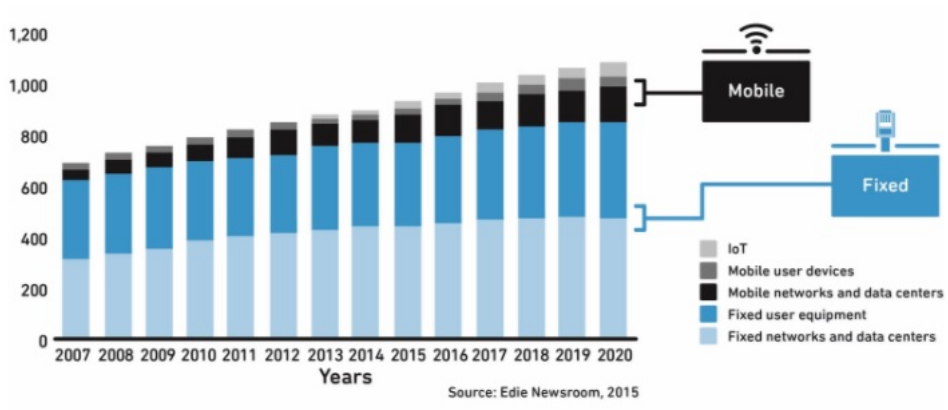

Figure 2.1. Global energy consumption for network infrastructure and user equipment during operational phase (TWh). Reproduced from [34]

Until recently, energy management for networking devices, in wired networks, has not gained much consideration. Researchers have suggested more than one strategy to make these devices more energy-aware: link rate adaptation in case of

low traffic, sleeping during no traffic periods. Currently, however, there is a poor understanding of the specific energy savings obtained by choosing these techniques, and no hardware implementations. One obstacle to innovation in this domain is the lack of power measurements from live networks and a good perception of how consumed energy changes under different traffic loads and switch/router configuration settings.

We found four ways to collect their energy consumption data:

- The first is GAL (Green Abstraction Layer) framework (section 2.1).
- The second is KWAPI framework (section 2.2).
- The third is Energy Management (EMAN) framework (section 2.3).
- The last one uses energy consumption models of different devices with a monitoring traffic protocol (cited previously in 1.2) in order to evaluate the energy consumption of each device. There are different detailed models (section 2.4) concerning switches and routers, which show the devices' consumption with different traffic loads.

We have decided to use the last method to provide devices' power consumption data, since other methods need hardware components, implemented or added, to network devices. In fact, there are still few network devices that can calculate their own power consumption and it is expensive to apply watt-meters to other devices.

2.1 GAL Framework

Green Abstraction Layer (GAL) [28] provides an interface between the control framework (local or network) and the low-level physical resource management algorithms, where energy-saving policies are applied.

GAL has been developed to hide the implementation details of energy-saving methods and helps supply standard interfaces for communication between the hardware and its control framework. It provides a hierarchical representation of the device's internal organization, as we can see in figure 2.2, where the tree graph represents the components at different levels. As said in [28], the goal of the GAL is to provide:

- a common and simple way to represent the power management capacities available in heterogeneous data plane hardware;
- a framework for information trade between power-managed data plane objects and control processes;
- a reference control chain that supports a consistent hierarchical structure of multiple local and network-wide energy management protocols.

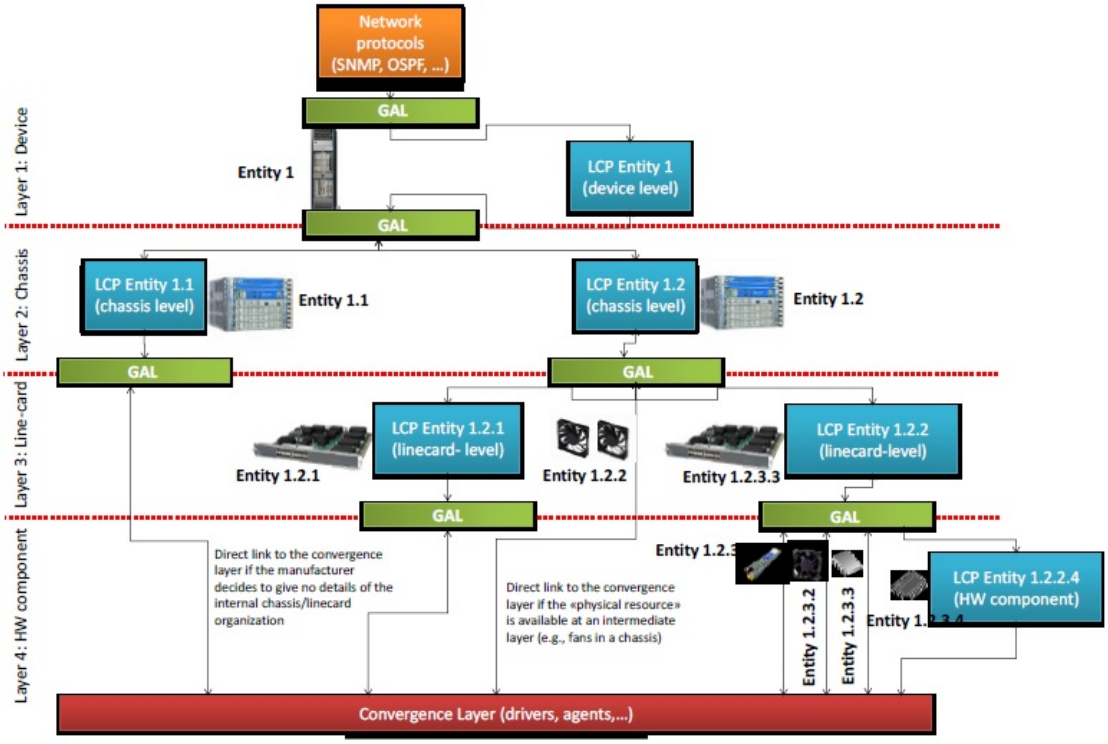

Figure 2.2. The hierarchical architecture of GAL. Reproduced from [28]

Standard Green Interface (GSI), GAL's northbound interface, allows the management of energy-sensitive hardware entities. GSI provides a variety of functions and data types to allow GAL's interface to work, even if the implementation of modules that interact with the hardware depends on the devices (e.g. network elements, network adapters, chips, fans etc.). The main tools offered by the GSI functions are:

- Discovery: it is employed to retrieve information about: available energy configurations and other information of the entity and a list of manageable components within the entity and their relations;
- Provisioning: it allows the setting of an energy configuration of the entity.
- Monitoring: it collects significant parameters of the physical device.

We make an example of GAL functioning in order to understand better how it works.

When a controller modifies the configuration of a logical link (e.g. the IP layer) by setting it into sleep mode, a command is sent to the interested device, through a network protocol. GAL receives the command, defines a highest-level Local Control Policy (LCP) and the physical resources bound to the logical link (e.g. which physical port). If there are no more active links, the layer-3 LCP puts the entire

line-card to sleep, otherwise, the LCP puts the physical interface into standby mode and decreases the performance of all the hardware components that process packets for that port. As we can guess the work of the controller is easier, it only defines the configuration of a logical link while GAL chooses how to manage each part of the physical device.

2.2 KWAPI Framework

The KWAPI framework [5] was created to supply power consumption data to the OpenStack platform.

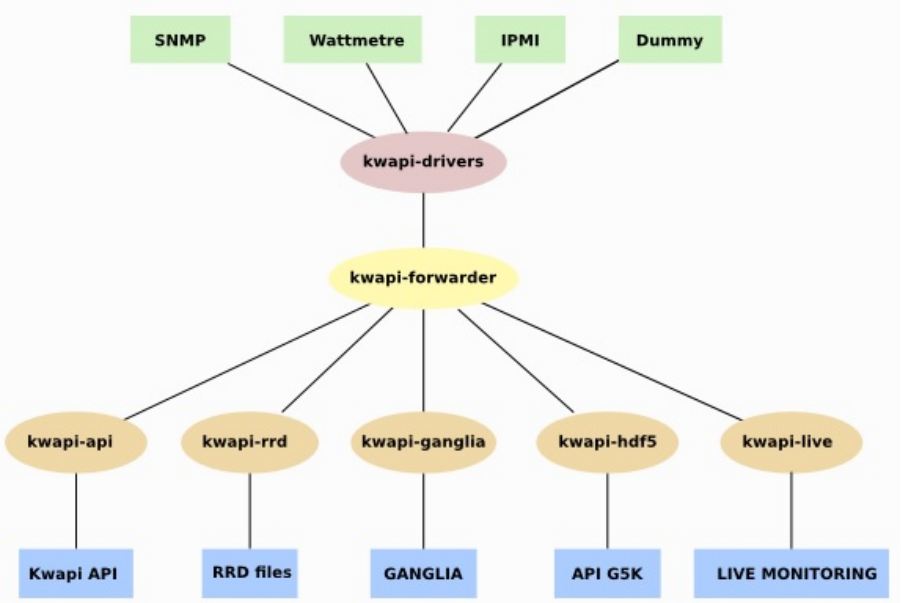

Figure 2.3. Kwapi architecture. Reproduced from [5]

Its architecture (fig.2.3) is based on two layers. The first one, the layer of drivers, recovers data from various devices. The second one, the layer of plugins, is responsible for the collection and elaboration of the data. These two layers communicate through a forwarder or a bus.

