NATIONAL & KAPODISTRIAN UNIVERSITY OF ATHENS

THE SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATION TECHNOLOGY AND TELECOMMUNICATIONS

INTERDEPARTMENTAL GRADUATE PROGRAM IN MANAGEMENT AND ECONOMICS OF TELECOMMUNICATION NETWORKS

THESIS

Software Defined Networking
OpenDaylight
Experimentation and Business Case

Achilleas P Grigoriadis

Supervisors: **Stathes Hadjiefthymiades,** Associate Professor
Dimitris Varoutas, Assistant Professor

ATHENS

DECEMBER 2014

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΟΙΚΟΝΟΜΙΚΗ ΤΩΝ ΤΗΛΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δίκτυα Καθοριζόμενα στο Λογισμικό OpenDaylight Διεξαγωγή Πειράματος και Επιχειρηματική Περίπτωση

Αχιλλέας Π Γρηγοριάδης

Επιβλέποντες: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής
Δημήτρης Βαρουτάς, Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2014

THESIS

Software Defined Networking OpenDaylight Experimentation and Business Case

Achilleas P Grigoriadis
R.N: MOP368

DECEMBER 2014

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δίκτυα Καθοριζόμενα στο Λογισμικό OpenDaylight Διεξαγωγή Πειράματος και Επιχειρηματική Περίπτωση

Αχιλλέας Π Γρηγοριάδης
Α.Μ: ΜΟΠ368

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ευστάθιος Χατζηευθυμιάδης**, Αναπληρωτής Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής
Δημήτρης Βαρουτάς, Επίκουρος Καθηγητής

ΔΕΚΕΜΒΡΙΟΣ 2014

ABSTRACT

Meeting current market requirements is almost impossible with traditional network architectures. Faced with flat or reduced budgets, enterprise IT and network departments are trying to squeeze the most from their networks using device management tools and manual timeconsuming processes. Software defined networking is the key solution to this situation. Software Defined Networking (SDN) is a new emerging network architecture where network control is decoupled from forwarding and is directly programmable. This migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity. Responsible for the control and orchestration of all these new functionalities will be the “Controller”. One of the most complete and fully operational controllers is OpenDaylight. OpenDaylight is an open source project with a modular, pluggable, and flexible controller platform at its core.

The OpenDaylight Project is a collaborative open source project that aims to accelerate adoption of Software-Defined Networking (SDN) and create a solid foundation for Network Functions Virtualization (NFV) for a more transparent approach that fosters new innovation and reduces risk.

The thesis begins exploring SDN technology and presents the basic architectural principles. Openflow protocol and Openflow enabled devices specifications will also be introduced. Following there is an analysis and a deep dive into OpenDaylight platform and its architectural framework and processes. A lab experiment will be performed as a practical example in the mentioned theory. Series of tests indicating the expected operation of SDN architecture will be performed using Openflow enabled switch and OpenDaylight controller platform. Finally follows an effort to give a financial aspect of how we can take advantage of SDN comparing it with standard networking technologies.

SDN offers a variety of new services that can be easily deployed, implemented and operate. Applications and services that now seem infeasible with SDN is just a challenge. Both providers and enterprises will benefit from these new services as revenues increase and costs reduce.

SUBJECT AREA: Software Defined Networking

KEYWORDS: Software Defined Networking, OpenDaylight, Openflow, SDN Controller, Northbound Interface

ΠΕΡΙΛΗΨΗ

Η ικανοποίηση των απαιτήσεων της σημερινής αγοράς είναι σχεδόν αδύνατη με τη χρήση των καθιερωμένων αρχιτεκτονικών δικτύωσης. Αντιμέτωποι με σταθερά ή χαμηλά Budget τα τμήματα πληροφορικής και δικτύων των επιχειρήσεων και των παρόχων προσπαθούν να συμπιέσουν όλο και περισσότερο τις υποδομές τους χρησιμοποιώντας εξειδικευμένα εργαλεία διαχείρισης και χρονοβόρες χειροκίνητες διαδικασίες. Η συμπίεση αυτή οδηγάει σιγά σιγά στην παροχή μη ολοκληρωμένων λύσεων και δεν ευνοεί την δημιουργία νέων αποδοτικών υπηρεσιών Η τεχνολογία Software Define Networking (SDN) είναι το κλειδί στην επίλυση του άνωθεν προβλήματος. Η συγκεκριμένη τεχνολογία είναι μια αναδυόμενη αρχιτεκτονική δικτύωσης στην οποία ο έλεγχος (control plane) έχει διαχωριστεί από την προώθηση (data plane) και είναι άμεσα προγραμματιζόμενος. Αυτή η μεταγωγή του ελέγχου από τις συσκευές δικτύου σε υπολογιστικές μηχανές προσφέρει τη δυνατότητα στην υποκείμενη υποδομή να αποσπαστεί για υλοποίηση εφαρμογών και δικτυακών υπηρεσιών, από την οποία προκύπτει ότι το δίκτυο πλέον φαίνεται σαν μια λογική ή εικονική οντότητα. Υπεύθυνος για τον έλεγχο και την ενορχήστρωση όλων αυτών των δυνατοτήτων θα είναι ο controller. Ένας από τους πιο ολοκληρωμένους και πλήρως λειτουργικούς controllers είναι ο OpenDaylight. Το OpenDaylight είναι ένα ανοιχτού κώδικα project με έναν ευέλικτο και ικανό τμηματοποίησης και απόσπασης controller στον πυρήνα του.

Το OpenDaylight project είναι ένα συνεργατικό project ανοικτού κώδικα που στοχεύει στην γρηγορότερη υιοθέτηση της τεχνολογίας Software Defined Networking. Ένας ακόμη στόχος του είναι να δημιουργήσει μια σταθερή βάση για την υλοποίηση Εικονικοποιημένων Δικτυακών Λειτουργιών η οποία προωθεί την καινοτομία και μειώνει το ρίσκο.

Η διπλωματική ξεκινάει εξερευνώντας την τεχνολογία SDN και παρουσιάζοντας τις βασικές αρχές της αρχιτεκτονικής της. Θα παρουσιαστούν επίσης το πρωτόκολλο Openflow και τα switch που είναι ικανά να το υποστηρίξουν (Openflow enabled). Θα ακολουθήσει μια ανάλυση του πλαισίου λειτουργίας της πλατφόρμας OpenDaylight παρουσιάζοντας πολλά από τα χαρακτηριστικά της καθώς επίσης και σειρά από τεστ σε περιβάλλον εργαστηρίου που αποδεικνύουν την αναφερθείσα λειτουργία του controller. Η ανάλυση κλείνει με μια προσπάθεια να δώσουμε την οικονομική άποψη και τα οφέλη που παρέχει η εν λόγω τεχνολογία δικτύωσης σε σύγκριση με τις υπάρχουσες.

Η τεχνολογία SDN προσφέρει μια πλειάδα νέων υπηρεσιών και εφαρμογών οι οποίες εύκολα αναπτύσσονται, εφαρμόζονται και λειτουργούν. Τέτοιου είδους εφαρμογές και λειτουργίες που σήμερα φαίνονται αδύνατες υλοποίησης με την τεχνολογία SDN είναι απλά μια πρόκληση. Οι πάροχοι και επιχειρήσεις θα επωφεληθούν καθώς αυτές οι νέες υπηρεσίες θα αυξήσουν τα έσοδά τους και θα μειώσουν το κόστος τους.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Software Defined Networking

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Software Defined Networking, OpenDaylight, Openflow, SDN Controller, Northbound Interface

Dedicated to my family and friends

Only Sky is the Limit

ACKNOWLEDGEMENTS

First of all, I would like to thank my Master's thesis advisor, Mr. Stathes Hadjiefthymiades, for his support and guidance.

I am especially grateful to my colleagues and friends for being patient and providing me with their insightful comments. Also Stavroula; she always believed in me and encouraged me in difficult times.

Finally, many thanks to my family for being always close.

TABLE OF CONTENTS

INTRODUCTION **16**

1. PREFACE **17**

1.1 Why SDN? 17

2. INTRODUCING SOFTWARE DEFINED NETWORKING **20**

3. OPENFLOW **22**

3.1 Openflow protocol messaging 23

4. OPENFLOW SWITCH **24**

4.1 Dedicated Openflow and Openflow enabled switches 24

4.2 Controller 26

4.3 Flow Table 27

- 4.3.1 Header Fields 27
- 4.3.2 Counters 28
- 4.3.3 Actions 29
- 4.3.4 Matching 31

4.4 Secure Channel 32

4.5 Openflow Protocol Overview 32

- 4.5.1 Controller-to-Switch 32
- 4.5.2 Asynchronous 33
- 4.5.3 Symmetric 33

4.6 Connection Setup 33

4.7 Connection Interruption 34

4.8 Encryption 34

4.9 Spanning Tree 34

4.10 Flow Table Modification Messages 35

4.11 Flow Removal 36

5. OPENFLOW CONTROLLER 37

5.1 Location 37

5.2 Flow 37

5.3 Behavior 37

6. OPENDAYLIGHT 38

6.1 OpenDaylight Hellium 39

6.2 Architectural Principles 42

6.3 Architectural Framework 44

6.4 Functional Overview 45

6.5 Evolution of the Controller Service Abstraction Layer 47

6.6 Switch Manager 48

6.7 GUI 48

6.8 High Availability 48

6.9 Topologies 50

6.10 Openflow Plugin Proposal 51

6.11 AD-SAL 51

6.12 MD-SAL: Architecture 53

6.12.1 MD SAL and AD SAL 54

6.12.2 MD-SAL Plugin 55

6.12.3 Plugin Development Process 56

6.12.4 "Flow Deleted" notification scenario 57

6.12.5 "Add Flow" scenario via NB REST API invocation 58

6.13 Producers and Consumers. Southbound and Northbound SAL plugins. 59

7. SDN OPENFLOW LAB 60

8. USE CASE FINANCIAL PROPOSAL 75

9. CONCLUSIONS 79

ABBREVIATIONS – ACRONYMS........ 81

REFERENCES........ 84

INDEX OF DIAGRAMS

Diagram 1: SDN and standard networking architecture revenue....................................78

INDEX OF PICTURES

Picture 1: Software define networking architecture logical view 20

Picture 2: Openflow switch example 26

Picture 3: Openflow commercial switches and access points.......................................26

Picture 4: Matching a packet 31

Picture 5: OpenDaylight Hydrogen controller platform..39

Picture 6: OpenDaylight Helium controller platform..40

Picture 7: OpenDaylight controller structure 44

Picture 8: OpenDaylight controller service abstraction layer 45

Picture 9: OpenDaylight controller service abstraction layer with Openflow plugin........46

Picture 10: Southbound plugins..47

Picture 11: Access information through northbound APIs 48

Picture 12: OpenDaylight controller web interface. Topology manager 51

Picture 13: Protocol plugins to Services 52

Picture 14: Application..53

Picture 15: AD-SAL and MD-SAL..54

Picture 16: Plugin development process 56

Picture 17: Flow deleted notification..57

Picture 18: Add flow through northbound REST API 58

Picture 19: Software engineer’s view of SAL 59

Picture 20: Lab design..60

Picture 21: Lab topology. Devices tab 61

Picture 22: Flows installed. Flow tab 62

Picture 23: Flows and communication verification 63

Picture 24: Add flow entry. OpenDaylight web interface 63

Picture 25: Flow1 details. Flows tab 64

Picture 26: Flow2 details. Flows tab 64

Picture 27: Flow details and statistics. Troubleshoot tab 65

Picture 28: Port details and statistics. Troubleshoot tab 66

Picture 29: Add new flow and stop communication 66

Picture 30: Mikrotik Openflow Flow table..67

Picture 31: Mikrotik Openflow Port statistics..67

Picture 32: Node information XML. Northbound interface 68

Picture 33: Flow details XML. Northbound interface...72

Picture 34: Port information XML. Northbound interface 73

Picture 35: OpenDaylight OSGi...74

Picture 36: Bandwidth Usage 76

INDEX OF TABLES

Table 1: Fields from packets used to match against flow entries....................................27

Table 2: Field lengths and the way they must be applied to flow entries........................28

Table 3: Required set of counters..28

Table 4: Actions...30

Table 5: NPV1 upgrading standard network architecture 76

Table 6: Enterprise's additive cost per year...77

Table 7: NPV2 SDN network architecture 77

Table 8: Revenue comparison. Standard vs SDN architecture......................................78

Introduction

This Thesis serves as the final outcome of my Master's Degree in Economics and Management of Telecommunications Networks (University of Athens).

The topic has been chosen based on the knowledge and experience I obtained in this two-year's course and focuses in presenting a new telecommunication's technology that will induce important changes on the computer networking universe.

The research, development, composition and final editing of this Thesis took place in Athens, Greece, between August and December 2014.

Its scope is to present and analyze Software defined networking (SDN); the most innovative technology of the last thirty years in Networking and Telecom industry.

My aim is to create a handful textbook for those interested in Networking and Telecommunications and encourage them to get further involved in the pioneering SDN theory and practices especially these using the powerful OpenDaylight controller.

1. PREFACE

After decades of data-networking as a decentralized and distributed process, recent years have seen the development of methods for centralizing data network control. The umbrella term for this centralizing of network control is “Software Defined Networking” (SDN). This term refers to the fact that software applications, like those that move virtual servers around in a data center, or orchestrate file transfers between data centers, define the operation of switching and routing nodes in the network.

This distributed model of networking in the Internet has been incredibly successful. An Internet that had just one central point of control could not have grown in the way the current Internet has. The way Internet operates is astonishing, because it seems almost organic – new network nodes graft on with a minimum of trouble, broken links self-heal, and the entire network spreads seamlessly over international borders.

This lack of a central point of control is absolutely key to the Internet’s reliability. The idea of ‘shutting down the Internet’ or ‘breaking the Internet’ has become the source of plenty of jokes, due to its sheer impossibility. These same control protocols that are used in the Internet have also taken hold in local and private networks over the last 15 years or so. There is no fundamental reason why an office LAN should use the same network control methods as the global Internet. It has just evolved that way because of ‘economies of scale’ in technical knowledge – engineers skilled in developing and operating Internet networking equipment can apply those same skills to LANs and private WANs.[1]

1.1 Why SDN?

The explosion of mobile devices and content, server virtualization, cloud services and big data are among the trends driving the networking industry to reexamine traditional network architectures. Many conventional networks are hierarchical, built with tiers of Ethernet switches arranged in a tree structure. This design made sense when client-server computing was dominant, but such a static architecture is ill-suited to the dynamic computing and storage needs of today’s enterprise data centers, campuses, and carrier environments. Some of the key computing trends driving the need for a new network paradigm include:

• Changing traffic patterns: Within the enterprise data center, traffic patterns have changed significantly. We use to have client-server applications where the bulk of the communication occurs between one client and one server while today’s applications access different databases and servers, creating a flurry of “east-west” machine-to-machine traffic before returning data to the end user device. At the same time, users are changing network traffic patterns as they push for access to corporate content and applications from any type of device (including their own), connecting from anywhere, at any time. Finally, many enterprise data centers managers are contemplating a utility computing model, which might include a private cloud, public cloud, or some mix of both, resulting in additional traffic across the wide area network.

• The “consumerization of IT”: Users are increasingly employing mobile personal devices such as smartphones, tablets, and notebooks to access the corporate network. New smart apps tend be more popular day by day.IT is under pressure to accommodate these personal devices in a fine-grained manner while protecting corporate data and intellectual property and meeting compliance mandates.

• Cloud services: Enterprises have enthusiastically embraced both public and private cloud services, resulting in unprecedented growth of these services. Enterprise business units now want the agility to access applications, infrastructure, and other IT resources on demand and à la carte. To add to the complexity, IT's planning for cloud services must be done in an environment of increased security, compliance, and auditing requirements, along with business reorganizations, consolidations, and mergers that can change assumptions overnight. Providing self-service provisioning, whether in a private or public cloud, requires elastic scaling of computing, storage, and network resources, ideally from a common viewpoint and with a common suite of tools.

• "Big data" means more bandwidth: Handling today's "big data" or mega datasets requires massive parallel processing on thousands of servers, all of which need direct connections to each other. The rise of mega datasets is fueling a constant demand for additional network capacity in the data center. Operators of high scale data center networks face the task of scaling the network to previously unimaginable size, maintaining any-to-many and any-to-any connectivity without disruptions.

1.2 Limitations of Current Networking Technologies

Meeting current market requirements is virtually impossible with traditional network architectures. Faced with flat or reduced budgets, enterprise IT departments are trying to squeeze the most from their networks using device-level management tools and manual processes. Carriers face similar challenges as demand for mobility and bandwidth explodes; profits are being eroded by escalating capital equipment costs and flat or declining revenue. Existing network architectures were not designed to meet the requirements of today's users, enterprises, and carriers; rather network designers are constrained by the limitations of current networks, which include:

• Complexity that leads to stasis: Networking technology to date has consisted largely of discrete sets of protocols designed to connect hosts reliably over arbitrary distances, link speeds, and topologies. To meet business and technical needs over the last few decades, the industry has tried to evolve current networking protocols to deliver higher performance and reliability, broader connectivity, and more security.

Protocols tend to be defined in isolation, however, with each solving a specific problem and without the benefit of any fundamental abstractions. This has resulted in one of the primary limitations of today's networks: complexity. For example, to add or move any device, operations engineers must touch multiple devices such as: switches, routers, firewalls, authentication servers, etc. and update ACLs, VLANs, quality of services (QoS), and other protocol-based mechanisms using the proper management tools. In addition, network topology, vendor switch model, and software version all must be taken into account. Due to this complexity, today's networks are relatively static as IT seeks to minimize the risk of service disruption.

The static nature of networks is in stark contrast to the dynamic nature of today's server environment, where server virtualization has greatly increased the number of hosts requiring network connectivity and fundamentally altered assumptions about the physical location of hosts. Prior to virtualization, applications resided on a single server and primarily exchanged traffic with select clients. Today, applications are distributed across multiple virtual machines (VMs), which exchange traffic flows with each other. VMs migrate to optimize and rebalance server workloads, causing the physical end points of existing flows to change (sometimes rapidly) over time. VM migration challenges many aspects of traditional networking, from addressing schemes and namespaces to the basic notion of a segmented, routing-based design.

In addition to adopting virtualization technologies, many enterprises today operate an IP converged network for their triple play service (voice, data, and video traffic). While existing networks can provide differentiated QoS levels for different services, the provisioning of those resources is highly manual. IT must configure each vendor's equipment separately, and adjust parameters such as ACLs and QoS on a per-session, per-application basis. Because of its static nature, the network cannot dynamically adapt to changing traffic, application, and user demands.

Inconsistent policies: To implement a network-wide policy, IT may have to configure thousands of devices and mechanisms. For example, every time a new virtual machine is brought up, it can take hours, in some cases days, for IT to reconfigure ACLs across the entire network. The complexity of today's networks makes it very difficult for IT to apply a consistent set of access, security, QoS, and other policies to increasingly mobile users, which leaves the enterprise vulnerable to security breaches, non-compliance with regulations, and other negative consequences.

Inability to scale: As demands on the data center rapidly grow, so too must the network grow. However, the network becomes vastly more complex with the addition of hundreds or thousands of network devices that must be configured and managed. IT has also relied on link oversubscription to scale the network, based on predictable traffic patterns; however, in today's virtualized data centers, traffic patterns are incredibly dynamic and therefore unpredictable.

Mega-operators, such as Google, Yahoo!, and Facebook, face even more daunting scalability challenges. These service providers employ large-scale parallel processing algorithms and associated datasets across their entire computing pool. As the scope of end-user applications increases (for example, crawling and indexing the entire world wide web to instantly return search results to users), the number of computing elements explodes and data-set exchanges among compute nodes can reach petabytes. These companies need so-called hyperscale networks that can provide high-performance, low-cost connectivity among hundreds of thousands— potentially millions—of physical servers. Such scaling cannot be done with manual configuration.

To stay competitive, carriers must deliver ever-higher value, better-differentiated services to customers. Multi-tenancy further complicates their task, as the network must serve groups of users with different applications and different performance needs. Key operations that appear relatively straightforward, such as steering a customer's traffic flows to provide customized performance control or on-demand delivery, are very complex to implement with existing networks, especially at carrier scale. They require specialized devices at the network edge, thus increasing capital and operational expenditure as well as time-to-market to introduce new services.

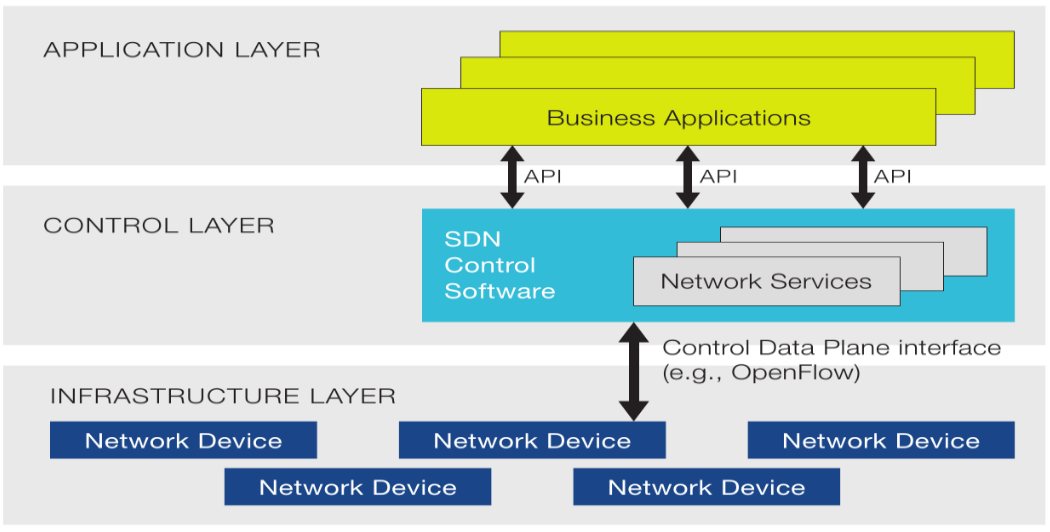
Vendor dependence: Carriers and enterprises seek to deploy new capabilities and services in rapid response to changing business needs or user demands. However, their ability to respond is hindered by vendors' equipment product cycles, which can range to three years or more. Lack of standard, open interfaces limits the ability of network operators to tailor the network to their individual environments.

This mismatch between market requirements and network capabilities has brought the industry to a tipping point. In response, the industry has created the Software-Defined Networking (SDN) architecture and is developing associated standards [2].SDN has great potential to change the way networks operate, and Openflow in particular has been touted as a "radical new idea in networking" [3].

2. INTRODUCING SOFTWARE DEFINED NETWORKING

Software Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. This migration of control, formerly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity.

Picture 1 depicts a logical view of the SDN architecture. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network and make the proper networking decisions. As a result, the network appears to the applications and policy engines as a single, logical switch. With SDN, enterprises and carriers gain vendor-independent control over the entire network from a single logical point, which greatly simplifies the network design and operation. SDN also greatly simplifies the network devices themselves, as they no longer need to understand and process thousands of protocol standards and implementations but only accept instructions from the SDN controllers.



Picture 1: Software define networking architecture logical view

Perhaps most importantly, network operators and administrators can programmatically configure this simplified network abstraction rather than having to hand-code tens of thousands of lines of configuration scattered among thousands of devices. In addition, leveraging the SDN controller’s centralized intelligence, IT can alter network behavior in real-time and deploy new applications and network services in a matter of hours or days, rather than the weeks or months needed today. By centralizing network state in the control layer, SDN gives network engineers the flexibility to configure, manage, secure, and optimize network resources via dynamic, automated SDN applications. Moreover, they can write these applications themselves and not wait for features to be embedded in vendors’ proprietary and closed software environments in the middle of the network.

In addition to abstracting the network, SDN architectures support a set of APIs that make it possible to implement common network services, including routing, multicast, security, access control, bandwidth management, traffic engineering, quality of service, processor and storage optimization, energy usage, and all forms of policy management, custom tailored to meet business objectives. For example, SDN architecture makes it easy to define and enforce consistent policies across both wired and wireless connections on a campus.

Likewise, SDN makes it possible to manage the entire network through intelligent orchestration and provisioning systems. The Open Networking Foundation is studying open APIs to promote multi-vendor management, which opens the door for on-demand resource allocation, self-service provisioning, truly virtualized networking, and secure cloud services.

Thus, with open APIs between the SDN control and applications layers, business applications can operate on an abstraction of the network, leveraging network services and capabilities without being tied to the details of their implementation. SDN makes the network not so much “application-aware” as “application-customized” and applications not so much “network-aware” as “network-capability-aware”. As a result, computing, storage, and network resources can be optimized [2].

3. OPENFLOW

Openflow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture. Over time, many software defined networking (SDN) protocols will likely emerge, but for now, the Openflow is the mostly commonly used SDN protocol. Openflow currently operates in the almost all SDN aware devices. In an SDN with a centralized control plane, the Openflow protocol carries the message between SDN controllers and the underlying network infrastructure, bringing network applications to life. So far, vendors and enterprises have made swift advancements in Openflow product development and network design strategies.[4] Openflow, a switching technology that began in 2008 as a Stanford University research project, is now gathering a lot of interest from network device vendors and managers of large switched networks. With the Openflow protocol, a form of software-defined networking, a network can be managed as a whole rather than as a number of individual devices. A management application executing on the controller interfaces to all of the switches in the network, making it possible to configure forwarding paths that utilize all available bandwidth. By interfacing to cloud management software, the application can guarantee that bandwidth is in place as workloads are created or moved.

The **Openflow** specification defines a **protocol** between the controller and the switches and a set of operations on the switches. The controller-to-switch protocol runs over either Transport Layer Security (TLS) or an unprotected TCP connection. Commands from the controller to the switch specify how packets are to be forwarded and configures parameters such as VLAN priorities. Messages from switches inform the controller when links go down or when a packet arrives with no specified forwarding instructions.

Forwarding instructions are based on a flow, which consists of all packets sharing a common set of characteristics. A large variety of parameters can be specified to define a flow. Possible criteria include the switch port where the packet arrived, the source Ethernet port, source IP port, VLAN tag, destination Ethernet or IP port, and a number of other packet characteristics. The controller specifies to the switch the set of parameters that define each flow and how packets that match the flow should be processed.

Each switch maintains a number of flow tables, with each table containing a list of flow entries. Each flow entry contains a match field that defines the flow, a counter and a set of instructions. Entries in the match field contain either a specific value against which the corresponding parameter in the incoming packet is compared or a value indicating that the entry is not included in this flow's parameter set.

Flow tables are numbered beginning with table zero, with incoming packets first compared to flow table entries in table zero. When a match is found, the flow counter is incremented and the specified set of instructions is carried out.

A new flow must be created when a packet arrives that does not match any flow table entry. The switch may have been configured to simply drop packets for which no flow has been defined, but in most cases, the packet will be sent to the controller. The controller then defines a new flow for that packet and creates one or more flow table entries. It then sends the entry or entries to the switch to be added to flow tables. Finally, the packet is sent back to the switch to be processed as determined by the newly created flow entries.

Flow table instructions modify the action set associated with each packet. Packets begin processing with an empty action set. Actions can specify that the packet be forwarded through a specified port or modify packet TTL, VLAN, MPLS tags or packet QOS.

Instructions in the first flow table may carry out an action on the packet or add actions to be carried out later. Instructions may also direct packet processing to continue by comparing it to entries in another flow table. A flow entry in a subsequent table may contain instructions that add further actions, delete or modify actions added earlier or carry out actions.

An instruction may also add a value called metadata to a packet before sending it to the next flow table. That value becomes an additional parameter to be matched against the metadata value in flow table entries in the next table. Processing continues table by table until all specified instructions have been completed and the packet has been forwarded.

An instruction may specify a group identifier. Groups provide an efficient way to direct that the same set of actions must be carried out on multiple flows. Group operations are defined within the switch by entries in the group table. Each entry consists of its identifier value, a group type, a counter and a set of action buckets. Group type specifies whether all action buckets should be executed, which is useful for implementing broadcast or multicast, or that only specific buckets are to be executed.

3.1 Openflow protocol messaging

The protocol consists of three types of messages: controller-to-switch, asynchronous and symmetric.

Controller-to-switch messages are sent by the controller to:

- Specify, modify or delete flow definitions
- Request information on switch capabilities
- Retrieve information like counters from the switch
- Send a packet back to a switch for processing after a new flow is created

Asynchronous messages are sent by the switch to:

- Send the controller a packet that does not match an existing flow
- Inform the controller that a flow has been removed because its time to live parameter or inactivity timer has expired
- Inform the controller of a change in port status or that an error as occurred on the switch

Symmetric messages can be sent by both the switch and the controller and are used for:

- Hello messages exchanged between controller and switch on startup
- Echo messages used to determine the latency of the controller-to-switch connection and to verify that the controller-to-switch connection is still operative
- Experimenter messages to provide a path for future extensions to Openflow technology.[5]

4. OPENFLOW SWITCH

The basic idea is the fact that most modern Ethernet switches and routers already contain flow-tables (typically built from TCAMs) that run at line-rate to implement firewalls, NAT, QoS, and to collect statistics. While each vendor’s flow-table is different, it has been identified an interesting common set of functions that run in almost all switches and routers. Openflow exploits this common set of functions.

Openflow provides an open protocol to program the flow table in different switches and routers. A network administrator can partition traffic into production and research flows. Researchers can control their own flows - by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today.

The datapath of an Openflow Switch consists of a Flow Table, and an action associated with each flow entry. The set of actions supported by an Openflow Switch is extensible, but below we describe a minimum requirement for all switches. For high-performance and low-cost the data-path must have a carefully prescribed degree of flexibility.

This means forgoing the ability to specify arbitrary handling of each packet and seeking a more limited, but still useful, range of actions. Therefore, later in the paper, define a basic required set of actions for all Openflow switches.

An Openflow Switch consists of at least three parts:

1. A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow

2. A Secure Channel that connects the switch to a remote control process (called the controller), allowing commands and packets to be sent between a controller and the switch using

3. The Openflow Protocol, which provides an open and standard way for a controller to communicate with a switch. By specifying a standard interface (the Openflow Protocol) through which entries in the Flow Table can be defined externally, the Openflow Switch avoids the need for researchers to program the switch.

