

**University
of Debrecen**

**Faculty of
Informatics**

**Department of Informatics
Systems and Networks**

Analysis of virtual core network technics in Internet of Things



Supervisor

Zoltan Gal, PhD

Associate Professor

Student

Mohanad Al-Zaidi

Computer Science Engineering, MSc

Debrecen, 2023

I agree to submit it for review

Bírálatra való beküldéséhez hozzájárulok

Outline

1. Introduction and personal motivation	2
2. An introduction to the Internet of Things	2
2.1. Architecture of the IoT	3
2.2. Virtualization techniques in IoT	8
2.3. Intelligence of the future era of IoT	14
3. Analysis of the virtualization effect on IoT systems	19
3.1. Presentation of the measurement environment and of the methodology applied	20
3.2. Extraction of status data series and evaluation of the virtualization effects.....	21
3.3. Description of the developed analysis scripts to analyse the traffic.....	29
3.4. Interpretation of the phenomenon.....	45
4. Conclusions and applicability of the results in practice	48
List of Figures	50
List of Tables	52
Acknowledgement	53
References	54

1. Introduction and personal motivation

In the upcoming years, more gadgets and items are anticipated to be connected to the internet due to the rapid expansion in the IT sector. The Internet of Objects (IoT) must be implemented effectively and smoothly due to the growing global population, personal desires to connect more things to the internet, and demands to link anything, anywhere, at any time. Due to the enormous volume of data generated, extra network access and core equipment had to be installed. There are many technical obstacles in the way of such vast IoT systems being integrated and operating without a hitch. Virtualization can reduce the complexity of individual solutions, even though it can be difficult to build a single solution to handle all of the problems.

Virtualization is a software technology that logically abstracts the network's underlying hardware components. It separates packet forwarding and decision-making. Control became independent of the hardware, which made it simple to maintain, upgrade, and provide consumers with more effective end-to-end data transmission services.

Creating a lab environment for simulation and testing the result is an essential thing to do before implementing a technology in the real world. This fact is the essential motivation for me to do this research study. In this study, I have set up a lab setting to look at a model of a few network devices and IoT end-user equipment. These devices are connected to Software Defined Network (SDN) virtualization technology, which uses the Open Flow protocol to manage network devices.

The project included several versions of data analysis and each one produced many output graphs. As the modeling purpose is to simplify the real environment or to obtain samples and examine the behaviors of the system, the analysis conducted in this project can be used on a large scale in the real environment and developed further for different purposes.

2. An introduction to the Internet of Things

The Internet of Things (IoT) is a sort of network that links everything to the Internet through information sensing equipment in order to perform information exchange and communications

for smart recognition, locating, tracking, monitoring, and administration. The Internet of Things (IoT) has evolved into a network of equipment of all shapes and sizes, including automobiles, smartphones, home appliances, toys, cameras, medical machines, industrial structures, animals, humans, and constructions, all of them are linked and communicate and share information according to established protocols. The purpose of the Internet of Things is to enable connection between things and anything and anybody at anytime, anywhere, and via the use of any service. There is a diverse range of communication technologies available. Despite the variety of IoT, I will briefly examine common IoT architecture and Virtualization techniques in IoT.

2.1. Architecture of the IoT

Different stakeholders propose different IoT designs, indicating that there is no standardised architecture certified by an authorised agency. The absence of a defined architecture adds to the ambiguity that makes a precise description of an IoT system difficult to come across. In an IoT system, the hardware unit falls under one or more of the following components:

- Processing parts.
- Storage parts.
- Communication parts.
- Sensors/actuators.

We must now add the software, middleware components, and related protocols that offer the methods of connecting and controlling the hardware, as well as support for service discovery, to create a fully operating system or systems. IoT architectural components may vary, depending on the application context. In this chapter, I'll go through the components of a general IoT scenario, and the two most prevalent architectures (three-layer and five-layer architectures).

2.1.1. The component of a general IoT scenario

A typical IoT situation can be described with that of a generic who needs to engage with a (potentially distant) physical thing.

2.1.1.1. User

A goal-oriented human or live digital entity (like a service, an application, or a software outcome). Interaction with the physical world is required to accomplish the aim. The Internet of Things (IoT) facilitates this contact.

2.1.1.2. Physical entity

A "physical entity" is a definite, recognizable aspect of the physical world that the user is interested in in order to achieve his or her objective. From people or animals to automobiles, from shop or logistic chain products to computers, from electronic equipment to closed or open locations, physical things may be nearly anything. A virtual entity is used to represent physical entities in the digital realm. Virtual entities have two basic qualities in the context of the Internet of Things:

- They are digital entities that are linked to one actual object that they symbolise. While each virtual entity should have only one actual object, it is possible for the same physical entity to be connected with many virtual entities, such as a separate representation for each application domain or IT system. Each virtual entity has to have a unique ID that identifies the thing it represents. Digital entities may be active (for example, software code) or passive (for example, data).
- Digital entities are synchronised representations of a collection of physical entity features or qualities. This implies that important digital parameters describing the physical entity's attributes may be updated whenever the actual entity changes. Changes that influence the virtual entity, on the other hand, may materialise in the physical entity.

2.1.1.3. Device

The linkage between the virtual and physical entities is accomplished via the employment of a "device." This may be accomplished via embedding, attaching, or just positioning the device near the physical entity. Devices may be classified into one of the following categories based on their functionality.

- **Tags:** One of the hallmarks of the Internet of Things is its ubiquity, which is achieved by the unique identification of things linked to the Internet. Tags are attached to the "objects" to provide this unique identification. Tags are utilised by readers, which are specialised sensors. They exist only to aid in the identifying procedure. For delivering this unique identification of "things," RFID is an ideal option. An RFID transponder or RFID tag is used to convey data that is positioned on the item that has to be recognized. A pairing element (like a coil or microwave antenna) and an electronic microchip (smaller than 0.33 mm) are usually used. Tags may be read-only, read/write, or

read/write/rewrite depending on how their data is encoded, and they can be passive, semi-passive, or active depending on their power source and how they are utilised. Tags don't need a built-in power source since they get their power from the electro-magnetic field created by readers.

- **An interrogator (reader)** examines the sent data (it can be on a device that is embedded in a wall). Readers are bulkier, more costly, and more power-hungry than tags. The reader delivers a low-power radio signal to energise the tag in the most popular kind of device. The tag then selectively reflects energy and so communicates some data back to the reader, revealing its identification, position, and any other pertinent data. The majority of tags are passive and only activate when they are inside the interrogator's coverage area. They stay inactive outside of this zone. Readers may receive and read information on the tag, which is subsequently sent to a computer database.
- **Sensors** collect data on the physical entity they are monitoring. The information in this context varies from the physical entity's identification to measurements of the physical entity's physical condition. Sensors, similar to other devices, may be connected to or otherwise implanted in a physical entity's physical structure, or they can be put in the environment to monitor entities indirectly. A camera that identifies people's faces is an example of the latter.
- **Actuators** may change a physical entity's physical state. Actuators may move basic physical entities (translate, rotate, etc.) or activate/deactivate functionality in more complicated ones.

2.1.1.4. Sensor Operating Systems

TinyOS and Contiki, for example, were intended for wireless sensor networks (WSN) and might be utilised for IoT. However, it seems that the majority of OSs planned for use in WSN fail to fulfil one or more of the IoT criteria. The creators of RIOT hope to have filled the gap between WSN and IoT OS needs.

Event-driven and multi-threaded OSs are two separate design paradigms for sensor nodes. Every action that an OS must do in an event-driven system is prompted by an event (like timer, an interruption indicating fresh sensor information or an incoming radio packet). The multi-threaded OS divides execution time among the many tasks, which are implemented as threads.

When switching from one thread to another, the previous context must be kept and the new context must be restored.

Table 1. Comparisons of different operating systems (S: unconditional support, NS: Not support, PS: Partial support)

OS	RAM	ROM	C	C++	Multi-Threading	Modularity	Real-Time
Contiki	<2KB	<30KB	PS	NS	PS	PS	PS
TinyOS	<1kB	<4kB	NS	NS	PS	NS	NS
Linux	1MB	1MB	S	S	S	S	PS
RIOT	1.5kB	5kB	S	S	S	S	S

2.1.1.5 Middleware

Middleware software connects RFID interrogators to corporate applications. Middleware is used for a variety of purposes: It optimises the performance of hardware like interrogators by configuring and managing it. It also filters duplicate tag readings and aggregates data before passing it on to back-end applications.

Middleware may be installed on a dedicated computer at each facility where RFID interrogators are utilised, on each interrogator, or on a networking appliance. Because they're installed near to the network's edge at a production plant or distribution centre, as an example, these solutions are commonly referred to as "edge-ware." To connect with the readers, middleware may also be installed in a data centre with a wide area network (WAN).

2.1.1.6 Resources

Resources are parts of software that offer data about real world objects or allow devices to be controlled. There are two sorts of resources:

- On-device resources are software that is installed locally on a device and is hosted on it. They contain executable code for obtaining, analysing, and storing sensor data, in addition to the actuator control code.
- Network resources, such as back-end or cloud-based databases, are resources that are accessible someplace on the network.

Storages are a special sort of resource since they retain information from other resources and provide information about physical objects. This may contain things like location and state-tracking data, static data like product type information, and a variety of other things. Storages may be placed on-device or on the network since they are resources.

- On-device storages often store data on one or a few physical things, like the one they are monitoring.
- Network-based storages compile data about a huge number of physical objects.

Because not all known information about an object is, or even can be, delivered by devices, human users may update the information in storage.

2.1.1.7 Service

The service offers a well-defined and standardised interface with all of the required functionality for communicating with physical entities and processes. All of this is accomplished by the use of the internet.

By accessing the device's hosted resources, services reveal the device's capability.

2.1.2 Three-layer Architecture of the IoT

- **Perception layer:** Known as a physical layer or sensing layer. In this layer, embedded sensors gather data from devices and communicate it to the network layer. It performs like the skin of a living entity, employing its five senses to identify objects and collect information. Tags, sensors, actuators, and the physical entity are all included in this layer.
- **Network layer:** this layer is in charge of connecting smart things, network devices, and servers. It's used to send and analyse data from the Perception Layer. 3G, WiFi, Bluetooth, Infrared, Zigbee, and other technologies function in this layer. The interrogator or reader is located on this layer.
- **Application layer** is in charge of providing the user with application-specific services. Data is processed at this layer in order to offer service and make choices. Middleware, Resources and Service components belong to this layer.



Figure 1. Three layers architecture of IoT

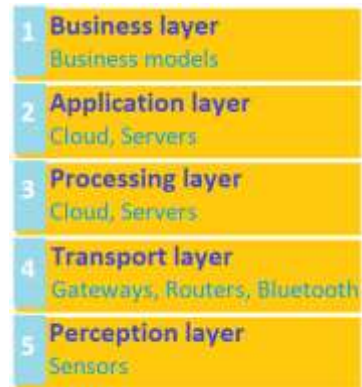


Figure 2. Five layers architecture of IoT

2.1.3 Five-layer Architecture of the IoT

- **Perception, Transport (Network), Application layers** have the same functionality that they have in three-layer architecture.
- **Processing layer** is responsible for storing, analysing, and processing large volumes of data.
- **Business layer** oversees the whole Internet of Things system, including apps, business and profit models, and user privacy.

2.2. Virtualization techniques in IoT

The Internet of Things (IoT) allows the connection to anything, anywhere, at any time, resulting in pervasive networks of heterogeneous objects. Laptops, cell phones, household appliances, industrial systems, e-health gadgets, surveillance equipment, precision agricultural sensors, and other Internet-connected accessories approached approximately 45,000,000,000, with a financial contribution of USD 14,000,000,000. Because there will be so many devices, enormous amounts of data will be generated, necessitating the installation of additional network access and core equipment. Numerous technological challenges exist for the smooth operation and incorporating such massive IoT systems, including security, privacy, heterogeneity of devices and applications, edge access, topological structure of IoT nodes, communication protocols, data collection and analysis. Furthermore, the possible mobility of IoT devices results in dynamic alteration in topology, which may need flow re-establishment. Although it may be difficult to develop a single solution to address all of the issues, virtualization and

programmability of hardware and software resources may considerably minimise the complexity of individual solutions.

Virtualization is a software technology that logically abstracts the underlying hardware components in a network. It separates the decision-making and packet forwarding. The control is decoupled from the hardware, making it easy to adapt, administer, and update. In recent years, the abstraction has expanded beyond hardware, with software integrated in hardware being virtualized as separate virtual function parts.

The majority of conventional networks are stiff and immovable. Heterogeneity, scalability, and interoperability have been important issues because of the obvious Internet's fast expansion. Software-Defined Networks (SDN) and Network Function Virtualization (NFV) are two common approaches to network virtualization. SDN provides a programmable network by centralising the control operations of routing devices; as a result, the physical infrastructure simply serves as a forwarding (data) plane, while the controller determines flow and policy information. Although other solutions offer comparable functionality, OpenFlow (OF) is the core standard for interaction between the control and data planes. The main benefit of SDN is that it allows for network-wide policy, flow, and configuration enforcement via a single point of programmability. Additionally, resource efficiency, network slice virtualization, security, and vendor independence may be accomplished. NFV is a framework for functions abstraction from specific hardware to a virtual environment, such as firewalls, load balancing, route computation, and so on. NFV's main advantages include the ability to replace specialised hardware with commodity servers.

Virtual network functions (VNFs) enable SDN services such as security functions, load balancing, data gathering and analysis, etc. This not only improves scalability and flexibility for delivering vendor-independent commodities at lower costs, but it also optimises network devices' computation, memory, storage, and networking capabilities. SDN and NFV are complementary rather than competing with each other (despite the fact that they are maintained by distinct standards groups). As a result, the primary advantages of both technologies are intertwined. NFV may help SDN get closer to cloud-based virtualization of the SDN controller and other network applications. Similarly, SDN can apply traffic engineering choices made by VNFs because of its programmable network connection.

2.2.1 IoT based on Software-Defined Network

IoT based on SDN means that the virtualization of the network that has an IoT device may improve routing efficiency, network management, and resource optimization to keep up with the growing needs of IoT networks. Traditional network issues such as heterogeneity, scalability, inefficient service delivery (lack of dynamic services), slow adaptation to new services (Time spent upgrading the network), and a lack of user experience guarantees are expected to be addressed by SDN solutions in the IoT environment. To that end, many SDN-based IoT architectures have been presented in a number of publications up till now. In addition to commercial solutions, scholarly literature has a few recommendations and solutions. We divide them into three categories: architectural, security, and management.