A Driver Manager controls the driver layer running the configuration file and, for each entry found, initializing a thread. Each entry consists of a list of probes, the type of driver to use and the relevant parameters. All data retrieved from the drivers are sent, in JSON format, to the forwarder, using ZeroMQ as the transport layer. ZeroMQ [19] is a high-performance asynchronous messaging library that provides to the developers the instruments to build their message queuing system in a distributed and concurrent way.

As said in [5], "the Forwarder operates by using a publish/subscribe pattern,

where the drivers are publishers and the plugins are subscribers." It can work locally or in a distributed architecture. Locally means that, for example, the drivers and the plugins are on the same machine, even if this configuration is not recommended. In the distributed architecture, instead, thanks to a gateway machine that connects isolated networks, a plugin can receive several measurements from remote drivers using the forwarder.

As said before, the forwarder provides data to the plugin that processes measurements. The plugins are:

- REST API: it permits to an external system to access real-time data.
- RRD: it is a plugin that stores the information received from the drivers in RRD files. The advantage of such files is that they render efficiently graphs.
- Live plugins: they are based on Flask and RRDtool to give a web interface.
  - Flask [2] is a micro web framework. It doesn't force or presume developers to use a particular library or instrument. It supports extensions that can add application characteristics as if they were implemented in Flask itself.
  - RRDtool [14] is used for high-performance data logging and graphing system for time series data.
- HDF5 plugin [4]: it is used to store detailed metric with Kwapi and allows to store several months of energy consumption data of numerous probes.
- GANGLIA plugin: it allows communication with GANGLIA server. Ganglia [3] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids.

2.3 EMAN Framework

Informational RFC [31] was published in September 2014, in which a framework is defined by providing energy management for devices within, or connected to, communication networks. The first thing that improves the energy efficiency of the network is to allow the devices to communicate their energy consumption. The EMAN framework provides an information model, for electrical equipment (Energy Object), that helps us to deal with this problem. The information model is structured in this way:

There is the identification of the Energy Object. The Energy Object has a Universally Unique Identifier (UUID) that uniquely and persistently identifies it. Furthermore, it can use a unique human-readable printable name that could be: textual DNS name, MAC address of the device, interface ifName or a text string.

Then there is the identification regarding the context of Energy Objects. This category describes how an Energy Object is used, namely, it indicates if the Object

is mainly a consumer, producer, meter, distributor or store of energy.

Finally, the power measurement is described. Where its attributes describe power, energy, and demand measurements.

The IETF has decided to use SNMP protocol to implement this framework and it can be implemented in other protocols.

2.3.1 SNMP MIB of EMAN Framework

This subsection illustrates SNMP MIBs created to monitor the energy consumption of devices present in the network.

The Energy Monitoring MIB [32] has the tools to measure and collect the power consumption data and the Energy Object Context MIB [33] allows to identify and understand relationships among devices.

2.3.1.1 Energy Monitoring MIB

The Energy Monitoring MIB is composed of two MIB modules: the ENERGY-OBJECT-MIB that collects power and energy measurements and the POWER-ATTRIBUTES-MIB that addresses on power quality measurements for Energy Objects.

ENERGY-OBJECT-MIB: this module is made of five tables.

The first one (tab.2.1) is the eoMeterCapabilitiesTable. It helps us understand the other tables regarding ENERGY-OBJECT-MIB and POWER-ATTRIBUTES-MIB, available for each Energy Object.

Entity (OID)	**Description**
eoMeterCapabilitiesEntry(1) [entPhysicalIndex]	An entry describes the metering capability of an Energy Object
eoMeterCapability	An indication of the energy-monitoring capabilities supported by this agent

Table 2.1: eoMeterCapabilitiesTable. Reproduced from [32]

The eoPowerTable (tab.2.2) is the second table. It collects the power consumption and indicates the units, sign, measurement accuracy, and related objects of the Energy Object.

Entity (OID)	**Description**
eoPowerEntry(1) [entPhysicalIndex]	An entry describes the power usage of an Energy Object
eoPower(1)	It indicates ower measured, positive value if the device consume energy otherwise negative

eoPowerNamePlate(2)	It indicates the rated maximum consumption for the fully populated Energy Object
eoPowerUnitMultiplier(3)	The magnitude of watts for the usage value in eoPower and eoPowerNameplate
eoPowerAccuracy(4)	This object indicates a percentage value representing the assumed accuracy of the usage reported by eoPower
eoPowerMeasurementCaliber (5)	It specifies how the usage value reported by eoPower was obtained.
eoPowerCurrentType(6)	This object indicates whether the eoPower for the Energy Object reports alternating current, direct current, or unknown
eoPowerMeasurementLocal (7)	It describe whether the measurement is made at the device itself or from another entity
eoPowerAdminState(8)	This object specifies the desired Power State and the Power State Set for the Energy Object
eoPowerOperState(9)	This object specifies the current operational Power State and the Power State Set for the Energy Object
eoPowerStateEnterReason (10)	This string object describes the reason for the eoPowerAdminState transition

Table 2.2: eoPowerTable. Reproduced from [32]

The value of eoPowerMeasurementCaliber can be:

- Unavailable.
- Unknown: it means that how the usage was determined is unknown. In some cases, the usage is calculated through other devices, so it is not known whether the usage reported is actual, estimated, or static.
- Actual: it indicates that the usage was measured through some hardware or direct physical means.
- Estimated: it means that the usage is an estimation based on the device type, state, and/or current utilization using some algorithm or heuristic.
- Static: it indicates that the usage is reported by external tables, specifications, and/or model information.

The third table (tab.2.3) is the eoPowerStateTable. It describes information and statistics about the supported power states for each Energy Object.

Entity (OID)	**Description**
eoPowerStateEntry(1) [entPhysicalIndex, eoPowerStateIndex]	This entry displays max usage values at every single possible Power State supported by the Energy Object
eoPowerStateIndex(1)	The index of the Power State of the Energy Object within a Power State Set
eoPowerStateMaxPower(2)	The maximum power usage for the power state indicated in eoPowerStateIndex

eoPowerStatePowerUnitMultiplier(3)	The magnitude of watts for the usage value in eoPowerStateMaxPower
eoPowerStateTotalTime(4)	The total time, in hundredths of a second, spent in a particular Power State
eoPowerStateEnterCount(5)	The number of time that an entity has visited a particular Power State since the last reset

Table 2.3: eoPowerStateTable. Reproduced from [32]

The Power State Sets have different standards and implementations. An Energy Object supports more than one Power State Set implementation at the same time.

At the moment, in the framework, there are three defined Power State Set:

- IEEE1621 [17]. The IEEE1621 Power State Set is composed of three simply states: on, off, or sleep.
- DMTF [12]. The Distributed Management Task Force (DMTF) standards organization was based on the CIM (Common Information Model) in order to define a power profile standard.
- EMAN [31]. The EMAN Power States are defined starting from the Power States defined in IEEE1621 with the supplement of the Power States defined in ACPI and DMTF.
  The Advanced Configuration and Power Interface (ACPI) specification supplies an open standard that can be used by the operating systems in order to discover, configure, monitor and power manage of our computer hardware.

Tab. 2.4 lists the EMAN Power States and shows the equivalent Power States of the other standards.

IEEE1621	DMTF	ACPI	EMAN
off	Off-Hard	G3/S5	mechoff
off	Off-Soft	G2/S5	softoff
off	Hibernate	G1/S4	hibernate
sleep	Sleep-Deep	G1/S3	sleep
sleep	Sleep-Light	G1/S2	standby
sleep	Sleep-Light	G1/S1	ready
on	on	G0/S0/P5	lowMinus
on	on	G0/S0/P4	low
on	on	G0/S0/P3	mediumMinus
on	on	G0/S0/P2	medium
on	on	G0/S0/P1	highMinus
on	on	G0/S0/P0	high

Table 2.4: Power State Sets. Reproduced from [31]

The fourth table (tab.2.5) is the eoEnergyParametersTable. With this table, the manager can set up parameters to collect energy measurements.