It is useful to categorize switches into dedicated Openflow switches that do not support normal Layer 2 and Layer 3 processing, and Openflow-enabled general purpose commercial Ethernet switches and routers, to which the Open-Flow Protocol and interfaces have been added as a new feature.

4.1 Dedicated Openflow and Openflow enabled switches

A **dedicated Openflow** Switch is a dumb datapath element that forwards packets between ports, as defined by a remote control process. Picture 2 shows an example of an Openflow Switch. In this context, flows are broadly defined, and are limited only by the capabilities of the particular implementation of the Flow Table. For example, a flow could be a TCP connection, or all packets from a particular MAC address or IP address, or all packets with the same VLAN tag, or all packets from the same switch port. For experiments involving non-IPv4 packets, a flow could be defined as all packets matching a specific (but non-standard) header. Each flow-entry has a simple action associated with it; the three basic ones (that all dedicated Openflow switches must support) are:

1. Forward this flow’s packets to a given port (or ports).This allows packets to be routed through the network. In most switches this is expected to take place at line-rate.

2. Encapsulate and forward this flow’s packets to a controller. Packet is delivered to Secure Channel, where it is encapsulated and sent to a controller. Typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Or in some experiments, it could be used to forward all packets to a controller for processing.

3. Drop this flow’s packets. Can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts.

An entry in the Flow-Table has three fields: (1) A packet header that defines the flow, (2) The action, which defines how the packets should be processed, and (3) Statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow (to help with the removal of inactive flows).

In the first generation “Type 0” switches, the flow header is a 10-tuple shown in Table 1. A TCP flow could be specified by all ten fields, whereas an IP flow might not include the transport ports in its definition. Each header field can be a wildcard to allow for aggregation of flows, such as flows in which only the VLAN ID is defined would apply to all traffic on a particular VLAN.

The detailed requirements of an Openflow Switch are defined by the Openflow Switch Specification.

Openflow-enabled switches. Some commercial switches, routers and access points will be enhanced with the Openflow feature by adding the Flow Table, Secure Channel and Openflow Protocol. Typically, the Flow Table will re-use existing hardware, such as a TCAM; the Secure Channel and Protocol will be ported to run on the switch’s operating system.

Picture 3 shows a network of Openflow-enabled commercial switches and access points. In this example, all the Flow Tables are managed by the same controller; the Openflow Protocol allows a switch to be controlled by two or more controllers for increased performance or robustness. Openflow-enabled switches must isolate experimental traffic (processed by the Flow Table) from production traffic that is to be processed by the normal Layer 2 and Layer 3 pipeline of the switch. There are two ways to achieve this separation. One is to add a fourth action:

4. Forward this flow’s packets through the switch’s normal processing pipeline.

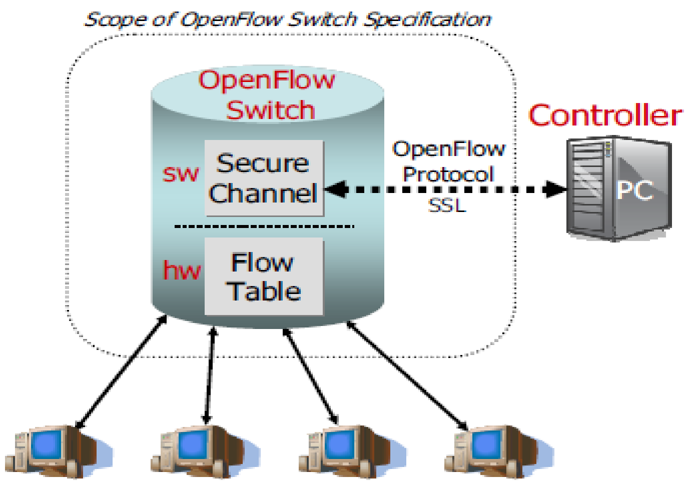
The other is to define separate sets of VLANs for experimental and production traffic. Both approaches allow normal production traffic that isn’t part of an experiment to be processed in the usual way by the switch. All Openflow-enabled switches are required to support one approach or the other; some will support both.

Additional features. If a switch supports the header formats and the four basic actions mentioned above (and detailed in the Openflow Switch Specification), then we call it a “Type 0” switch. We expect that many switches will support additional actions, for example to rewrite portions of the packet header (e.g., for NAT, or to obfuscate addresses on intermediate links), and to map packets to a priority class. Likewise, some Flow Tables will be able to match on arbitrary fields in the packet header, enabling experiments with new non-IP protocols. As a particular set of features emerges, we will define a “Type 1” switch.

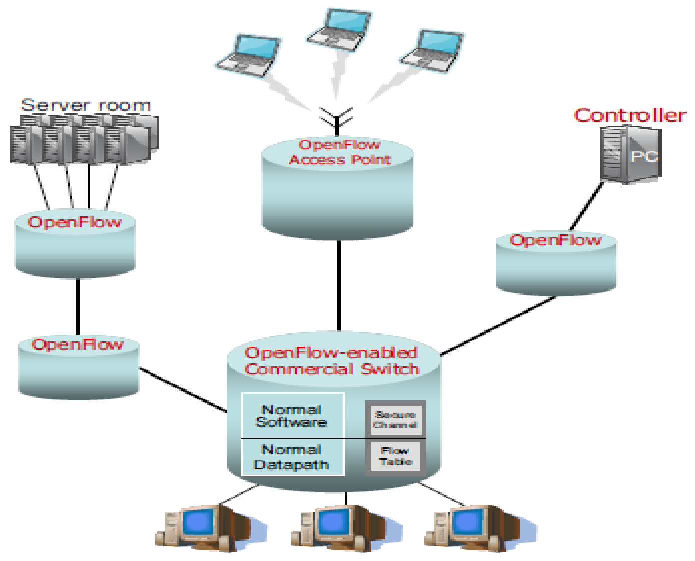
4.2 Controller

Based on Openflow switch specification a controller is responsible for adding and removing flow-entries from the Flow Table .A static controller might be a simple application running on a PC or server to statically establish flows to interconnect networking devices and hosts as needed. For example there are cases the flows resemble VLANs in current networks—providing a simple mechanism to isolate experimental traffic from the production network. Viewed this way, Openflow is a generalization of VLANs. One can also imagine more sophisticated controllers that dynamically add/remove flows as an experiment progresses.

In one usage model, a researcher might control the complete network of Openflow Switches and be free to decide how all flows are processed. A more sophisticated controller might support multiple researchers, each with different accounts and permissions, enabling them to run multiple independent experiments on different sets of flows. Flows identified as under the control of a particular researcher (e.g., by a policy table running in a controller) could be delivered to a researcher's user-level control program which then decides if a new flow-entry should be added to the network of switches.



Picture 2: Openflow switch example



Picture 3: Openflow commercial switches and access points

4.3 Flow Table

This section describes the components of flow table entries and the process by which incoming packets are matched against flow table entries.

Header Fields	Counters	Actions

Each flow table entry (see Table 1) contains:

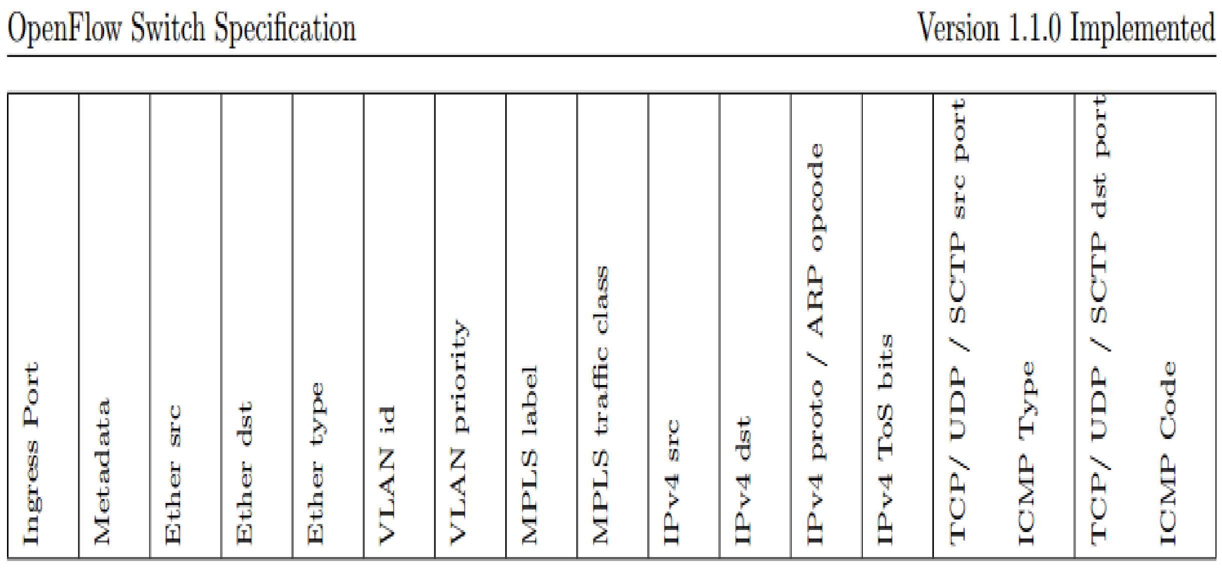
- header fields to match against packets
- counters to update for matching packet
- actions to apply to matching packets

4.3.1 Header Fields

Table 1 shows the header fields an incoming packet is compared against. Each entry contains a specific value, or ANY, which matches any value. If the switch supports subnet masks on the IP source and/or destination fields, these can more precisely specify matches. The fields in the Openflow are listed in Table 1 and details on the properties of each field are described in Table 2.

Switch designers are free to implement the internals in any way convenient provided that correct functionality is preserved. For example, while a flow may have multiple forward actions, each specifying a different port, a switch designer may choose to implement this as a single bitmask within the hardware forwarding table.

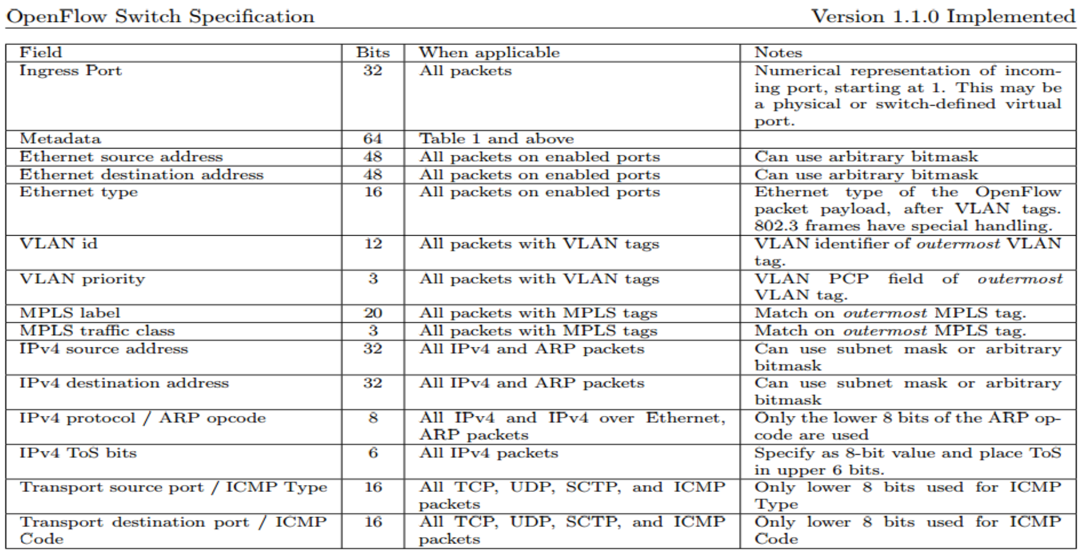
Table 1: Fields from packets used to match against flow entries



OpenFlow Switch Specification Version 1.1.0 Implemented

Ingress Port	Metadata	Ether src	Ether dst	Ether type	VLAN id	VLAN priority	MPLS label	MPLS traffic class	IPv4 src	IPv4 dst	IPv4 proto / ARP opcode	IPv4 ToS bits	TCP/ UDP / SCTP src port ICMP Type	TCP/ UDP / SCTP dst port ICMP Code

Table 2: Field lengths and the way they must be applied to flow entries



OpenFlow Switch Specification Version 1.1.0 Implemented

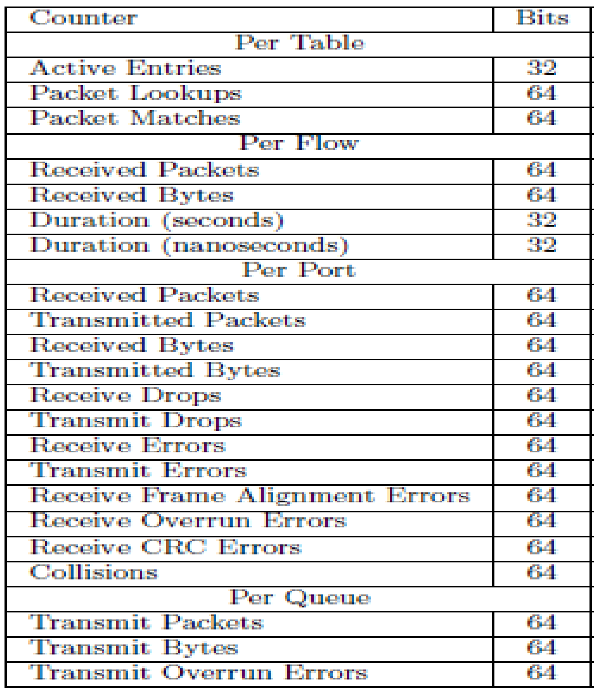
Field	Bits	When applicable	Notes
Ingress Port	32	All packets	Numerical representation of incoming port, starting at 1. This may be a physical or switch-defined virtual port.
Metadata	64	Table 1 and above	
Ethernet source address	48	All packets on enabled ports	Can use arbitrary bitmask
Ethernet destination address	48	All packets on enabled ports	Can use arbitrary bitmask
Ethernet type	16	All packets on enabled ports	Ethernet type of the OpenFlow packet payload, after VLAN tags. 802.3 frames have special handling.
VLAN id	12	All packets with VLAN tags	VLAN identifier of *outermost* VLAN tag.
VLAN priority	3	All packets with VLAN tags	VLAN PCP field of *outermost* VLAN tag.
MPLS label	20	All packets with MPLS tags	Match on *outermost* MPLS tag.
MPLS traffic class	3	All packets with MPLS tags	Match on *outermost* MPLS tag.
IPv4 source address	32	All IPv4 and ARP packets	Can use subnet mask or arbitrary bitmask
IPv4 destination address	32	All IPv4 and ARP packets	Can use subnet mask or arbitrary bitmask
IPv4 protocol / ARP opcode	8	All IPv4 and IPv4 over Ethernet, ARP packets	Only the lower 8 bits of the ARP opcode are used
IPv4 ToS bits	6	All IPv4 packets	Specify as 8-bit value and place ToS in upper 6 bits.
Transport source port / ICMP Type	16	All TCP, UDP, SCTP, and ICMP packets	Only lower 8 bits used for ICMP Type
Transport destination port / ICMP Code	16	All TCP, UDP, SCTP, and ICMP packets	Only lower 8 bits used for ICMP Code

4.3.2 Counters

Counters are maintained per-table, per-flow, per-port and per queue. Openflow compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges.

Table 3 contains the required set of counters. Duration refers to the time the flow has been installed in the switch. The Receive Errors field includes all explicitly specified errors, including frame, overrun, and CRC errors, plus any others. Counters wrap around with no overflow indicator. In this document, the phrase byte refers to 8-bit octets.

Table 3: Required set of counters



Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

4.3.3 Actions

Each flow entry is associated with zero or more actions that dictate how the switch handles matching packets. If no forward actions are present, the packet is dropped. Action lists for inserted flow entries MUST be processed in the order specified. However, there is no packet output ordering guaranteed within a port. For example, an action list may result in two packets sent to two different VLANs on a single port. These two packets may be arbitrarily re-ordered, but the packet bodies must match those generated from a sequential execution of the actions. A switch may reject a flow entry if it cannot process the action list in the order specified, in which case it should immediately return an unsupported flow error. Ordering within a port may vary between vendor switch implementations.

A switch is not required to support all action types just those marked as “Required Actions" below. When connecting to the controller, a switch indicates which of the “Optional Actions" it supports. Openflow-compliant switches come in two types: Openflow-only, and Openflow-enabled.

Openflow-only switches support only the required actions below, while Openflow enabled switches, routers, and access points may also support the **NORMAL** action. Either type of switch can also support the **FLOOD** action.

Required Action: Forward. Openflow switches must support forwarding the packet to physical ports and the following virtual ones:

- **ALL**: Send the packet out all interfaces, not including the incoming interface.
- **CONTROLLER**: Encapsulate and send the packet to the controller.
- **LOCAL:** Send the packet to the switches local networking stack.
- **TABLE**: Perform actions in flow table. Only for packet-out messages.
- **IN PORT:** Send the packet out the input port.

Optional Action: Forward. The switch may optionally support the following virtual ports:

- **NORMAL**: Process the packet using the traditional forwarding path supported by the switch (i.e., traditional L2, VLAN, and L3 processing.) The switch may check the VLAN field to determine whether or not to forward the packet along the normal processing route. If the switch cannot forward entries for the Openflow-specific VLAN back to the normal processing route, it must indicate that it does not support this action.
- **FLOOD**: Flood the packet along the minimum spanning tree, not including the incoming interface.

The controller will only ask the switch to send to multiple physical ports simultaneously if the switch indicates it supports this behavior in the initial handshake.

Optional Action: Enqueue. The enqueue action forwards a packet through a queue attached to a port. Forwarding behavior is dictated by the configuration of the queue and is used to provide basic Quality-of-Service (QoS) support.

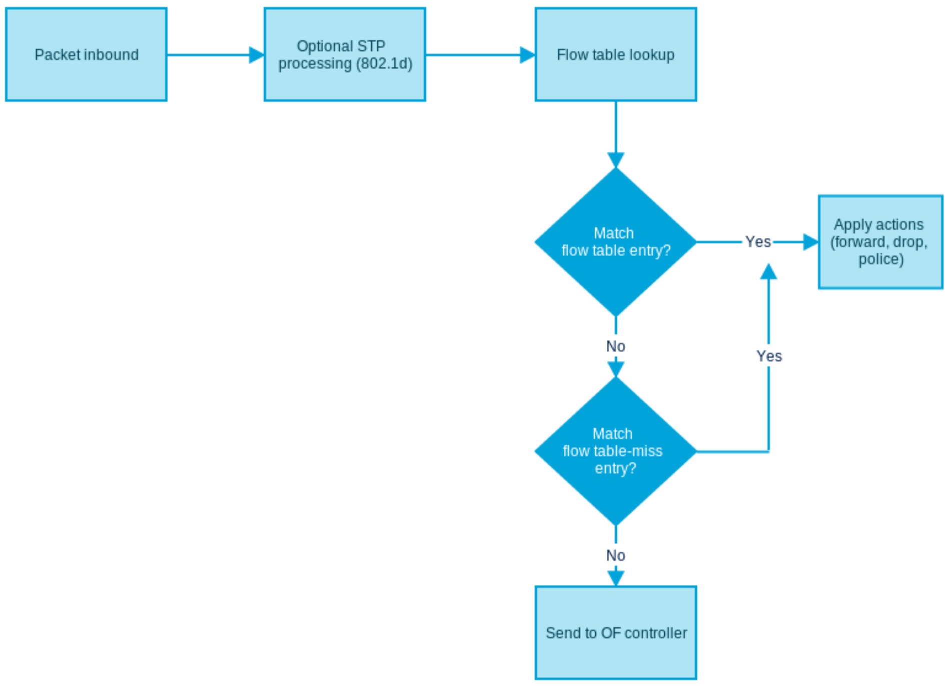
Required Action: Drop. A flow-entry with no specified action indicates that all matching packets should be dropped.

Optional Action: Modify-Field. While not strictly required, the actions shown in Table 4 greatly increase the usefulness of an Openflow implementation. To aid integration with existing networks, we suggest that VLAN modification actions be supported.

Table 4: Actions

Action	Associated Data	Description
Set VLAN ID	12 bits	If no VLAN is present, a new header is added with the specified VLAN ID and priority of zero. If a VLAN header already exists, the VLAN ID is replaced with the specified value.
Set VLAN priority	3 bits	If no VLAN is present, a new header is added with the specified priority and a VLAN ID of zero. If a VLAN header already exists, the priority field is replaced with the specified value.
Strip VLAN header	-	Strip VLAN header if present.
Modify Ethernet source MAC address	48 bits: Value with which to replace existing source MAC address	Replace the existing Ethernet source MAC address with the new value
Modify Ethernet destination MAC address	48 bits: Value with which to replace existing destination MAC address	Replace the existing Ethernet destination MAC address with the new value.
Modify IPv4 source address	32 bits: Value with which to replace existing IPv4 source address	Replace the existing IP source address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applicable to IPv4 packets.
Modify IPv4 destination address	32 bits: Value with which to replace existing IPv4 destination address	Replace the existing IP destination address with new value and update the IP checksum (and TCP/UDP checksum if applicable). This action is only applied to IPv4 packets.
Modify IPv4 ToS bits	6 bits: Value with which to replace existing IPv4 ToS field	Replace the existing IP ToS field. This action is only applied to IPv4 packets.
Modify transport source port	16 bits: Value with which to replace existing TCP or UDP source port	Replace the existing TCP/UDP source port with new value and update the TCP/UDP checksum. This action is only applicable to TCP and UDP packets.
Modify transport destination port	16 bits: Value with which to replace existing TCP or UDP destination port	Replace the existing TCP/UDP destination port with new value and update the TCP/UDP checksum This action is only applied to TCP and UDP packets.

4.3.4 Matching


Packet inbound
Optional STP processing (802.1d)
Flow table lookup
Match flow table entry?
Yes
Apply actions (forward, drop, police)
No
Yes
Match flow table-miss entry?
No
Send to OF controller

Picture 4: Matching a packet

On receipt of a packet, an Openflow Switch performs the functions shown in Picture 4. Header fields used for the table lookup depend on the packet type as described below.

- Rules specifying an ingress port are matched against the physical port that received the packet.
- The Ethernet headers as specified earlier are used for all packets.
- If the packet is a VLAN (Ethernet type 0x8100), the VLAN ID and PCP fields are used in the lookup.
- (Optional) For ARP packets (Ethernet type equal to 0x0806), the lookup fields may also include the contained IP source and destination fields.
- For IP packets (Ethernet type equal to 0x0800), the lookup fields also include those in the IP header.
- For IP packets that are TCP or UDP (IP protocol is equal to 6 or 17), the lookup includes the transport ports.
- For IP packets that are ICMP (IP protocol is equal to 1), the lookup includes the Type and Code fields.
- For IP packets with nonzero fragment offset or More Fragments bit set, the transport ports are set to zero for the lookup.

A packet matches a flow table entry if the values in the header fields used for the lookup (as defined above) match those defined in the flow table. If a flow table field has a value of ANY, it matches all possible values in the header.

To handle the various Ethernet framing types, matching the Ethernet type is handled in a slightly different way. If the packet is an Ethernet II frame, the Ethernet type is handled in the expected way. If the packet is an 802.3 frame with a SNAP header and Organizationally Unique Identifier (OUI) of 0x000000, the SNAP protocol id is matched against the flows Ethernet type. A flow entry that specifies an Ethernet type of 0x05FF, matches all Ethernet 802.2 frames without a SNAP header and those with SNAP headers that do not have an OUI of 0x000000.

Packets are matched against flow entries based on prioritization. An entry that specifies an exact match (i.e., it has no wildcards) is always the highest priority. All wildcard entries have a priority associated with them. Higher priority entries must match before lower priority ones. If multiple entries have the same priority, the switch is free to choose any ordering. Higher numbers have higher priorities.

For each packet that matches a flow entry, the associated counters for that entry are updated. If no matching entry can be found for a packet, the packet is sent to the controller over the secure channel.

4.4 Secure Channel

The secure channel is the interface that connects each Openflow switch to a controller. Through this interface, the controller configures, manages, receives events and send packets to the switch. Between the datapath and the secure channel, the interface is implementation specific, however all secure channel messages must be formatted according to the Openflow protocol.

Support for multiple simultaneous controllers is currently undefined.

4.5 Openflow Protocol Overview

As mentioned earlier the Openflow protocol supports three message types, *controller-to-switch, asynchronous, and symmetric,* each with multiple sub-types. Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch. Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state.

Symmetric messages are initiated by either the switch or the controller and sent without solicitation. The message types used by Openflow are described below.

4.5.1 Controller-to-Switch

Controller/switch messages are initiated by the controller and may or may not require a response from the switch.

Features: Upon Transport Layer Security (TLS) session establishment, the controller sends a features request message to the switch. The switch must reply with a features reply that specifies the capabilities supported by the switch.

Configuration: The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.

Modify-State: Modify-State messages are sent by the controller to manage state on the switches. Their primary purpose is to add/delete and modify flows in the flow tables and to set switch port properties.

Read-State: Read-State messages are used by the controller to collect statistics from the switches flow-tables, ports and the individual flow entries.

Send-Packet: These are used by the controller to send packets out of a specified port on the switch.

Barrier: Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

4.5.2 Asynchronous

Asynchronous messages are sent without the controller soliciting them from a switch. Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or error. The four main asynchronous message types are described below.

Packet-in: For all packets that do not have a matching flow entry, a packet-in event is sent to the controller (or if a packet matches an entry with a "send to controller" action). If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in events contain some fraction of the packet header (by default 128 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffering) must send the full packet to the controller as part of the event.

Flow-Removed: When a flow entry is added to the switch by a flow modify message, an idle timeout value indicates when the entry should be removed due to a lack of activity, as well as a hard timeout value that indicates when the entry should be removed, regardless of activity. The flow modify message also specifies whether the switch should send a flow removed message to the controller when the flow expires. Flow modify messages which delete flows may also cause flow removed messages.

Port-status: The switch is expected to send port-status messages to the controller as port configuration state changes. These events include change in port status (for example, if it was brought down directly by a user) or a change in port status as specified by 802.1D.

Error: The switch is able to notify the controller of problems using error messages.

4.5.3 Symmetric

Symmetric messages are sent without solicitation, in either direction.

Hello: Hello messages are exchanged between the switch and controller upon connection startup.

Echo: Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They can be used to indicate the latency, bandwidth, and/or liveness of a controller-switch connection.

Vendor: Vendor messages provide a standard way for Openflow switches to for additional functionality within the Openflow message type space. This is a staging area for features meant for future Openflow revisions.

4.6 Connection Setup

The switch must be able to establish the communication at a user-configurable (but otherwise fixed) IP address, using a user-specified port. Traffic to and from the secure channel is not checked against the flow table. Therefore, the switch must identify incoming traffic as local before checking it against the flow table. Future versions of the protocol specification will describe a dynamic controller discovery protocol in which the IP address and port for communicating with the controller is determined at runtime.

When an Openflow connection is first established, each side of the connection must immediately send an OFPT_HELLO message with the version field set to the highest Openflow protocol version supported by the sender. Upon receipt of this message, the recipient may calculate the Openflow protocol version to be used as the smaller of the version number that it sent and the one that it received.

If the negotiated version is supported by the recipient, then the connection proceeds. Otherwise, the recipient must reply with an OFPT_ERROR message with a type field of OFPET_HELLO_FAILED, a code field of OFPHFC_COMPATIBLE, and optionally an ASCII string explaining the situation in data, and then terminate the connection.

4.7 Connection Interruption

In the case that a switch loses contact with the controller, as a result of an echo request timeout, TLS session timeout, or other disconnection, it should attempt to contact one or more backup controllers. The ordering of the controller IP addresses is not specified by the protocol.

If some number of attempts to contact a controller (zero or more) fail, the switch must enter “emergency mode" and immediately reset the current TCP connection. In emergency mode, the matching process is dictated by the emergency flow table entries (those marked with the emergency bit when added to the switch). All normal entries are deleted when entering emergency mode.

Upon connecting to a controller again, the emergency flow entries remain. The controller then has the option of deleting all flow entries, if desired.

The first time a switch starts up; it is considered to be in emergency mode. Configuration of the default set of flow entries is outside the scope of the Openfllow protocol.

4.8 Encryption

The switch and controller communicate through a TLS connection. The TLS connection is initiated by the switch on startup to the controller's server, which is located by default on TCP port 6633. The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key. Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

4.9 Spanning Tree

Openflow switches may optionally support 802.1D Spanning Tree Protocol. Those switches that do support it are expected to process all 802.1D packets locally before performing the flow lookup. A switch that implements STP must set the OFPC_STP bit in the 'capabilities' field of its OFPT_FEATURES_REPLY message.

A switch that implements STP must make it available on all of its physical Ports, but it need not implement it on virtual ports (e.g. OFPP_LOCAL).

Port status, as specified by the spanning tree protocol, is then used to limit packets forwarded to the OFP_FLOOD port to only those ports along the spanning tree. Port changes as a result of the spanning tree are sent to the controller via port-update messages. Note that forward actions that specify the outgoing port or OFP_ALL ignore the port status set by the spanning tree protocol. Packets received on ports that are disabled by spanning tree must follow the normal flow table processing path.

Switches that do not support 802.1D spanning tree must allow the controller to specify the port status for packet flooding through the port-mod messages.

Because all switches register to the OpenFlow controller, the controller can create a complete overview of all switches and links between the switches so it can calculate the shortest path. The major difference with 802.1aq is that the switches do not cooperate together and decide the shortest path but communicate to the controller for this operation.

4.10 Flow Table Modification Messages

Flow table modification messages can have the following types:

```
enum ofpfiflowfimodficommand {
        OFPFC_ADD, /* New flow. */
        OFPFC_MODIFY, /* Modify all matching flows. */
        OFPFC_MODIFY_STRICT, /* Modify entry strictly matching wildcards */
        OFPFC_DELETE, /* Delete all matching flows. */
        OFPFC_DELETE_STRICT /* Strictly match wildcards and priority. */
};
```

For ADD requests with the OFPFF_CHECK_OVERLAP flag set, the switch must first check for any overlapping flow entries. Two flow entries overlap if a single packet may match both, and both entries have the same priority. If an overlap conflict exists between an existing flow entry and the ADD request, the switch must refuse the addition and respond with an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMFC_OVERLAP code.