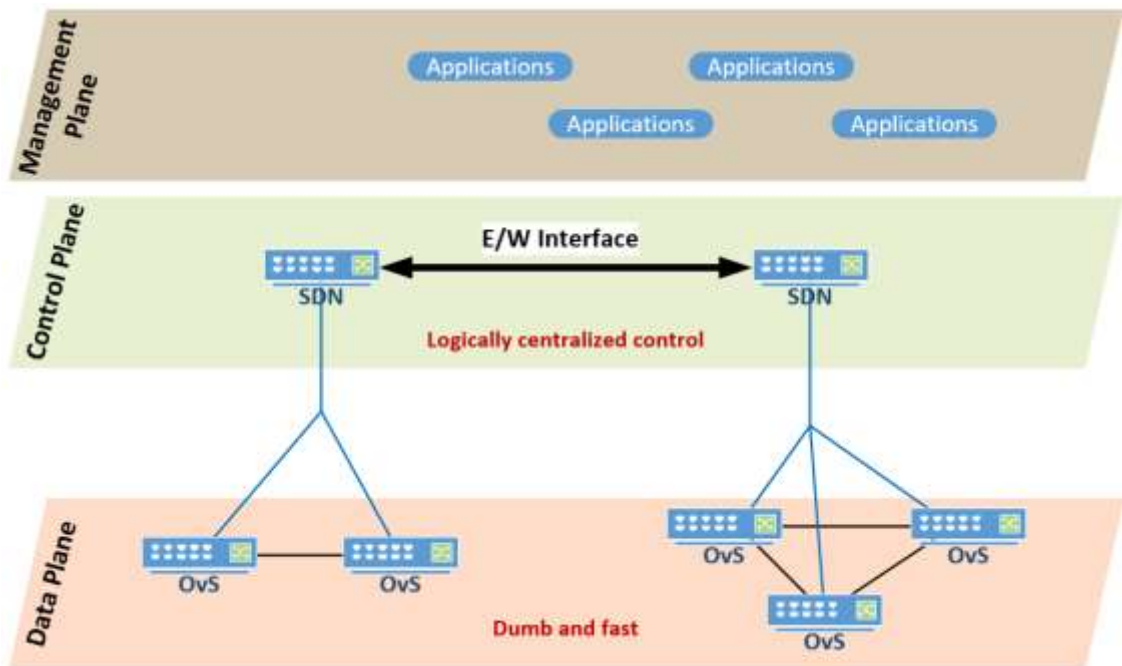


Figure 3. A generic SDN-based architecture for IoT

The services supplied in the control plane and the data plane are clearly separated in SDN-based IoT architecture. The data plane provides the means for forwarding traffic to the intended destination, whereas the control plane specifies the administration of network traffic. SDN-based architecture for IoT can be illustrated as shown in figure 3 in which it contains 3 layers (data, control, and management planes). The Management Layer enables the admin/analyst to

determine how control procedures will be regulated by human users as well as the SDN controller. IoT devices are connected in the data plane.

The SDN control plane communicates with devices using OpenFlow software and APIs. Devices' operating systems need to be compatible and enabled for SDN communication. OpenFlow works above the normal network routing protocols like OSPF.

2.2.2 Network Function Virtualization for IOT

The technologies of Network Function Virtualization and Software Defined Networking (SDN) are complimentary. They don't need or rely on one another; instead, they help each other work better. VNFs, or virtual network functions, are provided by NFV. VNFs include deep packet inspection, routing, security, and traffic management. Those processes can be coupled to create network services for IoT. Figure 4 depicts a hybrid SDN/NFV architecture for IoT that highlights how SDN and NFV work together to offer dependable connectivity and make IoT platforms easier to use.

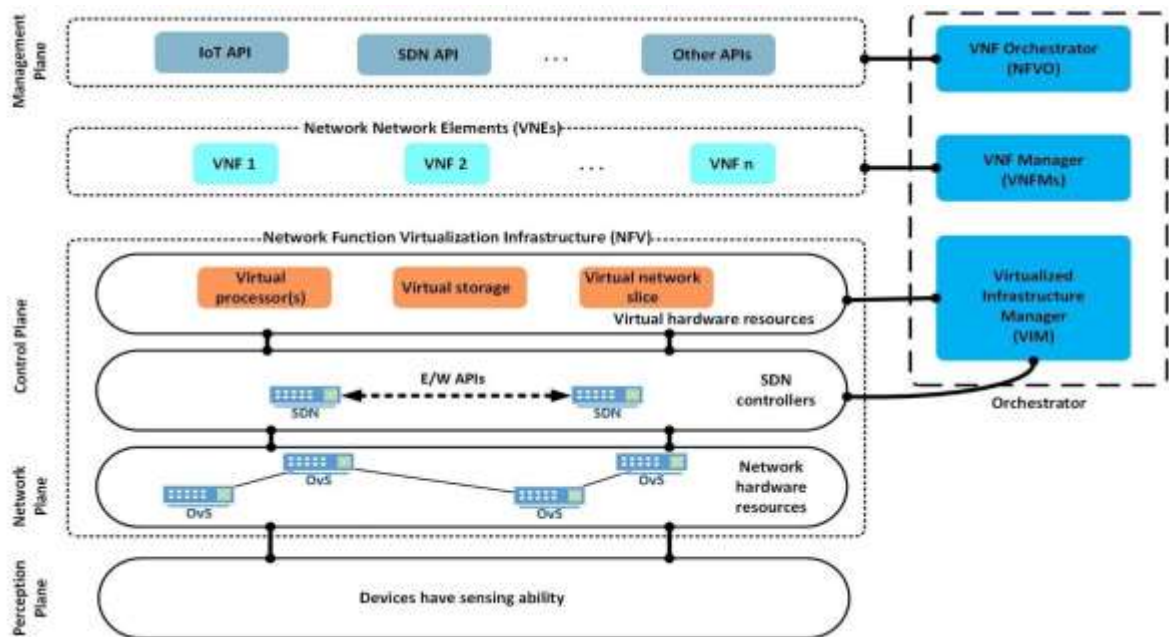


Figure 4. A standard SDN-IoT architecture with NFV

The architecture is made up of three parts: (1) NFVI, which consists of the networking hardware and software resources needed to link and maintain carrier networks, (2) VNFs, which are in charge of managing specific network functionality that runs on one or more virtual machines

(VMs), and (3) the Management and Orchestration plane, which facilitates connectivity between services from different NFVI modules, VNFs, and APIs from the Management Plane and coordinates their use.

The SDN controller enforces efficient packet flow rules and all of these pieces work together to enable long-term network virtualization with unbroken network connection.

It's worth noting that VNF has usually been used in conjunction with SDN. SDN often uses virtualization technologies, either as application processes or controller activities, for management tasks for IoT networks.

2.2.3 Security solutions

When creating solutions like SDN and NVF to make the management and troubleshooting easier, security concerns need to be taken into the consideration.

2.2.3.1 SDN-based security solutions for IoT Networks

Traditional security solutions for external threats, such as firewalls, are implemented at the network verge. However, because of the constant changes in network topology caused by IoT nodes joining and leaving, they must be re-configured and updated on a regular basis. Similarly, new internal risks such as rogue devices or software or hardware vulnerabilities need constant monitoring and patching. As a result, as technology advances, it can be necessary to assess the security requirements for both internal and external threats. For SDN-based IoT security challenges, many solutions have been presented. The security challenges may be divided into four categories: architectural and protocol-related vulnerabilities, flow-based security issues, Usecase-specific security issues, and other network-related attacks and vulnerabilities.

Architecture and Protocol Vulnerabilities: In an SDN context, new flow attacks that inject a large number of mismatched packets into the routing system may block connection between IoT devices and servers. This causes the control plane (by overloading the controller) and data plane (by overloading resources at OF switches) to handle an excessive volume of data packets. There are multiple presented solutions to address this problem. Including but not limited to:

- 1) The controller serves as a security middleware, filtering new-flow vulnerabilities including DDoS, switch-to-controller communication flooding, and flow-table flooding. Large-scale deployment, on the other hand, might be difficult for it.

- 2) There is a comparable technique for OpenFlow-enabled systems, which enables for quick creation and deployment of modules to identify and mitigate risks.
- 3) There is a proposal of forming groups of sensor nodes in a dispersed network scenario, which is suitable for an IoT cluster setting. Rather than utilising a static firewall to thwart a potential attack.

Consequently, a SDN-based routing protocol and a dynamic firewall called Distributed Smart Firewall that can perform the functions of an SDN controller.

Flow-based Security: Data traffic and flow characteristics are monitored using SDN gateways in a distributed framework. There are ideas for a mechanism for SDN-based IoT gateways to detect and decrease abnormal activity and improve QoS. Examples include, but are not limited to:

- 1) A system that includes an SDN-enabled gateway and a cloud-based controller for detecting fraudulent IoT activities in home networks.
- 2) An intrusion detection system that uses traffic patterns to forecast abnormalities for improved access control. The model detects deviations from normal based on a given input, and the policy generator sets up the rules for such flows. The solution is unusual in that it determines anomalies using regression models.

Practical security case: Basic security functionalities are missing from a large number of IoT-based household equipment, such as motion/monitoring sensors, smoke alarms, and smart lighting.

An Intrusion Detection and Mitigation system has been proposed, which provides a network-level preventive mechanism against malicious or suspicious objects from external network domains attempting to enter the Smart Home environment. This proposal calls for the use of machine learning to identify attacks based on learnt signatures. Although it has a lot of potential, it produces a lot of traffic and is only available on a few smart devices.

2.2.3.2 NFV-based security solutions for IoT

Primary security solutions of NFV is the use of virtual functions to extract the security modules from fixed hardware/software locations and implement them in the cloud. They may

simply be redeployed, scaled, and optimised for new applications from there. Machine learning techniques for safeguarding diverse IoT devices in a home network may be implemented using virtual functions. Virtual functions are deployed on verge gateways, which are where IoT devices send their data.

The following are the primary security enablers that NFV solutions in IoT may address:

- Firewall and Intrusion Detection: Rather than implementing the individual components (ex: flow filtering, signature detection, etc.) at the edge or controller, virtual functions may modularize and implement them as software abstractions. These are quite scalable.
- Virtual packet analyzers may be modified for different protocols and applications. Furthermore, using cloud-based machine learning algorithms may considerably boost analytical output.
- Authentication and Authorization: When compared to typical IoT or SDN-controller-based systems, AAA may become more adaptable. This may be thought of as a virtual trust authority that can be tailored to certain domains and applications.

2.3. Intelligence of the future era of IoT

The future era of the Internet of Things (IoT) will combine hyper-connectivity, edge computing, and artificial intelligence (AI). IoT software products will use AI techniques like machine learning (ML) and neural networks (NNs) to improve data treatment and integrate robotic devices, drones, autonomous vehicles, etc. These software products will result in new services, and experiences that will benefit businesses, consumers, and industries. As we progress toward the implementation of intelligent things that have the ability to specify the interactions between autonomous systems and people in a smooth manner, a more human centred viewpoint will enable us to maximise the effects of the future era of IoT technologies and software products.

2.3.1 The future era of the Internet of Things

To enable the IoT, heterogeneous technologies are employed for sensing, gathering, storing, acting, processing, inferring, transmitting, producing alerts, managing, and analysing data. New IoT systems that depend on real time data analysis at the verge rely on smart solutions with built-in intelligence, connectivity, and computation abilities for verge devices. These new IoT

systems are leaving centralised cloud processing and toward distributed intelligent edge processing. Traditional centralised cloud processing systems are ideal for non real time applications that need high data rates, a lot of storage and computing capacity, are expensive, and able to be utilised for data analytics and AI processing.

Distributed edge solutions, in contrast, place computations at the network's edge, where data is created. They are ideal for real time services and may be utilised for basic ultra-fast analytical workloads. Because no personal information is maintained in centralised servers and each user keeps full control over his or her data, the collecting, storage, and processing of data at the network's verge in a distributed manner helps to greater privacy of user's data. Through the Internet of Things (IoT), common physical things in the world are converted into information ecologies that improve our lives. The Internet of Things shapes the future of the Internet, with consequences for security and privacy (personal liberties), and it also has the potential to bridge the digital gap. Expanded reliance on the connectivity network by AI and IoT, combined with the severity of security concerns, raises their vulnerabilities. How emerging technologies adapt to these threats will determine the Internet's continued and future success as an engine of economic and social innovation. Combining AI and IoT opens up new possibilities, shaping a fresh set of services and scientific advances to the enhancement of human intellect and its fusion with the physical and digital world.

In order to properly design and implement the IoT technologies of the future era, such as AI, DLTs, hyper-connectivity, distributed edge processing, end to end distributed security, and autonomous systems – robotics, as shown in Figure 5, there will need to be more human-centred safeguard put in place.



Figure 5. Technology convergence of the future era of IoT

IoT combines the primary attributes of the technology of the future internet, mobile systems, and everywhere connection with those of industrial control systems in the context of industry digitization. Capabilities for sensing, actuating, and controlling for digitising industry applications, interoperability, platform integration, and standards are necessary. Instead of being simply extensions of the internet, IoT devices and systems that rely on greater sensing, and processing capability at the verge are already becoming a natural element of the future internet. In a hyper-connected world, where every device has the power to perceive its surroundings and communicate data, the Internet of Things is promising.

When diverse systems are connected, a typical IoT solution could result in various implementations in which their efficiency has to be tested and verified. Practically, this testing and verification requires different suppliers. In a variety of settings. Cloud computing's centralised services are being superseded by IoT edge distributed services that depend on AI approaches. With multi access edge processing and popular hyper-connectivity (5G and after), the Internet of Things can now process large amounts of data generated by its connections and use it for intelligent purposes by sophisticated AI techniques that can receive less data and use fewer processing and memory resources.

2.3.2 Edge computing as a part of future IoT technologies

The number of cellular IoT connections is expected to reach three and half billion globally by 2023. The number of linked devices in a plant will grow dramatically as assets, equipment, vehicles, and processes are digitised. In a typical smart factory, the number of linked devices required is expected to be a half per square metre. This figure is based on the number of prospective use cases and assets that could gain from a connection. The percentage of each sort of linked device varies depending on whether the location is automated or not. As we progress toward a higher level of automation, the number of 5G linked devices will grow. To accommodate enormous data volumes and real time important data, also to assure constant and safe connection, high bandwidth and consistently low latency are required. This necessitates a shift in IoT digital infrastructure. According to Gartner, by 2025, eighty percent of businesses will have closed their regular data centres, up from ten percent in 2018. The fundamental driver of this infrastructure evolution is workload placement, which is influenced by a range of business needs. Edge computing is at the top of Gartner's 2018 Hype Cycle for Cloud Computing, so there's plenty of room for false starts and demotivation before standards and best

practices are established and widespread adoption can begin. Edge computing is the decentralised alternative to today's hyper-scale clouds and traditional data centres. Technology innovators design distributed computing systems along a continuum from the core to the edge to maximise application possibilities and user experience.

2.3.3 Artificial intelligence as a part of future IoT technologies

Artificial intelligence refers to work aimed at making machines intelligent, with intelligence defined as a property that allows an object to function effectively and predictably in its surroundings. AI refers to a machine's ability to execute operations like learning, decision-making, or other intelligent human functions. IoT-based sensor data can be processed utilising approaches that depend on AI to deliver new data insights and develop new kinds of knowledge in medical technology, oil and gas industry, manufacturing, energy sector, mining, engineering, policy- and decision-making in governments and corporations, and many other sectors. When AI and IoT technologies are merged, both on the verge devices and on the servers, the benefits of both can be increased. All machine learning approaches can extract insights from data in order to analyse and anticipate future IoT device connections in advance. Because of its capacity to swiftly extract insights from data, AI is a star in the IoT. Machine learning enables data collected by smart sensors and devices, such as temperature, movement, humidity, air quality, pollution, vibration, and sound, to automatically identify trends and identify anomalies.