Entity (OID)	**Description**
eoEnergyParametersEntry(1) [eoEnergyParametersIndex]	An entry controls an energy measurement in eoEnergyTable. An Energy Object can have multiple eoEnergyParametersIndex
eoEnergyObjectIndex(1)	This index identifies the Energy Object
eoEnergyParametersIndex(2)	The index of the Energy Parameters setting for collection of energy measurements for an Energy Object
eoEnergyParametersIntervalLength(3)	It indicates the length of time in hundredths of a second over which to compute the average eoEnergyConsumed measurement
eoEnergyParametersIntervalNumber (4)	The number of intervals maintained in the eoEnergyTable
eoEnergyParametersIntervalMode(5)	A control object to define the mode of interval calculation for the computation of the average eoEnergyConsumed or eoEnergyProvided measurement
eoEnergyParametersIntervalWindow (6)	The length of the duration window between the starting time of one sliding window and the next starting time
eoEnergyParametersSampleRate(7)	The sampling rate, in milliseconds, at which the Energy Object should poll power usage in order to compute the average eoEnergyConsumed
eoEnergyParametersStorageType(8)	This variable indicates the storage type for this row
eoEnergyParametersStatus(9)	It is used to start or stop energy usage logging

Table 2.5: eoEnergyParametersTable. Reproduced from [32]

The eoEnergyParametersIntervalMode entry can have three values that define three ways for energy measurement collection:

- Period (fig.2.4): With this value the measurements are periodic and non-overlapped.

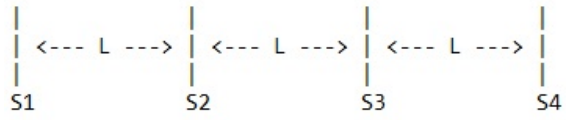

Figure 2.4. Periodic interval mode. Reproduced from [32]

Where eoEnergyParametersIntervalLength defines the value of L.

- Sliding windows (fig.2.5): This value indicates that the measurements are periodic and overlapped.

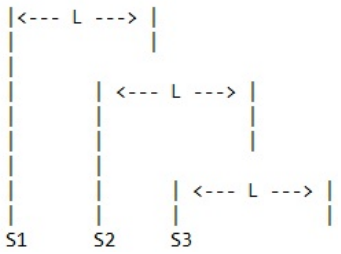

Figure 2.5. Sliding window interval mode. Reproduced from [32]

In this case S2 = S1 + eoEnergyParametersIntervalWindow.

- Total (fig.2.6): It specifies a continuous measurement since the last reset.

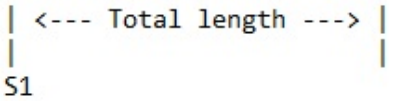

Figure 2.6. Total interval mode. Reproduced from [32]

The value of eoEnergyParametersIntervalNumber should be one and eoEnergyParametersIntervalLength is not considered.

The fifth table (tab.2.6) is the eoEnergyTable. It supplies a log of the energy and demand information.

Entity (OID)	**Description**
eoEnergyEntry(1) [eoEnergyParametersIndex, eoEnergyCollectionStartTime]	An entry describing energy measurements
eoEnergyCollectionStartTime (1)	This object specifies the start time of the energy measurement sample
eoEnergyConsumed(2)	The energy consumed in units of watt-hours for the Energy Object over the defined interval
eoEnergyProvided(3)	The energy produced in units of watt-hours for the Energy Object over the defined interval
eoEnergyStored(4)	The difference of the energy consumed and energy produced for an Energy Object in units of watt-hours for the Energy Object over the defined interval
eoEnergyUnitMultiplier(5)	The magnitude of watt-hours for the energy field in eoEnergyConsumed, eoEnergyProvided, eoEnergyStored, eoEnergyMaxConsumed, and eoEnergyMaxProduced
eoEnergyAccuracy(6)	This object indicates a percentage accuracy of Energy usage reporting
eoEnergyMaxConsumed(7)	This object is the maximum energy observed in eoEnergyConsumed since the monitoring started or was reinitialized
eoEnergyMaxProduced(8)	This object is the maximum energy ever observed in eoEnergyEnergyProduced since the monitoring started

eoEnergyDiscontinuityTime (9)	It specifies the time of the last interruption of total energy measurement

Table 2.6: eoEnergyTable. Reproduced from [32]

POWER-ATTRIBUTES-MIB: this module is composed of three tables.

The eoACPwrAttributesTable (tab.2.7) is the first table. It informs the manager about power quality available for each Energy Object.

Entity (OID)	**Description**
eoACPwrAttributesEntry(1) [entPhysicalIndex]	This is a sparse extension of the eoPowerTable with entries for power attributes measurements or configuration
eoACPwrAttributesConfiguration (1)	Configuration describes the physical configurations of the power supply lines: single phase, three-phase delta or, three-phase Y
eoACPwrAttributesAvgVoltage (2)	A measured value for average of the voltage measured over an integral number of AC cycles
eoACPwrAttributesAvgCurrent (3)	A measured value for average of the current measured over an integral number of AC cycles
eoACPwrAttributesFrequency (4)	A measured value for the basic frequency of the AC circuit
eoACPwrAttributesPowerUnitMultiplier (5)	The magnitude of watts for the usage value in eoACPwrAttributesTotalActivePower, eoACPwrAttributesTotalReactivePower, and eoACPwrAttributesTotalApparentPower measurements
eoACPwrAttributesPowerAccuracy (6)	This object indicates a percentage value representing the presumed accuracy of active, reactive, and apparent power usage reporting
eoACPwrAttributesTotalActivePower (7)	A measured value of the actual power delivered to or consumed by the load
eoACPwrAttributesTotalReactivePower (8)	A measured value of the reactive portion of the apparent power
eoACPwrAttributesTotalApparentPower (9)	A measured value of the voltage and current that determines the apparent power
eoACPwrAttributesTotalPowerFactor (10)	A measured value ratio of the real power flowing to the load versus the apparent power
eoACPwrAttributesThdCurrent (11)	A calculated value for the current total harmonic distortion
eoACPwrAttributesThdVoltage (12)	A calculated value for the voltage total harmonic distortion

Table 2.7: eoACPwrAttributesTable. Reproduced from [32]

The second table, eoACPwrAttributesDelPhaseTable (tab.2.8), is used to set up the parameters of energy and demand measurement collection.

Entity (OID)	Description
eoACPwrAttributesDelPhaseEntry(1) [entPhysicalIndex, eoACPwrAttributesDelPhaseIndex]	An entry describes power measurements of a phase in a DEL three-phase power
eoACPwrAttributesDelPhaseIndex (1)	A phase angle typically corresponding to 0, 120, 240
eoACPwrAttributesDelPhaseToNextPhaseVoltage (2)	A measured value of phase to next phase voltages
eoACPwrAttributesDelThdPhaseToNextPhaseVoltage (3)	A calculated value for the voltage total harmonic distortion for phase to next phase

Table 2.8: eoACPwrAttributesDelPhaseTable. Reproduced from [32]

The last table (tab.2.9) is the eoACPwrAttributesWyePhaseTable. It is used to indicate information and statistics about the supported Power States for each Energy Object.

Entity (OID)	Description
eoACPwrAttributesWyePhaseEntry(1) [entPhysicalIndex, eoACPwrAttributesWyePhaseIndex]	This table describes measurements of a phase in a WYE three-phase power system. Three entries are required for each supported entPhysicalIndex entry
eoACPwrAttributesWyePhaseIndex (1)	A phase angle typically corresponding to 0, 120, 240
eoACPwrAttributesWyePhaseToNeutralVoltage (2)	A measured value of phase to neutral voltage
eoACPwrAttributesWyeCurrent (3)	A measured value of phase currents
eoACPwrAttributesWyeActivePower (4)	A measured value of the actual power delivered to or consumed by the load
eoACPwrAttributesWyeReactivePower (5)	A measured value of the reactive portion of the apparent power
eoACPwrAttributesWyeApparentPower (6)	A measured value of the voltage and current determines the apparent power
eoACPwrAttributesWyePowerFactor (7)	A measured value ratio of the real power flowing to the load versus the apparent power for this phase
eoACPwrAttributesWyeThdCurrent (9)	A calculated value for the voltage total harmonic distortion for phase to phase
eoACPwrAttributesWyeThdPhaseToNeutralVoltage (10)	A calculated value of the voltage total harmonic distortion (THD) for phase to neutral

Table 2.9: eoACPwrAttributesWyePhaseTable. Reproduced from [32]

2.3.1.2 Energy Object Context MIB

The Energy Object Context MIB module is used to identify the Energy Objects and to understand the context in which they work and understanding the relationships between Energy Objects too. This MIB is composed of two tables: eoTable and eoRelationTable.