For valid (non-overlapping) ADD requests, or those with no overlap checking, the switch must insert the flow entry at the lowest numbered table for which the switch supports all wildcards set in the flow_match struct, and for which the priority would be observed during the matching process. If a flow entry with identical header fields and priority already resides in any table, then that entry, including its counters, must be removed, and the new flow entry added.

If a switch cannot find any table in which to add the incoming flow entry, the switch should send an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMFC_ALL_TABLES_FULL code. If the action list in a flow mod message references a port that will never be valid on a switch, the switch must return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_BAD_OUT_PORT code. If the referenced port may be valid in the future, e.g. when a linecard is added to a chassis switch, or a port is dynamically added to a software switch, the switch may either silently drop packets sent to the referenced port, or immediately return an OFPBAC_BADfiOUT_PORT error and refuse the flow mod. For MODIFY requests, if a flow entry with identical header fields does not current reside in any table, the MODIFY acts like an ADD, and the new flow entry must be inserted with zeroed counters. Otherwise, the actions field is changed on the existing entry and its counters and idle time fields are left unchanged.

For DELETE requests, if no flow entry matches, no error is recorded, and no flow table modification occurs. If flow entries match, and must be deleted, then each normal entry with the OFPFF_SEND_FLOW_REM flag set should generate a flow removed message. Deleted emergency flow entries generate no flow removed messages.

MODIFY and DELETE flow mod commands have corresponding STRICT versions. Without STRICT appended, the wildcards are active and all flows that match the description are modified or removed. If STRICT is appended, all fields, including the wildcards and priority, are strictly matched against the entry, and only an identical flow is modified or removed. For example, if a message to remove entries is sent that has all the wildcard flags set, the DELETE command would delete all flows from all tables, while the DELETE STRICT command would only delete a rule that applies to all packets at the specified priority.

For non-strict MODIFY and DELETE commands that contain wildcards, a match will occur when a flow entry exactly matches or is more specific than the description in the flow mod command. For example, if a DELETE command says to delete all flows with a destination port of 80, then a flow entry that is all wildcards will not be deleted. However, a DELETE command that is all wildcards will delete an entry that matches all port 80 traffic. This same interpretation of mixed wildcard and exact header fields also applies to individual and aggregate flows stats.

DELETE and DELETE STRICT commands can be optionally filtered by output port. If the out_port field contains a value other than OFPP_NONE, it introduces a constraint when matching. This constraint is that the rule must contain an output action directed at that port. This field is ignored by ADD, MODIFY, and MODIFY STRICT messages.

Emergency flow mod messages must have timeout values set to zero. Otherwise, the switch must refuse the addition and respond with an ofp_error_msg with FPET_FLOW_MOD_FAILED type and OFPFMFC_BAD_EMERG_TIMEOUT code.

If a switch cannot process the action list for any flow mod message in the order specified, it MUST immediately return an OFPET_FLOW_MOD_FAILED:

OFPFMFC_UNSUPPORTED error and reject the flow.

4.11 Flow Removal

Each flow entry has an idle_timeout and a hard_timeout associated with it. If no packet has matched the rule in the last idle_timeout seconds, or it has been hard_timeout seconds since the flow was inserted, the switch removes the entry and sends a flow removed message. In addition, the controller is able to actively remove entries by sending a flow message with the DELETE or DELETE_STRICT command. Like the message used to add the entry, a removal message contains a description, which may include wild cards.[6]

5. OPENFLOW CONTROLLER

The controller is the main device; it is responsible for maintaining all the network rules and distributes the appropriate instructions to the network nodes (devices). In others words, the Openflow controller is responsible for determining how to handle packets without valid flow entries, and it manages the switch flow table by adding and removing flow entries over the secure channel using the Openflow protocol. The controller essentially centralizes the network intelligence, while the network maintains a distributed forwarding plane through Openfllow Switches and routers. This is the reason the controller provides an interface for managing, controlling and administrating the Switches' flow-tables. Typically, the Controller runs on network accessible server and there are different control configurations depending on:

5.1 Location

According to how the delegation of switch management to controllers is performed. One can distinguish two types of settings regarding the location of the controller. One of them would be a Centralized configuration where a single controller manages and configures all devices and another configuration possible would be the distributed configuration, where is available one controller for each set of switches.

5.2 Flow

Flow Routing: Every flow is individually set up by controller. Exact-match flow entries.

Flow table contains one entry per flow. Good for, for example, campus networks.

Aggregated: One flow entry covers large groups of flows. Wildcard flow entries. Flow table contains one entry per category of flows. Good for large number of flows, e.g. backbone.

5.3 Behavior

Reactive: First packet of flow triggers controller to insert flow entries. Efficient use of flow table. Every flow incurs small additional flow setup time. If control connection lost, switch has limited utility.

Proactive: Controller pre-populates flow table in switch. Zero additional flow setup time. Loss of control connection does not disrupt traffic.

Depending on the configuration, controllers are more sophisticated than others. It's possible to configure a simple controller that dynamically add/remove flows and where the researcher can control the complete network of Openflow Switches and is responsible to decide how all flows are processed. Also can be imagined a sophisticated controller which can support multiple researchers, each with different accounts and permissions, enabling them to run multiple independent experiments on different sets of flows. This flows can be identified as under the control of a particular researcher and can be delivered to a researcher's user-level control program which then decides if a new flow-entry should be added to the network of switches.[7]

6. OPENDAYLIGHT

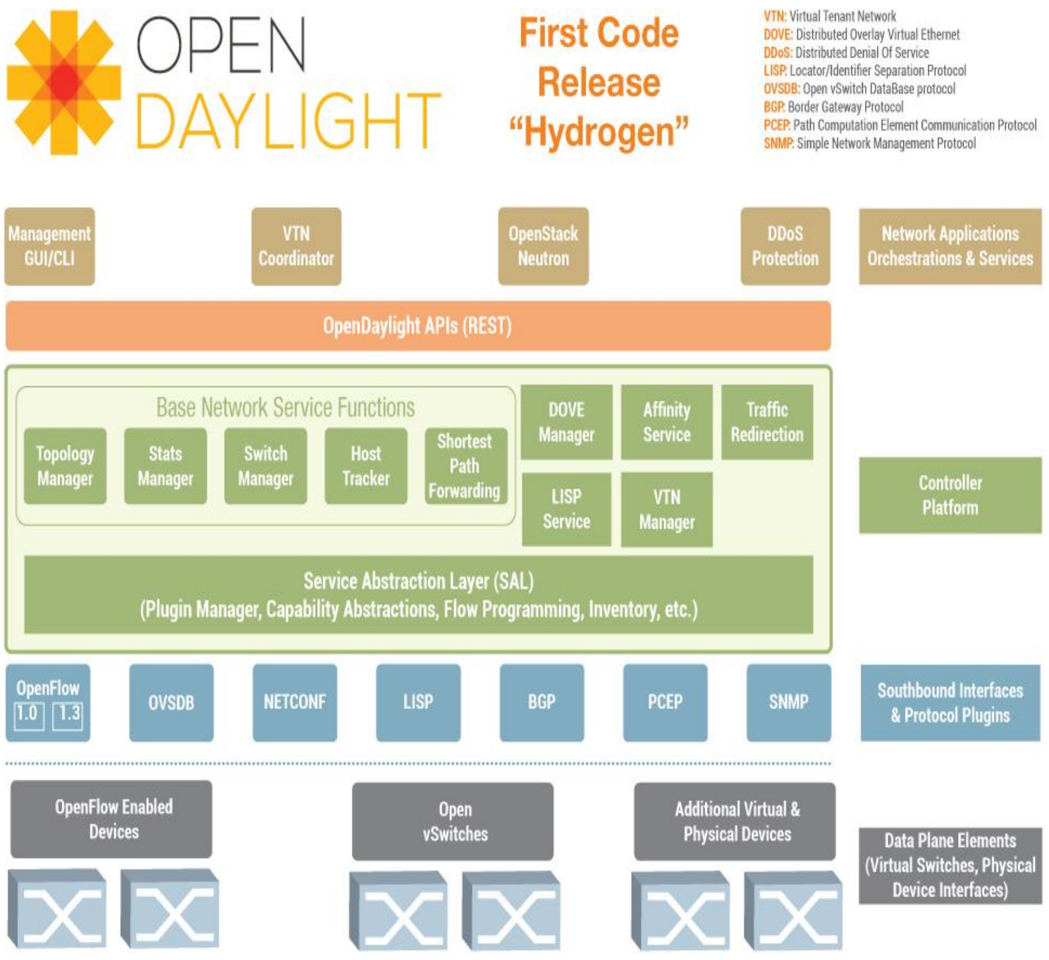
[8] Taken from the OpenDaylight Wiki Page

The OpenDaylight Project is a collaborative open source project that aims to accelerate adoption of Software-Defined Networking (SDN) and create a solid foundation for Network Functions Virtualization (NFV) for a more transparent approach that fosters new innovation and reduces risk. Founded by industry leaders and open to all, the OpenDaylight community is developing a common, open SDN framework consisting of code and blueprints.

SAN FRANCISCO, September 13, 2013 – The OpenDaylight Project, a community-led and industry-supported open source framework to advance Software-Defined Networking (SDN), today shared a first glimpse at the OpenDaylight SDN architecture aimed for the first release called “Hydrogen.” OpenDaylight is being built as a highly extensible and modular open source SDN platform to accelerate adoption across diverse and broad deployment use cases from enterprise IT to network providers to cloud service providers.

“The OpenDaylight community is developing an SDN architecture that supports a wide range of protocols and can rapidly evolve in the direction SDN goes, not based on any one vendor’s purposes,” said David Meyer, Technical Steering Committee chair, OpenDaylight Project. “As an open source project OpenDaylight can be a core component within any SDN architecture, putting the user in control. The community is working to further refine the Service Abstraction Layer to deliver an efficient application API that can be used over a broad collection of network devices so we can deliver a best-of-breed platform that will help users of all stripes realize the promise of SDN.”

With OpenDaylight, enterprise users and service providers can be fully vested in the SDN technology running their networks and have direct access to the people building it. To accommodate a wide range of use cases OpenDaylight Hydrogen includes new and legacy protocols such as OVSDB, Openflow 1.3.0, BGP and PCEP. It also includes multiple methods for network virtualization and two initial applications that leverage the features of OpenDaylight: Affinity Metadata Service to aid in policy management and Defense4All for Distributed Denial of Service (DDoS) attack protection. A plugin for OpenStack Neutron has been integrated, and the Open vSwitch Database project will allow management from within OpenStack.


OPEN DAYLIGHT
First Code Release "Hydrogen"
VTN: Virtual Tenant Network
DOVE: Distributed Overlay Virtual Ethernet
DDoS: Distributed Denial Of Service
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase protocol
BGP: Border Gateway Protocol
PCEP: Path Computation Element Communication Protocol
SNMP: Simple Network Management Protocol
Management GUI/CLI
VTN Coordinator
OpenStack Neutron
DDoS Protection
Network Applications Orchestrations & Services
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
Host Tracker
Shortest Path Forwarding
DOVE Manager
Affinity Service
Traffic Redirection
LISP Service
VTN Manager
Controller Platform
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
OpenFlow 1.0 1.3
OVSDB
NETCONF
LISP
BGP
PCEP
SNMP
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Picture 5: OpenDaylight Hydrogen controller platform

Projects were contributed by Cisco, ConteXtream, Ericsson, IBM, Industrial Technology Research Institute (ITRI), NEC, Pantheon, Plexxi, Radware and developers Brent Salisbury and Evan Zeller from the University of Kentucky.

"OpenDaylight has made great strides toward its goal of accelerating a common SDN platform. As the networking industry evolves to a software-defined world we are seeing open source development and design methodology as the driving force for modern architectures," said Inder Gopal, Board of Directors chair, OpenDaylight Project.

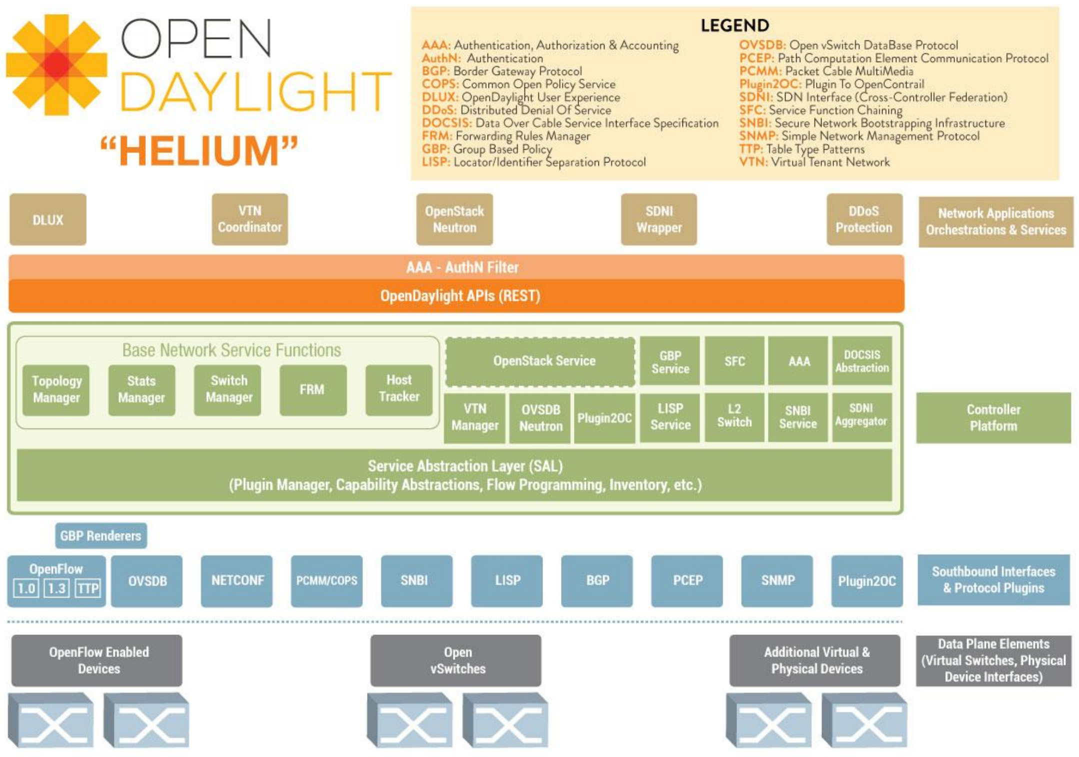
An OpenDaylight community exists; is open and everyone can participate and develop, test, design and give his effort in this whole project. The community is also willing to give answers and suggestions in every situation.

6.1 OpenDaylight Hellium

Software Defined Networking (SDN) separates the control plane from the data plane within the network, allowing the intelligence and state of the network to be managed centrally while abstracting the complexity of the underlying physical network. Great strides have been made within the industry toward this goal with standardized protocols such as Openflow. However, greater collaboration leveraging open source development best practices will significantly accelerate real, deployable solutions for the industry at large. Similarly, by evolving network services from an appliance model to one that leverages virtual compute, storage, and networking, Network Functions Virtualization (NFV) promises to drastically improve both the agility of when and where to run network functions as well as the cost structure of doing so.

SDN and NFV are a new way of deploying network infrastructure. A software-defined network adapts to the requirements of applications deployed on the network. Current generation networks and architectures are statically configured and vertically integrated. New generation applications such as Hadoop, video delivery, and virtualized network functions require networks to be agile and to flexibly adapt to application requirements.

From a high level view, software defined networking is commonly described in layers.


OPEN DAYLIGHT
"HELIUM"
LEGEND
AAA: Authentication, Authorization & Accounting
AuthN: Authentication
BGP: Border Gateway Protocol
COPS: Common Open Policy Service
DLUX: OpenDaylight User Experience
DDoS: Distributed Denial Of Service
DOCSIS: Data Over Cable Service Interface Specification
FRM: Forwarding Rules Manager
GBP: Group Based Policy
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
PCEP: Path Computation Element Communication Protocol
PCMM: Packet Cable MultiMedia
Plugin2OC: Plugin To OpenContrail
SDNI: SDN Interface (Cross-Controller Federation)
SFC: Service Function Chaining
SNBI: Secure Network Bootstrapping Infrastructure
SNMP: Simple Network Management Protocol
TTP: Table Type Patterns
VTN: Virtual Tenant Network
DLUX
VTN Coordinator
OpenStack Neutron
SDNI Wrapper
DDoS Protection
Network Applications Orchestrations & Services
AAA - AuthN Filter
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
FRM
Host Tracker
OpenStack Service
GBP Service
SFC
AAA
DOCSIS Abstraction
VTN Manager
OVSDB Neutron
Plugin2OC
LISP Service
L2 Switch
SNBI Service
SDNI Aggregator
Controller Platform
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
GBP Renderers
OpenFlow 1.0 1.3 TTP
OVSDB
NETCONF
PCMM/COPS
SNBI
LISP
BGP
PCEP
SNMP
Plugin2OC
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Picture 6: OpenDaylight Helium controller platform

Network Apps & Orchestration: The top layer consists of business and network logic applications that control and monitor network behavior. In addition, more complex solution orchestration applications needed for cloud and NFV (e.g. Firewall, NAT) thread services together and engineer network traffic in accordance with the needs of those environments.

Controller Platform: The middle layer is the framework in which the SDN abstractions can declare, providing a set of common APIs to the application layer (northbound interface) while implementing one or more protocols for command and control of the physical hardware within the network (typically referred to as the southbound interface).

Physical & Virtual Network Devices: The bottom layer consists of the physical & virtual devices, switches, routers, etc., that make up the connective fabric between all endpoints within the network. These devices can be either dedicated Openflow or Openflow enabled.

OpenDaylight is an open source project with a modular, pluggable, and flexible controller platform at its core. This controller is implemented purely in software and is contained within its own Java Virtual Machine (JVM). As such, it can be deployed on any hardware and operating system platform that supports Java.

The controller exposes open northbound APIs which are used by applications. OpenDaylight supports the OSGi framework and bidirectional REST for the northbound API. The OSGi framework is used for applications that will run in the same address space as the controller while the REST (web based) API is used for applications that do not run in the same address space (or even necessarily on the same machine) as the controller. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run algorithms to perform analytics, and then use the controller to orchestrate the new rules, if any, throughout the network.

Here we will refer simply what are OSGi and REST. OSGi is a dynamic module system for Java. It defines means to install, uninstall, update, start and stop modules. Those modules are called bundles, but are, in their simplest form, actually Java jar files with a special manifest. Modules can be installed, uninstalled etc. without stopping or restarting the Java VM. REST (Representational State Transfer) is a simple stateless architecture that generally runs over HTTP. REST involves reading a designated Web page that contains an XML file. The XML file describes and includes the desired content.

The controller platform itself contains a collection of dynamically pluggable modules to perform needed network tasks. There are a series of base network services for such tasks as understanding what devices are contained within the network and the capabilities of each, statistics gathering, etc. In addition, platform oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality.

The southbound interface is capable of supporting multiple protocols (as separate plugins), e.g. Openflow 1.0, Openflow 1.3, BGP-LS, etc. These modules are dynamically linked into a Service Abstraction Layer (SAL). The SAL exposes device services to which the modules north of it are written. The SAL determines how to fulfill the requested service irrespective of the underlying protocol used between the controller and the network devices.

Software Defined Network (SDN) is a new way of deploying network infrastructure. The SDN adapts to the requirements of applications deployed on the network. Current generation networks and architectures are statically configured and vertically integrated. New generation applications require networks to be agile and flexibly adapt to application requirements.

As networks get larger, especially in Massively Scalable Data Centers and Cloud, there is a large desire at ease-of-management and orchestration. This is leading to the need for programmatic interface (API) to the network to make it easier to write scripts as CLI and SNMP are not conducive to automation. There has been a change going on in IT and the CIO are getting influenced more by the Application and Server Admins and they who are used to the API and the ease of server management tools are demanding similar things from the network.

As application programmers desire their applications to be moved around in a data center or across clouds, it becomes imperative that the network becomes agile in meeting the requirements (bandwidth, services like load balancing, firewall) of the applications.

Network Abstraction and Virtualization is desired as it allows the network operators to operate the network at a higher level without worrying about the quirkiness of different products from the same or different vendors.

Additionally, there has been a desire from network operators for the ability to influence the forwarding and other network behavior based on their own algorithms and business logic. That means there is a need for the network to no longer be vertically integrated with the networking control logic coming only from the networking vendor.

Finally, there has been a desire to see the cost of networking gear come down especially amongst the Web Services and Cloud providers who build out large Data Centers. They thus view vendor neutrality and the rise of merchant silicon as leverage to be used against the networking vendors.

All these factors play into the growing interest in software defined networking and also results in a not-so-crisp definition of what SDN really means. However, it is clear that SDN is a 3-legged stool with the Network Applications (Apps) on the top written to open API, a Controller as the middle plane interacting with and managing network devices. Clearly there needs to be some sort of API or protocol needed for the Controller and the network devices to communicate with each other. Openflow is one such protocol which has come out of the efforts of Open Networking Foundation (ONF). The network devices support Agents which interpret the protocol and the API.

Central to the SDN effort is the Controller which provides the ability to deploy software to control the network gear and redeploy as needed. The vision is to have a modular Controller with a well published Northbound API for network Applications to write towards while utilizing southbound protocols such as Openflow to communicate with supported downstream network nodes. The industry and customers will benefit immensely by having an Open Source Controller with contributions from various industry players.

The OpenDaylight Controller supports not only the Openflow protocol but also other open protocols to allow communication with devices which have Openflow and/or respective Agents. It also includes a Northbound API to allow customer applications (software) which will work with the Controller in controlling the network. The Custom Apps cover a wide spectrum of solutions for solving customer needs across different vertical market segments.

The Controller architecture supports both the Hybrid Switch model as well as classical Openflow model of having a fully centralized Control Plane.

Hardware and Software Requirements:

OpenDaylight Controller is a JVM so it can run on any metal and OS provided it supports Java JVM 1.7+. We recommend the following:

Linux (Ubuntu or RHEL or Fedora or Any other popular Linux Distro that supports Java)

JVM 1.7+ (JAVA_HOME should be set to environment)

The Controller has a built in GUI. The GUI is implemented as an application using the same Northbound API as would be available for any other user application.

6.2 Architectural Principles

Runtime Modularity and Extensibility: Allow for a modular, extensible controller that supports installation, removal and updates of service implementations within a running controller, also known as in service software upgrade (ISSU)

Multiprotocol Southbound: Allow for more than one protocol interface with network elements with diverse capabilities southbound from the controller.

Service Abstraction Layer (SAL): Where possible, allow for multiple southbound protocols to present the same northbound service interfaces.

Open Extensible Northbound API: Allow for an extensible set of application-facing APIs both across runtimes via REST (level 3 API) and within the same runtime, i.e., via function calls (level 2 API). The set of accessible functions should be the same.

Support for Multitenancy/Slicing: Allow for the network to be logically (and/or physically) split into different slices or tenants with parts of the controller, modules, explicitly dedicated to one or a subset of these slices. This includes allowing the controller to present different views of the controller depending on which slice the caller is from.

Consistent Clustering: Clustering that gives fine-grained redundancy and scale out while insuring network consistency.

Open Extensible Northbound API

Allow for an extensible set of application-facing APIs both across runtimes via REST (level 3 API) and within the same runtime, i.e., via function calls (level 2 API). The set of accessible functions should be the same.

Here are the APIs exposed by the OpenDaylight projects:

OpenDaylight Controller:

AD-SAL REST and Java APIs

MD-SAL RESTCONF Northbound APIs

OpenDaylight Virtual Tenant Network - REST API

Open DOVE - Northbound API

Openflow Plugin - N/A

Affinity Metadata Service - N/A

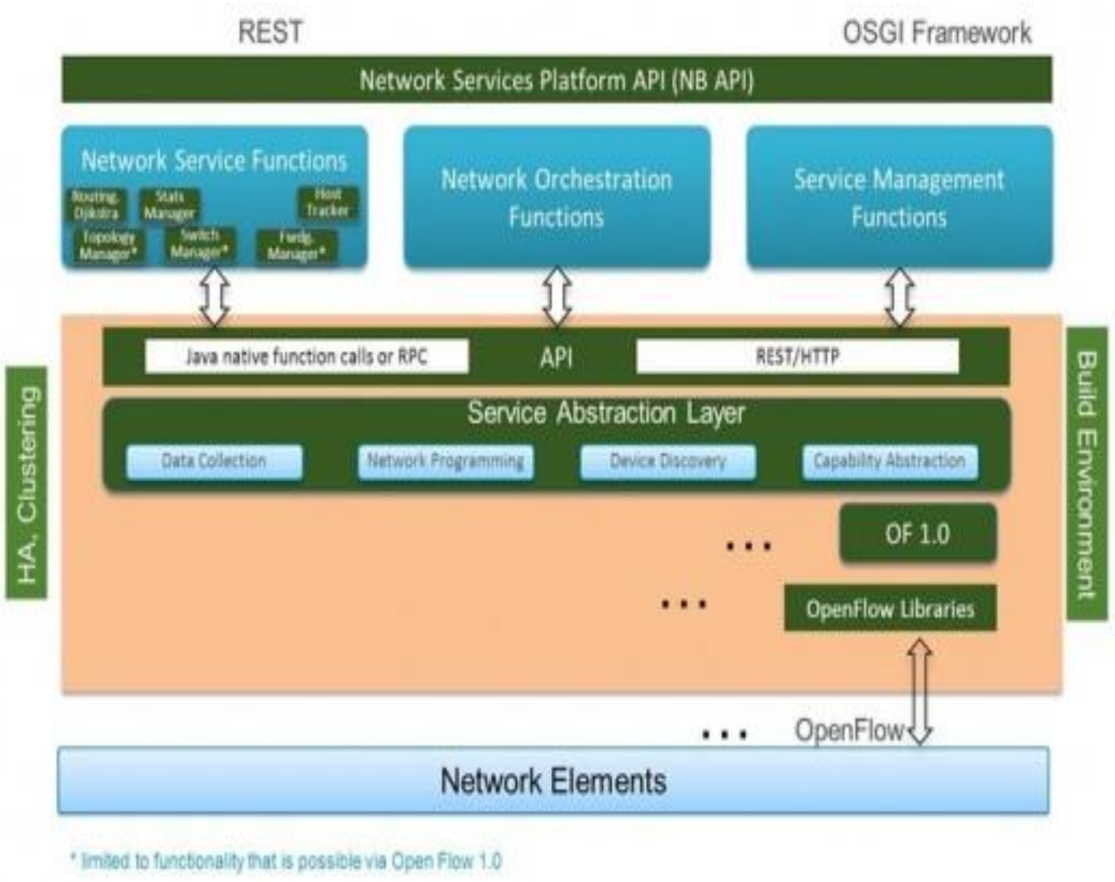
YANG Tools - Available YANG models

LISP Flow Mapping - Java API and REST API

OVSDB - REST API

6.3 Architectural Framework

The OpenDaylight Controller as it is pure software and basically a JVM it can run on any OS and hardware as long as it supports Java. The following picture shows the structure of the OpenDaylight Controller.



Picture 7: OpenDaylight controller structure

On the Southbound as stated earlier one can support multiple protocols (as plugins), e.g. Openflow 1.0, Openflow 1.3, BGP-LS, etc. The OpenDaylight Controller will start with an Openflow 1.0 Southbound plug in. Other OpenDaylight contributors would add to those as part of their contributions/projects. These modules are linked dynamically into a Service Abstraction Layer (SAL). The SAL exposes services to which the modules north of it are written. The SAL figures out how to fulfill the requested service irrespective of the underlying protocol used between the Controller and the network devices. This provides investment protection to the Applications as the Openflow and other protocols evolve over time. For the Controller to control devices in its domain it needs to know about the devices, their capabilities, reachability, etc. This information is stored and managed by the Topology Manager. The other components like ARP handler, Host Tracker, Device Manager and Switch Manager help in generating the topology database for the Topology Manager.

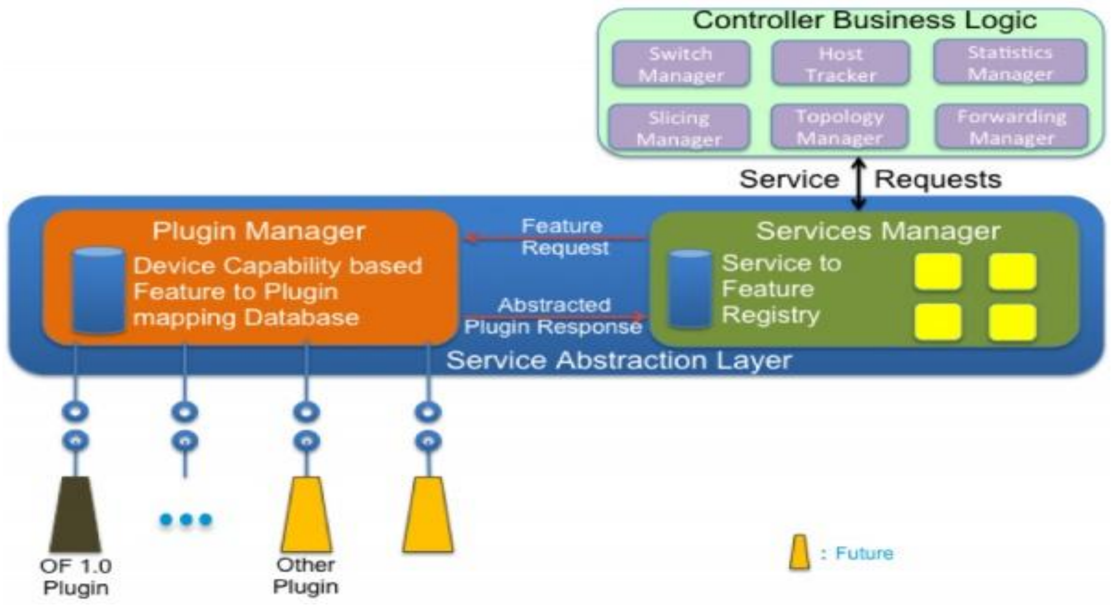
One of the key components is that the Controller exposes open Northbound APIs which can be used by Applications. OSGi framework and bidirectional REST is supported for the Northbound API. OSGi framework is used for applications that will run in the same address space as the Controller while the REST (web based) API is used for Apps that do not run in the same address space (or even the same metal) as the Controller. The business logic and algorithms reside in the Apps. These Apps use the Controller to gather network intelligence, runs its algorithm to do analytics and then use the Controller to orchestrate the new rules throughout the network.