Machine learning is proving to have significant benefits over conventional business intelligence technologies when it comes to analysing IoT data, such as the ability to make operational anticipations up to twenty times faster and with greater precision as compared to threshold based monitoring systems. AI techniques expand machine learning algorithms for complicated decisions based on recognizing patterns, self-learning, self-healing, context awareness, and autonomous decision-making that may be used for intelligent IoT devices. These will have an impact on future digital twin models and continuous learning implementations, as well as autonomous car applications and predictive maintenance.

2.3.4 Network and Communication

Low power short range networks for wireless IoT connectivity are expected to grow in popularity through 2025, coexisting with extensive IoT networks, while 5G networks will supply 1000 to 5000 times the capacity of today's 3G and 4G networks. As corporations push

boundaries of data, analytics, and intelligence, IoT technologies are extending established business models, resulting in the development of new ones. IoT devices will play a role in this development and will be a driving force behind it. Changes will be incorporated first into specific communication standards and networks, then into the communication and network structures established by these standards. 5G and the Internet of Things develop new abilities and use cases that will affect not only consumer services but also many sectors undergoing digital changes.

2.3.4.1 Network technology – hyper-connectivity beyond 5G

A key facilitator for the expansion of the Internet of Things will be the development of essential communication skills. By doing so, it will allow IoT use cases to expand beyond data collecting into complicated scenarios requiring precise actuation, automation, task performing, and crucial connections. 5G will enable new connectivity to become the incentive for future era of IoT services by enabling innovations like advanced modulation techniques for high wireless data rate transmission, automated network application, software-defined networking and network function virtualization, in addition to supporting IoT services that are designed for the edge and cloud. The requirements for critical IoT communications are myriad and varied, ranging from improved communication network robustness and dependability to extremely low latencies and high capacity, while also integrating the mission context with the ability of meeting rigorous energy efficiency requirements or being able to cover wide outdoor spaces, enclosed spaces, or fast moving vehicles.

One of the important components of 5G is the emphasis on the integration of diverse access technologies, including satellite communication systems. Satellites might be crucial in ensuring ubiquity and stability in far-off locations, as well as enabling new Internet of Things (IoT) services. Satellite connectivity is not built into IoT devices, and the needs of satellites are not taken into account while designing IoT protocols. As a result, cross-layer optimization is necessary to enable the collection of IoT data from satellites, load balancing, and unloading of terrestrial networks, allowing for seamless integration of IoT and satellite networks. By integrating security into the fundamental network architecture, a more dependable and secure network will be provided by 5G for IoT. Future networks will need to solve cell-to-cell and radiation interference, as well as create novel management models to govern roaming while taking advantage of the coexistence of diverse cell types and radio access technologies.

For increased efficiency in accessing network resources, new management protocols managing user assignment with reference to cells and technology will need to be adopted in the mobile core network. In remote places, satellite communications should be explored as a possible radio access technology. End-to-end latency will benefit from lowering delay and the numerous protocol translations as safety applications emerge. Mobile network densification puts a strain on the connection to the core network. Future networks, on the other hand, should integrate cloud utilisation techniques to maximise latency, security, energy efficiency, and accessibility.

In this situation, greater network flexibility is required, which can be achieved by combining cloud technologies with software-defined networks and network function virtualization, allowing network flexibility to incorporate new applications and properly arrange network resources (sharing processing resources, splitting data traffic, security rules, QoS parameters, mobility etc.).

By incorporating the Web of Things (WoT) into the emerging IoT, the growth and widespread of current communication technology might soon reach unheard-of heights. Humans, robots, and things, as well as groups of them, will be network users.

2.3.4.2 Communication technology

IoT devices, both on the consumer and enterprise sides, are primarily driving global connection growth. There are approximately 12 billion active IoT devices, and by 2025, there will be 22 billion.

802.11 Wi-Fi, Bluetooth, ZigBee, 6LowPAN, and Z-Wave protocols use several ISM radio bands simultaneously. Connectivity modules are built using integrated circuits (ICs), reference designs, and well-featured software that follow a flexible modular paradigm that handles a wide range of application areas.

3. Analysis of the virtualization effect on IoT systems

New technology must be tested before being used in the real world. The easiest technique to track the outcomes is to model the technology using a particular software environment. This chapter involves hands-on simulation work to construct a network topology containing IoT and

end-user devices. These gadgets are attached to a remote controller (SDN solution). Through OpenFlow traffic, this controller manages the topology's elements. As mentioned earlier, OpenFlow is the core standard for interaction between the control and data planes of network devices. The Openflow traffic between the controller and the network devices was extracted and handled in Python code for analysis. This project can give a picture of how networks can be controlled and how traffic can be extracted and analysed. This is a kernel for modern techniques in networks and the IoT world. Analysing the live traffic is the core of many new techniques like integrating Artificial Intelligence (AI) into networks.

3.1. Presentation of the measurement environment and of the methodology applied

I used several software tools to accomplish this project. The used tools are:

- Virtualbox to connect multiple virtual machines.
- GNS3 to build a network topology.
- OpenDayLight as SDN controller.
- Open Flow Manager to be connected over the controller and make controlling easier.
- Wireshark to extract traffic and Python language.

The measurements time duration was 311.80 seconds (for topology “a”), and 95.38 seconds (for topology “b”). The work steps are illustrated in table 2.

Table 2. Followed steps to accomplish the project

#	Steps
1	Prepare the environment to connect multiple virtual machines in my PC
2	Prepare GNS3 software
3	Connect SDN controller (OpenDayLight and Open Flow Manager platform)
4	Change rules in the controller affecting on the topology inside GNS3
5	Collect open-flow traffic via Wireshark
6	Create some scripts to analyse that traffic

3.2. Extraction of status data series and evaluation of the virtualization effects

Let's go through each point in the previous table. Points 1 to 5 for software environment preparation. Point 6 is for Data analysis.

3.2.1 Prepare the environment to connect multiple virtual machines in PC

As the project was constructed using a simulation environment, I installed VirtualBox software that creates multiple virtual machines.



Figure 6. VirtualBox software that contains 3 virtual machines

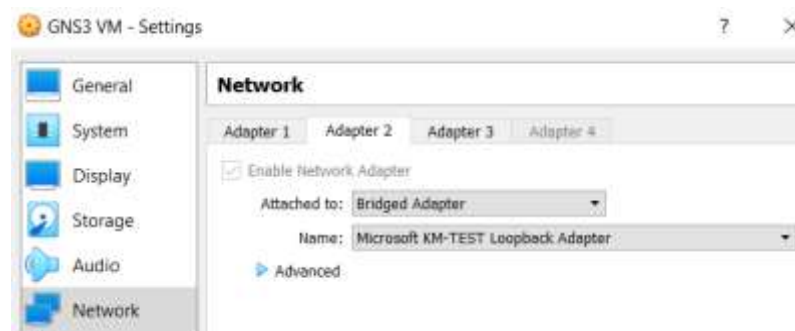


Figure 7. Configuring a virtual machines in VirtualBox to be connected to Microsoft KM-TEST Loopback adapter in my PC

As shown in figure 6, three virtual machines were created (Next pages of this chapter contain explanations of each one). All of them were configured to be connected to a new network adapter that was created on my PC as shown in figure 7 and figure 8.

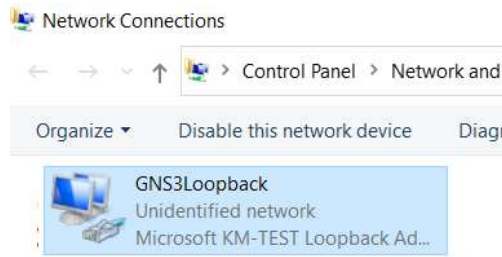


Figure 8. Microsoft KM-TEST Loopback adapter

Connecting different virtual machines to the same network adapter in my PC (KM-TEST Loopback) enables the connectivity between them.

3.2.2 Prepare GNS3 software

I installed GNS3 software and connected it to a virtual machine (GNS3 VM that is shown in figure 6 and 7). I brought the required devices' images inside GNS3. The most important one is (Open vSwitch) which has the ability to be controlled via an SDN controller.

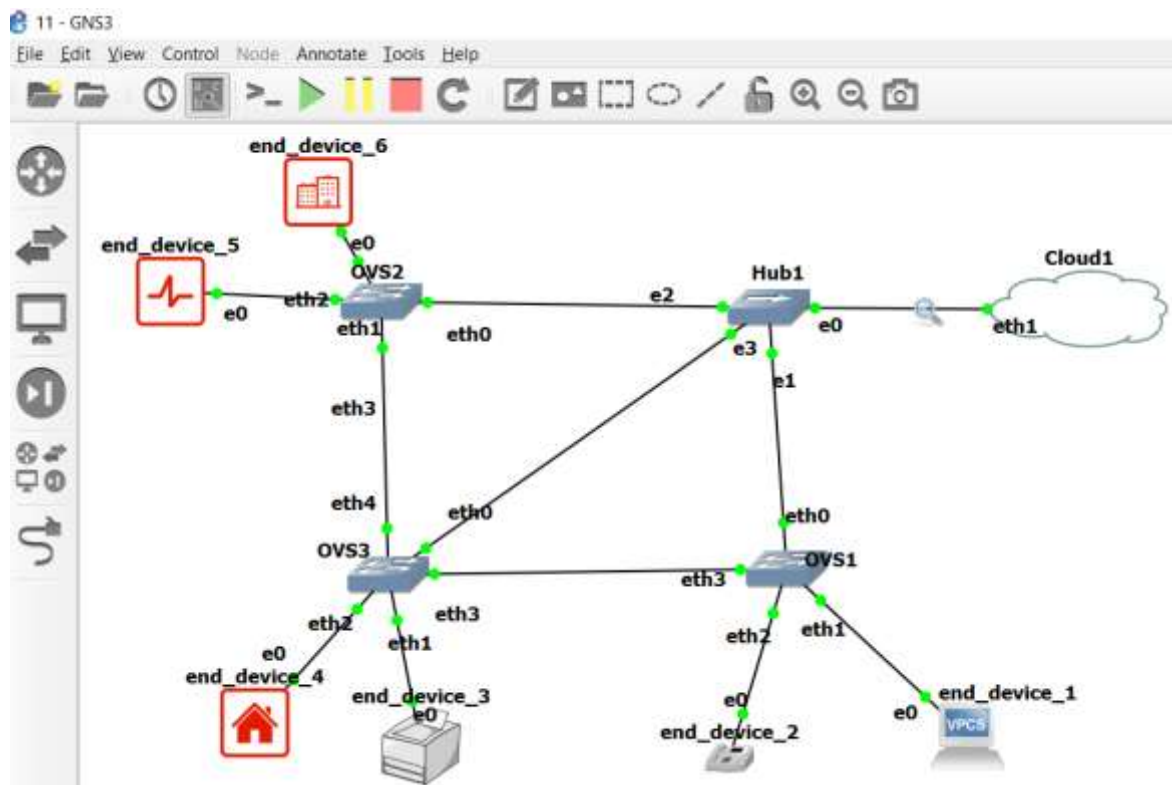


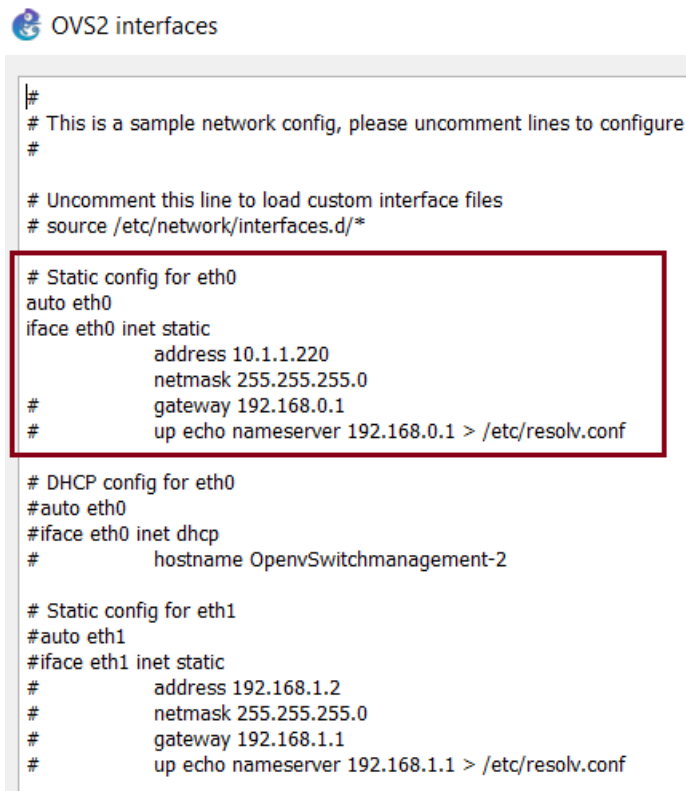
Figure 9. Created GNS3 topology

The below table explains the identification of the used nodes in this chapter.

Table 3. Identification of the nodes

Node name	IP address	Node's ID in the controller
OVS1	10.1.1.200	108226331345484
OVS2	10.1.1.220	227561361717582
OVS3	10.1.1.210	103892047702094
SDN controller	10.1.1.60	Not used

As shown in figure 9, I created a topology that contains multiple end user devices in addition to Open vSwitches which is connected to the KM-TEST loopback by a cloud item. I configured the KM-TEST Loopback adapter in my PC and the Open vSwitches inside the GNS3 with the same subnet range of IP address. So, the topology inside the GNS3 becomes reachable by my PC. IP address configuration in the Open vSwitch was implemented by simply right mouse click on each Open vSwitch, select (edit config), and then change the IP address as shown in figure 10.



```
#
# This is a sample network config, please uncomment lines to configure
#

# Uncomment this line to load custom interface files
# source /etc/network/interfaces.d/*

# Static config for eth0
auto eth0
iface eth0 inet static
    address 10.1.1.220
    netmask 255.255.255.0
#    gateway 192.168.0.1
#    up echo nameserver 192.168.0.1 > /etc/resolv.conf

# DHCP config for eth0
#auto eth0
#iface eth0 inet dhcp
#    hostname OpenvSwitchmanagement-2

# Static config for eth1
#auto eth1
#iface eth1 inet static
#    address 192.168.1.2
#    netmask 255.255.255.0
#    gateway 192.168.1.1
#    up echo nameserver 192.168.1.1 > /etc/resolv.conf
```

Figure 10. Configuring IP address in Open vSwitch in GNS3

Some additional console commands are required inside the Open vSwitch to connect it to the controller that have IP address (10.1.1.60) as the below,

- ovs-vsctl set-controller br0 tcp:10.1.1.60:6633
- ovs-vsctl set bridge br0 protocols=OpenFlow13

3.2.3 Connect SDN controller (OpenDayLight and Open Flow Manager platform)

I chose OpenDayLight as the SDN controller. I installed its virtual machine in VirtualBox software as shown in figure 6. It takes an IP address from the same range of Microsoft KM-TEST Loopback adapter in my PC and can be opened via internet browser.