The first one, eoTable (tab.2.10), has as its main aim the connection of this MIB module with the other MIBs present in the agent. The eoTable is essential to identify and understand the context of the Energy Object.

Entity (OID)	**Description**
eoEntry(1) [entPhysicalIndex]	An entry describes the attributes of an Energy Object
eoEthPortIndex(1)	This variable uniquely identifies the power Ethernet port to which a Power over Ethernet device is connected
eoEthPortGrpIndex(2)	This variable uniquely identifies the group containing the port to which a power over Ethernet device PSE is connected
eoLldpPortNumber(3)	This variable uniquely identifies the port component as defined by the lldpLocPortNum in the LLDP-MIB and LLDP-MED-MIB
eoMgmtMacAddress(4)	It specifies a Media Access Control address of the Energy Object
eoMgmtAddressType(5)	It specifies the eoMgmtAddress type (IPv4 or IPv6)
eoMgmtAddress(6)	It specifies the management address as an IPv4 address or IPv6 address of Energy Object
eoMgmtDNSName(7)	This object specifies a DNS name of the eoMgmtAddress
eoDomainName(8)	It specifies the name of an Energy Management Domain for the Energy Object
eoRoleDescription(9)	It specifies an administratively assigned name to indicate the purpose an Energy Object serves in the network
eoKeywords(10)	It specifies a list of keywords that can be used to group Energy Objects for reporting or searching
eoImportance(11)	It specifies a ranking of how important the Energy Object is compared with other Energy Objects in the same Energy Management Domain
eoPowerCategory(12)	This object describes the Energy Object category, which indicates the expected behaviour or physical property of the Energy Object
eoAlternateKey(13)	It specifies an alternate key string that can be used to identify the Energy Object
eoPowerInterfaceType(14)	This object describes the Power Interface for an Energy Object

Table 2.10: eoTable. Reproduced from [33]

The eoRelationTable table (tab.2.11) explains in a simple way the relationships with other Energy Objects.

Entity (OID)	Description
eoRelationEntry(1) [entPhysicalIndex, eoRelationIndex]	An entry in this table specifies the Energy relationship between Energy objects
eoRelationIndex(1)	It is an arbitrary index to identify the Energy Object related to another Energy Object
eoRelationID(2)	This object specifies the Universally Unique Identifier (UUID) of the peer (other) Energy Object
eoRelationship(3)	This object describes the relations between Energy Objects
eoRelationStatus(4)	The status controls and reflects the creation and activation status of a row in this table to specify energy relationship between Energy Objects
eoRelationStorageType(5)	This variable indicates the storage type for this row

Table 2.11: eoRelationTable. Reproduced from [33]

2.4 Energy Consumption Models Bound to the Bit Rate

In this section, we are going to list the different models that we have found in the available literature. These studies bind the energy consumption of different network nodes and their bit rate.

Model 1 In the paper [23] the authors examine the consumption of energy in function of the used protocol and the dimensions of the L2 frames (64, 256, 512, 1518 bytes).
Analysing the tests results, it is found that the consumption of each interface $P_{int}(p, s, c)$ depends on the link utilization p, the size of the packets s and the overhead of the routing protocol c. The last term c can be ignored. So the energy consumption of each interface results to be:

$$\begin{aligned} P_{int}(p,s) &= P(\text{header processing}) + P(\text{packet transferring}) \\ &= E_{HP} \cdot (\text{packets per second}) + E_{PT} \cdot (\text{bits per second}) \\ &= \left(E_{HP} \cdot \frac{p \cdot R}{s}\right) + (E_{PT} \cdot p \cdot R) \\ &= p \cdot R \cdot \left(\frac{E_{HP}}{s} \cdot E_{PT}\right) \end{aligned} \tag{2.1}$$

Where, $E_{HP}[Joule]$ is the energy consumed for the packet header processing, $E_{PT}[Joule/bit]$ is the energy required to transfer a bit and R is the maximum capacity of the link.

Model 2 The authors of the paper [20], in order to study their analytical model, change the packets dimensions (100, 500, 1000, 1500 bytes) and the link utilization.

The network devices used for the test are an enterprise switch, an edge switch, a metro router and an edge router. Their resulting model is the following:

$$\begin{aligned} P &= P_{idle} + E_p \cdot R_{pkt} + E_{S\&F} \cdot R_{byte} \\ &= P_{idle} + E_{inc,byte} \cdot R_{byte} \end{aligned} \tag{2.2}$$

Where $E_{inc,byte}$ is the incremental energy per-byte, defined as:

$$E_{inc,byte} = \frac{E_p}{L} + E_{S\&F} \tag{2.3}$$

E_p is the per-packet processing energy consumption, $E_{S\&F}$ is the per-byte store and forward energy consumption and L is the packets dimension in bytes.

Clearly, in a network with generic traffic, we can't know the size of each received packet, so, we are using the expectation $E[L]$, and the previous formula becomes:

$$E[P] = P_{idle} + R_{byte} \cdot \left(\frac{E_p}{E[L]} + E_{S\&F} \right) \tag{2.4}$$

In order to evaluate $E[L]$ the authors use two packet length distributions: one uniform between [64-1518] bytes with $E[L]$ equal to 791 byte, and one trimodal in 64, 576, 1518 bytes with $E[L]$ equal to 944 byte. In both cases, we have an error of the estimated value respect to the measured one of 0.2%.

The result values that we can use in tests with generic traffic are listed in the table 2.12.

Device	P_{idle} **(W)**	E_p **nJ/pkt**	$E_{S\&F}$ **nJ/byte**
Enterprice Ethernet Switch	36.2	40	0.28
Edge Ethernet Switch	631	1571	9.4
Metro IP Router	352	1375	14.4
Edge IP Router	576	1707	10.2

Table 2.12: Summary of measurement results for the different devices. Reproduced from [20]

Model 3 The following paper [27] shows a test bench that can be used to compare the energy characteristics of the different network devices. In order to prove the validity of the test bench, the authors create an energy consumption model (2.5).

$$\begin{aligned} P_{switch} = P_{chessis} + N_{linecard} \cdot P_{linecard} + \\ \sum_{i=0}^{configs} Nport_{configs_i} \cdot P_{configs_i} \cdot utilizFactor \end{aligned} \tag{2.5}$$

$P_{linecard}$ represents the power consumed by the linecard when all ports are turned off, and $N_{linecard}$ is the real number of linecards presents within the switch. The variable $configs$ is the number of configurations for the port line rate. $P_{configs_i}$ is the power for a port running at maximum rate and $utiliznFactor$ is the scaling factor that represents the utilization of each port, which can be between 0 and 1.

During the tests the authors noted some interesting behaviours in common for edge switches, core and edge routers:

- the port capacity affects considerably energy consumption, particularly with a high number of ports;
- when the traffic throughput for each port is set, the packets dimension impacts on general consumption;
- the number of TCAM inputs doesn't affect general consumption;
- the firmware update of the devices reduces energy consumption.

Model 4 The authors of the paper [24] study the power consumption of two standard router to obtain a general analytical model that can be used to optimize the network consumption in a framework. The model obtained is the following:

$$PC(X) = CC(x_0) + \sum_{i=0}^{N}(TP(x_{i0}, x_{i1}) + LCC(x_{i1})) \tag{2.6}$$

Where the energy consumption PC depends on vector X that defines the type of chassis, the installed linecards, the configuration and the traffic profile of the device. Indeed the function $CC(x)$ indicates the energy consumption of a particular type of chassis, N is the number of active linecards, $TP(x)$ is a scalar parameter that represents the router traffic utilization and, at the end, $LCC(x)$ corresponds to the energy cost of linecard turned on in a specific configuration.

Model 5 In this paper [22] the authors evaluate an energy model with two type of OpenFlow switches, a hardware switch (NEC PF 5240) and a software switch (Open vSwitch running on a server). The general model used is shown below:

$$P_{switch} = P_{base} + P_{config} + P_{protocol} + P_{OF} \tag{2.7}$$

P_{base} is the energy consumption of the switch with no traffic load and P_{config} is obtained in this way:

$$P_{config} = \sum_{i}^{N_{active\_Ports}} s_i \cdot P_{port} \tag{2.8}$$

Where P_{port} is the power consumption of the port at full load and s_i is the relative utilization of the port's configured speed, which can be between 0 and 1.

$P_{control}$ (2.7) is the energy consumption caused by the control traffic. It is equal to:

$$P_{control} = r_{Packet\_In} \cdot E_{Packet\_In} + r_{Flow\_Mod} \cdot E_{Flow\_Mod} \tag{2.9}$$

r_{Packet_In} represents the rate of outgoing $Packet\_In$ messages, E_{Packet_In} their energy consumption, whereas r_{Flow_Mod} is the rate of $Flow\_Mod$ packets.