The Controller has a built in GUI. The GUI is implemented as an application using the same Northbound API as would be available for any other user application. This can be called the first complete application developed using this Northbound API.

6.4 Functional Overview

Service Abstraction Layer

Multiple southbound protocols are linked consistently to the service modules and Apps through the Service Abstraction Layer (SAL) which is the heart of the modular design of the Controller.



Picture 8: OpenDaylight controller service abstraction layer

SAL is like a common understandable language between Northbound and Southbound interfaces.

The OSGi framework allows dynamically linking plugins for the evolving southbound protocols. The SAL provides basic services like Device Discovery which are used by modules like Topology Manager to build the topology and device capabilities. Services are constructed using the features exposed by the plugins (based on the presence of a plugin and capabilities of a network device). Based on the service request the SAL maps to the appropriate plugin and decides which is the most appropriate Southbound protocol to interact with a given network device. This decision can also be manually predefined. Each plugin is independent of each other and are loosely coupled with the SAL. Currently Openflow plugin is implemented but there is extensibility for other future plugins.

Topology Service is a set of services that allow conveying topology information like a new node a new link has been discovered and so on.

Data Packet services, in summary the possibility to deliver to applications the packets coming from the agents, if any.

Flow Programming service is supposed to provide the necessary logic to program in the different agents a Match/Actions rule.

Statistics service will export to the northbound API to be able to collect statistics at least per:

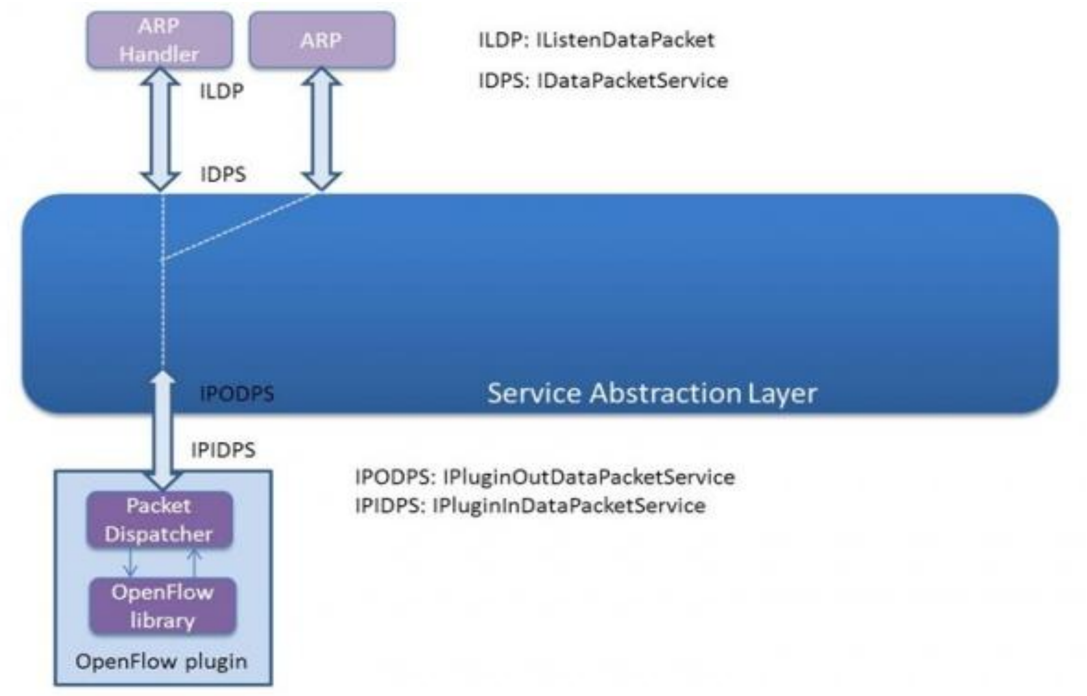
Flow Node, Connector (port),

Inventory service will provide APIs for returning inventory information about the node and node connectors for example

Resource service is a placeholder to query resource status

SAL Services: Data Packet Service

For better understanding of SAL let's have a look at the Data Packet Service with Openflow 1.0 plugin



Picture 9: OpenDaylight controller service abstraction layer with Openflow plugin

IListenDataPacket: is a service implemented by the Upper layer module or App (ARP Handler is one such module in the picture above) which want to receive data packets

IDataPacketService: This interface is implemented by the SAL and provides the service of sending and receiving packets from the Agent. This service will be registered in the OSGi service registry so that an application can retrieve it.

IPluginOutDataPacketService: This interface is exported by SAL when a Protocol Plugin wants to deliver a Packet toward the Application layer

IPluginInDataPacketService: This interface is exported by the Protocol Plugin and is used to send out the packets through SAL towards the Agent on the network devices.

Now let us see how the code execution takes place in the SAL with the Services and plugins:

Say the Openflow plugin receives an ARP packet that need to be dispatched to the ARP Handler Application

The Openflow Plugin will call IPluginOutDataPacketService to get the packet to the SAL.The ARP Handler Application would've registered to the IListenDataPacket Service. The SAL upon receiving the packet will handover the packet to the ARP Handler App.The Application can now process the packet.

For the reverse path of the Application sending a packet out, the execution flow would be:

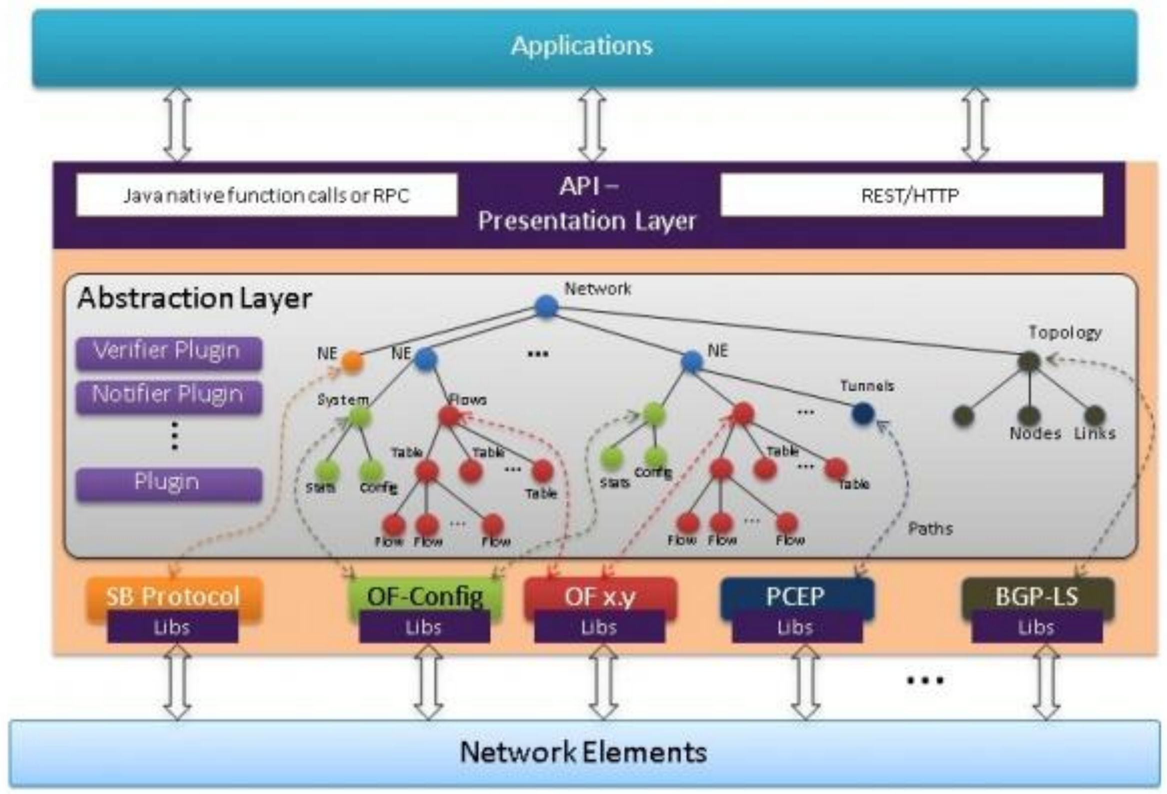
The Application constructs the packet and calls the interface IDataPacketService provided by SAL to send the packet. The Destination network device is to be provided as part of the API.

SAL will then call the IPluginInDataPacketService interface for a given Protocol plugin based on the destination network device (Openflow Plugin in this case).

The Protocol plugin will then ship the packet to the appropriate network element. The plugin will handle all protocol specific processing.

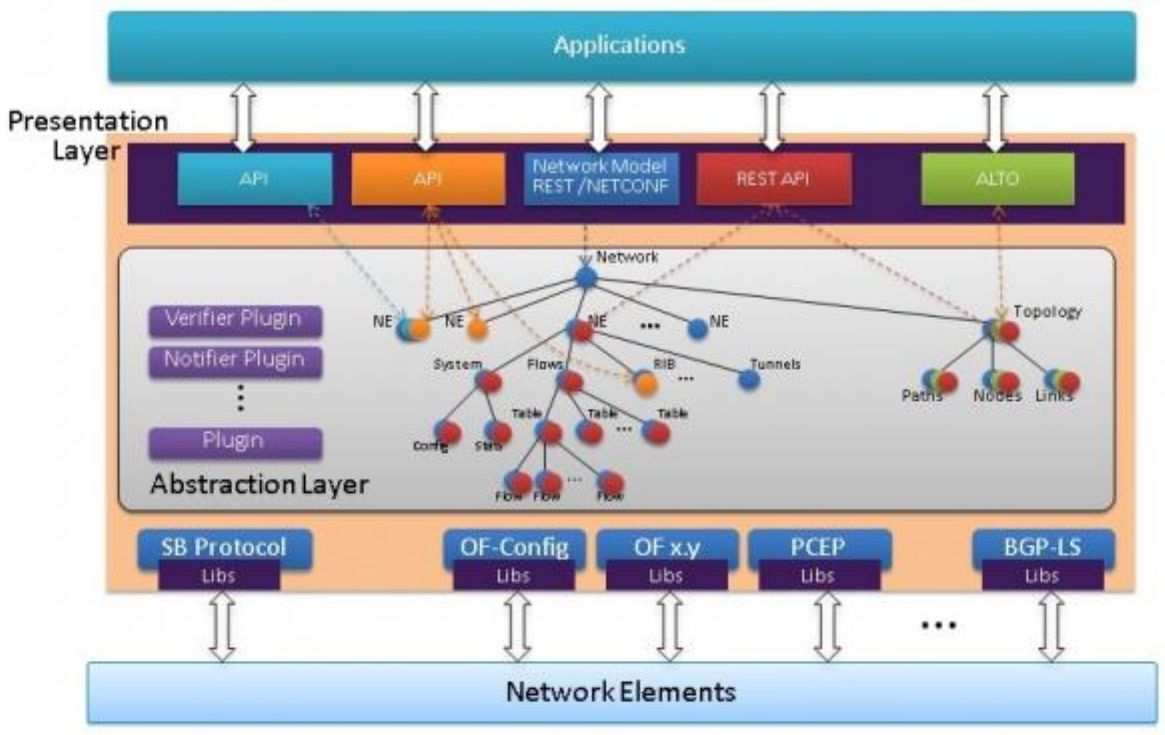
6.5 Evolution of the Controller Service Abstraction Layer

The SAL evolves into a model based approach, where a framework is provided to model the network, its properties and network devices, and dynamically map between services/applications using the north-bound APIs and protocol plugins providing the southbound APIs. The following figure shows how southbound plugins provide portions of the overall network model tree.



Picture 10: Southbound plugins

Picture 11 depicts the way applications can access information in the network model using the northbound API.



Picture 11: Access information through northbound APIs

6.6 Switch Manager

The Switch Manager API holds the details of the network element. As a network element is discovered, its basic attributes like what kind of device is , software version, capabilities ,etc. are stored in the data base by the Switch Manager

6.7 GUI

The GUI is implemented as an APP and uses the NB REST API to interact with the other modules of the Controller. This is the first complete App developed using the northbound API. This architecture thus ensures that whatever is possible with the GUI is also available via REST API and thus the Controller can be integrated easily into other management or orchestration systems that fit better every enterprise’s needs.

6.8 High Availability

High Availability in the distributed IP context is provided through several mechanisms:
• **Redundancy at the network level** (the “two of everything” approach, where redundant routers/switches and redundant paths in the network design allow for the failure of a link or element).
• **Redundancy at the element level** using redundant route processors/switch control modules. The redundant processors can work in either a stateless active/standby mode (which normally implies an interruption in forwarding if there is no alternative path) or through stateful mirroring of control process data (e.g., nonstop routing). [9]

The OpenDaylight Controller supports a Cluster based High Availability model. There are several instances of the OpenDaylight Controller which logically act as one logical controller. This not only gives a fine grain redundancy but also allows a scale-out model for linear scalability. To make the Controller highly available, we need to add resilience at: Controller level, by adding 1 or more controller instances in clustered fashion. For high end and high scale SDN implementations like an ISP we understand the importance of this High Availability model. As a common practice you have to take into consideration the following:

- Make sure the Open Flow enabled switches (OF-S elements) are multi-homed to multiple instances of the controller.
- Make sure the applications are multi-homed to the controller instances

The Openflow enabled Switches connect to two or more instances of the Controller via persistent point-to-point TCP/IP connection. On the northbound side the interaction between the controller and the applications is done via RESTful web services for all the Request-Response type of interaction, being those based on HTTP and being HTTP based on non-persistent connections between the server and the client, it's possible to leverage all the high-available techniques used to give resilience on the WEB, like:

- Provide the cluster of controller with a virtual IP to be reached via an anycast type of solution.
- Have the APP to talk to the cluster after a DNS request is done using a DNS round-robin technique.
- Deploy between the APPs and the cluster of controller an HTTP load-balancer that can then not only be used to provided resilience but also distributed the workload accordingly to the URL requested.

The interaction between the Controllers and the Open-Flow enabled switches is essentially to have one Openflow switch multi-homed to multiple controllers, so if one of the controllers goes down the other is ready to control the switch. This interaction has already been specified in the Openflow 1.2. To summarize it when having multiple controllers connected to one switch; the Openflow 1.2 specifications specify two modes of operations:

- Equal interaction: in this case all the controllers have read/write access to the switch, which means they have to synchronize in order not to step on each other feet.
- Master/Slave interaction: in this case there will be one master and multiple slaves (there could be still multiple equal as well)

In case of race conditions it seems safer the master/slave model where the switch will accept orders from only one controller (guaranteed). In other case you have to configure a very strict controller to controller interaction for internal synchronization.

For Controller (instance) to Controller (instance) interaction one needs to synchronize the following information amongst the instances:

- Topology in-memory database
- Switch and Host tracking database
- Configuration files

Master controller for a given Openflow switch, this could simply be based on simple metric like the highest IP address controller takes the master role, with designated backup being the next highest IP address.

- User database

It is assumed that the path calculation on each node can happen independently. If consistency is desired then we should include the paths in the information that need to be synchronized.

Apps using REST API use non-persistent connections between the App and the Controller (instance), which means that if a Controller instance goes down the App will reestablish a new connection on the next transaction. If the failure happens in the middle of a transaction then there will be an HTTP error and appropriate corrective action is taken.

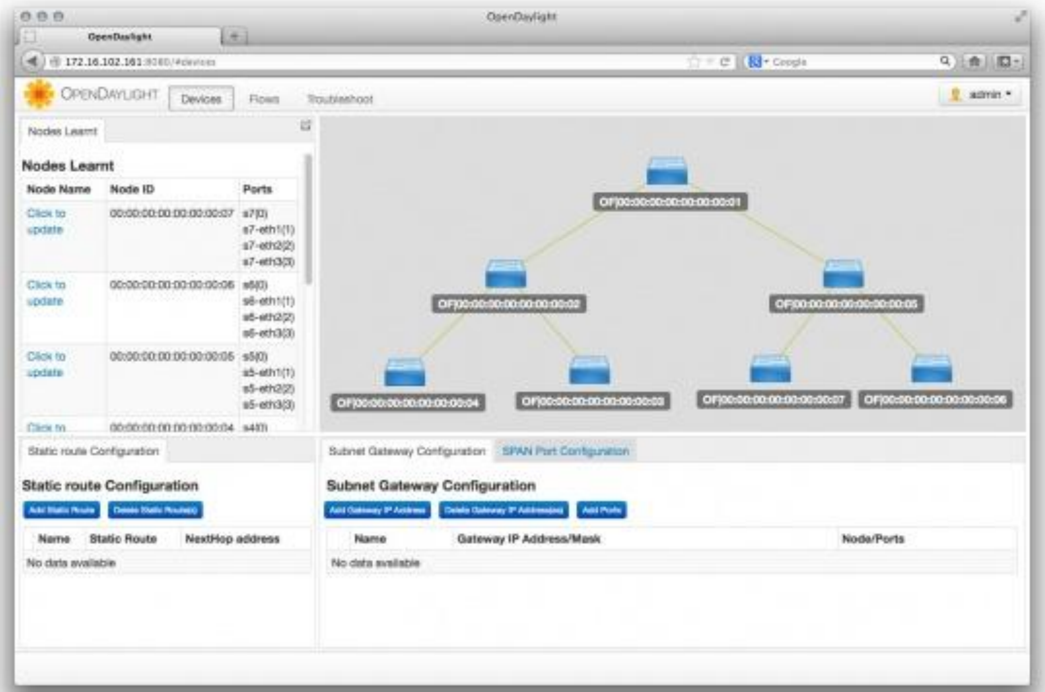
In case an App uses the OSGi framework then the App is running on one of the instances of the Controller. If that Controller instance goes down, the App goes down with it. However, it is the responsibility of the App to ensure its' own resiliency by having multiple instances and providing its own state synchronization between the instances. Such types of cases exist when you implement firewall clusters.

The Controller provides Clustering Services which the Controller modules can use to get state and event synchronization. It also provides a transaction API to maintain transactions across the nodes in a cluster.

6.9 Topologies

The OpenDaylight Controller provides you with a centralized logical view of their physical network topology, and enables you to coordinate forwarding rules changes with any of the network devices, on behalf of applications that directly manage network policies.

OpenDaylight Controller uses the LLDP (link layer discovery protocol) messages to discover the topology of the connected Openflow Devices. The Topology View tab provides a graphical view of the topology with switches and hosts (will be shown later). The OpenDaylight Controller's Topology Manager stores and manages information about the devices in the domain, including their capabilities and reachability. This information is stored and managed by the Topology Manager. Other components, including the ARP handler, Host Tracker, Device Manager, and Switch Manager help generate the topology database for the Topology Manager.


OpenDaylight
172.16.102.161:8080/#devices
OPENDAYLIGHT
Devices
Flows
Troubleshoot
admin
Nodes Learnt
OF|00:00:00:00:00:00:00:01
OF|00:00:00:00:00:00:00:02
OF|00:00:00:00:00:00:00:05
OF|00:00:00:00:00:00:00:04
OF|00:00:00:00:00:00:00:03
OF|00:00:00:00:00:00:00:07
OF|00:00:00:00:00:00:00:06
Static route Configuration
Subnet Gateway Configuration
SPAN Port Configuration
Add Static Route
Delete Static Route(s)
Add Gateway IP Address
Delete Gateway IP Address(es)
Add Ports

Nodes Learnt

Node Name	Node ID	Ports
Click to update	00:00:00:00:00:00:00:07	s7(0) s7-eth1(1) s7-eth2(2) s7-eth3(3)
Click to update	00:00:00:00:00:00:00:06	s6(0) s6-eth1(1) s6-eth2(2) s6-eth3(3)
Click to update	00:00:00:00:00:00:00:05	s5(0) s5-eth1(1) s5-eth2(2) s5-eth3(3)
Click to	00:00:00:00:00:00:00:04	s4(0)

Static route Configuration

Name	Static Route	NextHop address
No data available		

Subnet Gateway Configuration

Name	Gateway IP Address/Mask	Node/Ports
No data available		

Picture 12: OpenDaylight controller web interface. Topology manager

6.10 Openflow Plugin Proposal

Openflow is a vendor-neutral standard communications interface defined to enable interaction between the control and forwarding layers of SDN architecture (southbound). The Openflow plugin project intends to develop a plugin to support implementations of the Openflow specification as it develops and evolves. Specifically the project will continue to provide support for the existing Openflow 1.0 implementation, developing a plugin aiming to support newer versions of Openflow , and further supporting subsequent Openflow specifications. The plugin shall be implemented in such a way that existing and future Openflow protocol specifications can be easily integrated and published for the OpenDaylight controller.

6.11 AD-SAL

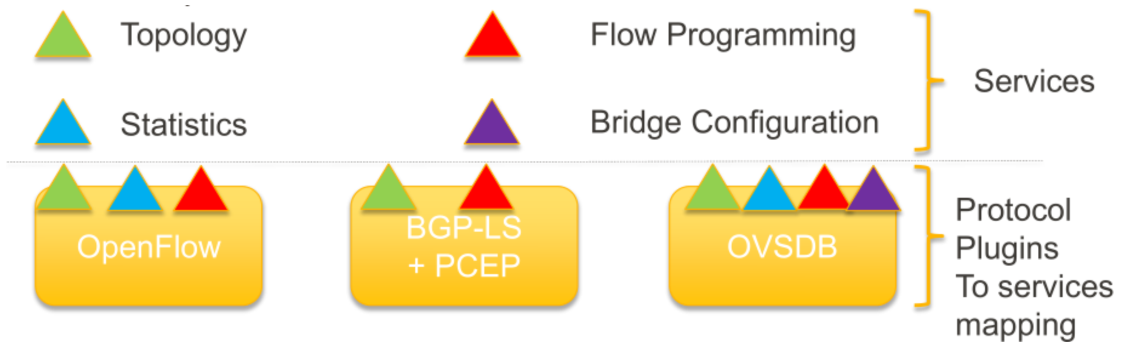
AD-SAL is a flavor of the Service Abstraction Layer created for the Controller project, whose task is primarily to create a layer against which the applications can be developed without the knowledge of the underlying SDN protocol. The logic has been created when there has been a need to control two different types of network elements in an SDN fashion, meaning by accessing them in a programmatic way. It was clear that irrespective of the different types of network element certain things would still be applicable, practical and simple examples are:

- Given a Network Element I want to know who my neighbors are and build the network graph
- Given a Network Element I want to know what interfaces are attached to it and what properties they carry etc.

Starting from this assumption that certain aspects of the SDN protocols would anyway be common no matter what the actual protocol is doing underneath prompted for the idea those aspects could be described in a generic way via a Java Contract (usually a set of Java Interfaces and supporting Objects that represent the data). This way of creating contracts that allows a consumer of the contract to be insulated by the possible implementation is rather common in the Java world, where for example there are contracts like:

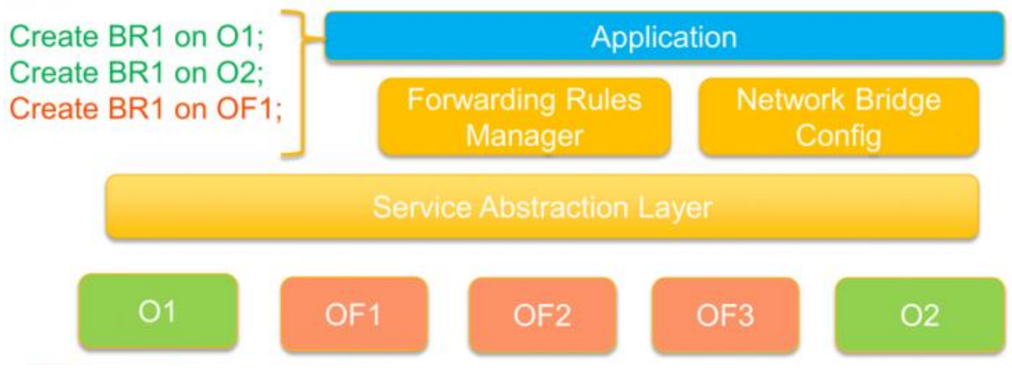
- Servlet 3.0
- JAX-RS

Which are described by a JSR (Java Specification Request) which then can be implemented by different providers? For example Servlet 3.0 contracts could be implemented by Tomcat or Jetty or other servlet container. By going by the same analogy the protocol plugins (i.e. the components that can understand an SDN protocol and convert to the common layer) can implement one or more of these contracts based on the capabilities. The Service Abstraction Layer defined in AD-SAL is just an adaptation layer to enable the protocol plugins to speak common aspects but only if they can do it. The service abstraction layer in fact doesn't force a lowest common denominator for all to be spoken, but rather given a certain aspect if the protocol plugins supports it will be supported in a common fashion. Let's take as example the picture below:



Picture 13: Protocol plugins to Services

In the picture there are 4 types of contracts, identified by the colored triangles, and there are 3 protocol plugins that provide implementation for those contracts. For example the Topology contract (the one used to learn the network graph) is implemented by all of the protocol plugins, as expected, while the "Bridge Configuration" contract is only implemented by OVSDB because the others are not capable of it. That is perfectly fine and actually expected because that allows not reverting to the lowest common denominator feature of all the protocol plugins. Now an application when trying to use a service on a given network element has to expect that a given contract may not be there, because protocol plugin cannot create stuff don't exist, but this allows to seamlessly plug in protocol plugins that implement a given contract and application would be able to leverage it due to the fact that they have been already programmed to understand it. The picture below explains how an application has to deal with the optional presence of a contract on a given network element:



Picture 14: Application

In picture 14 an application is trying to create a bridge on Nodes O1, O2 and OF1, the first two are capable of that contract, the third one is not, so the application has to take care of it and handle the return error code.

In summary AD-SAL is just enforcing that the protocol plugins speak a common language for words that can be expressed in a common language. The dictionary of the common language is not hardcoded, it can be extended supplying new contracts to AD-SAL such that new protocols plugin can implement it and applications can consume them.

6.12 MD-SAL: Architecture

Model-driven approach to service abstraction presents an opportunity to unify both northbound and southbound APIs and the data structures used in various services and components of an SDN Controller.

In order to describe the structure of data provided by controller components a domain-specific language, YANG (data modeling language for the NETCONF network configuration protocol), is proposed as the modeling language for service and data abstractions. Such language allows to:

- Modeling the structure of XML data and functionality provided by controller components
- Define semantic elements and their relationships
- Model all the components as a single system.

YANG is a modular language representing data structures in an XML tree format. The XML nature of YANG data model presents an opportunity for self-describing data, which controller components and applications using the controller's northbound APIs can consume in a raw format, along with the data's schema.

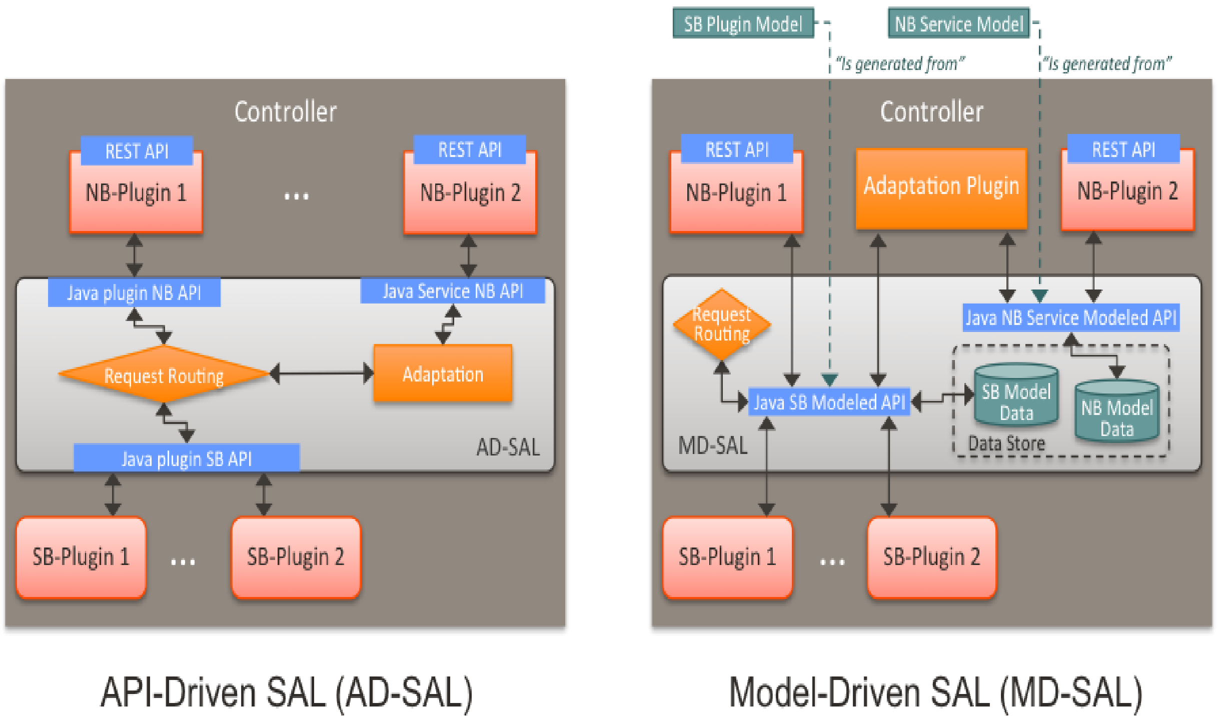
Utilizing a schema language simplifies development of controller components and application. A developer of a module that provides some functionality (a service, data, and functions/procedures) can define a schema and thus create simpler, statically typed APIs for the provided functionality, and thus lower the risk of incorrect interpretation of data structures exposed through the Service Abstraction Layer.