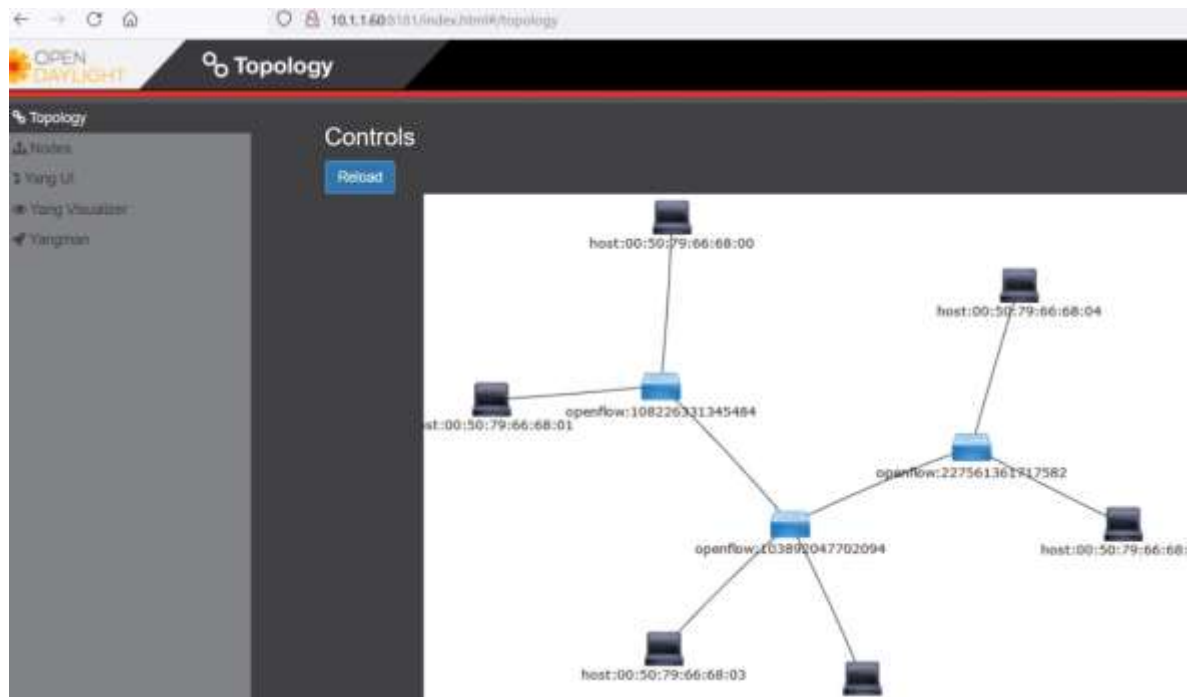


Figure 11. OpenDayLight page in web browser

As shown in figure 11, devices in the GNS3 topology appear in the OpenDayLight controller graphical interface. The displayed devices are Three Ovs switches and whenever we do ping from an end device to another, those end devices appear. To have better control on the traffic follow and create policies, I used another software which is (Open Flow Manager). I installed the virtual machine image of Open Flow Manager in the same way of installing OpenDayLight in the VirtualBox software as shown in figure 6.

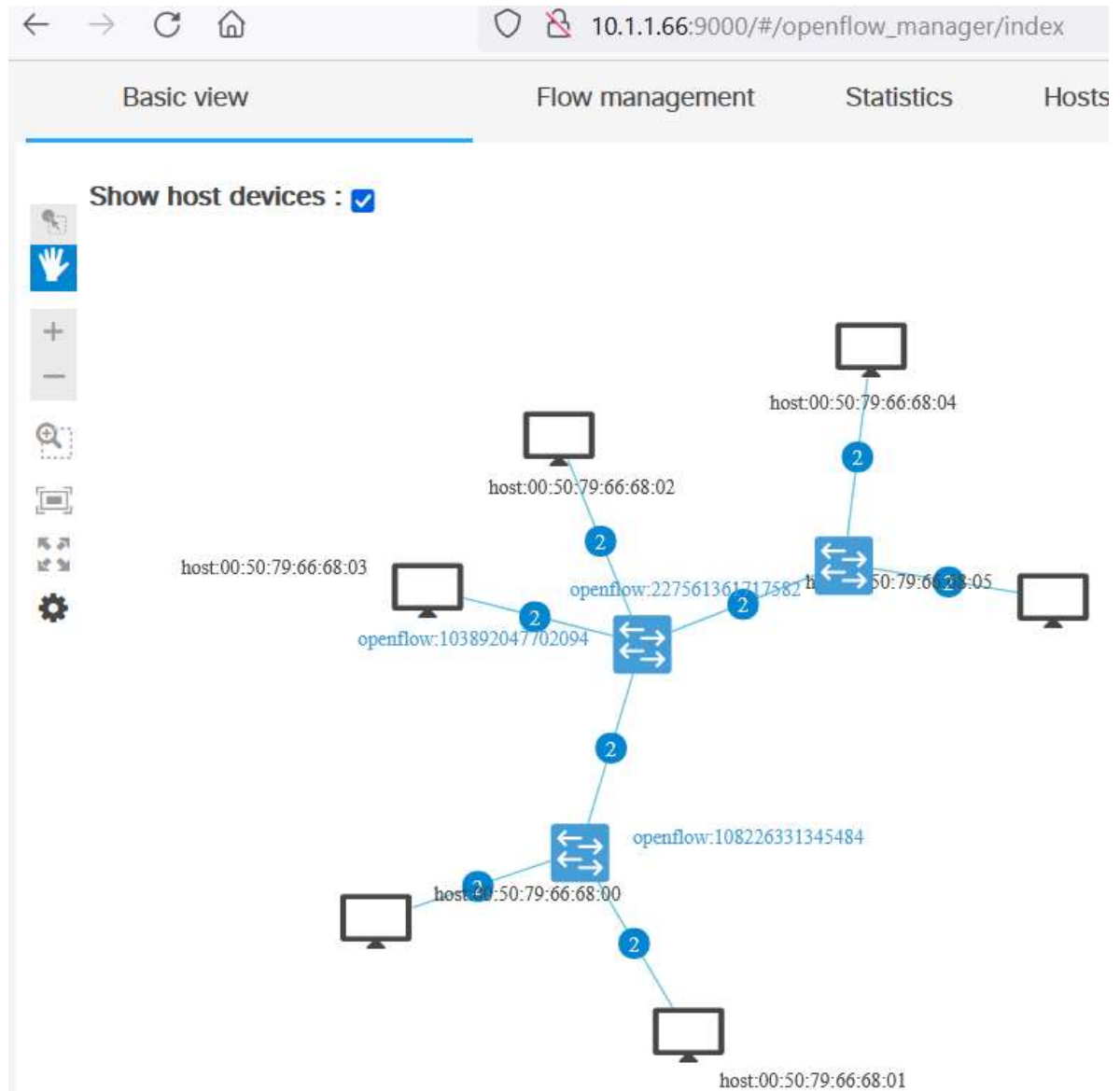


Figure 12. Open Flow Manager page in web browser (basic view tab)

If we navigate in the Open Flow Manager page and transport from (Basic view) to (Flow management), we can see multiple flows. Each flow represents a policy to allow traffic from a specific port to another. We can delete and add flows.

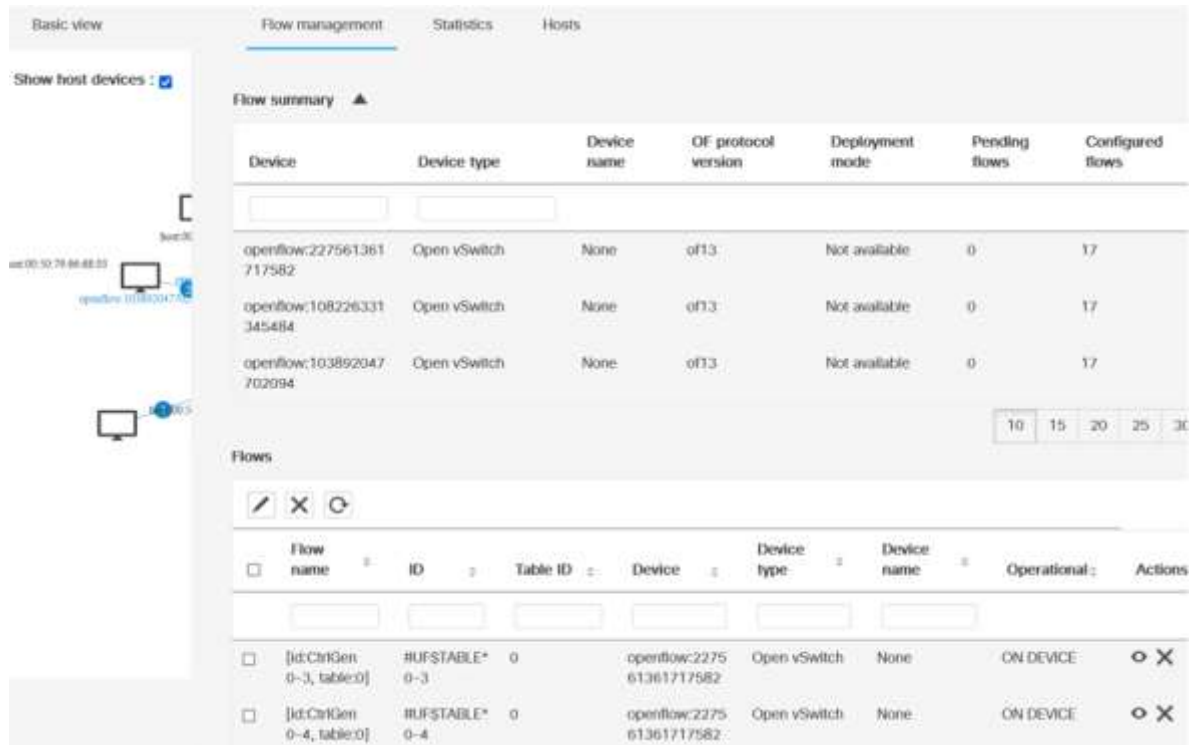


Figure 13. Open Flow Manager page in web browser (flow management tab)

3.2.4 Change rules in the controller affecting on the topology inside GNS3

If we remove all follows, end devices will not be able to reach each other. Figures 14, 15, and 16 show how to add new flow.



Figure 14. Adding new flow in Open Flow Manager (step 1)

SELECT DEVICE

+

Config ☐

Operational ☐

Device

-

-

openflow:227561361717582 [None] [Open vSwitch]

openflow:108226331345484 [None] [Open vSwitch]

openflow:103892047702094 [None] [Open vSwitch]

Show preview Send request Send all Back

Figure 15. Adding new flow in Open Flow Manager (step 2, select the device)

Cookie mask ☐

ADDED Priority

Match

ADDED In port

Metadata ☐

Metadata mask ☐

Ethernet type ☐

Source MAC ☐

Destination MAC ☐

Vlan ID ☐

Vlan priority ☐

Actions

ADDED Output port

General properties

Table 0

ID 10284

Priority 100

In port openflow:108226331345484:1

Actions

Output port

Output port openflow:108226331345484:3

Maximum length 0

Send actual flow (request)

Show preview Send request Send all Back

Figure 16. Adding new flow in Open Flow Manager (step 3, specify flow's rules)

Table number: the default table is 0

ID: any ID for the new flow

Priority: higher numbers give more priority. I put 100 and other numbers are also possible

Match: we can match different things, I choose to match the input interface

Action: I chose to allow a matched interface to communicate with a chosen output interface.

3.2.5 Collect open-flow traffic via Wireshark

After changing some flows to allow and deny some communication between end devices, I started a Wireshark capture in GNS3 in the link between the cloud and the hub as shown in figure 9. During the capture, I did a ping between devices. Then in Wireshark, I stopped the capture and saved a csv file as shown in figure 17.

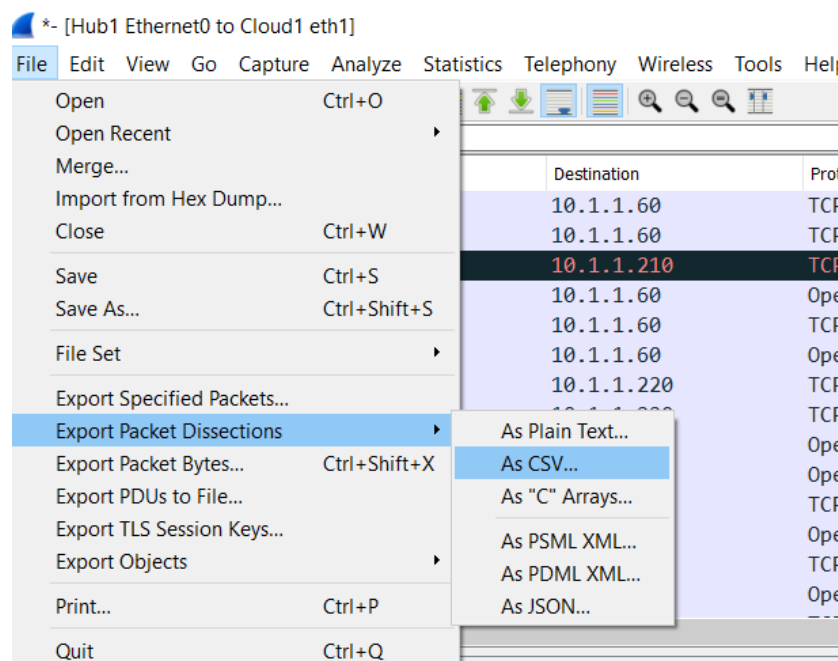


Figure 17. Saving a Wireshark capture as a csv file

No.	Time	Source	Destination	Protocol	Length	Info
1	0	10.1.1.60	10.1.1.210	OpenFlow	82	Type: OFPT_MULTIPART_REQUEST, OFPMP_TABLE
2	0.002196	10.1.1.210	10.1.1.60	TCP	1514	57860 > 6633 [ACK] Seq=1 Ack=17 Win=501 Len=1448 TSval=80930
3	0.002222	10.1.1.210	10.1.1.60	TCP	1514	57860 > 6633 [ACK] Seq=1449 Ack=17 Win=501 Len=1448 TSval=80930
4	0.002241	10.1.1.210	10.1.1.60	TCP	1514	57860 > 6633 [PSH, ACK] Seq=2897 Ack=17 Win=501 Len=1448 TSval=80930
5	0.002268	10.1.1.210	10.1.1.60	TCP	1514	57860 > 6633 [ACK] Seq=4345 Ack=17 Win=501 Len=1448 TSval=80930
6	0.002286	10.1.1.210	10.1.1.60	OpenFlow	386	Type: OFPT_MULTIPART_REPLY, OFPMP_TABLE

Figure 18. Screenshot of the saved csv file

3.3. Description of the developed analysis scripts to analyse the traffic

This is the last step of table 2. This part of the project is the analysing of data that was extracted previously. I analysed the traffic within multiple versions.