The energy consumption P_{OF}, produced from the OpenFlow traffic is calculated as follows:

$$P_{OF} = \sum_{i}^{N_{flows}} r_{packetts}(i) \cdot \left[\sum_{j}^{N_{matches}} \mu_{match}(i,j) \cdot e_{match}(j) + \sum_{k}^{N_{actions}} \mu_{action}(i,k) \cdot e_{action}(k) \right] \tag{2.10}$$

N_{flows} is the number of active flows and $r_{packet}(i)$ is the corresponding packet rate. For each packet processed there are a number of matches and possible actions. The binary matrix $\mu_{match}(i,j)$ represents the active matches and $e_{match}(j)$ is the energy consumed by a single match j. In the same way $\mu_{action}(k,i)$ is the binary matrix of the possible actions and $e_{action}(k)$ is the energy consumption caused by action k.

After the tests the authors simplify the P_{OF} equation, which becomes:

$$P_{OF} = \sum_{i}^{N_{flows}} r_{packetts}(i) \cdot \left(\max_{j}[\mu_{match}(i,j) \cdot e_{match}(j)] + \max_{k}[\mu_{action}(i,k) \cdot e_{action}(k)] \right) \tag{2.11}$$

With these approximations, there is an error less than 1% for the hardware switch and lower than 8% for the software switch respect to the measurements done.

Model 6 In this paper [26] the authors study a way to reduce the latency and the energy consumption of the OpenFlow switches, due to the lookup of the TCAM, using per-port packet prediction circuitry. They studied and compared two per-packet consumption models, one without prediction (2.12) and one with prediction (2.17).

Model without prediction. The power consumption model of the switch without prediction is:

$$E_{pkt} = E_{rx} + E_{lookup} + E_{xref} + E_{tx} \tag{2.12}$$

E_{rx} indicates the energy necessary for: the reception of the packet from the link, the extraction and elaboration of the fields necessary for the construction of a flow-key and saving the packet inside the input memory. E_{rx} is obtained in the following way:

$$E_{rx} = pkt\_size \cdot (E_{MAC} + E_{buf\_wr}) \tag{2.13}$$

E_{MAC} is the per-bit energy required for MAC and Serializer/Deserializer (SerDes), while E_{buf_wr} is the energy needed to write the packet in memory.

E_{lookup}, in 2.12, represents the energy consumption needed to search the forwarding instruction of the packet inside the TCAM, which is equal to:

$$\begin{aligned} E_{lookup} =& E_{TCAM\_search} + P_{in} \cdot E_{data\_rd} + \\ & (1 - P_{in}) \cdot (E_{TCAM\_wr} + E_{data\_wr}) \end{aligned} \tag{2.14}$$

P_{in} is the probability to find the flow-key in the TCAM. E_{TCAM_search} and E_{TCAM_wr} are the energy necessary to search and update the TCAM. E_{data_rd} and E_{data_wr} are the energy consumed to read and write associated TCAM flow memory and forwarding instructions.

E_{xfer}, in 2.12, indicates the energy required to read the packet from the inbound memory, all the steps needed to initiate a transfer of the packet across the switch fabric, and write the packet into the outbound memory. E_{xref} is equal to:

$$E_{xfer} = pkt\_size \cdot (E_{buf\_rd} + E_{fab} + E_{buf\_wr}) \tag{2.15}$$

E_{buf_rd} and E_{buf_wr} represent the per-bit energy to write and read the buffer. E_{fab} is the energy consumed by the crossbar fabric chip.

E_{tx}, in 2.12, is the energy required to read the packet from the outbound buffer and send it on the link. It is equal to:

$$E_{tx} = pkt\_size \cdot (E_{MAC} + E_{buf\_rd}) \tag{2.16}$$

Model with prediction. The power consumption model of the switch with prediction is:

$$E_{pkt} = E_{rx} + E_{predict} + E_{xref} + E_{tx} \tag{2.17}$$

$E_{predict}$ represents the energy consumed by the prediction phase. It is given by the following formula:

$$\begin{aligned} E_{predict} =& P_{hit} \cdot E_{cache\_hit} + P_{incorrect} \cdot E_{cache\_incorrect} + \\ & P_{miss} \cdot E_{cache\_miss} \end{aligned} \tag{2.18}$$

P_{hit}, $P_{incorrect}$ and P_{miss} are the probability that an entry in prediction cache is correctly found, incorrectly found or missing. Indeed E_{cache_hit}, $E_{cache\_incorrect}$ and E_{cache_miss} are the energy necessary to search and maintain the prediction cache when one of these events occur. They are calculated as follows:

$$E_{cache\_hit} = E[S] \cdot E_{cache\_search} + E_{cache\_rd} \tag{2.19}$$

$$\begin{aligned} E_{cache\_incorrect} =& E[S] \cdot E_{cache\_search} + E_{cache\_rd} + E_{xfer\_min} + \\ & E_{tx\_min} + E_{lookup} + E_{cache\_wr} \end{aligned} \tag{2.20}$$

$$E_{cache\_miss} = E[S] \cdot E_{cache\_search} + E_{lookup} + E_{cache\_wr} \tag{2.21}$$

$E[S]$ is the per-packet expected number of searches in the prediction cache. E_{cache_search} is the energy needed to search the signature CAM. E_{cache_rd} is the energy necessary to read the associated flow-key and forwarding RAM. E_{xfer_min} is the energy required to transfer the beginning of the packet across the switch fabric and into the output memory. E_{tx_min} is the energy required to transmit an initial fragment of the packet on the output link before being aborted. E_{cache_wr} is the energy needed to update the prediction cache.

Model 7 Its distinctive feature is that the energy consumption of the switch is bound to the wake-up and sleep time of the device [25].

$$P = P_{base} + P_{dynamic} \sum_{i=1}^{Nport} min(1, D \cdot p_i) \tag{2.22}$$

P_{base} is the energy consumption when there is no traffic. $P_{dynamic}$ is the difference between P_{base} and the energy consumption of the switch at maximum load divided by the number of ports. D is the increment of energy consumption in function of the increment of the traffic per-port p_i. It's defined as:

$$D = \sum_{L=L_{min}}^{L_{max}} \left(p(L) \cdot \frac{T_s + T_w + \frac{L}{R}}{\frac{L}{R}} \right) \tag{2.23}$$

L is the packet size in bits, $p(L)$ is the probability that the packet size is L and R is the link speed. T_w is the time necessary to wake up the port and T_s is the time required to put the port in Low Power Idle mode (LPI).

Chapter 3

Software Implementation and Simulation

In this chapter we will introduce the software needed for our simulations, then we will explain how to install and set them in order to function correctly. Later, we will show: the virtual network on which we performed our tests and how to implement it and then our algorithm necessary to collect the devices' incoming bytes and, from them, calculate the energy consumption of our virtual network devices.

3.1 Simulation Environment

In order to create the simulation environment we need the following software:

- VirtualBox [18] is a software application that allows creating a virtual environment in which it is possible to install, as a guest, additional operating systems;
- Mininet [7] is an SDN emulator. It allows to develop and simulate computer networks and distributed systems inside your laptop in order to test your own applications and systems;
- Net-SNMP [9] is a software set for using and deploying the SNMP protocol. It provides us different applications as shown in the tab. 3.1.

Application	**Description**
encode keychange	produce the KeyChange string for SNMPv3
snmptranslate	translate MIB OID names between numeric and textual forms
snmpget	communicates with a network entity using SNMP GET requests
snmpgetnext	communicates with a network entity using SNMP GETNEXT requests
snmpbulkget	communicates with a network entity using SNMP GETBULK requests

snmpwalk	retrieve a subtree of management values using SNMP GETNEXT requests
snmpbulkwalk	retrieve a subtree of management values using SNMP GETBULK requests
snmpset	communicates with a network entity using SNMP SET requests
snmptrap	sends SNMP TRAP or INFORM notification messages
snmpd	a SNMP agent that responds to SNMP requests for a given host
snmptrapd	a SNMP daemon that listens for SNMP TRAPs or INFORMs and logs or acts upon them
snmptest	communicates with a network entity using SNMP requests
mib2c	a MIB conversion utility that can translate MIB structures into other forms, such as C-code
tkmib	a perl/Tk interactive graphical MIB browser for SNMP

Table 3.1: SNMP applications included with Net-SNMP. Reproduced from:"en. wikipedia.org/wiki/Net-SNMP"

With VirtualBox is possible to create a virtual machine and install on it a Linux operating system (in this case we used Ubuntu 14.04 LTS 32 bit).