6.12.1 MD SAL and AD SAL

The overall Model-Driven SAL (MD-SAL) architecture is similar to the API-Driven SAL (AD-SAL). As with the AD-SAL, plugins can be data providers, or data consumers, or both just like the AD-SAL, the MD-SAL connects data consumers to appropriate data providers and facilitates data adaptation between them.

Now, in the AD-SAL, the SAL APIs, request routing between consumers and providers, and data adaptations are all statically defined at compile or build time. In the MD-SAL, the SAL APIs and request routing between consumers and providers are model defined, and data adaptations are provided by 'internal' adaptation plugins. The API code is generated from these models when a plugin is compiled. When the plugin OSGI bundle is loaded into the controller, the API code is loaded into the controller along with the rest of the plugin containing the model.

The AD-SAL and the MD-SAL are shown side-by-side in the following figure:



Picture 15: AD-SAL and MD-SAL

The AD-SAL provides request routing (selects an SB plugin based on service type) and optionally provides service adaptation, if an NB (Service, abstract) API is different from its corresponding SB (protocol) API. For example, picture 15, the AD-SAL routes requests from NB-Plugin 1 to SB Plugins 1 and 2. Note that the plugin SB and NB APIs in this example are essentially the same (although both of them need to be defined). Request routing is based on plugin type: the SAL knows which node instance is served by which plugin. When an NB Plugin requests an operation on a given node, the request is routed to the appropriate plugin which then routes the request to the appropriate node. The AD-SAL can also provide service abstractions and adaptations. For example, in the above figure, NB Plugin 2 is using an abstract API to access the services provided by SB Plugins 1 and 2. The translation between the SB Plugin API and the abstract NB API is done in the Abstraction module in the AD-SAL.

The MD-SAL provides request routing and the infrastructure to support service adaptation. However, it does not provide service adaptation itself: service adaptation is provided by plugins. From the point of view of MD-SAL, the Adaptation Plugin is a regular plugin. It provides data to the SAL, and consumes data form the SAL through APIs generated from models. An Adaptation Plugin basically performs model-to-model translations between two APIs. Request Routing in the MD-SAL is done on both protocol type and node instances; since node instance data is exported from the plugin into the SAL (the model data contains routing information).

The simplest MD-SAL APIs generated from models are functionally equivalent to AD-SAL function call APIs. Additionally, the MD-SAL can store data for models defined by plugins. Provider and consumer plugins can exchange data through the MD-SAL storage (more details in later sections). Data in the MD-SAL is accessed through getter and setter APIs generated from models. Note that this is in contrast to the AD-SAL, which is stateless.

Note that in the above figure, both NB AD-SAL Plugins provide REST APIs to controller client applications. Functionality provided by the MD-SAL is basically to facilitate the plumbing between providers and consumers. A provider or a consumer can register itself with the MD-SAL. A consumer can find a provider that it's interested in. A provider can generate notifications; a consumer can receive notifications and issue RPCs to get data from providers. A provider can insert data into SAL's storage; a consumer can read data from SAL's storage.

Note that the structure of SAL APIs is different in the MD-SAL from that in the AD-SAL. The AD-SAL typically has both NB and SB APIs even for functions or services that are mapped 1:1 between SB Plugins and NB Plugins. For example, in the current AD-SAL implementation of the Openflow Plugin and applications, the NB SAL APIs used by OF applications are mapped 1:1 onto SB OF Plugin APIs. The MD-SAL allows both the NB plugins and SB plugins to use the same API generated from a model. One plugin becomes an API (service) provider; the other becomes an API (service) Consumer. This eliminates the need to define two different APIs and to provide three different implementations even for cases where APIs are mapped to each other 1:1. The MD SAL provides instance-based request routing between multiple provider plugins.

6.12.2 MD-SAL Plugin

All plugins (protocol, application, adaptation, and others) have the same lifecycle. The life of a plugin has two distinct phases: design and operation.

In design phase, the plugin designer performs the following actions:

The designer decides which data will be consumed by the plugin and imports the SAL APIs generated from the API provider's models. Note that the topology model is just one possible data type that may be consumed by a plugin. A list of currently available data models and their APIs exist.

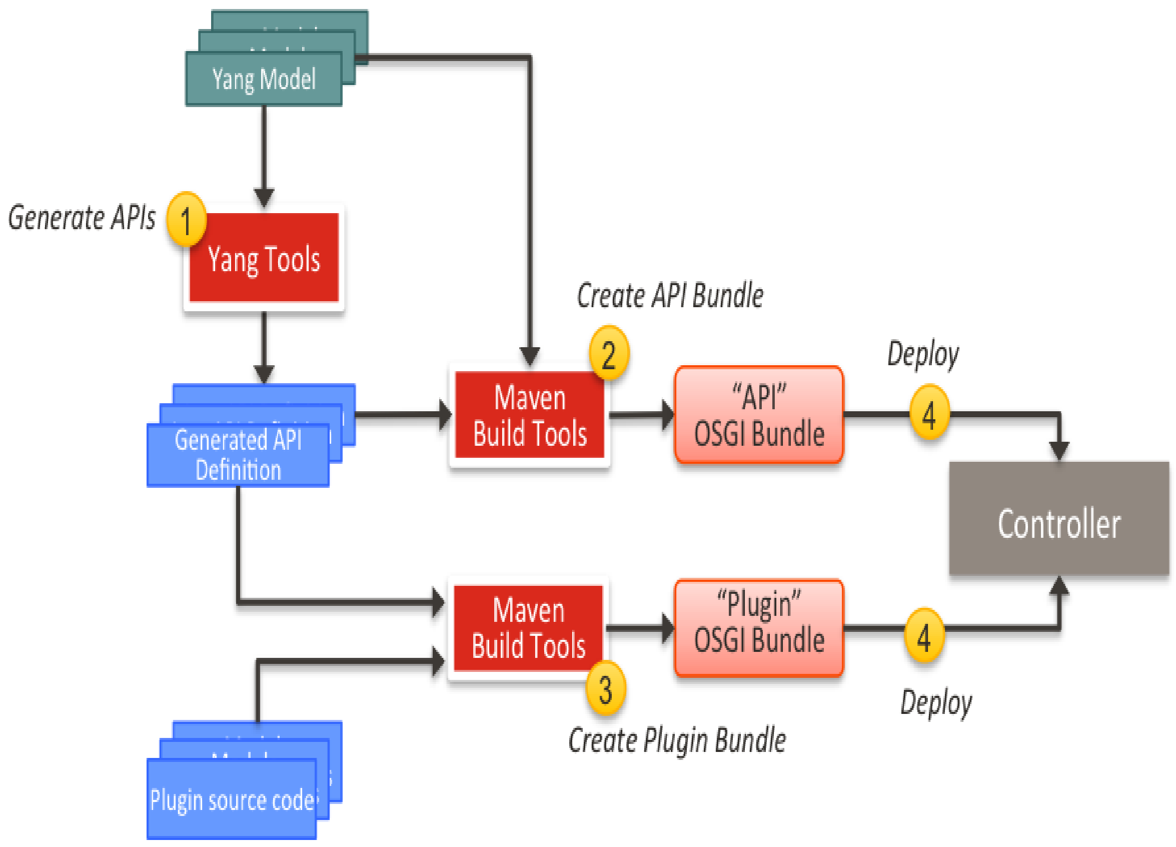
The designer decides which data and how will be provided by the plugin and designs the data model for the provided data. The data model (expressed in yang) is then run through the yang tools, which generate the SAL APIs for the model.

The implementations for the generated consumer and provider APIs, along with other plugin features and functionality, are developed. The resulting code is packaged in a "plugin" OSGI bundle. Note that a developer may package the code of a subsystem in multiple plugins or applications that may communicate with each other through the SAL.

These generated APIs and a set of helper classes are also built and packaged in an "API" OSGI bundle.

6.12.3 Plugin Development Process

The plugin development process is shown in the following figure.

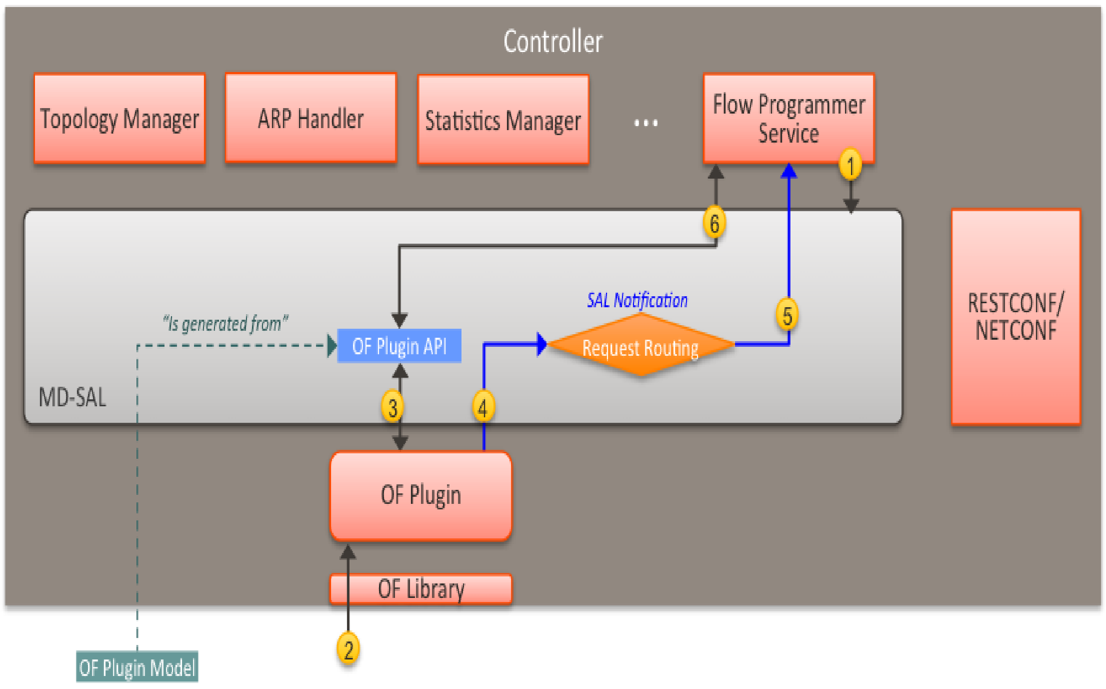


Picture 16: Plugin development process

The operation phase begins by the time the OSGi bundle is loaded and active. The operation of the OF Protocol plugin and OF applications, such as the Flow Programmer Service, the ARP Handler or the Topology Manager is good way to explain the plugin operation.

6.12.4 "Flow Deleted" notification scenario

The following figure shows a scenario where a “Flow Deleted” notification from a switch arrives at the controller.



Picture 17: Flow deleted notification

The scenario is as follows:

The Flow Programmer Service registers with the MD SAL for the “Flow Deleted” notification. This is done when the Controller and its plugins or applications are started.

A “Flow Deleted” Openflow packet arrives at the controller. The Openflow Library receives the packet on the TCP/TLS connection to the sending switch and passes it to the Openflow Plugin.

The OF Plugin parses the packet and uses the parsed data to create a “Flow Deleted” SAL notification. The notification is actually a “Flow Deleted” Data Transfer Object (DTO) that is created by means of methods from the model-generated OF Plugin API.

The Openflow Plugin sends the “Flow Deleted” into the SAL. The SAL routes the notification to registered consumers, in this case, the Flow Programmer Service.

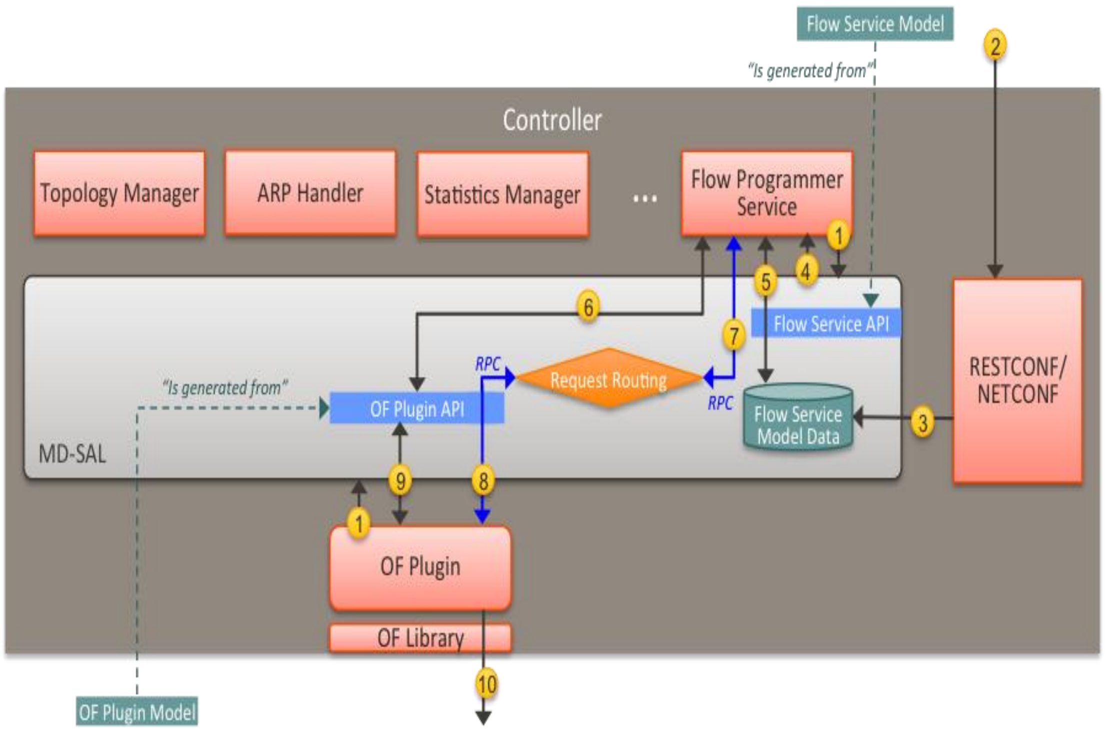
The Flow Programmer Service receives the notification containing the notification DTO.

The Flow Programmer Service uses methods from the model-generated Openflow Plugin API to get data from the immutable notification DTO received in Step 5. The processing is the same as in the AD-SAL.

Note that other packet-in scenarios, where a switch punts a packet to the controller, such as an ARP or an LLDP packet, are similar. Each Interested app registers for the respective notifications. The Openflow plugin generates the notification from received Openflow packets, and sends them to the SAL. The SAL routes the notifications to the registered recipients.

6.12.5 "Add Flow" scenario via NB REST API invocation

The following figure shows a scenario where an external application adds a flow by means of the NB REST API of the controller.



Picture 18: Add flow through northbound REST API

The scenario is as follows:

Registrations are performed when the Controller and its plugins or applications are started. a) The Flow Programmer Service registers with the MD SAL for Flow configuration data notifications, and b) The Openflow Plugin registers (among others) the ‘AddFlow’ RPC implementation with the SAL. Note that the RPC is defined in the OF Plugin model, and the API is generated during build time.

A client application requests a flow add through the REST API of the Controller. The client application provides all parameters for the flow in the REST call.

Data from the ‘Add Flow’ request are parsed, and a new flow is created in the Flow Service configuration data tree. Note also that the REST call returns success notification to the caller as soon as the flow data is written to the configuration data tree.

Since the Flow Programmer Service is registered to receive notifications for data changes in the Flow Service data tree, the MD-SAL generates a *‘data changed’* notification to the Flow Programmer Service.

The Flow Programmer Service reads the newly added flow and performs a flow add operation (which is basically the same as in the AD-SAL).

At some point during the flow addition operation, the Flow Programmer Service needs to tell the OF Plugin to add the flow in the appropriate switch. The Flow Programmer Service uses the OF Plugin generated API to create the RPC input parameter DTO for the “AddFlow” RPC of the OF Plugin.

The Flow Programmer Service gets the service instance (actually, a proxy), and invokes the “AddFlow” RPC on the service. The MD-SAL will route the request to the appropriate OF Plugin (which implements the requested RPC)

The “AddFlow” RPC request is routed to the OF Plugin, and the implementation method of the “AddFlow” RPC is invoked.

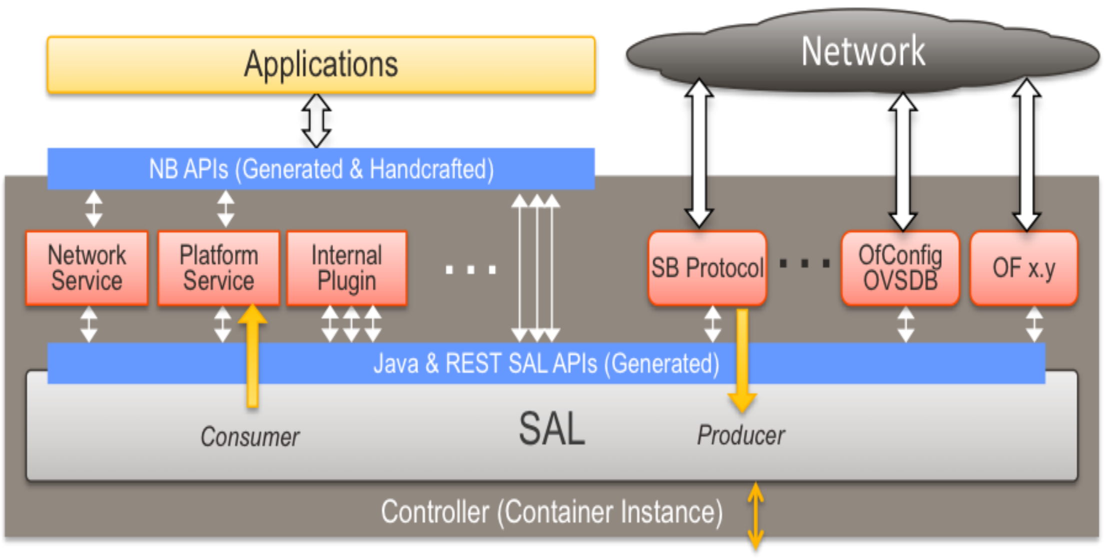
The “AddFlow” RPC implementation uses the OF Plugin API to read values from the DTO of the RPC input parameter. (Note that the implementation will use the getter methods of the DTO generated from the yang model of the RPC to read the values from the received DTO.)

The "AddFlow" RPC is further processed (pretty much the same as in the AD-SAL) and at some point, the corresponding flowmod is sent to the corresponding switch.

6.13 Producers and Consumers. Southbound and Northbound SAL plugins.

The big difference between southbound and northbound plugins is that southbound talk with a protocol to network devices while northbound with APIs to the related applications. In fact SAL cannot distinguish between south and north. The SAL is basically a way to exchange data and more an adaptation mechanism between plugins. The plugin SAL roles (consumer or producer) are defined with respect to the data being moved around or stored by the SAL. Producer implements an API and provides the data of the API: Consumer uses the API and consumes the data of the available API.

While 'northbound' and 'southbound' provide a network engineer's view of the SAL, 'consumer' and 'producer' provide a software engineer's view of the SAL, and is shown in the following figure:[8]



Picture 19: Software engineer’s view of SAL

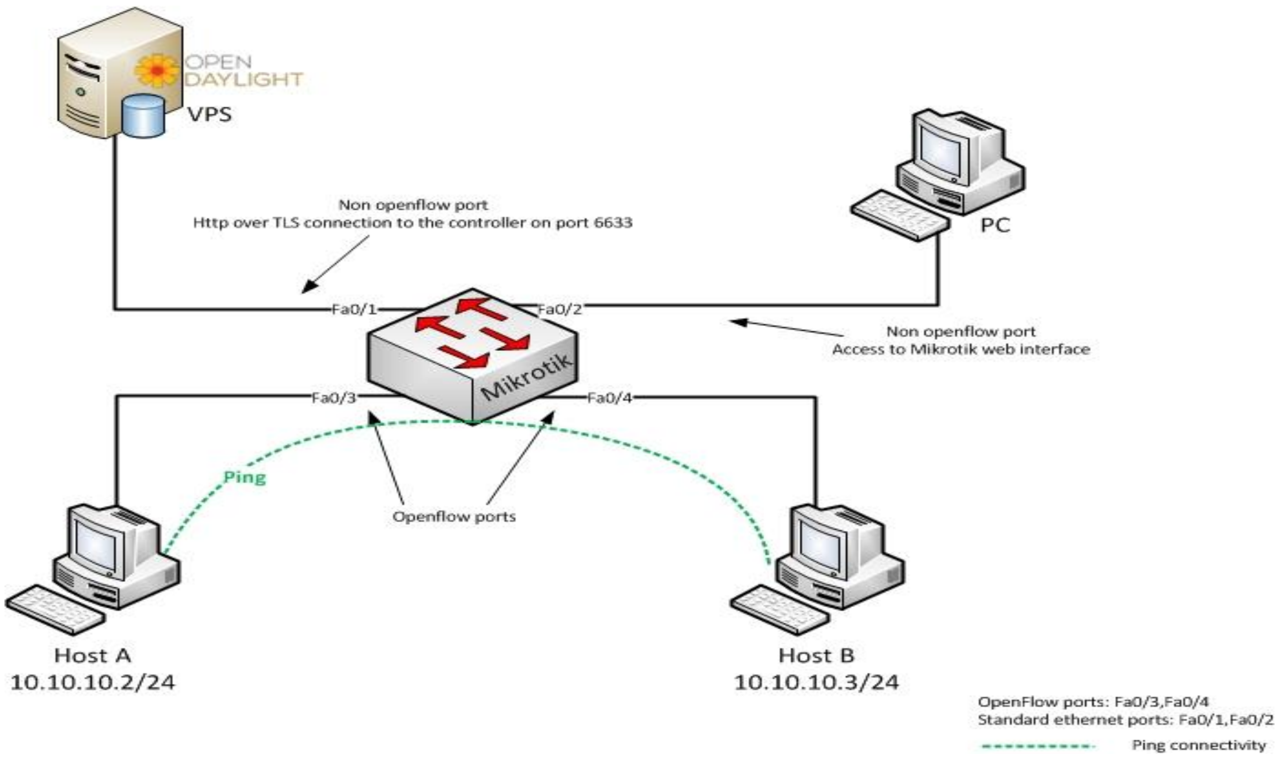
7. SDN OPENFLOW LAB

For this experiment we install OpenDaylight Controller on a VPS and control a mikrotik Openflow enabled switch. Three of the five ports of the switch are configured as Openflow ports. The IP of the controller is defined and configured on the Openflow switch and a HTTP over TLS session is established through one of the non-Openflow ports. Two hosts (PCs) are connected in the Openflow ports and one PC is connected in the last non-Openflow port in order to gain access in the mikrotik switch web interface.

The VPS details are:

- Operating System → Ubuntu 12.04 LTS
- CPU → 2,3 Ghz
- RAM → 512MB
- Switch details are:
- Model→ Mikrotik RB951Ui-2HnD
- Operating system→ RouterOS
- Openflow version → 1.0
- Host details are:
- Host 1 → laptop with ip address 10.10.10.2/24
- Host 2 → laptop with ip address 10.10.10.3/24

The design below depicts the connectivity between the elements.



Picture 20: Lab design

A communication between two PCs (icmp protocol/ping) based on flows installed on the Openflow switch through the use of the controller will be demonstrated.

Afterwards, flows that will break this communication based on specific header fields (matching source address) will be installed.

Finally, flow table and statistics per flow and port will be presented build on:

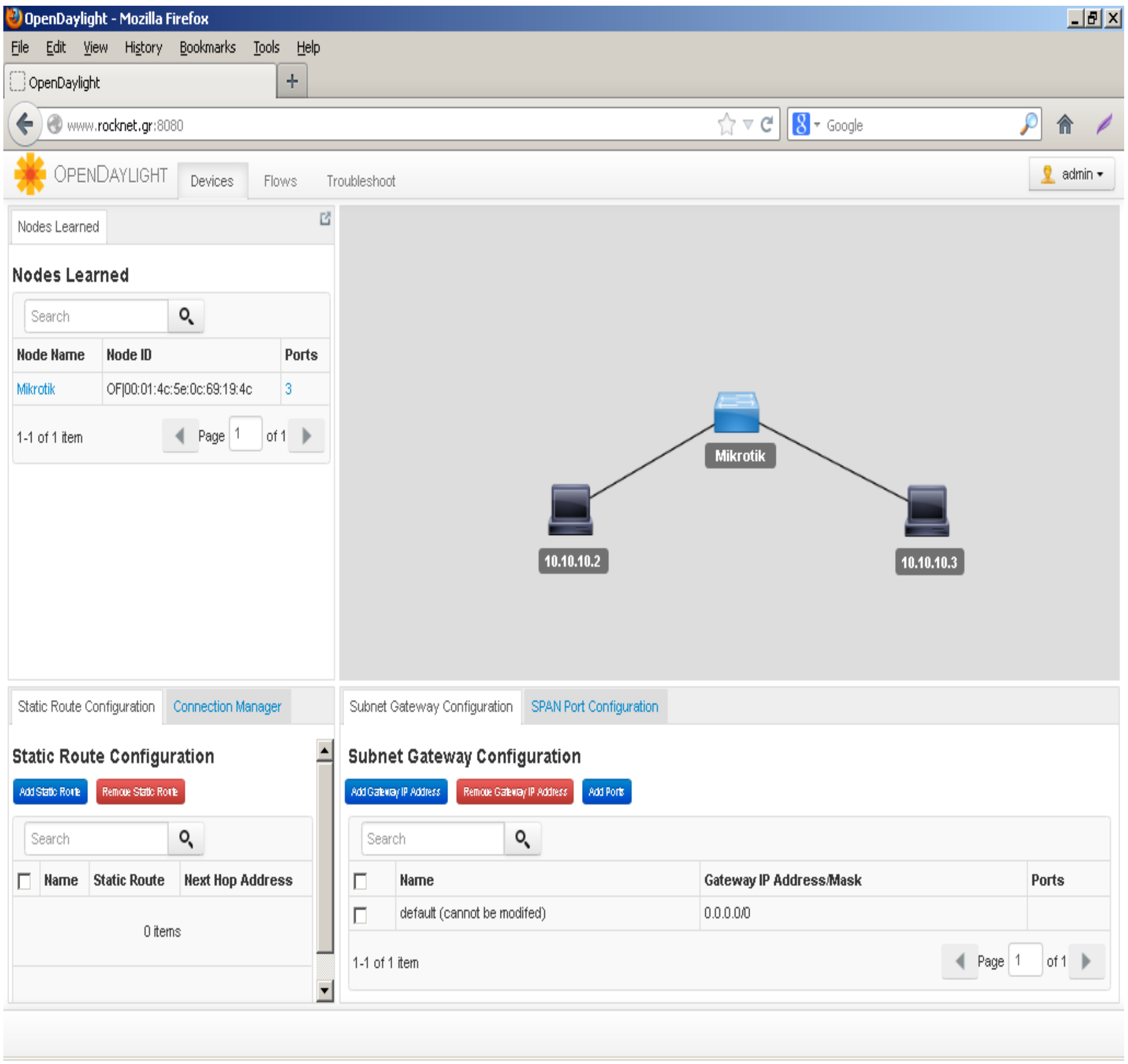
a) Switch’s web interface

b) Controller’s web interface

c) Controller’s northbound interface as xml (REST).

Step 1

The following picture shows the web interface of the OpenDaylight controller with the Mikrotik (Openflow) switch learned as a node. Three ports are discovered as Openflow ports. When you first connect the hosts two flows with priority 1 are installed in order to enable connectivity between them (default action flood). In order to start with no connectivity between the hosts we will install a new flow denying everything with higher priority.