3.3.1 Data analysis version 1 (get output graph for each message type)

I created Python script to do the below:

- Extract only the messages related to OpenFlow protocol,
- Separate them into multiple kinds depending on message information, source, and destination.
- Calculate the number of packets of each kind.

Let me divide the Python code to blocks and explain each one,

Block 1: importing libraries. This step can be done in different ways depending on the used software environment. I used Jupyter notebook software.

1	! python3 -m pip install pandas
2	! python3 -m pip install matplotlib
3	! python3 -m pip install seaborn

Block 2: defining symbols to use the libraries.

1	import pandas as pd
2	import matplotlib.pyplot as plt

3	import seaborn as sns
---	-----------------------

Block 3: reading the csv file.

1	df = pd.read_csv("Mohanadfile.csv")
2	df.head()

Block 4: creating a dictionary (df1) that takes all the components of the csv file and extract only messages that are related to OpenFlow protocol.

1	df1 = df.loc[df['Protocol'] == "OpenFlow"]
---	--

Block 5: removing the word (type) from the message's information as shown in figure 17.

1	df1['Info'] = df1['Info'].map(lambda x: x.lstrip("Type: "))
---	---

Block 6: creating a function to do the below,

- Using a (unique) function to make my PC know that each source address can have many packets. When the same source address sends a new message, my PC knows that this is not a new source.
- Print all the components of (df1) dictionary for each source IP separately.

1	df_sources_query = []
2	s = df1["Source"].unique()
3	for source in df1["Source"].unique():
4	print(source)
5	print(df1.loc[df1['Source'] == source])
6	df_sources_query.append(df1.loc[df1['Source'] == source])
7	print(df_sources_query)

Block 7: creating a dictionary that contains the below,

- Sources.
- Destinations.
- Messages.
- Messages count.

- Messages information which is a combination of multiple elements to be used in graph drawings

The purpose of this dictionary is to arrange the information of (df1) dictionary in block 1 as shown in figure 19.

Source	Destination	new message info	message info	message count
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_TABLE	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_FLOW	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_GROUP_DESC	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_GROUP	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_METER_CONFIG	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_METER	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_PORT_STATUS	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_MULTIPART_REQUEST OFPMP_QUEUE	10.1.1.60 x \n10.1.1.210\nOFPT_	57
10.1.1.60	10.1.1.210	OFPT_BARRIER_REQUEST	10.1.1.60 x \n10.1.1.210\nOFPT_	84

10.1.1.210	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_TABLE	10.1.1.210 x \n10.1.1.60\nOFPT_	19
10.1.1.210	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_FLOW	10.1.1.210 x \n10.1.1.60\nOFPT_	19
10.1.1.210	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_GROUP_DESC	10.1.1.210 x \n10.1.1.60\nOFPT_	19

10.1.1.220	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_TABLE	10.1.1.220 x \n10.1.1.60\nOFPT_	19
10.1.1.220	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_FLOW	10.1.1.220 x \n10.1.1.60\nOFPT_	19
10.1.1.220	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_GROUP_DESC	10.1.1.220 x \n10.1.1.60\nOFPT_	19
10.1.1.220	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_GROUP	10.1.1.220 x \n10.1.1.60\nOFPT_	19

10.1.1.200	10.1.1.60	OFPT_BARRIER_REPLY	10.1.1.200 x \n10.1.1.60\nOFPT_	28
10.1.1.200	10.1.1.60	OFPT_MULTIPART_REPLY OFPMP_TABLE	10.1.1.200 x \n10.1.1.60\nOFPT_	19

Figure 19. The result of block 7 of the Python code

```

1 df_query = {"source": [],
2             "distenation" : [],
3             'new_message_info' : [],
4             "message_info" : [],
5             "message_info_count" : []
6         }
7 s = df1["Source"].unique()
8 for source in df1["Source"].unique():
9     #print(source)
10    df_source = df1.loc[df1['Source'] == source]
11    for dist in df_source["Destination"].unique():
12        #print(dist)

```

13	df_dist = df_source.loc[df1['Destination'] == dist]
14	for message in df_dist['Info'].unique():
15	#print(message)
16	df_message = (df_source.loc[df1['Info'] == message])
17	df_message_count = df_message.shape[0]
18	#print(df_message.shape[0])
19	#df_query[source] = {dist : {message:df_message_count}}
20	df_query["source"].append(source)
21	df_query["distenation"].append(dist)
22	df_query["new_message_info"].append(message)
23	df_query["message_info"].append(source+' x \n' + dist + '\n' + message)
24	df_query["message_info_count"].append(df_message_count)
25	df_query

The below table explains the functions of the messages between the controller and the network devices.

Table 4. Meaning of the OFPT Messages

	Name of the Message	Function of the Message
1	OFPT_MULTIPART_REQUEST, OFPMP_FLOW and its reply	A negotiation to exchange information about the rules (flows) that will be applied on the OvSwitches by the controller.
2	OFPT_MULTIPART_REQUEST, OFPMP_TABLE and its reply	Exchanging information about the tables that combine flows (rules). So, each flow belongs to a specific table. The default table is 0.
3	OFPT_MULTIPART_REQUEST, OFPMP_GROUP and its reply	Exchanging statistics about groups.
4	OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC and its reply	Exchanging information about the way of listing the groups' statistics.
5	OFPT_MULTIPART_REQUEST, OFPMP_METER and its reply	Exchanging statistics about meters.

6	OFPT_MULTIPART_REQUEST, OFPMP_METER_CONFIG and its reply	Exchanging statistics about meters' configuration.
7	OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS and its reply	Exchanging information related to port status of the switches.
8	OFPT_MULTIPART_REQUEST, OFPMP_QUEUE and its reply	Exchanging of queue statistics for the switches' ports.
9	OFPT_BARRIER_REQUEST and its reply	Control packets to ensure that the messages follow the correct order of checking before being executed.
10	OFPT_PACKET_IN and OFPT_PACKET_OUT	Packets to update the controller by live information related to OvSwitches and the topology. (IN) means toward the controller while (OUT) means going out from the controller.

Block 8: drawing graph for each line in figure 18.

1	<code>data_frame = pd.DataFrame(df_query)</code>
2	<code>sns.catplot(data=data_frame, y='message_info_count', row='message_info',</code>
3	<code>kind='bar', height=2, aspect=2)</code>
4	<code>plt.show()</code>

The result of this code is multiple graphs. Each line in figure 19 is represented by a graph.

Figure 20 till 31 are samples of the outputs.

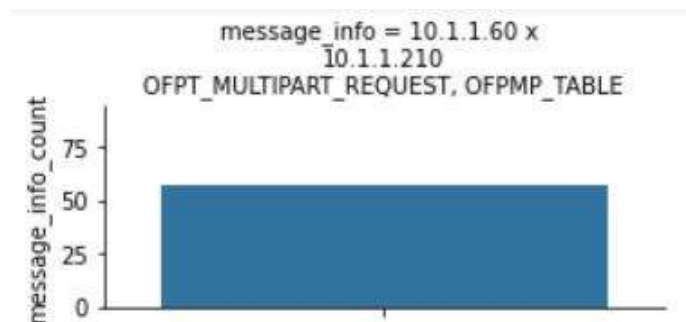


Figure 20. Output sample 1: a message between OVS3 and the controller repeated 57 times

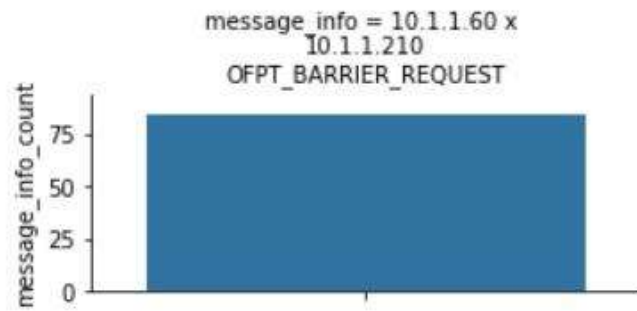


Figure 21. Output sample 2: a message between OVS3 and the controller repeated 84 times

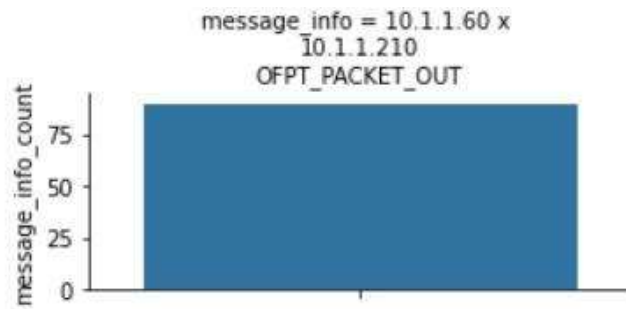


Figure 22. Output sample 3: a message between OVS3 and the controller repeated 90 times

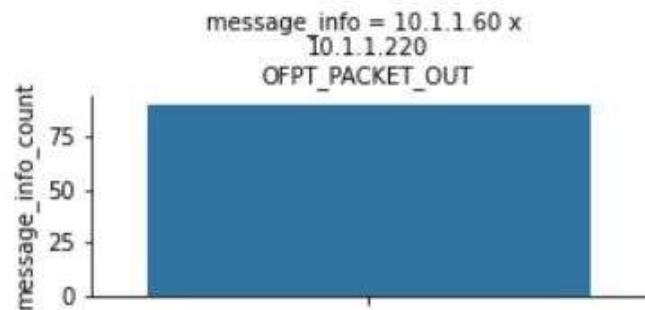


Figure 23. Output sample 4: a message between OVS2 and the controller repeated 90 times

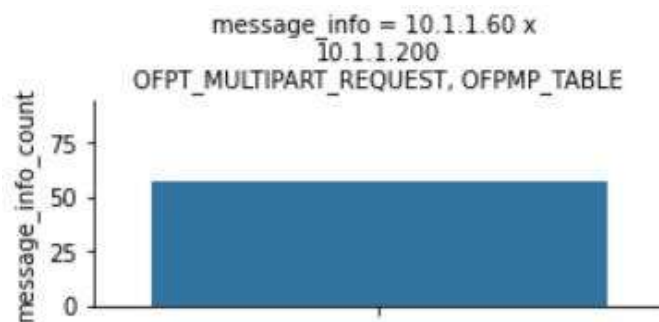


Figure 24. Output sample 5: a message between OVS1 and the controller repeated 90 times

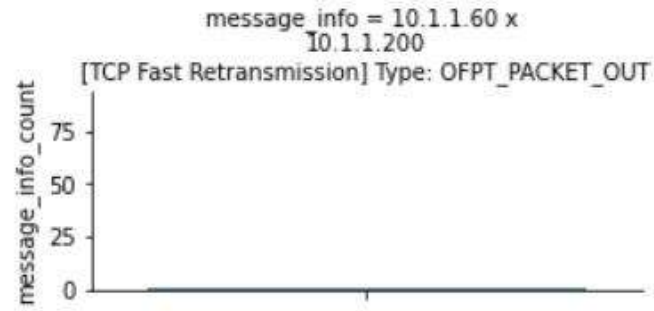


Figure 25. Output sample 6: a message between OVS1 and the controller appeared 1 time

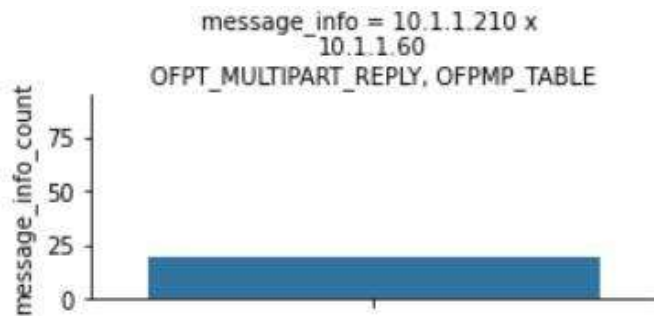


Figure 26. Output sample 7: a message between OVS3 and the controller repeated 19 times

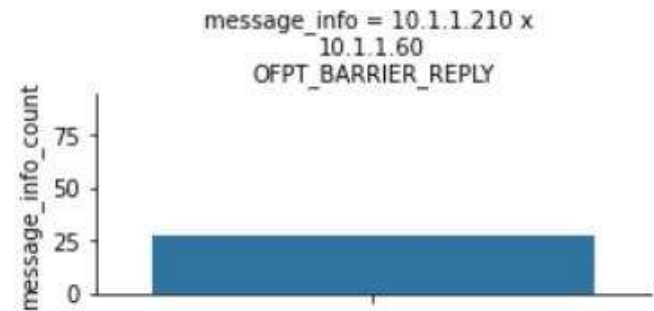


Figure 27. Output sample 8: a message between OVS3 and the controller repeated 28 times

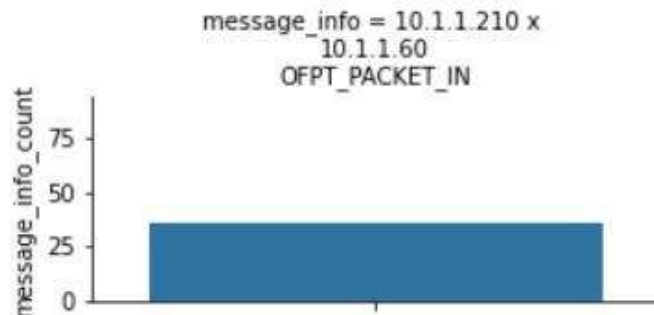


Figure 28. Output sample 9: a message between OVS3 and the controller repeated 36 times

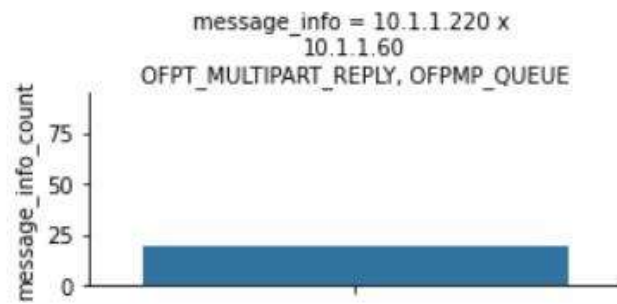


Figure 29. Output sample 10: a message between OVS2 and the controller repeated 19 times

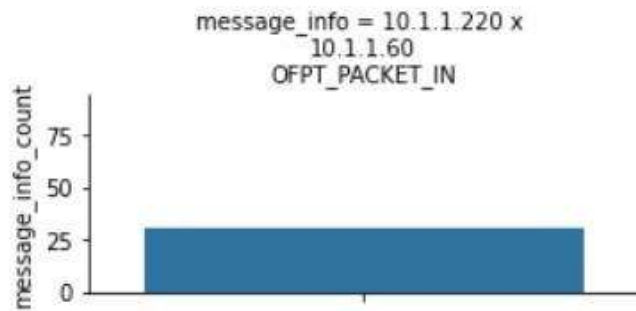


Figure 30. Output sample 11: a message between OVS2 and the controller repeated 31 times

