ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

SDN Southbound Interfaces

Συγγραφέας:

Βουτσάς Ιωάννης, ΑΜ: 5166

Υπεύθυνος Καθηγητής:

Χρήστος Ι. Μπούρας, Καθηγητής

Επιβλέπων:

Δρ. Ανδρέας Παπαζώης

ΠΑΤΡΑ, Σεπτέμβριος 2016

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον υπεύθυνο καθηγητή της διπλωματικής και καθηγητή του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής της Πολυτεχνικής σχολής του Πανεπιστημίου Πατρών κ. Χρήστο Ι. Μπούρα που με εμπιστεύτηκε και μου επέτρεψε να εκπονήσω αυτήν την εργασία.

Επίσης θα ήθελα να ευχαριστήσω θερμά τον Δρ. Ανδρέα Παπαζώη για την βοήθεια του και την υπομονή που έδειξε. Αποτέλεσε σημαντικό και στέρεο έρεισμα σε όλα τα στάδια της εκπόνησης της εργασίας.

Τέλος θα ήθελα να ευχαριστήσω βαθιά την οικογένεια μου, που με στήριξε και με βοήθησε πάντα με τους καλύτερους τρόπους καθ' όλη την διάρκεια της ακαδημαϊκής μου πορείας. Την διπλωματική αφιερώνω σε αυτούς.

Πάτρα, Σεπτέμβριος 2016

Ιωάννης Βουτσάς

Περίληψη

Το μεγαλύτερο μέρος της πληροφορίας σήμερα διακινείται μέσω των υπολογιστών και φυσικά των δικτύων δεδομένων υψηλών ταχυτήτων. Μπορούμε να πούμε ότι ζούμε στην Κοινωνία της Πληροφορίας λόγω της ταχύτητας, της αξιοπιστίας αλλά και της απόστασης που μπορεί να καλύψει μια ποσότητα πληροφορίας. Επομένως και ο τομέας της έρευνας γύρω από τα δίκτυα είναι θεμελιώδης για την βελτίωση της ποιότητας τους. Μάλιστα, λόγω του υψηλού κόστους συντήρησης, που συνεχώς αυξάνεται, υπάρχει η ανάγκη για μια αλλαγή που θα φέρει μεγαλύτερη ευελιξία με μικρότερο κόστος στα δίκτυα δεδομένων υψηλών ταχυτήτων.

Προς αυτή την κατεύθυνση στρέφεται το Software Defined Networking (SDN). Πρόκειται ουσιαστικά για μια σκέψη να διαχωριστεί το επίπεδο των δεδομένων με αυτό του ελέγχου του δικτύου. Πιο συγκεκριμένα, εισάγεται ένα επίπεδο μεγαλύτερης αφαίρεσης το οποίο ελέγχει και παίρνει αποφάσεις για το που θα κινηθεί η πληροφορία, ενώ στο χαμηλότερο αφαιρετικά επίπεδο τα δεδομένα προωθούνται με τον μέχρι σήμερα γνωστό τρόπο. Για να υλοποιηθεί αυτό λοιπόν, δημιουργείται η ανάγκη υλοποίησης ενός ελεγκτή που θα εμφωλεύει όλες αυτές τις λειτουργίες, οι οποίες προσφέρουν στα δίκτυα ευελιξία, αντοχή σε βλάβες μέσω απομόνωσης και αναδιάταξης της εικονικής τοπολογίας τους, μεγαλύτερη ταχύτητα και φυσικά, έναν εύκολο, κεντρικοποιημένο τρόπο διαχείρισης. Το Open Networking Operating System (ONOS) είναι ένα τέτοιο περιβάλλον που υλοποιεί έναν ελεγκτή και κάνει εφικτό όλο αυτό το εγχείρημα. Προσφέρει δυνατότητες για υλοποίηση διαφόρων πρωτοκόλλων επικοινωνίας, μπορεί να ενσωματώσει διαφόρων τύπου συσκευές και να διαχειρίζεται και ελέγχει ένα δίκτυο.

Μάλιστα αυτός είναι και ο σκοπός της εργασίας. Η υλοποίηση της επικοινωνίας μια συσκευής Cisco με τον ελεγκτή του ONOS, μέσω του πρωτοκόλλου Netconf.

Executive Summary

The biggest portion of information transferred today, is done by computers and of course through high-speed data networks. It is a fact that we live in the Information Society due to the speed, reliability as well as the distance that a piece of information can cover. Therefore, research related to high-speed data networks is fundamental for improving their quality. In fact, because of the very high cost of maintenance, that is perpetually growing, there is the need for a change that will make networks more flexible while reducing cost.