Then we run the virtual machine with Ubuntu and we update, the OS, writing by the terminal:

```
sudo apt-get update
```

Then, we install mininet executing this command line:

```
sudo apt-get install mininet
```

or these:

```
git clone git://github.com/mininet/mininet
cd mininet/util
sudo ./install.sh -nfv
```

Where the options used above are needed to install the following:

- Mininet dependencies and core files (n);
- OpenFlow (f);
- Open vSwitch (v);

The next step is to install the SNMP agent, that allows us to know different statistics of the mininet's virtual devices. To install it we execute these commands:

```
sudo apt-get install snmp
```

```
sudo apt-get install snmpd
```

We have to configure the agent, in order to do that, we enter in the `/etc/snmp` directory and rename the existing snmpd.conf file (eg. snmpd.conf.org), then, create a new snmpd.conf file in which we write:

```
rocomunity public
syslocation "pc"
syscontact s208290@studenti.polito.it
```

These rows define the system information. Up to now it is necessary to define only these information for our simulations but with the snmpd.conf file [6] we can modify: the behaviour of the agent, the authentication parameters, the access control, etc.

In order that the agent uses the snmpd.conf file and collects the information of all the devices, we have to modify `/etc/default/snmpd` file as follows:

from

```
# snmpd options (use syslog, close stdin/out/err)
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 127.0.0.1'
```

to

```
# snmpd options (use syslog, close stdin/out/err)
#SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid 127.0.0.1'
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid -c
          /etc/snmp/snmpd.conf'
```

At the end, we restart the agent using this command:

```
/etc/init.d/snmpd restart
```

Now we test the agent to verify if it works, we send an SNMP request through the terminal (eg. `snmpwalk -v 2c -c public 127.0.0.1 -Oe`) and if all the previous steps are performed correctly the answer is shown below.

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux a-Virtualbox 3.19.0-25-generic...
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks:  (864) 0:00:08.69
SNMPv2-MIB::sysContact.0 = STRING: s208290@studenti.polito.it
SNMPv2-MIB::sysName.0 = STRING: a-VirtualBox
SNMPv2-MIB::sysLocation.0 = STRING: "pc"
SNMPv2-MIB::sysORLastChange.0 = Timeticks:  (2) 0:00:00.02
SNMPv2-MIB::sysORID.1 = OID: SNMP-MPD-MIB::snmpMPDCompliance
SNMPv2-MIB::sysORID.2 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORID.3 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.5 = OID: TCP-MIB::tcpMIB
```

```
SNMPv2-MIB::sysORID.6 = OID: IP-MIB::ip
SNMPv2-MIB::sysORID.7 = OID: UDP-MIB::udpMIB
```

The last tool needed is a python library, provided by net-SNMP, which allows interacting with our SNMP agent [10]. This library is useful because it returns the agent's information in form of lists that are more easy to manage. In this way, we can easily create an algorithm that sends periodic requests to the agent and elaborates all its answers. The library is installed by using this command line:

```
sudo apt-get install python-netsnmp
```

3.2 Creation of SNMP Network and Energy Application

3.2.1 SNMP Network with Mininet

Mininet provides python API in order to create any network [8]. For our simulations, we create the wired network in fig. 3.1.

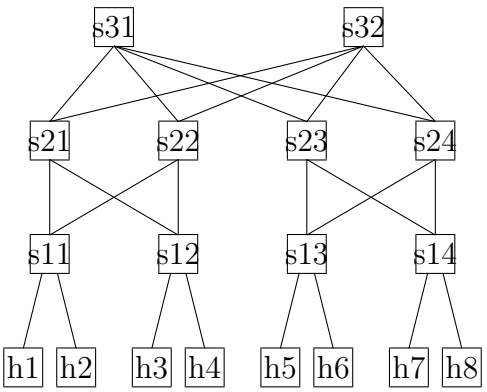

Figure 3.1. Fat-tree Network implemented.

It is composed of eight hosts, ten switches and a controller.

Mininet provides different controllers [1]. Two of them are implemented and start when the mininet network starts. The other two are remote and run independently from mininet. Besides we can use any remote controller, for example, Onos or OpenDaylight.

The implemented controllers are OpenFlow controller and Open vSwitch controller. The second one cannot control more than 16 switches.

The remote controllers are POX controller and Ryu controller. They can be installed when we install mininet, adding to the command `./install.sh` the options:

`-p` to install the POX controller [13];
`-y` to install the Ryu controller [15].

They are both open source and python-based controllers.

After the instructions to create and start the network, we add:

1. `ovs-vsctl set Bridge s11 stp_enable = true`

   This command enables spanning tree algorithm and it must be executed by all switches. This command is mandatory in complex networks managed by OpenFlow or Open vSwitch controller, in order to avoid loops.

2. `s11.cmd('/usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid -c /etc/snmp/snmpd.conf')`

   By using this command the agent starts to collect information about the switch.

There are three ways to ask information about the switch (eg. 1.3.6.1.2.1.2.2.1.2 is ifDesc object):

- by ubuntu terminal:

  ```
  snmpwalk -v 2c -c public IPswitch 1.3.6.1.2.1.2.2.1.2
  ```

- in the python script:

  ```
  c0.cmd('snmpwalk -v 2c -c public IPswitch 1.3.6.1.2.1.2.2.1.2')
  ```

  with this instruction the controller c0 sends an SNMP request;

- from mininet command-line interface (CLI):

  ```
  sh snmpwalk -v 2c -c public IPswitch 1.3.6.1.2.1.2.2.1.2
  c0 snmpwalk -v 2c -c public IPswitch 1.3.6.1.2.1.2.2.1.2
  ```

  the first string results being the same as the first way of asking information (ubuntu terminal), the second string results being the same as the second way of asking information (python script).

In all the cases mentioned above the answer is in fig. 3.2.

```
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifDescr.3 = STRING: s11-eth1
IF-MIB::ifDescr.4 = STRING: s11-eth2
IF-MIB::ifDescr.5 = STRING: s12-eth1
IF-MIB::ifDescr.6 = STRING: s12-eth2
IF-MIB::ifDescr.7 = STRING: s11-eth3
IF-MIB::ifDescr.8 = STRING: s12-eth3
```

Figure 3.2. Snmpwalk answer

3.2.2 Energy Application

Our aim is to calculate the energy consumption of every switch in the network, in order to obtain it, we use SNMP that counts the number of incoming bytes from every interface (ifHCInOctets). In this way, we can calculate the incoming rate of the switch. By using an energy model 2.4 we are able to evaluate the energy consumption of each network device.

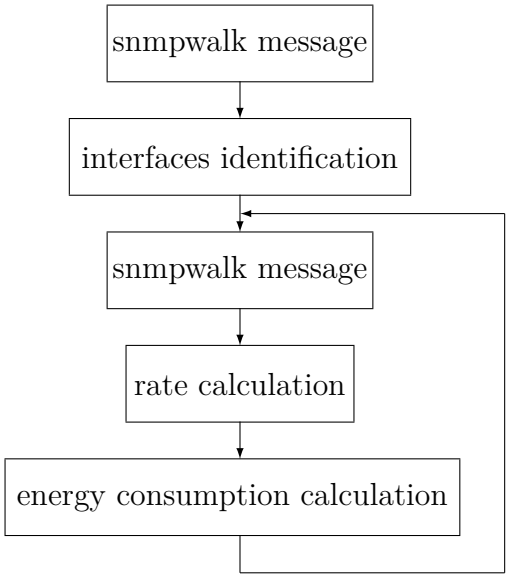

Figure 3.3. Application diagram.

At the beginning of our script we open a session:

```
stringsession = netsnmp.Session(DestHost='127.0.0.1', Version=2,
                Community='public')
```

As we can see, in the string is defined: the IP address of the receiver, the SNMP version used and the community (defined in snmpd.conf). After that, we send a snmpwalk request to the agent with object ifDescr, in this way we can identify all the interfaces of the switches and their position in the answer's list (fig. 3.4).