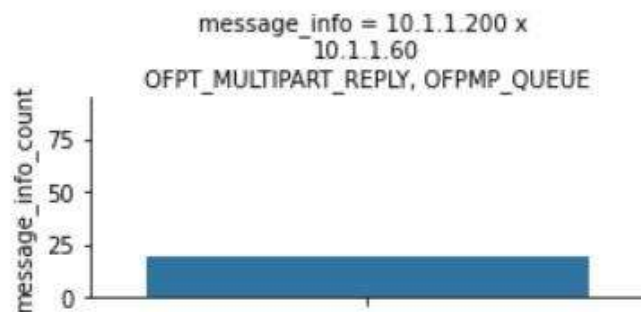


Figure 31. Output sample 12: a message between OVS1 and the controller repeated 19 times

3.3.2 Data analysis version 2 (check the effect of changing the flow rules)

I removed block 8 and edited Block 7 of the previous script to be as the below,

1	df_query = {"source": [],
2	"distenation" : [],

3	'new_message_info' : [],
4	"message_info" : [],
5	"message_info_count" : []
6	}
7	s = df1["Source"].unique()
8	for source in df1["Source"].unique():
9	#print(source)
10	df_source = df1.loc[df1['Source'] == source]
11	for dist in df_source["Destination"].unique():
12	#print(dist)
13	df_dist = df_source.loc[df1['Destination'] == dist]
14	for message in df_dist['Info'].unique():
15	#print(message)
16	df_message = (df_source.loc[df1['Info'] == message])
17	df_message_count = df_message.shape[0]
18	#print(df_message.shape[0])
19	#df_query[source] = {dist : {message:df_message_count}}
20	df_query["source"].append(source)
21	df_query["distenation"].append(dist)
22	df_query["new_message_info"].append(message)
23	df_query["message_info"].append(dist + '\n' + message)
24	df_query["message_info_count"].append(df_message_count)
25	message_info = df_query["message_info"]
26	message_info_count = df_query["message_info_count"]
27	fig = plt.figure(figsize = (10, 15))
28	color = ['red']
29	plt.barh(message_info, message_info_count, height = 0.4, color=color)
30	plt.ylabel("message type")
31	plt.xlabel("No. of message sent")

32	<code>plt.title('source %s' %source)</code>
33	<code>for index,data in enumerate(message_info_count):</code>
34	<code>plt.text(x=data+1 , y =index , s=f"{data}" , fontdict=dict(fontsize=10))</code>
35	<code>plt.tight_layout()</code>
36	<code>plt.show()</code>
37	<code>df_query.clear()</code>
38	<code>df_query = {"source": [],</code>
39	<code>"distenation" : [],</code>
40	<code>'new_message_info' : [],</code>
41	<code>"message_info" : [],</code>
42	<code>"message_info_count" : []</code>
43	<code>}</code>

This edition of the code is to create a single output figure for each source device. I applied this code to the topology shown in figure 32.

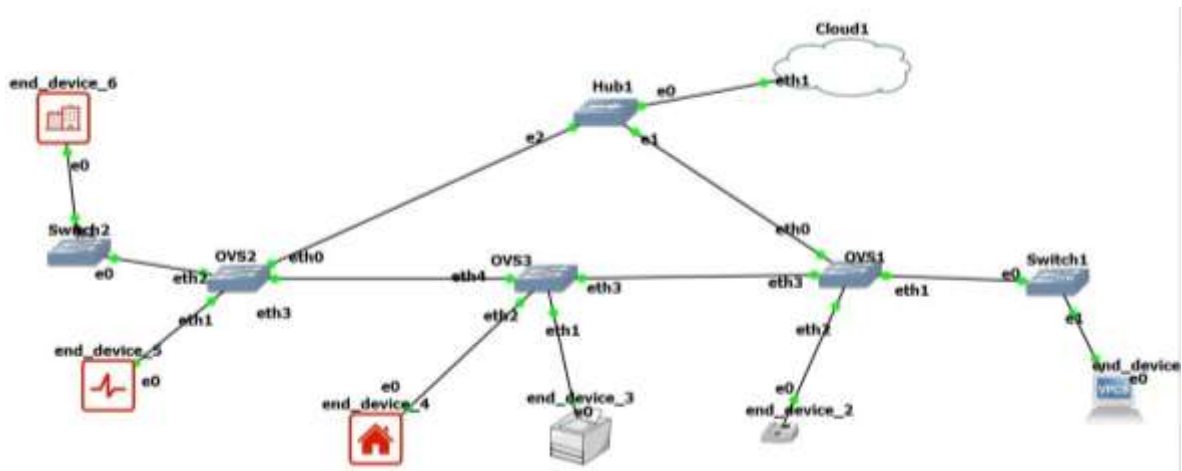


Figure 32. Topology "a" to which script version 2 was applied

For the above figure, before extracting the data via Wireshark and applying the script on the extracted data, I removed many allowed flows in the controller (Open Flow Manager). The removed flows are related to OvSwitch1 (10.1.1.200) to compare the output of this switch and OvSwitch2. The outputs are shown in figures 33, 34, and 35.

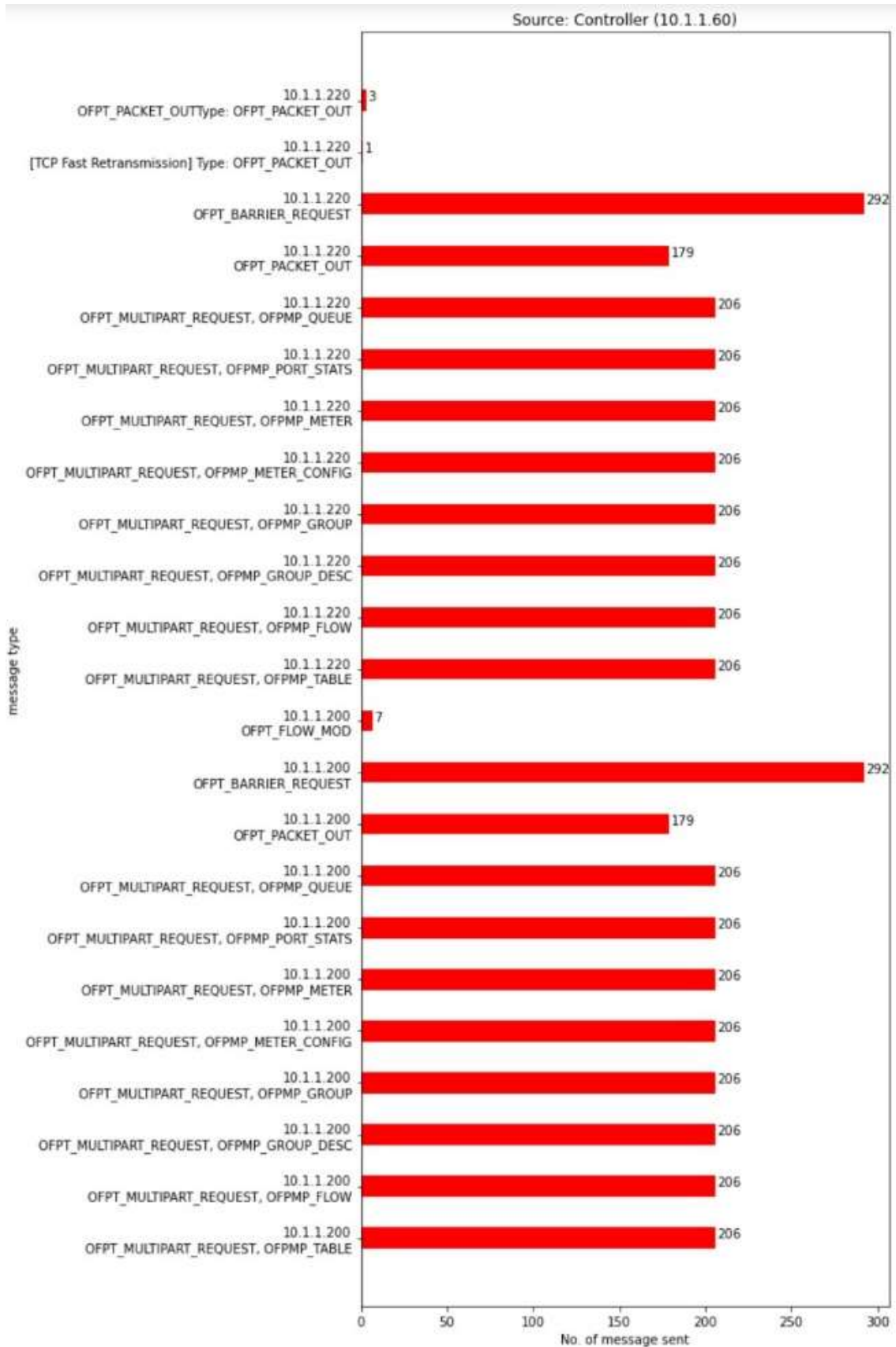


Figure 33. Output graph 1 of script version 2 (Messages from the controller)

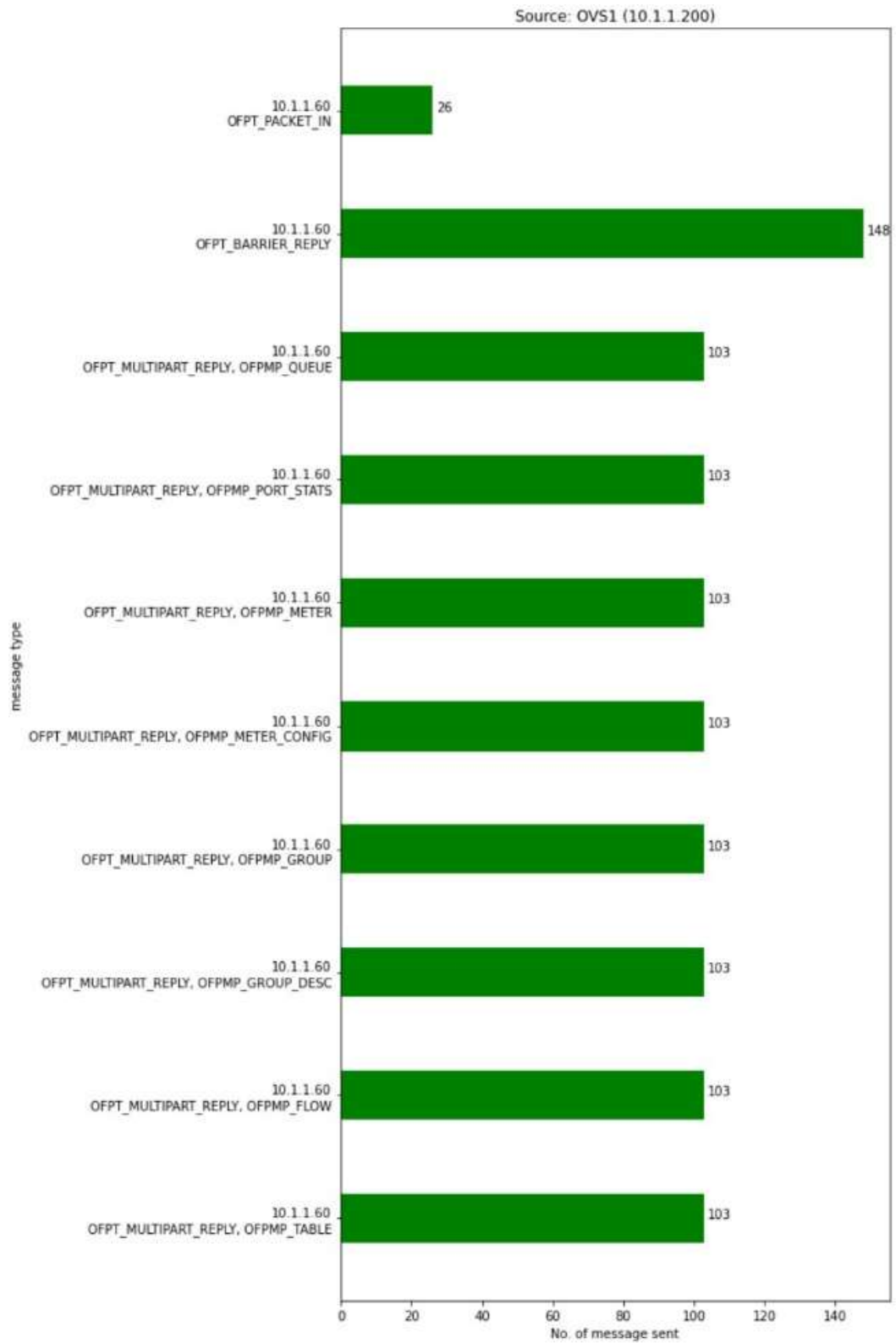


Figure 34. Output graph 2 of script version 2 (Messages from OVS1)