Nodes Learned

Node Name	Node ID	Ports
Mikrotik	OF|00:01:4c:5e:0c:69:19:4c	3

1-1 of 1 item Page 1 of 1

Static Route Configuration

	Name	Static Route	Next Hop Address

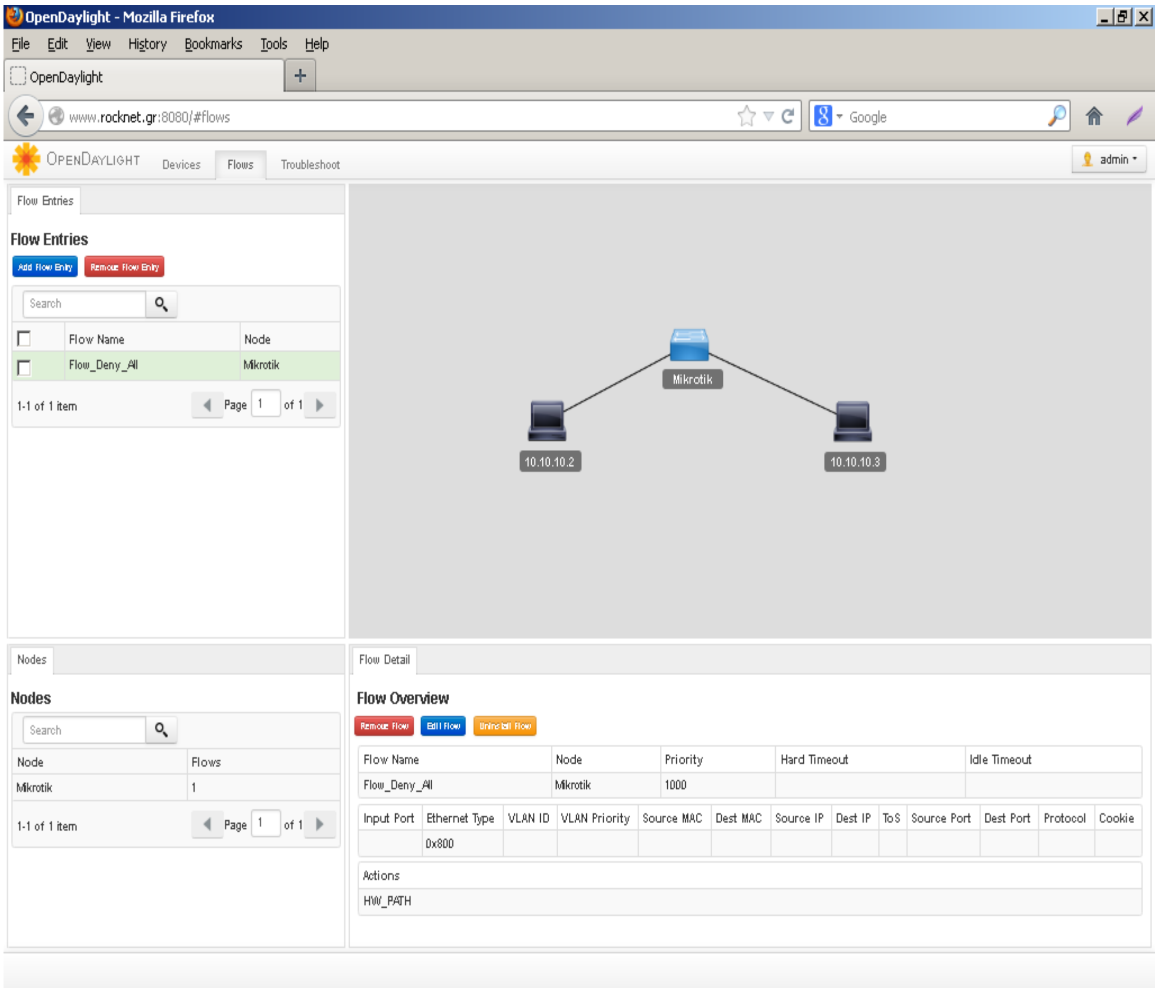
0 items

Subnet Gateway Configuration

	Name	Gateway IP Address/Mask	Ports
	default (cannot be modifed)	0.0.0.0/0	

1-1 of 1 item Page 1 of 1

Picture 21: Lab topology. Devices tab



Flow Entries

	Flow Name	Node
☐	Flow_Deny_All	Mikrotik

1-1 of 1 item Page 1 of 1

Nodes

Node	Flows
Mikrotik	1

1-1 of 1 item Page 1 of 1

Flow Detail

Flow Overview

Flow Name	Node	Priority	Hard Timeout	Idle Timeout
Flow_Deny_All	Mikrotik	1000		

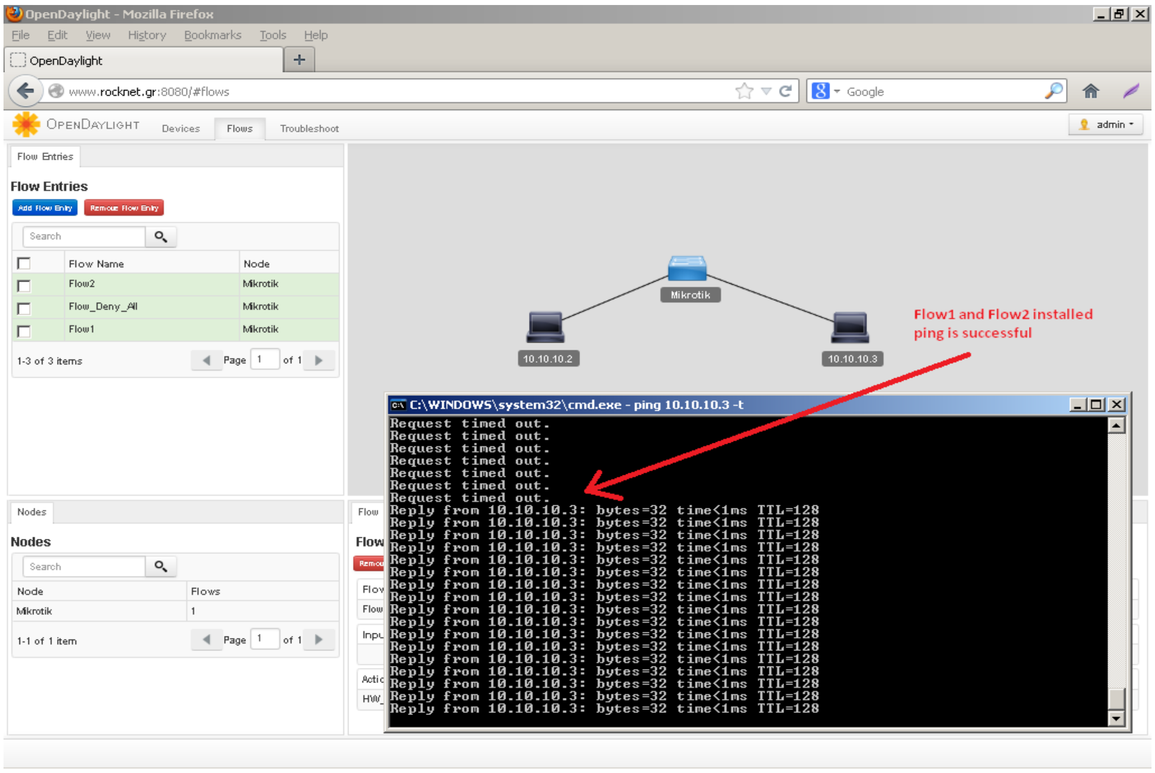
Input Port	Ethernet Type	VLAN ID	VLAN Priority	Source MAC	Dest MAC	Source IP	Dest IP	ToS	Source Port	Dest Port	Protocol	Cookie
	0x800											

Actions
HW_PATH

Picture 22: Flows installed. Flow tab

Step 2

As every packet is dropped achieving zero communication we need to enable the ping between host1 and host2. Two flows, each for every direction, are necessary. The flows are installed from the OpenDaylight's web interface and will enable the connectivity matching specific source/destination ip addresses. When installed the ping is successful.


OpenDaylight - Mozilla Firefox
File Edit View History Bookmarks Tools Help
OpenDaylight
www.rocknet.gr:8080/#flows
Google
OpenDaylight
Devices
Flows
Troubleshoot
admin
Mikrotik
10.10.10.2
10.10.10.3
Flow1 and Flow2 installed
ping is successful

Flow Entries

	Flow Name	Node
☐	Flow2	Mikrotik
☐	Flow_Deny_All	Mikrotik
☐	Flow1	Mikrotik

1-3 of 3 items Page 1 of 1

Nodes

Node	Flows
Mikrotik	1

1-1 of 1 item Page 1 of 1

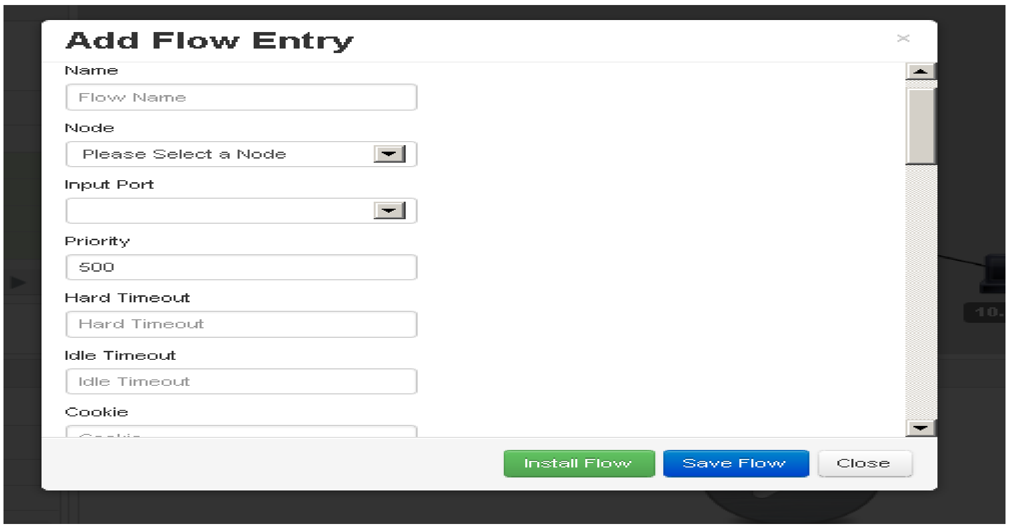
```
C:\WINDOWS\system32\cmd.exe - ping 10.10.10.3 -t
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
```

Picture 23: Flows and communication verification

Adding a flow OpenDaylight web interface menu

Extra fields appear with the side bar

.



Add Flow Entry

Name: Flow Name

Node: Please Select a Node

Input Port:

Priority: 500

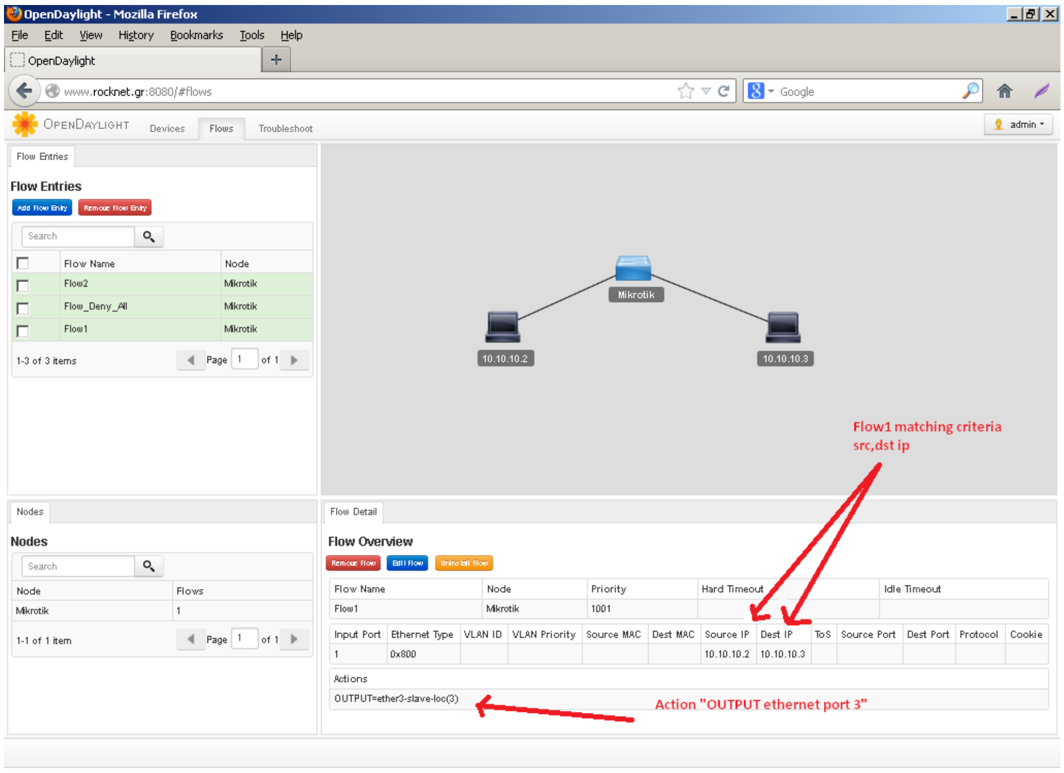
Hard Timeout: Hard Timeout

Idle Timeout: Idle Timeout

Cookie: Cookie

Picture 24: Add flow entry. OpenDaylight web interface

The flow details appear in the next two pictures.


OpenDaylight - Mozilla Firefox
File Edit View History Bookmarks Tools Help
OpenDaylight
www.rocknet.gr:8080/#flows
Google
OpenDaylight
Devices
Flows
Troubleshoot
admin
Mikrotik
10.10.10.2
10.10.10.3
Flow1 matching criteria
src,dst ip
Action "OUTPUT ethernet port 3"

Flow Entries

	Flow Name	Node
	Flow2	Mikrotik
	Flow_Deny_All	Mikrotik
	Flow1	Mikrotik

1-3 of 3 items Page 1 of 1

Nodes

Node	Flows
Mikrotik	1

1-1 of 1 item Page 1 of 1

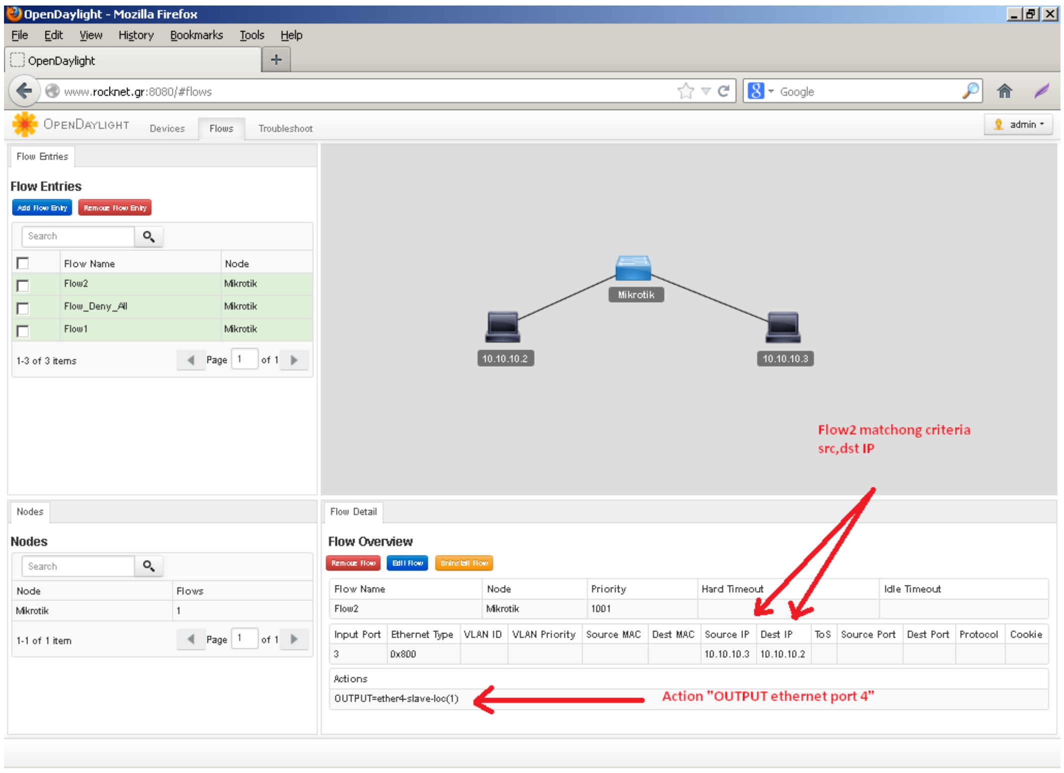
Flow Overview

Flow Name	Node	Priority	Hard Timeout	Idle Timeout
Flow1	Mikrotik	1001		

Input Port	Ethernet Type	VLAN ID	VLAN Priority	Source MAC	Dest MAC	Source IP	Dest IP	ToS	Source Port	Dest Port	Protocol	Cookie
1	0x800					10.10.10.2	10.10.10.3					

Actions
OUTPUT=ether3-slave-loc(3)

Picture 25: Flow1 details. Flows tab


OpenDaylight - Mozilla Firefox
File Edit View History Bookmarks Tools Help
OpenDaylight
www.rocknet.gr:8080/#flows
Google
OpenDaylight
Devices
Flows
Troubleshoot
admin
Mikrotik
10.10.10.2
10.10.10.3
Flow2 matchong criteria
src,dst IP
Action "OUTPUT ethernet port 4"

Flow Entries

	Flow Name	Node
	Flow2	Mikrotik
	Flow_Deny_All	Mikrotik
	Flow1	Mikrotik

1-3 of 3 items Page 1 of 1

Nodes

Node	Flows
Mikrotik	1

1-1 of 1 item Page 1 of 1

Flow Overview

Flow Name	Node	Priority	Hard Timeout	Idle Timeout
Flow2	Mikrotik	1001		

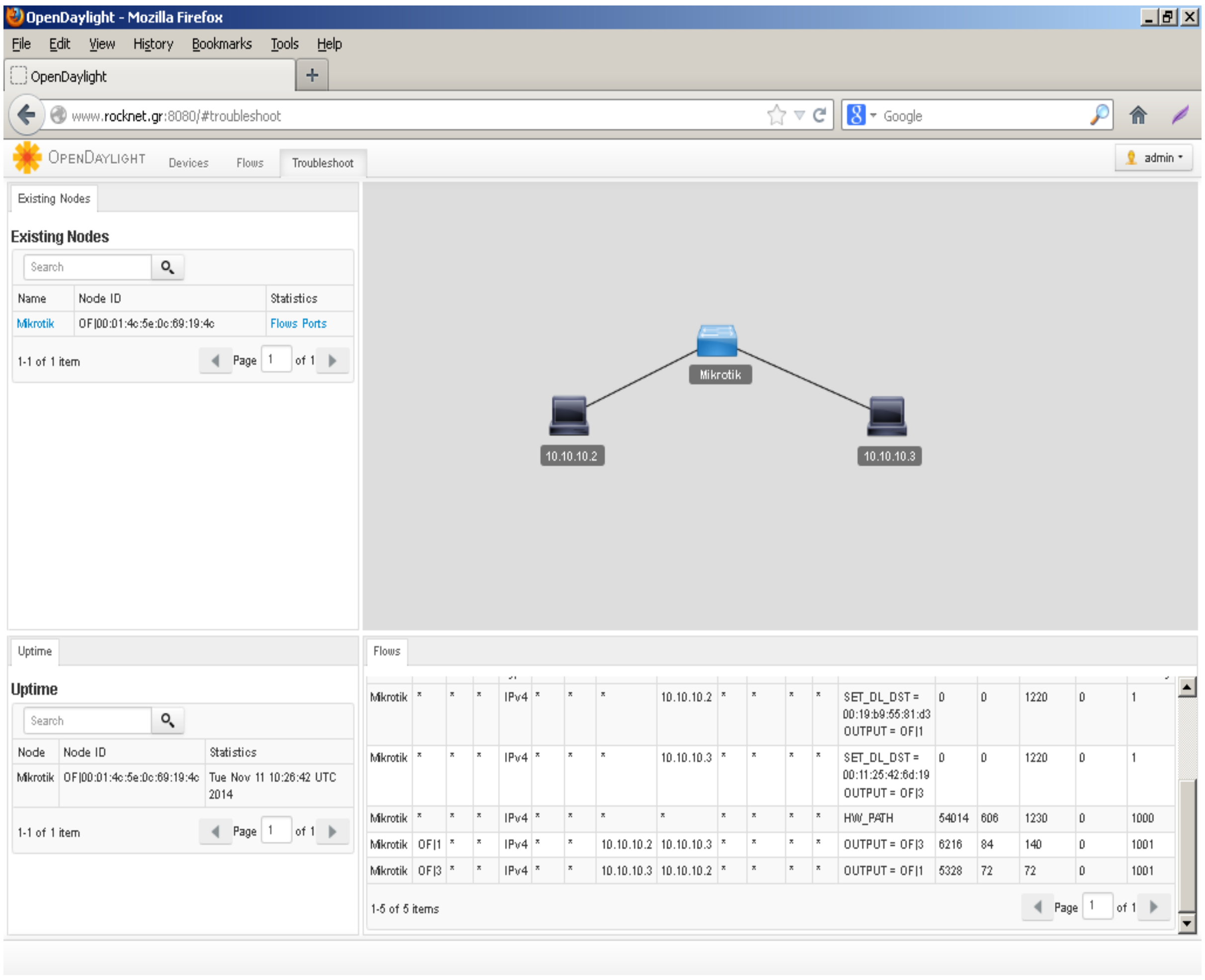
Input Port	Ethernet Type	VLAN ID	VLAN Priority	Source MAC	Dest MAC	Source IP	Dest IP	ToS	Source Port	Dest Port	Protocol	Cookie
3	0x800					10.10.10.3	10.10.10.2					

Actions
OUTPUT=ether4-slave-loc(1)

Picture 26: Flow2 details. Flows tab

Step 3

From the “troubleshoot” tab of the OpenDaylight controller we can verify the flows’ details and statistics like those of packet counters and timestamps.



Existing Nodes

Name	Node ID	Statistics
Mikrotik	OF|00:01:4c:5e:0c:69:19:4c	Flows Ports

1-1 of 1 item Page 1 of 1

Uptime

Node	Node ID	Statistics
Mikrotik	OF|00:01:4c:5e:0c:69:19:4c	Tue Nov 11 10:26:42 UTC 2014

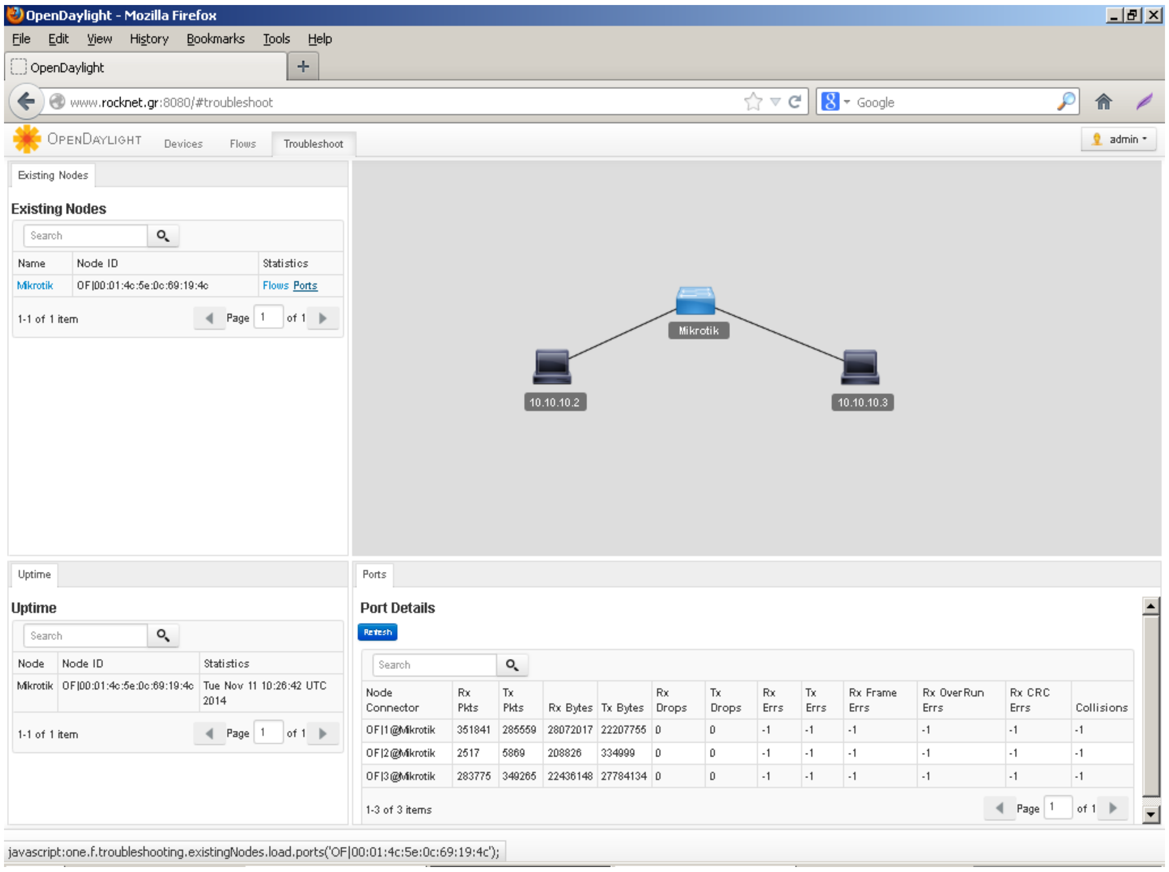
1-1 of 1 item Page 1 of 1

Mikrotik	*	*	*	IPv4	*	*	*	10.10.10.2	*	*	*	*	SET_DL_DST = 00:19:b9:55:81:d3 OUTPUT = OF|1	0	0	1220	0	1
Mikrotik	*	*	*	IPv4	*	*	*	10.10.10.3	*	*	*	*	SET_DL_DST = 00:11:25:42:6d:19 OUTPUT = OF|3	0	0	1220	0	1
Mikrotik	*	*	*	IPv4	*	*	*	*	*	*	*	*	HW_PATH	54014	606	1230	0	1000
Mikrotik	OF|1	*	*	IPv4	*	*	10.10.10.2	10.10.10.3	*	*	*	*	OUTPUT = OF|3	6216	84	140	0	1001
Mikrotik	OF|3	*	*	IPv4	*	*	10.10.10.3	10.10.10.2	*	*	*	*	OUTPUT = OF|1	5328	72	72	0	1001

1-5 of 5 items Page 1 of 1

Picture 27: Flow details and statistics. Troubleshoot tab

Additionally statistics per port


OpenDaylight - Mozilla Firefox
File Edit View History Bookmarks Tools Help
OpenDaylight
www.rocknet.gr:8080/#troubleshoot
Google
OPENDAYLIGHT
Devices
Flows
Troubleshoot
admin
Mikrotik
10.10.10.2
10.10.10.3
Refresh

Existing Nodes

Name	Node ID	Statistics
Mikrotik	OF|00:01:4c:5e:0c:69:19:4c	Flows Ports

1-1 of 1 item Page 1 of 1

Uptime

Node	Node ID	Statistics
Mikrotik	OF|00:01:4c:5e:0c:69:19:4c	Tue Nov 11 10:26:42 UTC 2014

1-1 of 1 item Page 1 of 1

Port Details

Node Connector	Rx Pkts	Tx Pkts	Rx Bytes	Tx Bytes	Rx Drops	Tx Drops	Rx Errs	Tx Errs	Rx Frame Errs	Rx OverRun Errs	Rx CRC Errs	Collisions
OF|1@Mikrotik	351841	285559	28072017	22207755	0	0	-1	-1	-1	-1	-1	-1
OF|2@Mikrotik	2517	5869	208826	334999	0	0	-1	-1	-1	-1	-1	-1
OF|3@Mikrotik	283775	349265	22436148	27784134	0	0	-1	-1	-1	-1	-1	-1

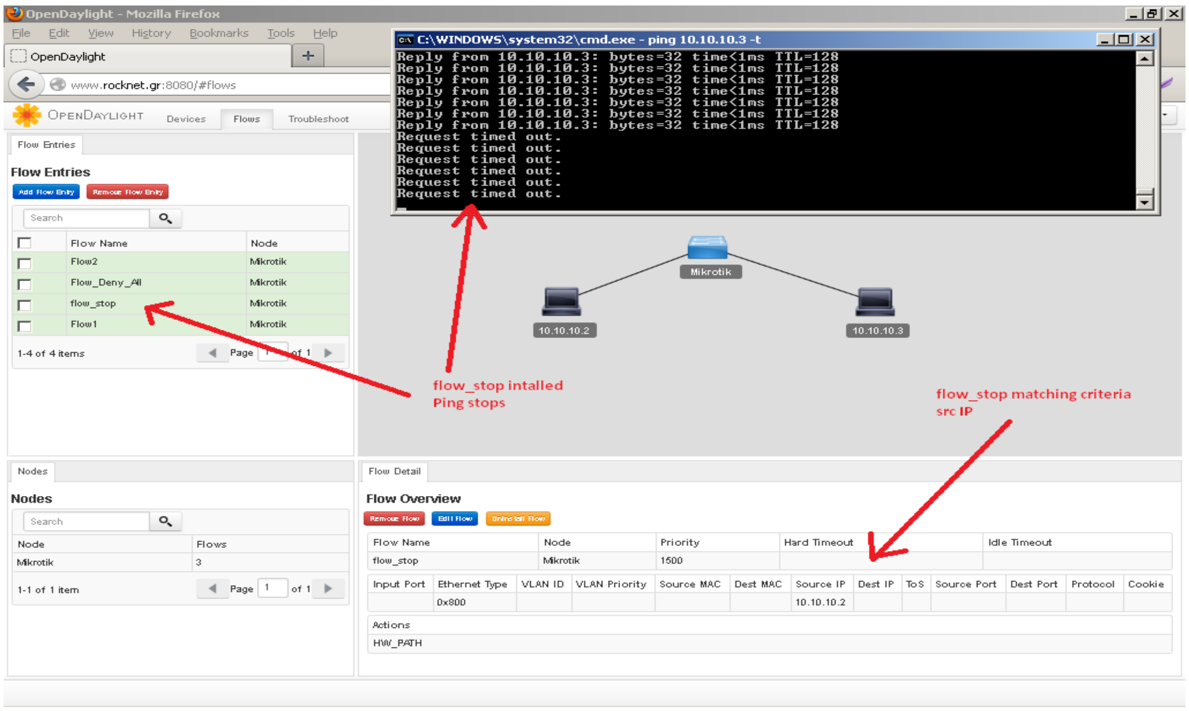
1-3 of 3 items Page 1 of 1

javascript:one.f.troubleshooting.existingNodes.load.ports('OF|00:01:4c:5e:0c:69:19:4c');