```
for i in range(0, l):
   if resp[l] == 's11-eth1':
      interfs11[0] = l
   elif resp[l] == 's11-eth2':
      interfs11[1] = l
   elif resp[l] == 's11-eth3':
      interfs11[2] = l
   elif resp[l] == 's11-eth4':
      interfs11[3] = l
   elif resp[l] == 's12-eth1':
      interfs12[0] = l
```

Figure 3.4. Interface identification

We have to do this because, as we can see in fig. 3.2, there are other interfaces that are useless (eg. `lo, eth0`).

Now, following the diagram in fig. 3.3, we ask the agent the number of bytes received by all interfaces. Therefore we create an infinite cycle in which every 20 seconds the application sends a snmpwalk request to the agent and calculates the ingress rate (3.1) and energy consumption (3.2).

We calculate the ingress rate in $[bit/s]$ of each switch as shown in equation 3.1.

$$R_{s11} = \sum_{i}^{N_{interf}} \frac{\Delta ifHCInOctets_i \cdot 8}{\Delta t} \tag{3.1}$$

In order to verify our results, we use a traffic simulator (iperf). Iperf simulates a client/server communication and calculates the bandwidth. In order to use iperf, we have to declare a host (e.g. h1) as UDP server with this command:

```
h1 iperf -s -u &
```

and another host (e.g. h4) as a client with:

```
h4 iperf -c h1 -t 60 -b 600m
```

Where, `-t` is used to define the duration of the test in seconds and `-b` is used to set the bandwidth of UDP traffic, in this example 600 Mbit/sec.

As we can see in fig. 3.5 and in the last column of the fig. 3.6 our program gives us acceptable results considering the fact that it has different measurement intervals compared with iperf.

```
------------------------------------------------------------
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams
UDP wondow size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.4 port 44062 connected with 10.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec  4.23 GBytes   606 Mbits/sec
[  3] Sent 3092006 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec  4.23 GBytes   605 Mbits/sec   0.003 ms 5043/3092005 (0.16%)
[  3]  0.0-60.0 sec  183 datagrams received out-of-order
```

Figure 3.5. Bandwidth of iperf.

```
rate of s11 interfaces [0, 0, 0, 0]
rate of s11 interfaces [22, 0, 37, 508308266]
rate of s11 interfaces [22, 0, 288, 624086546]
rate of s11 interfaces [0, 0, 0, 623585282]
rate of s11 interfaces [22, 0, 0, 624099787]
```

Figure 3.6. Rate of our program.

The program is written ad-hoc for the network shown above (fig. 3.1), so we can make some considerations that cannot be used in a real network. For example, by knowing the name of interfaces we can easily identify them (fig. 3.4). Another approximation is the interval of time that we use to calculate the rate, in a real network we have to consider the RTT time.

To evaluate the energy consumption of the switches from s11 to s14 we use the model 2.4:

$$E[P] = P_{idle} + R_{bit} \cdot \left(\frac{E_p}{E[L]} + E_{S\&F} \right) \Big/ 8 \tag{3.2}$$

with $P_{idle} = 631\ W$, $E_p = 1571\ nJ/pkt$ and $E_{S\&F} = 9.4\ nJ/byte$, which are the parameters' values of the edge ethernet switch, as we can see in tab. 2.12.

The authors of the paper [20] measure the power consumption with different packet dimensions (100, 500, 1000, 1500 bytes) to develop the energy model of the

edge ethernet switch, as we can see in the figure 3.7.

In order to verify our results, we run different tests with different packets sizes (100, 500, 1000, 1500 bytes). We do the tests using iperf with the option -M in the client instruction, as shown below.

```
h4 iperf -c h1 -t 60 -M 500
```

This option allows setting TCP maximum segment size (MTU - 40 bytes), in this example 500 bytes.

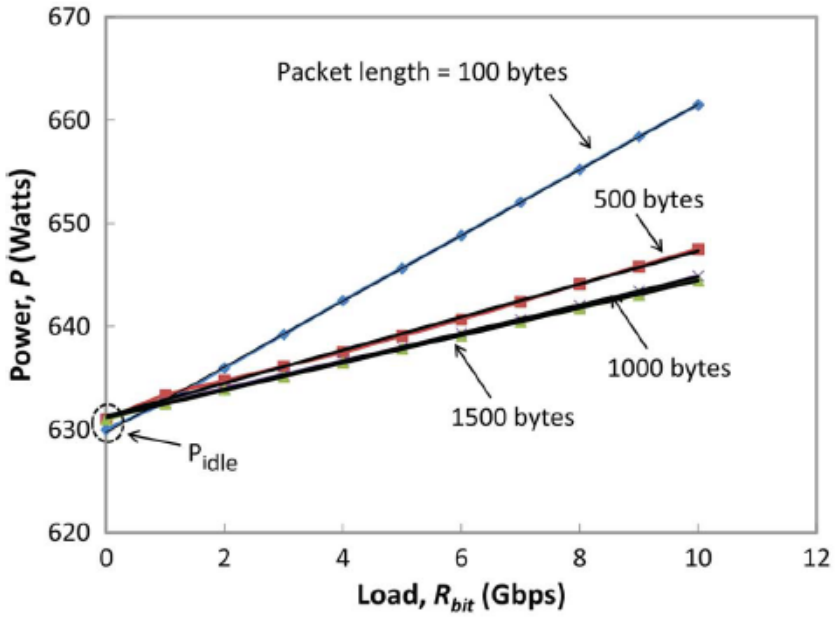

Figure 3.7. Energy consumption of edge ethernet switch. Reproduced from [20]

Results of our tests are shown in figure 3.8. Comparing the two graphs we can see that in both cases when the packets size increases the straight lines' slope decrease. In particular, when traffic's packets size is 100 bytes, we can see that the switch's energy consumption increases quickly up to exceed 660 watts, with a bit rate of 10Gbps. Instead, with the same bit rate, when the traffic's packets size is 500, 1000 or 1500 bytes the energy consumption of the switch is around 645 watts.

Tab. 3.2 reports the energy consumptions of the edge switch for a bit rate of 10Gbps. In the first column there are the results obtained from the paper [20], and in the second column there, are the results of our simulations.

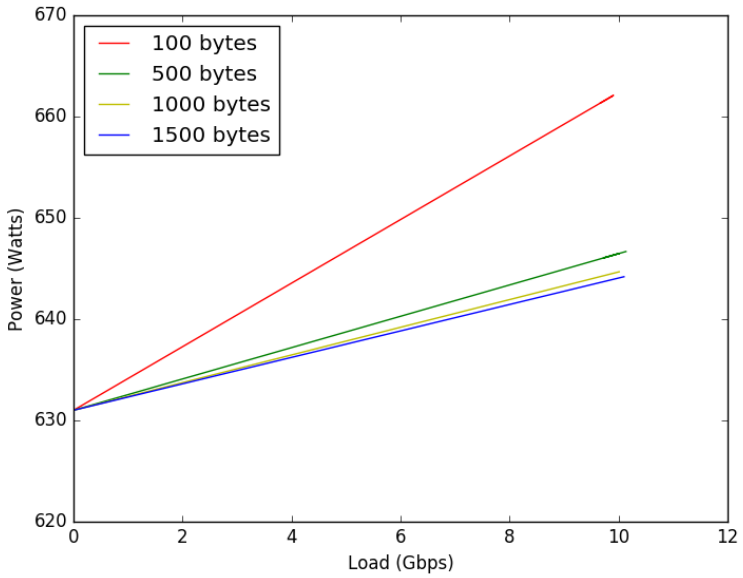

Figure 3.8. Energy consumption of edge ethernet switch from our tests.

packets dimensions [Bytes]	**paper's results [Watts]**	**tests' results [Watts]**
100	661.9	662.38
500	646.8	646.68
1000	644.5	644.71
1500	644.2	644.06

Table 3.2: Energy consumption of the edge switch for a bit rate of 10Gbps

The differences between our tests' results and the energy consumptions measured in [20] are less than 0.5 watts, which means that we have an error less than 0.1%, so our results can be considered acceptable.

Chapter 4

Conclusions

In this study, we tried to present and simulate an algorithm that, through a southbound API, provides to the SDN controller the energy consumption data of each managed network device. To achieve this goal we started by analysing the state of the art of the SDN network then we researched frameworks available for measuring/estimating the energy consumption of the different network devices.

At the end of the research, we decided to use for our simulations, the SNMP protocol as the southbound API for the collection of the devices' incoming bytes. From these, we obtained the incoming rate and, finally, we were able to estimate the energy consumption of the device through an analytical model [20] that binds the incoming rate and its energy consumption.

In our opinion, the method considered in this study has points in favour and, at the same time, limitations that can be summarized, respectively, in the easy integration in the current controllers, since they already measure the rate of the different nodes (however to reach a good estimate of energy consumption it is essential to accurately measure the rate of devices and to have or create an accurate model), but the fact that the SDN controller should have a database with all analytical models of each controlled device, can be a limitation, this because at the same rate, each device has a different energy consumption that depends not only by the model but also by the producer.

From our study carried out, in conclusion, we can assert that this method is a good alternative until the devices themselves become able to measure their energy consumption and provide it to southbound protocols.

We could, in future, consider extending the use of this method for the estimation of the energy consumption regarding wireless devices and integrate it with remote controllers such as ONOS and OpenDayLight.

Appendix A

Simulation Algorithms

A.1 Network Creation Algorithm

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.link import Intf
from mininet.log import setLogLevel, info
import os

#in this function we creat the network
def simulateNetwork():

net = Mininet( topo=None )

info( '*** Adding controller\n' )
c0 = net.addController(name='c0')

info( '*** Add switches\n')
s11 = net.addSwitch('s11')
s12 = net.addSwitch('s12')
s13 = net.addSwitch('s13')
s14 = net.addSwitch('s14')

s21 = net.addSwitch('s21')
s22 = net.addSwitch('s22')
s23 = net.addSwitch('s23')
s24 = net.addSwitch('s24')

s31 = net.addSwitch('s31')
s32 = net.addSwitch('s32')
```

```
info( '*** Add hosts\n')
h1 = net.addHost('h1')
h2 = net.addHost('h2')
h3 = net.addHost('h3')
h4 = net.addHost('h4')
h5 = net.addHost('h5')
h6 = net.addHost('h6')
h7 = net.addHost('h7')
h8 = net.addHost('h8')

# we report only an example for each network level
info( '*** Add links\n')

# link between host and first level swithes
net.addLink(h1, s11)
net.addLink(h2, s11)

# link between first and second level switches
net.addLink(s21, s11)
net.addLink(s22, s11)
net.addLink(s21, s12)
net.addLink(s22, s12)

# link between second and third level switches
net.addLink(s31, s21)
net.addLink(s31, s23)
net.addLink(s31, s22)
net.addLink(s31, s24)

info( '*** Starting network\n')
net.start()

enableSTP()

info( '*** Starting SNMP agent in h1\n')
s11.cmd('/usr/sbin/snmpd -Lsd -Lf /dev/null -u snmp -I -smux
         -p /var/run/snmpd.pid -c /etc/snmp/snmpd.conf')

CLI(net)

net.stop()

# in this function we enable the spanning tree algorithm for all switches
def enableSTP():
```

```
for x in range(1,5):
    cmd = "ovs-vsctl set Bridge %s stp_enable=true" % ("s1" + str(x))
    os.system(cmd)
    print cmd
    cmd = "ovs-vsctl set Bridge %s stp_enable=true" % ("s2" + str(x))
    os.system(cmd)
    print cmd

for x in range(1, 3):
    cmd = "ovs-vsctl set Bridge %s stp_enable=true" % ("s3" + str(x))
    os.system(cmd)
    print cmd

#with this function we start the application
if __name__ == '__main__':
    setLogLevel( 'info' )
    simulateNetwork()
```