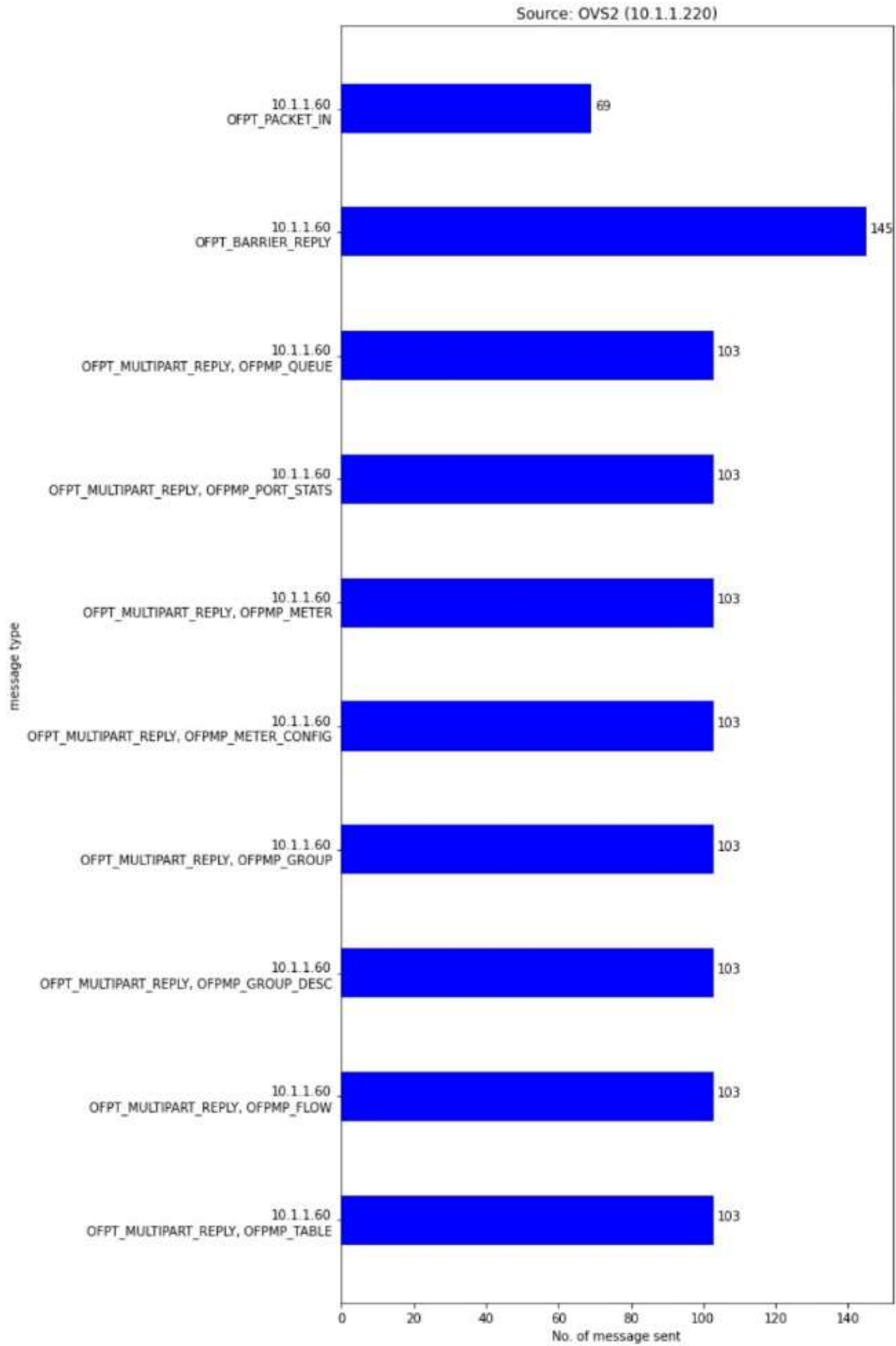


Figure 35. Output graph 3 of script version 2 (Messages from OVS2)

As shown in previous three figures, there are request messages generated from the controller and there are reply messages generated from OvSwitches toward the controller. If we compare between figure 34 and 35, we can conclude that reducing the allowed flows for OvSwitch1 (10.1.1.200) results in reducing the number of OFPT_PACKET_IN messages.

3.3.3 Data analysis version 3 (Flows analysis of Openflow's packets over time)

OFPT_PACKET_IN and OFPT_PACKET_OUT messages are the final steps of the communication between the controller and OvSwitches. Those messages occur only after the connection is established and OpenFlow protocol works properly. I extracted part of the output data from the topology in figure 32 and manually analysed the messages and drew them in time order as shown in figures 36 and 37.

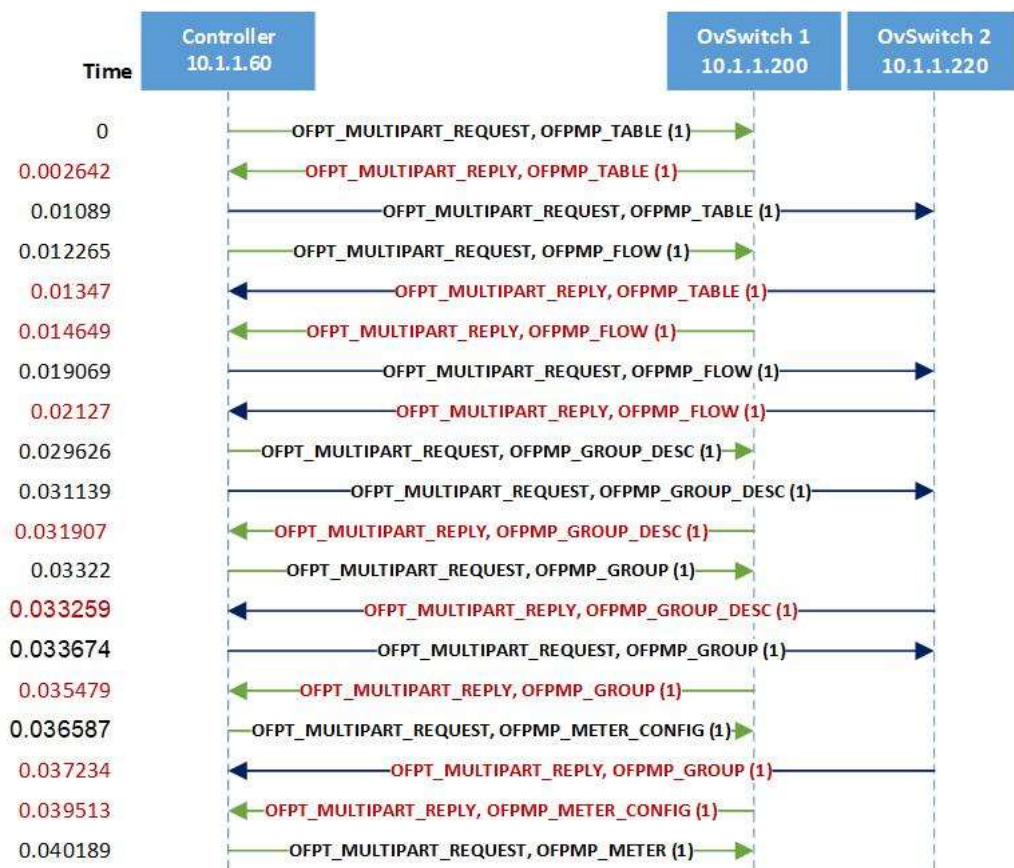


Figure 36. Controller's messages in Topology "a", part 1/2

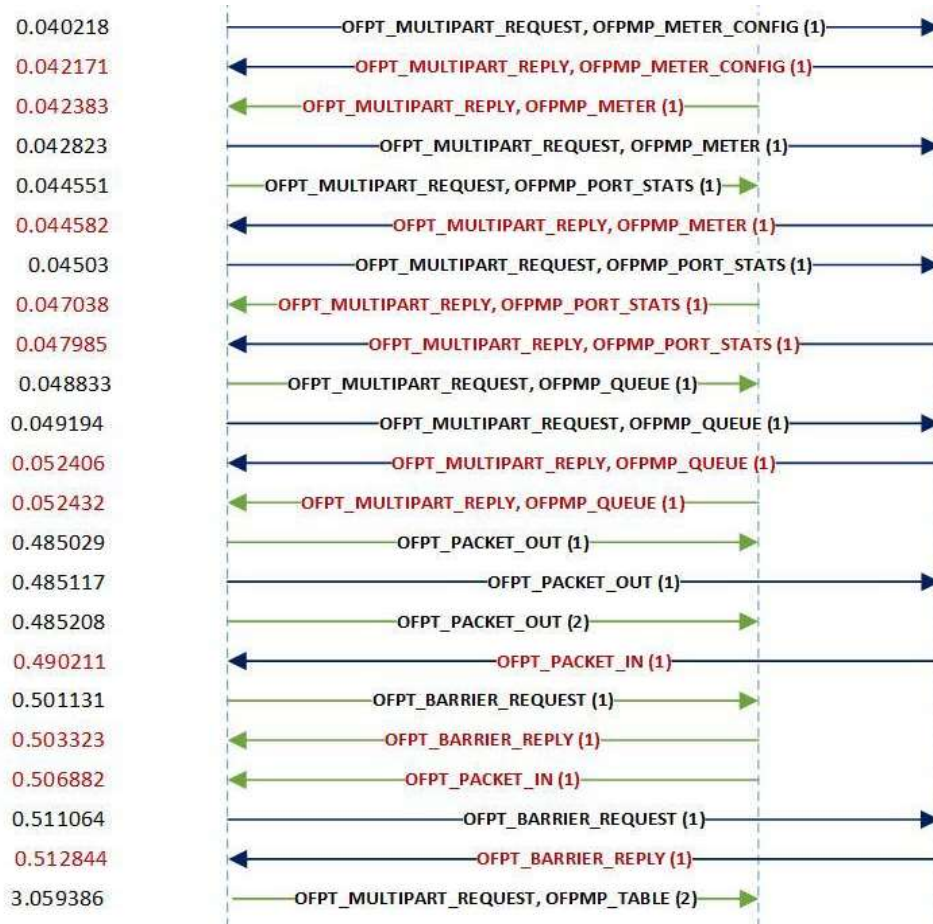


Figure 37. Controller's messages in Topology "a", part 2/2

I repeated the same previous manual analysis for a new topology that is shown in figure 38.

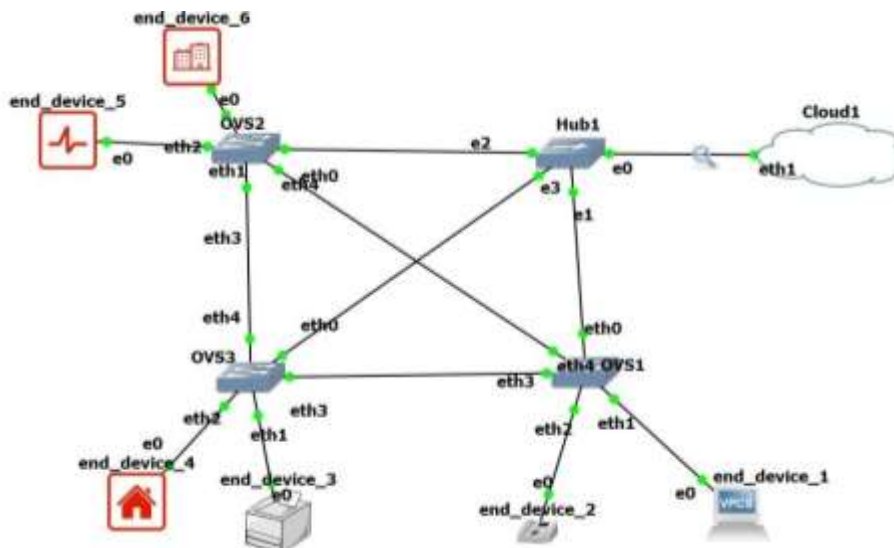


Figure 38. Topology "b"