Picture 28: Port details and statistics. Troubleshoot tab

Step 4

A new flow denying all packets matching host's 1 source IP address is installed with higher priority and as a result the ping will no longer be successful.



```
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Reply from 10.10.10.3: bytes=32 time<1ms TTL=128
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

Flow Entries

	Flow Name	Node
☐	Flow2	Mikrotik
☐	Flow_Deny_All	Mikrotik
☐	flow_stop	Mikrotik
☐	Flow1	Mikrotik

1-4 of 4 items Page 1 of 1

Nodes

Node	Flows
Mikrotik	3

1-1 of 1 item Page 1 of 1

Flow Overview

Flow Name	Node	Priority	Hard Timeout	Idle Timeout
flow_stop	Mikrotik	1500		

Input Port	Ethernet Type	VLAN ID	VLAN Priority	Source MAC	Dest MAC	Source IP	Dest IP	ToS	Source Port	Dest Port	Protocol	Cookie
	0x800					10.10.10.2						

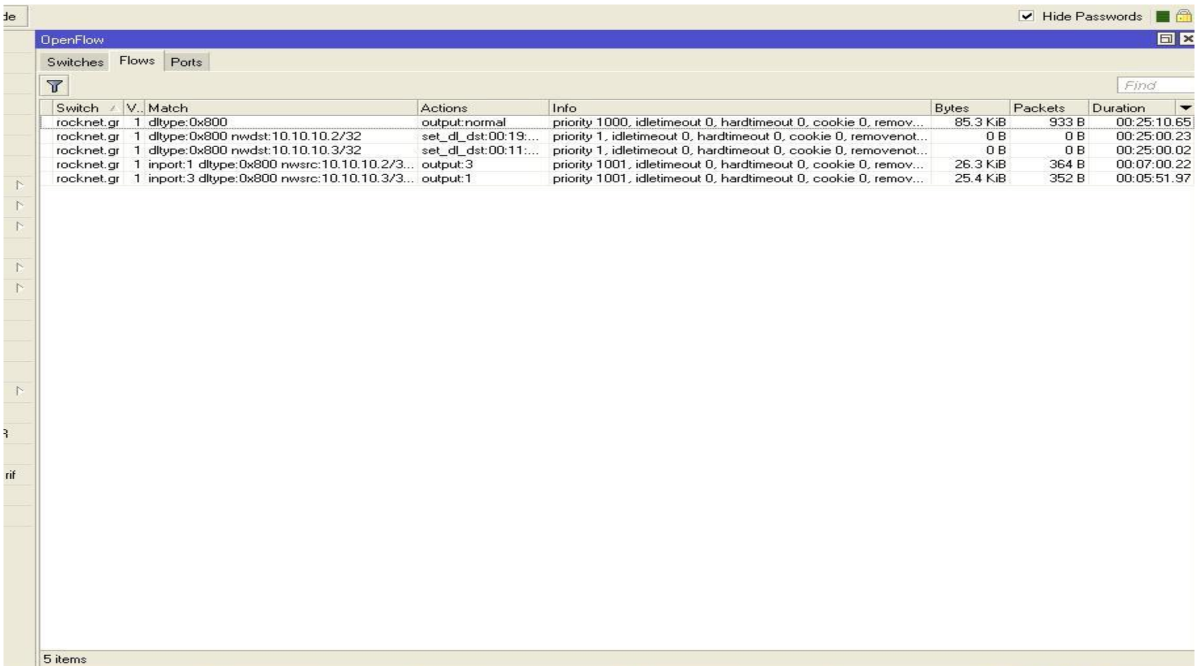
Actions
HW_PATH

Picture 29: Add new flow and stop communication

Step 5

We want to verify if all these flow details appear in the Openflow switch (mikrotik). In order to do so we connect to its web interface and observe the Openflow tab.

The five flows that appear on the controller are naturally installed in the Openflow switch.

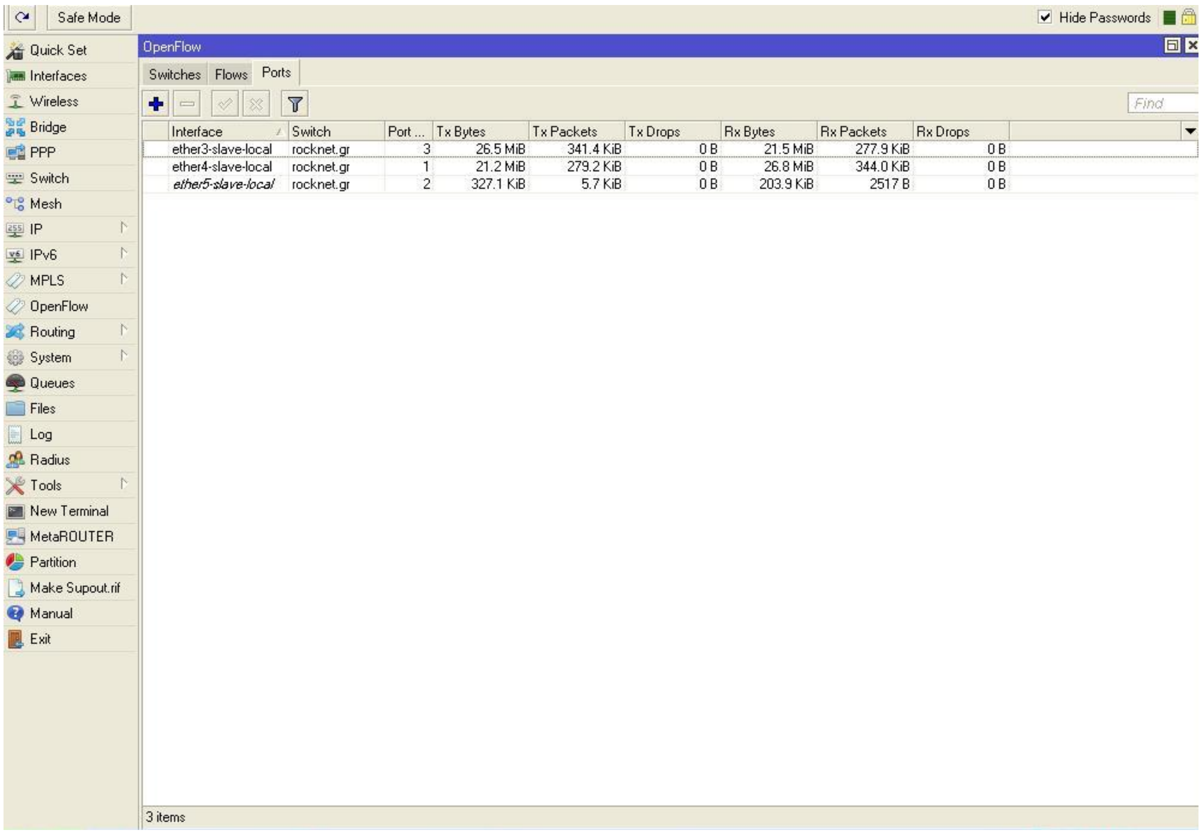


Switch	V..	Match	Actions	Info	Bytes	Packets	Duration
rocknet.gr	1	dltype:0x800	output:normal	priority 1000, idletimeout 0, hardtimeout 0, cookie 0, remov...	85.3 KiB	933 B	00:25:10.65
rocknet.gr	1	dltype:0x800 nwdst:10.10.10.2/32	set_dl_dst:00:19:...	priority 1, idletimeout 0, hardtimeout 0, cookie 0, removenot...	0 B	0 B	00:25:00.23
rocknet.gr	1	dltype:0x800 nwdst:10.10.10.3/32	set_dl_dst:00:11:...	priority 1, idletimeout 0, hardtimeout 0, cookie 0, removenot...	0 B	0 B	00:25:00.02
rocknet.gr	1	inport:1 dltype:0x800 nwsrc:10.10.10.2/3...	output:3	priority 1001, idletimeout 0, hardtimeout 0, cookie 0, remov...	26.3 KiB	364 B	00:07:00.22
rocknet.gr	1	inport:3 dltype:0x800 nwsrc:10.10.10.3/3...	output:1	priority 1001, idletimeout 0, hardtimeout 0, cookie 0, remov...	25.4 KiB	352 B	00:05:51.97

5 items

Picture 30: Mikrotik Openflow Flow table

Additionally statistics per port



Interface	Switch	Port ...	Tx Bytes	Tx Packets	Tx Drops	Rx Bytes	Rx Packets	Rx Drops
ether3-slave-local	rocknet.gr	3	26.5 MiB	341.4 KiB	0 B	21.5 MiB	277.9 KiB	0 B
ether4-slave-local	rocknet.gr	1	21.2 MiB	279.2 KiB	0 B	26.8 MiB	344.0 KiB	0 B
ether5-slave-local	rocknet.gr	2	327.1 KiB	5.7 KiB	0 B	203.9 KiB	2517 B	0 B

3 items

Picture 31: Mikrotik Openflow Port statistics

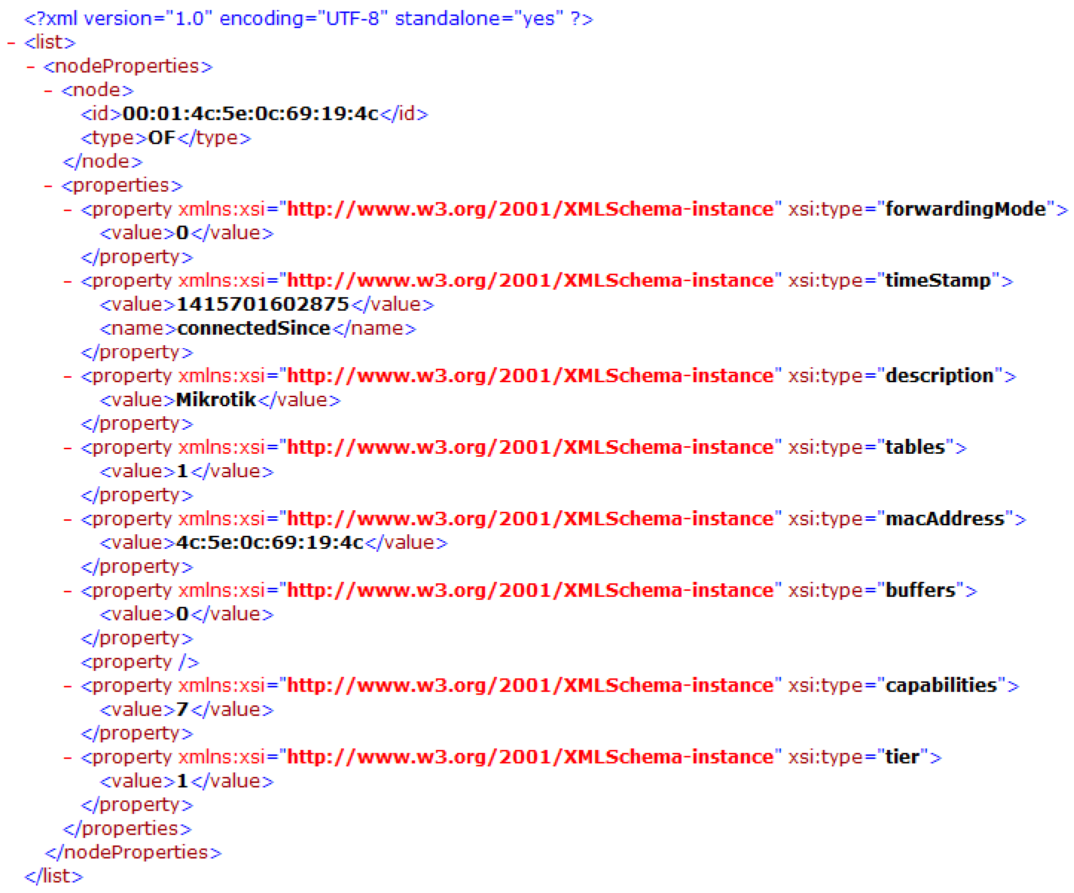
Step 6

We want to verify if these flow details can be exported from OpenDaylight’s northbound interface. Below are three xml files presenting:

- Node information
- Flow information
- Port information

These files are the output of proper http requests in the controller’s url.

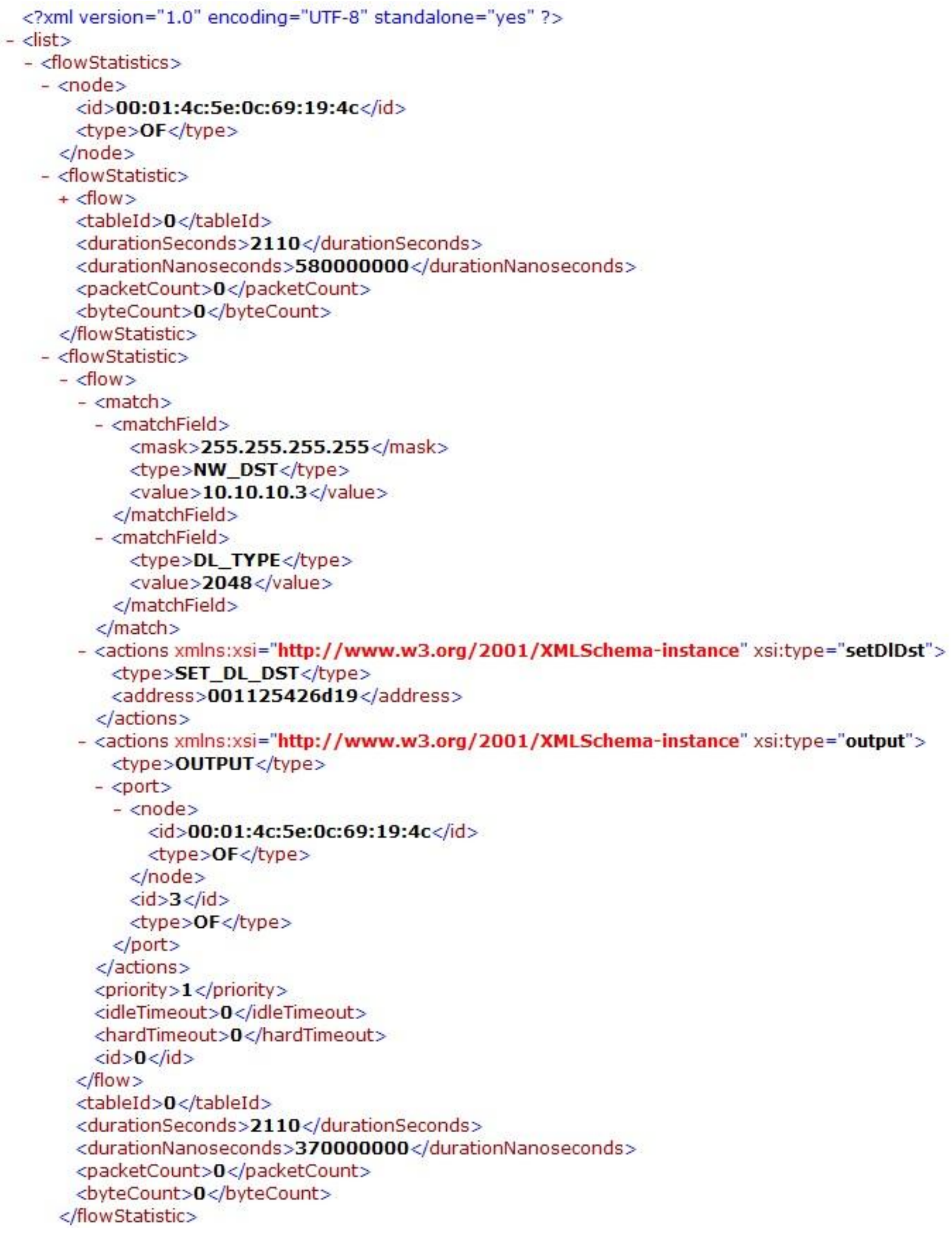
Node information



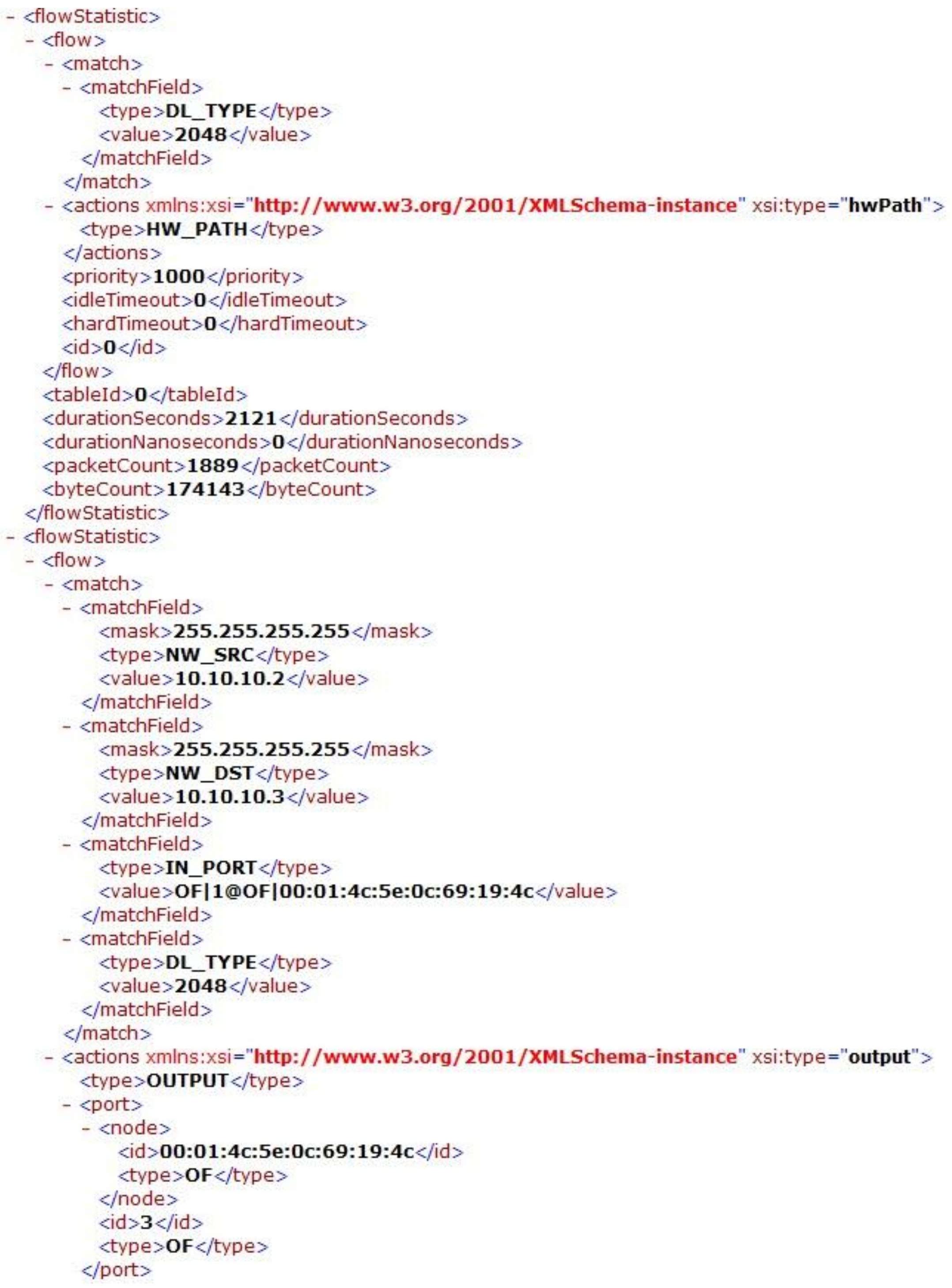
```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <list>
  - <nodeProperties>
    - <node>
        <id>00:01:4c:5e:0c:69:19:4c</id>
        <type>OF</type>
      </node>
    - <properties>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="forwardingMode">
          <value>0</value>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="timeStamp">
          <value>1415701602875</value>
          <name>connectedSince</name>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="description">
          <value>Mikrotik</value>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="tables">
          <value>1</value>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="macAddress">
          <value>4c:5e:0c:69:19:4c</value>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="buffers">
          <value>0</value>
        </property>
        <property />
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="capabilities">
          <value>7</value>
        </property>
      - <property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="tier">
          <value>1</value>
        </property>
      </properties>
    </nodeProperties>
  </list>
```

Picture 32: Node information XML. Northbound interface

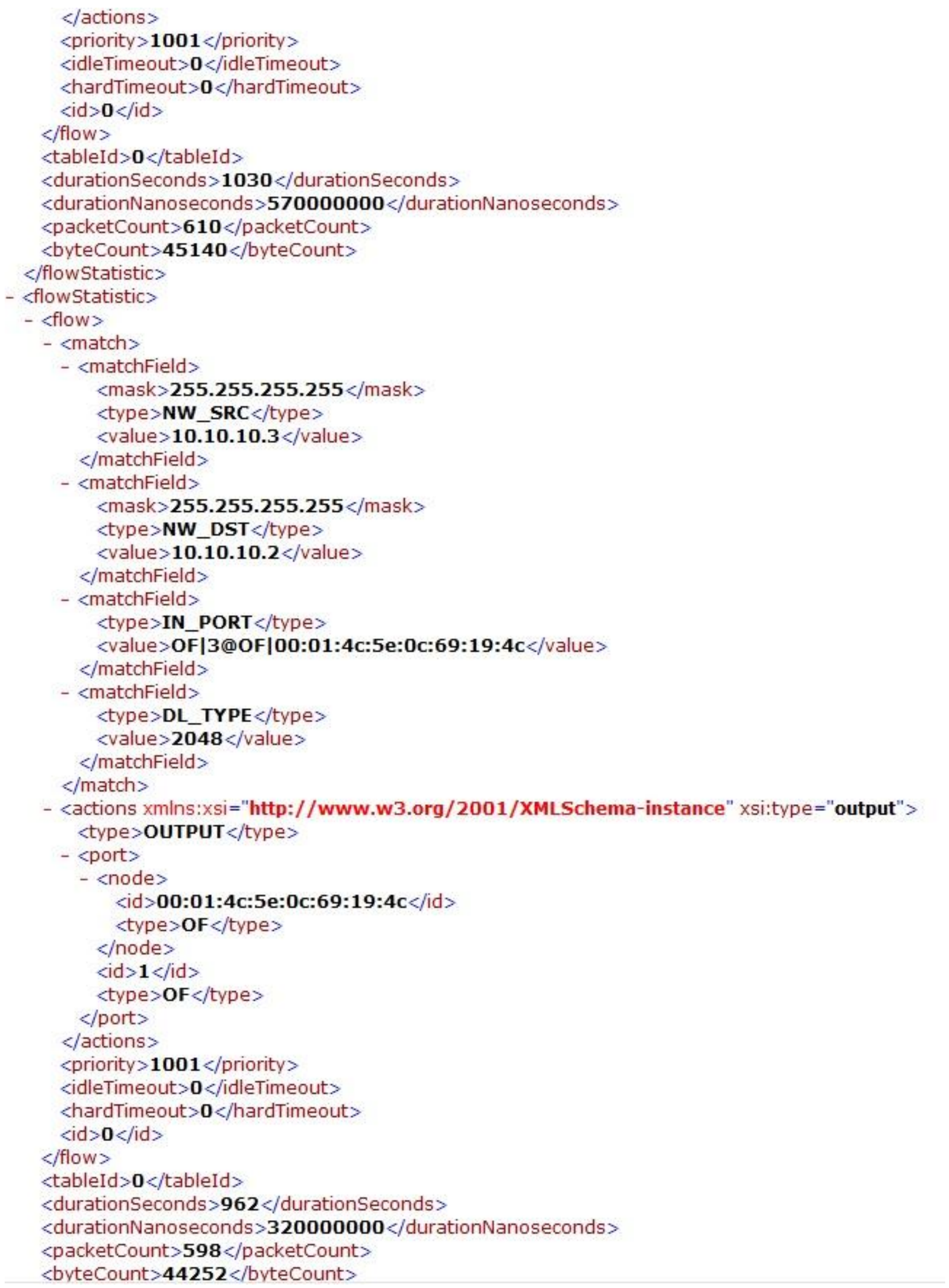
Flow details



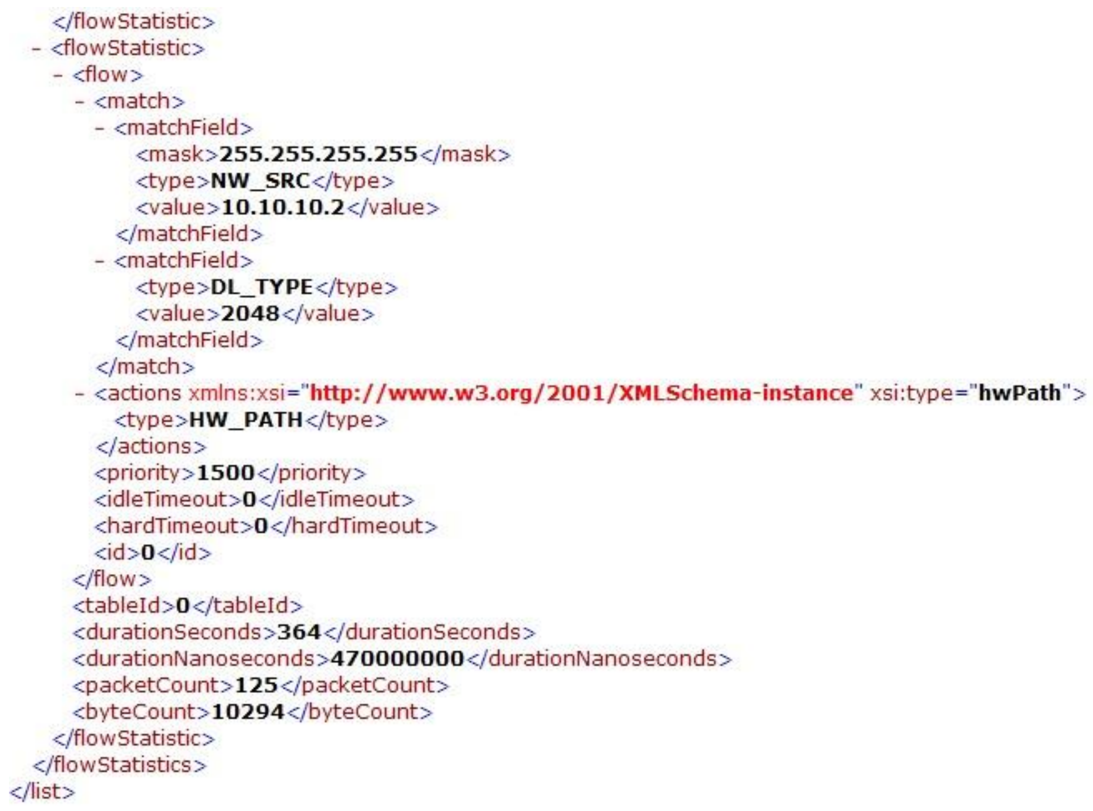
```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <list>
  - <flowStatistics>
    - <node>
        <id>00:01:4c:5e:0c:69:19:4c</id>
        <type>OF</type>
      </node>
    - <flowStatistic>
      + <flow>
        <tableId>0</tableId>
        <durationSeconds>2110</durationSeconds>
        <durationNanoseconds>580000000</durationNanoseconds>
        <packetCount>0</packetCount>
        <byteCount>0</byteCount>
      </flowStatistic>
    - <flowStatistic>
      - <flow>
        - <match>
          - <matchField>
              <mask>255.255.255.255</mask>
              <type>NW_DST</type>
              <value>10.10.10.3</value>
            </matchField>
          - <matchField>
              <type>DL_TYPE</type>
              <value>2048</value>
            </matchField>
          </match>
        - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="setDlDst">
            <type>SET_DL_DST</type>
            <address>001125426d19</address>
          </actions>
        - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="output">
            <type>OUTPUT</type>
          - <port>
            - <node>
                <id>00:01:4c:5e:0c:69:19:4c</id>
                <type>OF</type>
              </node>
              <id>3</id>
              <type>OF</type>
            </port>
          </actions>
          <priority>1</priority>
          <idleTimeout>0</idleTimeout>
          <hardTimeout>0</hardTimeout>
          <id>0</id>
        </flow>
        <tableId>0</tableId>
        <durationSeconds>2110</durationSeconds>
        <durationNanoseconds>370000000</durationNanoseconds>
        <packetCount>0</packetCount>
        <byteCount>0</byteCount>
      </flowStatistic>
```



```
- <flowStatistic>
  - <flow>
    - <match>
      - <matchField>
          <type>DL_TYPE</type>
          <value>2048</value>
        </matchField>
      </match>
    - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="hwPath">
        <type>HW_PATH</type>
      </actions>
      <priority>1000</priority>
      <idleTimeout>0</idleTimeout>
      <hardTimeout>0</hardTimeout>
      <id>0</id>
    </flow>
    <tableId>0</tableId>
    <durationSeconds>2121</durationSeconds>
    <durationNanoseconds>0</durationNanoseconds>
    <packetCount>1889</packetCount>
    <byteCount>174143</byteCount>
  </flowStatistic>
- <flowStatistic>
  - <flow>
    - <match>
      - <matchField>
          <mask>255.255.255.255</mask>
          <type>NW_SRC</type>
          <value>10.10.10.2</value>
        </matchField>
      - <matchField>
          <mask>255.255.255.255</mask>
          <type>NW_DST</type>
          <value>10.10.10.3</value>
        </matchField>
      - <matchField>
          <type>IN_PORT</type>
          <value>OF|1@OF|00:01:4c:5e:0c:69:19:4c</value>
        </matchField>
      - <matchField>
          <type>DL_TYPE</type>
          <value>2048</value>
        </matchField>
      </match>
    - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="output">
        <type>OUTPUT</type>
      - <port>
        - <node>
            <id>00:01:4c:5e:0c:69:19:4c</id>
            <type>OF</type>
          </node>
          <id>3</id>
          <type>OF</type>
        </port>
```



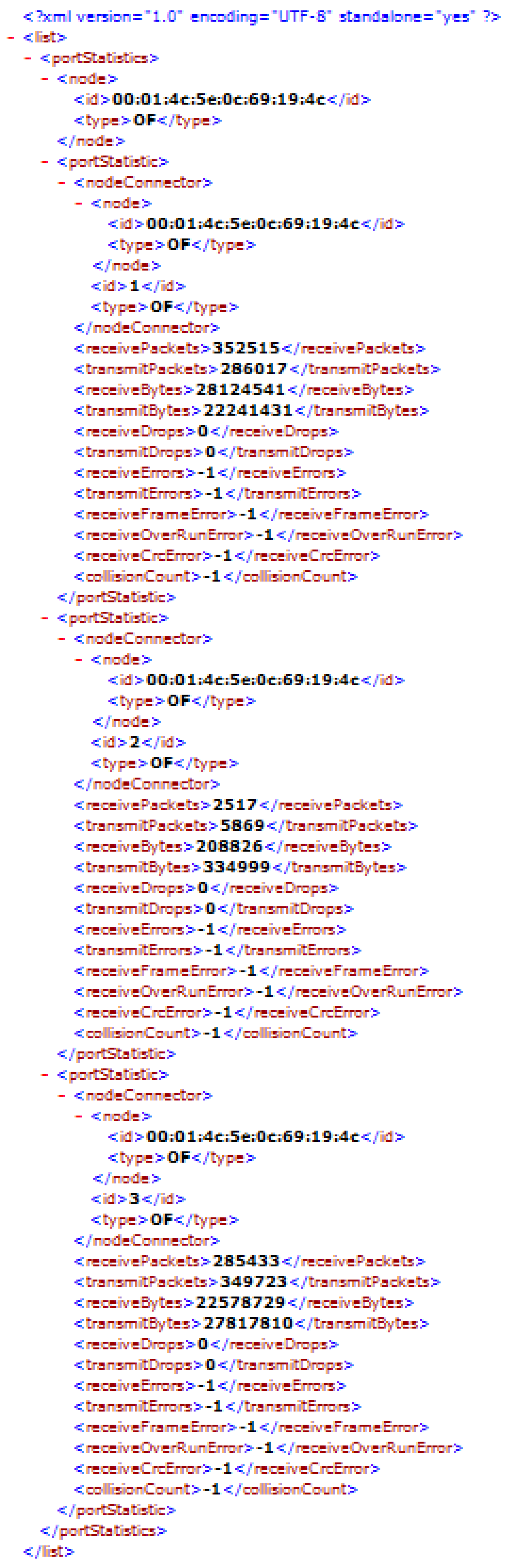
```
          </actions>
          <priority>1001</priority>
          <idleTimeout>0</idleTimeout>
          <hardTimeout>0</hardTimeout>
          <id>0</id>
        </flow>
        <tableId>0</tableId>
        <durationSeconds>1030</durationSeconds>
        <durationNanoseconds>570000000</durationNanoseconds>
        <packetCount>610</packetCount>
        <byteCount>45140</byteCount>
      </flowStatistic>
    - <flowStatistic>
      - <flow>
        - <match>
          - <matchField>
              <mask>255.255.255.255</mask>
              <type>NW_SRC</type>
              <value>10.10.10.3</value>
            </matchField>
          - <matchField>
              <mask>255.255.255.255</mask>
              <type>NW_DST</type>
              <value>10.10.10.2</value>
            </matchField>
          - <matchField>
              <type>IN_PORT</type>
              <value>OF|3@OF|00:01:4c:5e:0c:69:19:4c</value>
            </matchField>
          - <matchField>
              <type>DL_TYPE</type>
              <value>2048</value>
            </matchField>
          </match>
        - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="output">
            <type>OUTPUT</type>
          - <port>
            - <node>
                <id>00:01:4c:5e:0c:69:19:4c</id>
                <type>OF</type>
              </node>
              <id>1</id>
              <type>OF</type>
            </port>
          </actions>
          <priority>1001</priority>
          <idleTimeout>0</idleTimeout>
          <hardTimeout>0</hardTimeout>
          <id>0</id>
        </flow>
        <tableId>0</tableId>
        <durationSeconds>962</durationSeconds>
        <durationNanoseconds>320000000</durationNanoseconds>
        <packetCount>598</packetCount>
        <byteCount>44252</byteCount>
```