A.2 Energy Consumption Algorithm

```
#!/usr/bin/python

import netsnmp
import time
import string
import matplotlib.pyplot as plt

session = netsnmp.Session(DestHost='127.0.0.1', Version=2, Community='public')

#set the IOD into SNMP request
interfaces = netsnmp.VarList( netsnmp.Varbind('.1.3.6.1.2.1.2.2.1.2',))
#send the SNMPWALK request and save the answer list in resp
resp = session.walk(interfaces)

l = len(resp)

interfs11 = [0, 0, 0, 0]
# we save the interface position of s11 switch in the answer list
for i in range(0, l):
    if resp[i] == 's11-eth1':
        interfs11[0] = i
    elif resp[i] == 's11-eth2':
        interfs11[1] = i
    elif resp[i] == 's11-eth3':
```

```
        interfs11[2] = i
    elif resp[i] == 's11-eth4':
        interfs11[3] = i

tlinks11 = [0, 0, 0, 0]

rates11 = [0, 0, 0, 0]

t1 = 0
a = 0
Pidle = 631
Ep = 1571*10**-9
Esf = 9.4*10**-9
pktDim=1500

#start the cycle to collect incoming byte of each interface and calculate
#the incoming rate and energy consumption
while a<5 :

    Trates11 = 0

    Traffic = netsnmp.VarList(netsnmp.Varbind('.1.3.6.1.2.1.31.1.1.1.6',))

    res = session.walk(Traffic)

    t2 = int(time.time())

    for i in range(0, 4):

        if(a>0):
            #in this equation we calculate the incoming rate [bits/s]
            of each interface
            rates11[i] = ( (int(res[interfs11[i]])*8) - tlinks11[i]) / (t2 - t1)

            #here we calculate the total incoming rate of switch s11
            Trates11 = Trates11 + (rates11[i])

        tlinks11[i] = int(res[interfs11[i]])*8

        if (tlinks11[i]<0):
            print('input error')
            break

    #this equation calculate the energy consumption of switch s11
    P11 = Pidle + Trates11* (((Ep/ pktDim )+Esf)/8)
```

```
print (Trates11)
print (P11)

t1=t2

time.sleep(15)
a= a + 1
```

Bibliography

[1] Automating controller startup. "http://mininet.org/blog/2013/06/03/automating-controller-startup/".

[2] Flask. "http://flask.pocoo.org/docs/0.10/".

[3] Ganglia. "http://ganglia.info/".

[4] The hdf goup. "https://www.hdfgroup.org/".

[5] Kwapi-g5k framework. "http://kwapi-g5k.readthedocs.org/en/latest/architecture.html".

[6] Manpage of snmpd.conf net-snmp. "http://www.net-snmp.org/docs/man/snmpd.conf.html".

[7] Mininet. "http://mininet.org/".

[8] Mininet python api reference manual. "http://mininet.org/api/".

[9] Net-snmp. "http://www.net-snmp.org/".

[10] Net-snmp library. "https://github.com/haad/net-snmp/tree/master/python".

[11] Openflow switch specification. "https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf".

[12] Power state management profile. "http://www.dmtf.org/sites/default/files/standards/documents/DSP1027_2.0.0.pdf".

[13] Pox wiki. "https://github.com/noxrepo/pox".

[14] Rrdtool. "http://oss.oetiker.ch/rrdtool/index.en.html".

[15] Ryu sdn framework. "http://osrg.github.io/ryu/".

[16] sflow. "http://sflow.org/about/index.php".

[17] Standard for user interface elements in power control of electronic devices employed in office/consumer environments.

[18] Virtualbox. "https://www.virtualbox.org/".

[19] Zeromq. "http://zeromq.org/".

[20] Robert W. A. Ayre Arun Vishwanath, Kerry Hinton and Rodney S.Tucker. Modeling energy consumption in high-capacity routers and switches.

[21] IEEE Fernando M. V. Ramos Member IEEE Paulo Verissimo Fellow-IEEE Christian Esteve Rothenberg Member IEEE Siamak Azodolmolky Senior Member IEEE Diego Kreutz, Member and IEEE Steve Uhlig, Member. Software-defined networking: A comprehensive survey.

[22] David Hausheer Fabian Kaup, Sergej Melnikowitsch. Measuring and modeling the power consumption of openflow switches.

[23] Hong-Shik PARK Jaewon AHN. Measurement and modeling the power consumption of router interface.

[24] Paul Barford Cristian Estan David Tsiang Steve Wright Joseph Chabarek, Joel Sommers. "power awareness in network design and routing".

[25] Z. Zhao J. A. Maestro A. Vishwanath A. Sanchez-Macian P. Reviriego, V. Sivaraman and C. Russell. An energy consumption model for energy efficient ethernet switches.

[26] Matthew Farrens Paul T. Congdon, Prasant Mohapatra and Venkatesh Akella. Simultaneously reducing latency and power consumption in openflow switches.

[27] Sujata Banerjee Priya Mahadevan, Puneet Sharma and Parthasarathy Ranganathan. A power benchmarking framework for network devices.

[28] Franco Davoli Lorenzo Di Gregorio Pasquale Donadio Leonardo Fialho-Martin Collier Alfio Lombardo Diego Reforgiato Recupero Tivadar Szemethy Raffaele Bolla, Roberto Bruschi. The green abstraction layer: A standard power management interface for next-generation network devices.

[29] RFC-3411. An architecture for describing simple network management protocol (snmp) management frameworks. "http://www.ietf.org/rfc/rfc3411.txt".

[30] RFC-3954. Cisco systems netflow services export version 9. "http://www.ietf.org/rfc/rfc3954.txt".

[31] RFC-7326. Energy management framework. "http://www.ietf.org/rfc/rfc7326.txt".

[32] RFC-7460. Monitoring and control mib for power and energy. "http://www.ietf.org/rfc/rfc7460.txt".

[33] RFC-7461. Energy object context mib. "http://www.ietf.org/rfc/rfc7461.txt".

[34] David Schnaufer. Energy efficiency in the telecommunications network. "https://www.rfglobalnet.com/doc/energy-efficiency-in-the-telecommunications-network-0001".