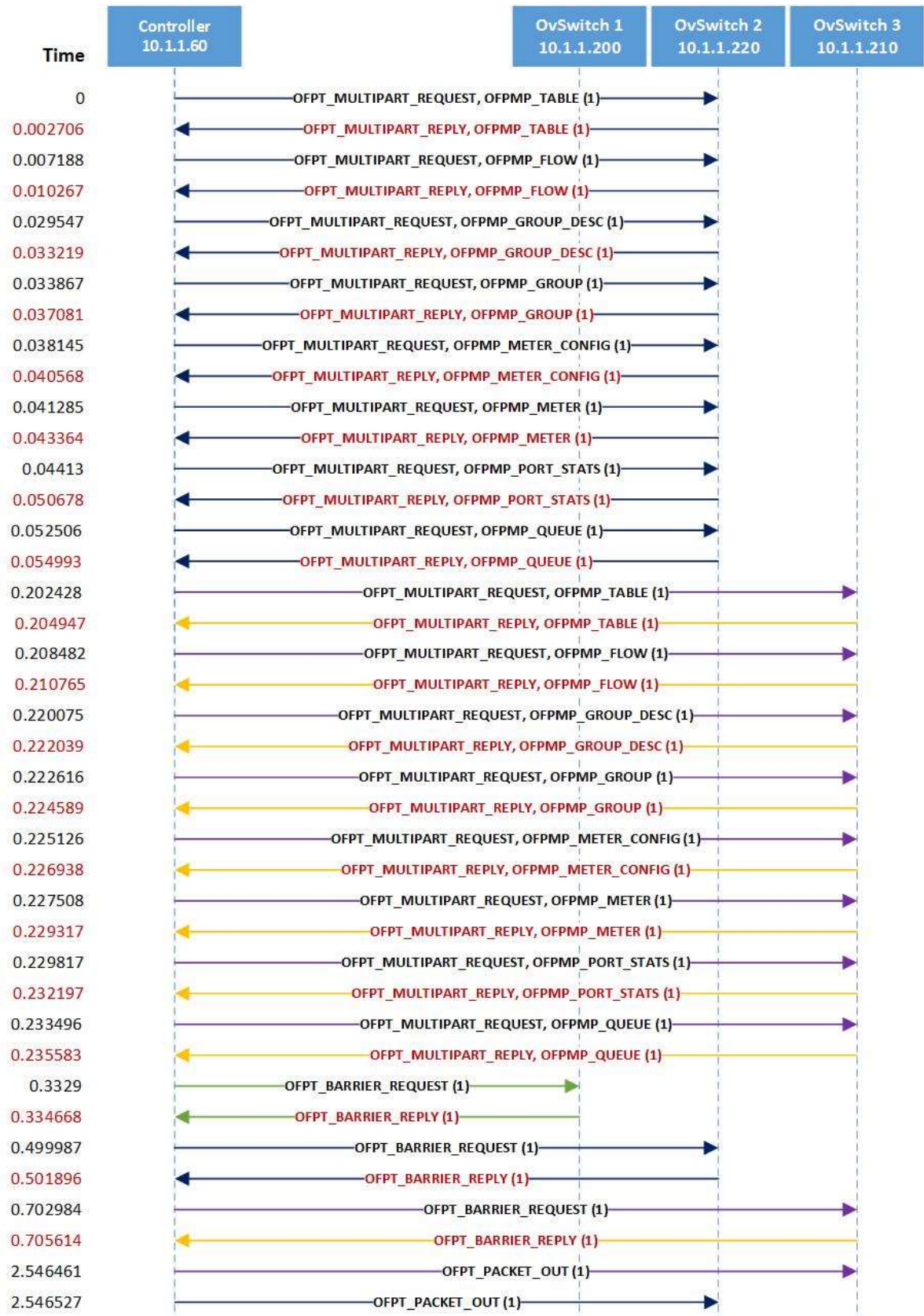


Figure 39. Controller's messages in Topology "b", part 1/2

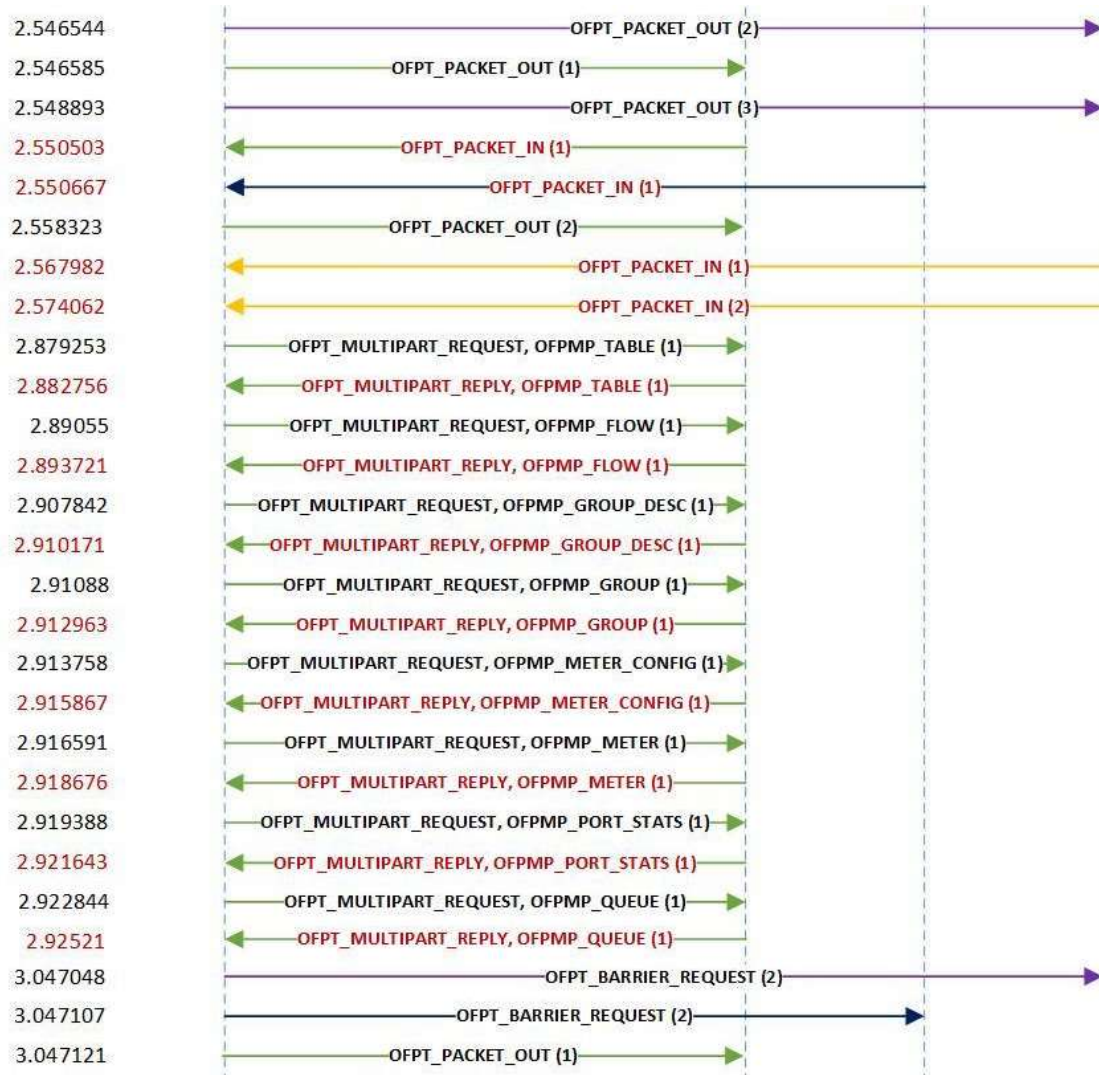


Figure 40. Controller's messages in Topology "b", part 2/2

3.4. Interpretation of the phenomenon

IoT result in network enlargement. As the network grows, managing and troubleshooting the network become tougher. Virtualization using an SDN solution makes the management of the network easier. What happened in my project is that the processing and forwarding functions of network devices were separated. Network devices (Open vSwitches) become just for forwarding the traffic without any processing. The OpenDayLight and Open Flow Manager software, which serves as the SDN controller, handle all processing. Openflow is the protocol that enables the SDN controller to manage devices. As shown in this chapter, the traffic captures contain Openflow packets. The flows are the policies that are configured in the controller, and the relevant processing took place in the controller as per those flows, as illustrated in figures

13 to 16. The switches' role is just to forward the traffic in accordance with the choices made by those flows. Additionally, in this project, the communication messages between the SDN controller and the network devices were extracted and processed in Python codes for analysis. The employed Python scripts demonstrated the behavior of those messages and how their amount was affected by changing the rules in the SDN controller. Figure 34 shows how the amount of OFPT_PACKET_IN packets from OVS1 (10.1.1.200) to the controller reduced due to reducing the allowed flows for this switch via the controller.

In Table 5 and Table 6, I represent the ratio of control and data messages conform to the following formulae:

$$r_i = NC_i / ND, i = 1, \dots, 11$$

Where r_i is the ratio of control messages type i , NC_i is the number of control messages type i and ND is the number of data plus control messages i ($i = 1, \dots, 11$), respectively.

Table 5 for topology “a” in which the measurements time duration was 311.80 seconds. Table 6 for topology “b” in which the measurement time duration was 95.38 seconds.

Table 5. Ratio of the No. of control packets/data plus control packets in % (Topology “a”)

Message type	OVS1, r [%]	OVS2, r [%]	CTRL, r [%]
OFPT_MULTIPART_REQUEST, OFPMP_TABLE and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_FLOW and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_GROUP and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_METER_CONFIG and its reply	5.79	5.17	5.05

OFPT_MULTIPART_REQUEST, OFPMP_METER and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_PORT_STATS and its reply	5.79	5.17	5.05
OFPT_MULTIPART_REQUEST, OFPMP_QUEUE and its reply	5.79	5.17	5.05
OFPT_BARRIER_REQUEST and its reply	8.32	7.28	7.16
OFPT_PACKET_IN and OFPT_PACKET_OUT	1.46	3.46	4.48
Total	61.89	52.27	57.09

The effect of reducing the allowed flows for OVS1 can explain the outputs of table 5 that the total control messages ratio of OVS2 is just 52.27% while it is 61.89% for OVS1. So, OVS2 has less control messages and more data messages as compared with OVS1. The data messages ratios are 47.73% and 38.11% for switches OVS2 and OVS1 respectively.

The ratio of OFPT_PACKET_IN control messages in OVS2 is more than twice as they are in OVS1. It is 3.46% and 1.46% in OVS2 and OVS1 respectively.

The controller has not only control messages but also data messages like HTTP. That's why the total control messages ratio of the controller is not 100% but 57.09%.

Table 6. Ratio of the No. of control packets/data plus control packets in % (Topology "b")

Message type	OVS1, r [%]	OVS2, r [%]	OVS3, r [%]	CTRL, r [%]
OFPT_MULTIPART_REQUEST, OFPMP_TABLE and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMP_FLOW and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMP_GROUP_DESC and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST,	5.00	5.12	5.00	5.30

OFPMMP_GROUP and its reply				
OFPT_MULTIPART_REQUEST, OFPMMP_GROUP_DESC	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMMP_METER_CONFIG and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMMP_METER and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMMP_PORT_STATS and its reply	5.00	5.12	5.00	5.30
OFPT_MULTIPART_REQUEST, OFPMMP_QUEUE and its reply	5.00	5.12	5.00	5.30
OFPT_BARRIER_REQUEST, and OFPT_BARRIER_REPLY	11.60	8.30	8.20	7.60
OFPT_PACKET_IN and OFPT_PACKET_OUT	3.34	3.30	4.50	5.22
Total	59.94	57.68	57.70	60.52

The outputs of table 6 can be explained by the fact that when we don't change things in the controller, all switches have about equal ratios of control messages.

4. Conclusions and applicability of the results in practice

This research intends to support the seamless implementation of IoT system in networks by testing a practical model. Network devices managed by an SDN virtualization solution were included in the tested model. The idea of separating the control and forwarding functions in network devices was tested. The tests and analysis that were accomplished in this project can be considered when implementing real large scale solutions.

The project's usage of data analysis and Python scripts may provide a picture of the management of networks and the extraction and analysis of traffic. This is the kernel for modern techniques in networks and IoT like, integrating Artificial Intelligence (AI) into networks. Moreover, similar data analysis used in this project can be developed in real environment for

security solutions by knowing the usual traffic and detecting any abnormal messages and generating alarms for potential security threats.

This research gives samples of extracting and evaluating control messages from different topologies. In topology “a” where the controller is connected to two switches (OVS1, and OVS2), the allowed flows for OVS1 were decreased. In topology “b” where the controller is connected to three switches (OVS1, OVS2, and OVS3), everything kept as the default in the controller. The changes in allowed flows didn’t effect on the number of OFPT_BARRIER_REQUEST, and OFPT_BARRIER_REPLY control messages as they are the mostly used in both samples. Except the OFPT_PACKET_IN and OFPT_PACKET_OUT, all other control messages are approximately within the same ratio.

In real world, service provider companies may face technical problem that affect the service due to the lack of organizing and managing the network. So, virtualization can enhance the service. Additionally, programmability might aid in a quicker time to problem discovery and resolution. Analyzing is important for design and maintain the network.

List of Figures

Figure 1. Three layers architecture of IoT	8
Figure 2. Five layers architecture of IoT	8
Figure 3. A generic SDN-based architecture for IoT	10
Figure 4. A standard SDN-IoT architecture with NFV	11
Figure 5. Technology convergence of the future era of IoT	15
Figure 6. VirtualBox software that contains 3 virtual machines	21
Figure 7. Configuring a virtual machines in VirtualBox to be connected to Microsoft KM-TEST Loopback adapter in my PC.....	21
Figure 8. Microsoft KM-TEST Loopback adapter.....	22
Figure 9. Created GNS3 topology	22
Figure 10. Configuring IP address in Open vSwitch in GNS3.....	23
Figure 11. OpenDayLight page in web browser	24
Figure 12. Open Flow Manager page in web browser (basic view tab).....	25
Figure 13. Open Flow Manager page in web browser (flow management tab)	26
Figure 14. Adding new flow in Open Flow Manager (step 1)	26
Figure 15. Adding new flow in Open Flow Manager (step 2, select the device)	27
Figure 16. Adding new flow in Open Flow Manager (step 3, specify flow's rules).....	27
Figure 17. Saving a Wireshark capture as a csv file.....	28
Figure 18. Screenshot of the saved csv file	29
Figure 19. The result of block 7 of the Python code	31
Figure 20. Output sample 1: a message between OVS3 and the controller repeated 57 times	33
Figure 21. Output sample 2: a message between OVS3 and the controller repeated 84 times	34
Figure 22. Output sample 3: a message between OVS3 and the controller repeated 90 times	34
Figure 23. Output sample 4: a message between OVS2 and the controller repeated 90 times	34
Figure 24. Output sample 5: a message between OVS1 and the controller repeated 90 times	34
Figure 25. Output sample 6: a message between OVS1 and the controller appeared 1 time	35
Figure 26. Output sample 7: a message between OVS3 and the controller repeated 19 times	35
Figure 27. Output sample 8: a message between OVS3 and the controller repeated 28 times	35
Figure 28. Output sample 9: a message between OVS3 and the controller repeated 36 times	35
Figure 29. Output sample 10: a message between OVS2 and the controller repeated 19 times	36
Figure 30. Output sample 11: a message between OVS2 and the controller repeated 31 times	36
Figure 31. Output sample 12: a message between OVS1 and the controller repeated 19 times	36

Figure 32. Topology "a" to which script version 2 was applied.....	38
Figure 33. Output graph 1 of script version 2 (Messages from the controller)	39
Figure 34. Output graph 2 of script version 2 (Messages from OVS1).....	40
Figure 35. Output graph 3 of script version 2 (Messages from OVS2).....	41
Figure 36. Controller's messages in Topology "a", part 1/2	42
Figure 37. Controller's messages in Topology "a", part 2/2	43
Figure 38. Topology "b"	43
Figure 39. Controller's messages in Topology "b", part 1/2.....	44
Figure 40. Controller's messages in Topology "b", part 2/2.....	45

List of Tables

Table 1. Comparisons of different operating systems (S: unconditional support, NS: Not support, PS: Partial support).....	6
Table 2. Followed steps to accomplish the project.....	20
Table 3. Identification of the nodes	23
Table 4. Meaning of the OFPT Messages	32
Table 5. Ratio of the No. of control packets/data plus control packets in % (Topology “a”)	46
Table 6. Ratio of the No. of control packets/data plus control packets in % (Topology “b”)	47

Acknowledgement

This work has been supported by QoS-HPC-IoT Laboratory at the University of Debrecen, Hungary.

References

- [1] IEEE, IoT Towards Definition Internet of Things Revision 1, 27 May, 2015
- [2] Deepika Navani; Sanjeev Jain; Maninder Singh Nehra , The Internet of Things (IoT): A Study of Architectural Elements. *IEEE Xplore*: 12 April 2018, DOI: 10.1109/SITIS.2017.83
- [3] Sarah A. Al-Qaseemi; Hajer A. Almulhim; Maria F. Almulhim; Saqib Rasool Chaudhry, IoT architecture challenges and issues: Lack of standardization, *IEEE Xplore*: 19 January 2017, DOI: 10.1109/FTC.2016.7821686
- [4] Iqbal Alam, Kashif Sharif, A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV, *ACM Computing Surveys*, April 2020, DOI: 10.1145/3379444
- [5] Carlos Gonzalez, Salim Mahamat Charfadine, Olivier Flauzac, and Florent Nolot. 2016. SDN-based security framework for the IoT in distributed grid. In *Proceedings of the 2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech'16)*. IEEE, 1–5. DOI: <https://doi.org/10.1109/splitech.2016.7555946>
- [6] Seungwon Shin, Phillip A. Porras, Vinod Yegneswaran, Martin W. Fong, Guofei Gu, and Mabry Tyson. FRESKO: Modular composable security services for software-defined networks. In *Proceedings of 20th Annual Network & Distributed System Security Symposium (NDSS'13)*, 2013.
- [7] Tong Xu, Deyun Gao, Ping Dong, Hongke Zhang, Chuan Heng Foh, and Han-Chieh Chao. 2017. Defending against new-flow attack in SDN-based Internet of Things. *IEEE Access* 5 (2017), 3431–3443. DOI: <https://doi.org/10.1109/access.2017.2666270>
- [8] Arunan Sivanathan, Daniel Sherratt, Hassan Habibi Gharakheili, Vijay Sivaraman, and Arun Vishwanath. 2016. Low-cost flow-based security solutions for smart-home IoT devices. In *Proceedings of the 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS'16)*. IEEE, 1–6. DOI: <https://doi.org/10.1109/ants.2016.7947781>
- [9] Hongda Li, Feng Wei, and Hongxin Hu. 2019. Enabling dynamic network access control with anomaly-based IDS and SDN. In *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization (SDN-NFVSec'19)*. ACM, 13–16. DOI: <https://doi.org/10.1145/3309194.3309199>
- [10] Mehdi Nobakht, Vijay Sivaraman, and Roksana Boreli. 2016. A host-based intrusion detection and mitigation framework for smart home IoT using OpenFlow. In *Proceedings of the 2016 11th International Conference on Availability, Reliability and Security (ARES'16)*. IEEE, 147–156. DOI: <https://doi.org/10.1109/ares.2016.64>

- [11] Keyur K Patel, Sunil M Patel, Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges, Computer Science, 2016, , DOI 10.4010/2016.1482
- [12] Ovidiu Vermesan, Next Generation Internet of Things – Distributed Intelligence at the Edge and Human-Machine Interactions, River Publishers, New York, DOI: <https://doi.org/10.1201/9781003338963>, 2022.