```
    </flowStatistic>
  - <flowStatistic>
    - <flow>
      - <match>
        - <matchField>
            <mask>255.255.255.255</mask>
            <type>NW_SRC</type>
            <value>10.10.10.2</value>
          </matchField>
        - <matchField>
            <type>DL_TYPE</type>
            <value>2048</value>
          </matchField>
        </match>
      - <actions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="hwPath">
          <type>HW_PATH</type>
        </actions>
        <priority>1500</priority>
        <idleTimeout>0</idleTimeout>
        <hardTimeout>0</hardTimeout>
        <id>0</id>
      </flow>
      <tableId>0</tableId>
      <durationSeconds>364</durationSeconds>
      <durationNanoseconds>470000000</durationNanoseconds>
      <packetCount>125</packetCount>
      <byteCount>10294</byteCount>
    </flowStatistic>
  </flowStatistics>
</list>
```

Picture 33: Flow details XML. Northbound interface

Port Information

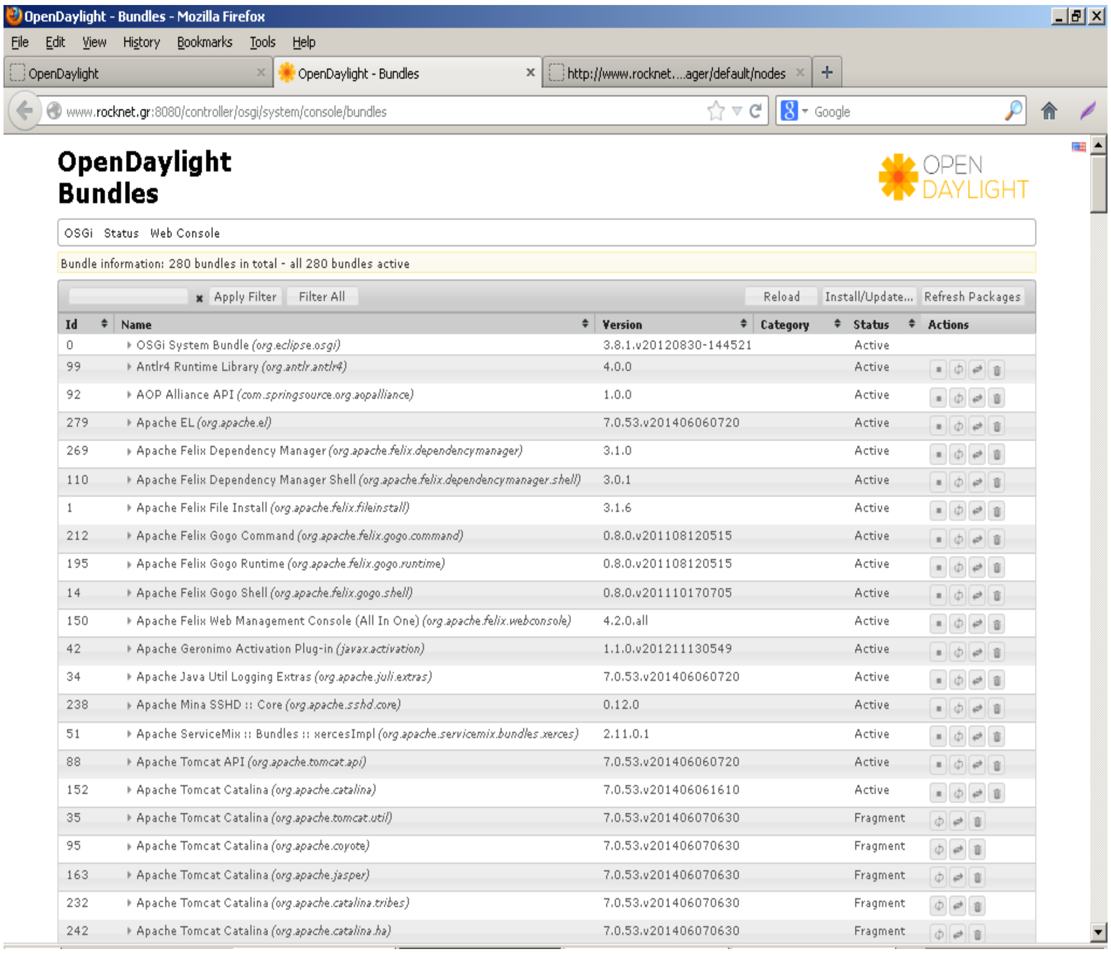


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <list>
  - <portStatistics>
    - <node>
        <id>00:01:4c:5e:0c:69:19:4c</id>
        <type>OF</type>
      </node>
    - <portStatistic>
      - <nodeConnector>
        - <node>
            <id>00:01:4c:5e:0c:69:19:4c</id>
            <type>OF</type>
          </node>
          <id>1</id>
          <type>OF</type>
        </nodeConnector>
        <receivePackets>352515</receivePackets>
        <transmitPackets>286017</transmitPackets>
        <receiveBytes>28124541</receiveBytes>
        <transmitBytes>22241431</transmitBytes>
        <receiveDrops>0</receiveDrops>
        <transmitDrops>0</transmitDrops>
        <receiveErrors>-1</receiveErrors>
        <transmitErrors>-1</transmitErrors>
        <receiveFrameError>-1</receiveFrameError>
        <receiveOverRunError>-1</receiveOverRunError>
        <receiveCrcError>-1</receiveCrcError>
        <collisionCount>-1</collisionCount>
      </portStatistic>
    - <portStatistic>
      - <nodeConnector>
        - <node>
            <id>00:01:4c:5e:0c:69:19:4c</id>
            <type>OF</type>
          </node>
          <id>2</id>
          <type>OF</type>
        </nodeConnector>
        <receivePackets>2517</receivePackets>
        <transmitPackets>5869</transmitPackets>
        <receiveBytes>208826</receiveBytes>
        <transmitBytes>334999</transmitBytes>
        <receiveDrops>0</receiveDrops>
        <transmitDrops>0</transmitDrops>
        <receiveErrors>-1</receiveErrors>
        <transmitErrors>-1</transmitErrors>
        <receiveFrameError>-1</receiveFrameError>
        <receiveOverRunError>-1</receiveOverRunError>
        <receiveCrcError>-1</receiveCrcError>
        <collisionCount>-1</collisionCount>
      </portStatistic>
    - <portStatistic>
      - <nodeConnector>
        - <node>
            <id>00:01:4c:5e:0c:69:19:4c</id>
            <type>OF</type>
          </node>
          <id>3</id>
          <type>OF</type>
        </nodeConnector>
        <receivePackets>285433</receivePackets>
        <transmitPackets>349723</transmitPackets>
        <receiveBytes>22578729</receiveBytes>
        <transmitBytes>27817810</transmitBytes>
        <receiveDrops>0</receiveDrops>
        <transmitDrops>0</transmitDrops>
        <receiveErrors>-1</receiveErrors>
        <transmitErrors>-1</transmitErrors>
        <receiveFrameError>-1</receiveFrameError>
        <receiveOverRunError>-1</receiveOverRunError>
        <receiveCrcError>-1</receiveCrcError>
        <collisionCount>-1</collisionCount>
      </portStatistic>
    </portStatistics>
  </list>
```

Picture 34: Port information XML. Northbound interface

We conclude that OpenDaylight's web interface, Mikrotik switch's web interface and OpenDaylight's northbound interface give identical details for all flows installed.

OSGi web interface appear and its capabilities appear below


OpenDaylight - Bundles - Mozilla Firefox
File Edit View History Bookmarks Tools Help
OpenDaylight
OpenDaylight - Bundles
http://www.rocknet....ager/default/nodes
www.rocknet.gr:8080/controller/osgi/system/console/bundles
Google
OPEN DAYLIGHT
OSGi Status Web Console
Apply Filter
Filter All
Reload
Install/Update...
Refresh Packages

OpenDaylight Bundles

Bundle information: 280 bundles in total - all 280 bundles active

Id	Name	Version	Category	Status	Actions
0	OSGi System Bundle *(org.eclipse.osgi)*	3.8.1.v20120830-144521		Active	
99	Antlr4 Runtime Library *(org.antlr.antlr4)*	4.0.0		Active	
92	AOP Alliance API *(com.springsource.org.aopalliance)*	1.0.0		Active	
279	Apache EL *(org.apache.el)*	7.0.53.v201406060720		Active	
269	Apache Felix Dependency Manager *(org.apache.felix.dependencymanager)*	3.1.0		Active	
110	Apache Felix Dependency Manager Shell *(org.apache.felix.dependencymanager.shell)*	3.0.1		Active	
1	Apache Felix File Install *(org.apache.felix.fileinstall)*	3.1.6		Active	
212	Apache Felix Gogo Command *(org.apache.felix.gogo.command)*	0.8.0.v201108120515		Active	
195	Apache Felix Gogo Runtime *(org.apache.felix.gogo.runtime)*	0.8.0.v201108120515		Active	
14	Apache Felix Gogo Shell *(org.apache.felix.gogo.shell)*	0.8.0.v201110170705		Active	
150	Apache Felix Web Management Console (All In One) *(org.apache.felix.webconsole)*	4.2.0.all		Active	
42	Apache Geronimo Activation Plug-in *(javax.activation)*	1.1.0.v201211130549		Active	
34	Apache Java Util Logging Extras *(org.apache.juli.extras)*	7.0.53.v201406060720		Active	
238	Apache Mina SSHD :: Core *(org.apache.sshd.core)*	0.12.0		Active	
51	Apache ServiceMix :: Bundles :: xercesImpl *(org.apache.servicemix.bundles.xerces)*	2.11.0.1		Active	
88	Apache Tomcat API *(org.apache.tomcat.api)*	7.0.53.v201406060720		Active	
152	Apache Tomcat Catalina *(org.apache.catalina)*	7.0.53.v201406061610		Active	
35	Apache Tomcat Catalina *(org.apache.tomcat.util)*	7.0.53.v201406070630		Fragment	
95	Apache Tomcat Catalina *(org.apache.coyote)*	7.0.53.v201406070630		Fragment	
163	Apache Tomcat Catalina *(org.apache.jasper)*	7.0.53.v201406070630		Fragment	
232	Apache Tomcat Catalina *(org.apache.catalina.tribes)*	7.0.53.v201406070630		Fragment	
242	Apache Tomcat Catalina *(org.apache.catalina.ha)*	7.0.53.v201406070630		Fragment	

Picture 35: OpenDaylight OSGi

8. USE CASE FINANCIAL PROPOSAL

From a financial point of view designing a network based on SDN switches is approximately 30% more economic than designing one with standard networking equipment. The cost for the controller in SDN architecture is negligible in comparison with the overall budget required for the switches and routers. As mentioned in the beginning of this research SDN gains ground in regards to operational cost, provisioning complexity and network real-time monitoring. All implementation and migration processes as well as troubleshooting is completed faster and operations engineers do not need to have especially high expertise. Finally a fully SDN net will require lower CAPEX, lower OPEX and is capable of offering much more services (especially VAS: value added services).

Below lies a brief financial analysis of how SDN introduces new services from which both ISPs and enterprises (corporate customers) will benefit. OpenDaylight controller and Openflow devices will be the components of our SDN network architecture. Smart and friendly apps can be built using the northbound interface while the Openflow enabled devices communicate to the southbound. A theoretical example based on many assumptions comparing SDN network architecture vs. standard network architecture will be presented:

An ISP is offering point-to-point circuits to corporate customer between Athens and Salonika. Adsl users (residential) are served from Salonika to Athens. Datacenters and international peerings are located in Athens.

Assumptions:

- No operational and engineering costs are measured as considered the same on both architectures.
- The ISP will either offer standard network services or SDN network services.
- In a ten year plan only one corporate customer will be provisioned in the ISP and there will be no need for core bandwidth upgrade (due to traffic as Adsl customers will remain constant).
- Power consumption, licensing, life cycle etc. will not be considered.
- The prices of the services will not change in a ten-year plan.

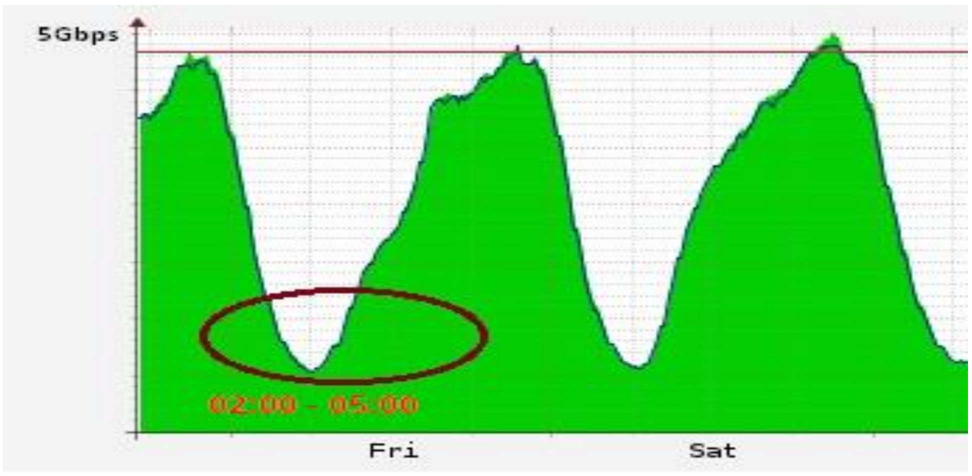
(Further assumptions of minor significance are made in order to consider the whole environment stable)

Dynamic Bandwidth Management Use Cases

There is an unfulfilled demand to provide enterprises with occasional high-bandwidth services such as Hybrid Cloud backup and storage, data center to data center replication, disaster recovery or workload migration. The demand is unfulfilled because service providers’ networks are often provisioned to peak traffic loads because of an inability to time shift or space shift traffic. For most service providers, the average network utilization rate leaves approximately 50 percent of capacity unused. This unused capacity is money left on the table because of the inability to offer unused capacity dynamically to currently unreached market segments as well as the existing customer base. Unused capacity can be offered using scavenger or reserve classes of service. These classes of service provide options such as where to present demand: load placement; when to present the demand: calendaring; and how to route the demand: policy. In addition, service providers’ business models require that bandwidth is dedicated to the enterprise for a lengthy period (typically three years) to cover the cost of deploying the bandwidth. It is difficult, therefore, for enterprises to make a long-term expense commitment for occasional high-bandwidth usage, resulting in enterprises suffering network service degradation during these occasional peak service usage periods [9].

In our example an ISP is offering circuits between Athens and Salonika. The core interconnections between the two cities are 5x1gigabit. 90% of this BW is consumed from ADSL users and the traffic rate through the day goes as follows:

We observe that between 02:00 – 06:00 75% of the BW is unused.



Picture 36: Bandwidth Usage

A new company offering cloud services needs an 1 Gigabit circuit between Athens and Salonika in order to replicate data between its two sites every Sunday 03:00-05:00. With standard networking architecture we can only provide them with 1 Gigabit circuit with annual fee. The company will pay 1 Gigabit circuit just for using it once a week for 2 hours. The ISP based on the service layer agreement will have to upgrade its core infrastructure to support the new bandwidth requirements. The 5 Gigabit circuit Athens-Salonika will need to be upgraded to 6 Gigabit. The value of the investment will be too high.

Briefly some financial results based on a 10 year plan:

The annual fee for 1 Gigabit circuit is 60.000 euro.

The cost for the ISP to upgrade its core to 6 Gbps is 500.000 euro.

For a ten year plan the company will spend 600.000 euros and the profits of the ISP will accede 38955 euros based on NPV with a discount rate 2%. The investment will begin to be profitable the 10th year.

Additionally this is too risky for the ISP in case the company decides to pull out before the 10th year.

Table 5: NPV1 upgrading standard network architecture

i=0,02 (Discount rate) , t=10years (time of the cash flow)											
Year	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
(1+i)^t	1,00	1,02	1,04	1,06	1,08	1,10	1,13	1,15	1,17	1,20	1,22
Cash Flow	-500000,00	58823,53	57670,13	56539,34	55430,73	54343,85	53278,28	52233,61	51209,42	50205,32	49220,90
NPV(0,10)	**38955,1004**										
Revenues		-441176,5	-383506,3	-326967	-271536	-217192	-163914	-111681	-60471,11357	-10265,8	38955,1

What if we can offer a service both the ISP and the company will benefit from?

With SDN enabled network the ISP can easily offer flexible services. A “BW on demand” service will fit the company’s requirements.

As stated already the network is 75% unused between 02:00 – 07:00. The company will acquire a “BW on demand” service of 1Gbps once a week for two hours (the time needed for the data replication). This service will cost 450 euro/week (per data replication).

It is not necessary for the ISP to upgrade its core connections since no service license agreement is now needed (annual fee 1 Gbps circuit). The service license agreement is 1 Gbps every Sunday 03:00-05:00.

Briefly some financial results based on a 10 year plan:

Every year has 52 weeks so the company will pay 450 euro*52 = 23400 euro per year.

Table 6: Enterprise’s additive cost per year

Year	Price
1	23400
2	46800
3	70200
4	93600
5	117000
6	140400
7	163800
8	187200
9	210600
10	234000

By acquiring a “BW on demand” service the company will spend 234.000 euros when standard network architecture costs 600.000 euros. Therefore the company will save 366.000 euros.

The ISP investment is 50.000 euros (including upgrading the networking devices to support Openflow, train the operations engineers and develop the “BW on Demand” service).

The ISP does need to upgrade its core as it offers “BW on Demand” with SDN.

As a result, profit (for the ISP) starts from the third year and in the end of the tenth year will be 160.192 euro. Let us note here that as no big investment has been made the ISP risks are minor if the corporate customer decides to leave the service.

Table 7: NPV2 SDN network architecture

i=0,02 (Discount rate) , t=10years (time of the cash flow)											
Year	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
(1+i)^t	1,00	1,02	1,04	1,06	1,08	1,10	1,13	1,15	1,17	1,20	1,22
Cash Flow	-50000,00	22941,18	22491,35	22050,34	21617,98	21194,10	20778,53	20371,11	19971,67	19580,07	19196,15
NPV(0,10)	**160192,4891**										
Revenues		-27058,8	-4567,47	17482,86858	39100,85155	60294,95	81073,48	101444,6	121416,3	140996,3	160192,5

Table 8: Revenue comparison. Standard vs SDN architecture

Year	Revenues	
	Standard Networking	SDN Networking
1	-441176,4706	-27058,82353
2	-383506,3437	-4567,474048
3	-326967,0036	17482,86858
4	-271536,2781	39100,85155
5	-217192,4295	60294,9525
6	-163914,1466	81073,48284
7	-111680,5358	101444,591
8	-60471,11357	121416,2657
9	-10265,79762	140996,3389
10	38955,10037	160192,4891

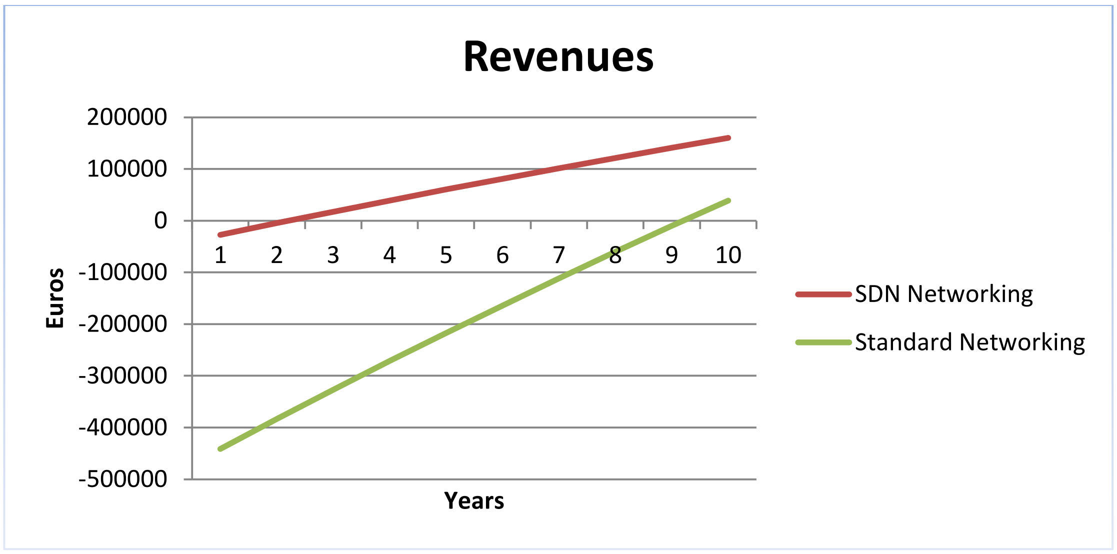


Diagram 1: SDN and standard networking architecture revenue

9. CONCLUSIONS

Trends such as user mobility, server virtualization, IT-as-a-Service, and the need rapidly to respond to changing business conditions place significant demands on the network demands that today's conventional network architectures can't handle. Future networks will become increasingly more heterogeneous, interconnecting users and applications over networks ranging from wired, infrastructure-based wireless (e.g., cellular–based networks, wireless mesh networks), to infrastructure-less wireless networks (e.g. mobile ad-hoc networks, vehicular networks). In the meantime, mobile traffic has been increasing exponentially over the past several years, and is expected to increase 18–fold by 2016, with more mobile-connected devices than the world's population, which is already a reality [10]. Software-Defined Networking provides a new, dynamic network architecture that transforms traditional network backbones into rich service-delivery platforms.

By decoupling the control–and data planes, programmable networks enable customized control, an opportunity to eliminate middleboxes, as well as simplified development and deployment of new network services and protocols.[11] An SDN approach fosters network virtualization, enabling IT staff to manage their servers, applications, storage, and networks with a common approach and tool set. Whether in a carrier environment or enterprise data center and campus, SDN adoption can improve network manageability, scalability, and agility.
The future of networking will rely more and more on software, which will accelerate the pace of innovation for networks as it has in the computing and storage domains. SDN promises to transform today's static networks into flexible, programmable platforms with the intelligence to allocate resources dynamically, the scale to support enormous data centers and the virtualization needed to support dynamic, highly automated, and secure cloud environments. With its many advantages and astonishing industry momentum, SDN is on the way to becoming the new norm for networks.[12] The need of a complete control platform is mandatory, this platform that discriminate among others is OpenDaylight. SDN presents both significant challenges and unlimited opportunities for the future of transferring information and communicating across the globe. The companies coming together understand that the best way to address this historical moment in their industry is to do it together. Collaborative open development and open source software are the driving force behind modern architectures and well recognized for accelerating technology innovation and adoption.

OpenDaylight will provide a common platform on top of which vendor products and services can be built, giving vendors the room to innovate and compete and provide users with the best solutions at a rapid pace. SDN is expected to reduce the network OpEx by simplifying operations, optimizing resource usage through centralized data/algorithms, and simplifying network software upgrades. It also significantly cuts down a network operator's CapEx, since a commercial-off-the-shelf (COTS) server with a high-end CPU is much cheaper than a high-end router [13]. As the SDN paradigm gains momentum, the migration from existing IP routers to SDN-compliant equipment, e.g., Openflow switches, is becoming eminent. In data centers, SDN can be already fully integrated into the network architecture, by migrating the switching and routing infrastructure entirely to SDN. For a medium to large-scale ISP, on the other hand, a viable approach is to gradually migrate to SDN, for instance, over a multi-period cycle spanning couple of years [14].

SDN is a new concept that has the opportunity to revolutionize networking and telecom industry. Although similar concepts have appeared in the pass with questionable success, the current market status, the evolution of both telecoms and IT technologies and the increasing requirement for network flexibility are expected to make SDN a de facto standard in telecoms weather these are in developing or developed markets.[15]

ABBREVIATIONS – ACRONYMS

ACL	Access Control List
AD-SAL	API Driven Service Abstraction Layer
ADSL	Asymmetric Digital Subsrcibers Line
API	Application Programming Interface
ARP	Address Resolution Protocol
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol Linkstate Distribution
BW	Bandwidth
CAPEX	Capital Expenditure
CIO	Chief Information Officer
CLI	Command Line Interface
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
DNS	Domain Name System
DTO	Data Transfer Object
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISP	Internet Service Provider
IT	Information Technology
JVM	Java Virtual Machine
LLDP	Link Layer Discovery Protocol
MAC	media access control address
MD-SAL	Model Driver Service Abstraction Layer
MPLS	Multi Protocol Label Switching
NAT	Netwrok Address Translation

NB	Northbound
NFV	Network Functions Virtualization
NPV	Net Present Value
OF	Openflow
ONF	Open Networking Fundation
OPEX	Operational expenditure
OS	Operating System
OSGi	Open Service Gateway initiative
OVSDB	Open vSwitch Database Management Protocol
PCEP	Path Computation Element Protocol
QoS	Quality of Service
REST	Representational state transfer
RPC	Remote Procedure Call
SAL	Service Abstraction Layer
SB	Southbound
SDN	Software Defined Networking
SNAP	Subnetwork Access Protocol
SNMP	Simple Network Management Protocol
STP	Spanning Tree Protocol
SW	Software
TCAM	Ternary content-addressable memory
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPS	Virtual Private Server

XML	Extensible Markup Language

REFERENCES

[1] Allied Telesis, *Demystifying Software-Defined Networking (SDN*), Available: http://www.alliedtelesis.com/userfiles/file/WP_Demystifying_SDN_RevA.pdf.2014, Last accessed 10th Dec 2014

[2] Open Networking Foundation, *Software –Defined Networking: The New Norm for Networks, April*,2012

[3] Thomas A. Limoncelli. *Openflow: a radical new idea in networking Community*, August 2012.

[4] Techtarget,*Openflow-protocol-tutorial-SDN-controllers-and-applications-emerge Available:* http://searchsdn.techtarget.com/guides/Openflow-protocol-tutorial-SDN-controllers-and-applications-emerge ,2014.Last accessed 12th Oct 2014

[5] Techtarget, *Openflow protocol primer: Looking under the hood,* Available: http://searchsdn.techtarget.com/feature/Openflow-protocol-primer-Looking-under-the-hood ,2014.Last accessed 10th Oct 2014

[6] Open Networking Foundation, *Openflow Switch Specification, Version 1.0.0*,December 2009

[7] Guillermo Romero de Tejada Muntaner, *Evaluation of Openflow Controllers*, October 2012

[8] The Linux Foundation, OpenDaylight Wiki Page , Available: http://wiki.OpenDaylight.org/view/Main_Page .Last accessed 12th Oct 2014

[9] ACG Research, *Business Case for Cisco SDN for the WAN*,2014

[10] Thomas D.Nadeue, Ken Gray, *An Authoritative Review of Network Programmability Technologies. Software Define Networking,* August 2013

[11] Bruno Nunes Astuto, Marc Mendon_ca, Xuan Nam Nguyen, Katia Obraczka, Thierry Turletti, *A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks*, January 2014

[12] Cisco visual networking index: *Global mobile data traffic forecast update, 2011–2016. Technical report, Cisco*, February 2012

[13] Metaswitch Networks. *PCE - An Evolutionary Approach to SDN.* Available:http://www.metaswitch.com/sites/default/files/metaswitch-white-paper-pce-an-evolutionary-approach-to-sdn.pdf ,2012.Last accessed 14th Oct 2014.

[14] Tamal Das, Marcel Caria and Admela Jukan, Marco Hoffmann *A Techno-economic Analysis of Network Migration to Software-Defined Networking*, October 2013

[15] Informa telecoms & media, Juniper networks, *Mobile SDN: The future is virtual*. July 2013