This is the direction that Software Defined Networking (SDN) is looking to. It is actually an idea about separating the data plane from the control plane of a single network. In particular, there is a higher abstraction layer that is controlling and taking decisions about how to handle the information flows while on the other hand, in the lower abstraction level the flows remain the same as currently known. For this purpose, there is the need for implementing a controller that will nest all these operations, providing to networks flexibility, stability, resistance to damage by isolating and rescheduling the virtual topology, better speed and of course an easy centralized way for managing the network. Open Networking Operating System (ONOS) is an environment that implements a controller and carries out this idea. It provides the means for implementing various protocols, it can embed devices of various types and can also manage and control a network.

In fact, that is the purpose of this diploma thesis. Implementing the communication between a Cisco device and an Onos controller, under Netconf protocol.

Περιεχόμενα

Πίνακας Εικόνων

Ακρωνύμια

SDN: Software Defined Networking

ONF: Open Network Foundation

TCP: Transmission Control Protocol

API: Application programming interface

NAT: Network address translation

QoS: Quality of Service

RPC: Remote procedure call

XML: Extensible Markup Language

DTD: Document Type Definition

XSLT: EXtensible Stylesheet Language

SSH: Secure Shell

TLS: Transport Layer Security

RADIUS: Remote Authentication Dial-In User Service

URI: Uniform Resource Identifier

ONOS: Open Network Operating System

NFV: Network function virtualization

DNS: Domain Name System

IP: Internet Protocol

MAC: Media Access Control

PoE: Power over Ethernet

YAML: YAML Ain't Markup Language

YANG: Yet Another Next Generation

SNMP: Simple Network Management Protocol

MIB: Management Information Base

IDE: Integrated Development Environment

OSGi: Open Services Gateway initiativ

IOS: Internetwork Operating System

1.Εισαγωγή

Software Defined Networking

Ως μια νέα αναδυόμενη τεχνολογία, το Software Defined Networking (SDN), είναι δύσκολο να οριστεί ξεκάθαρα και αυστηρά. Δεν υπάρχει ακόμη η σιγουριά στην επιστημονική κοινότητα για το που θα καταλήξει, όμως σίγουρα μπορούμε να επισημάνουμε κάποια θεμελιώδη στοιχεία του και μέσω αυτών, να κατανοήσουμε τι είναι το SDN [1]. Το βασικότερο από αυτά είναι και ο ορισμός που χρησιμοποιεί ο Open Network Foundation (ONF, ο οργανισμός που διαχειρίζεται τα πρότυπα Openflow):

"*SDN is the physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices*".

Για να γίνει περισσότερο κατανοητός αυτός ο διαχωρισμός μπορούμε να θεωρήσουμε μια συσκευή στην οποία η διαδικασία που αποφασίζει το πώς θα διαχειριστεί η συσκευή το κάθε πακέτο που βρίσκεται στις εισόδους της, είναι το επίπεδο ελέγχου(control plane). Από την άλλη, τα κυκλώματα τα οποία κάνουν εφικτή την αποστολή και την λήψη αυτών των πακέτων, αποτελούν το επίπεδο των δεδομένων (data plane). Επιπλέον λιγότερο συχνά αναφέρεται και το επίπεδο διαχείρισης (management plane) που αφορά την ρύθμιση των συσκευών. Σε ένα μοντέλο SDN, το control plane και το management plane, «παραμένουν» με την συσκευή, ενώ αντίθετα το control plane αποκόπτεται και υλοποιείται σε μια περιοχή-κέντρο του δικτύου με την έννοια της πρόσβασης από ομάδες συσκευών.

Κι ενώ το επίπεδο των δεδομένων παραμένει όπως το ξέραμε μέχρι σήμερα, δηλαδή με τα TCP packets όπου καθοδηγούνται από τα διάφορα tables των routers, ο ελεγκτής έρχεται να προστεθεί πάνω σε αυτόν τον σχεδιασμό αναλαμβάνοντας κάποιες διεργασίες που θα προσφέρουν τελικά στο δίκτυο τα πλεονεκτήματα του SDN. Τέτοιες διεργασίες είναι η συλλογή πληροφοριών και δυνατοτήτων για τις συσκευές, η συλλογή δεδομένων και η στατιστική μελέτη ή εκτέλεση αλγορίθμων που μακροσκοπικά θα κάνουν την κίνηση στο δίκτυο πιο αποδοτική.

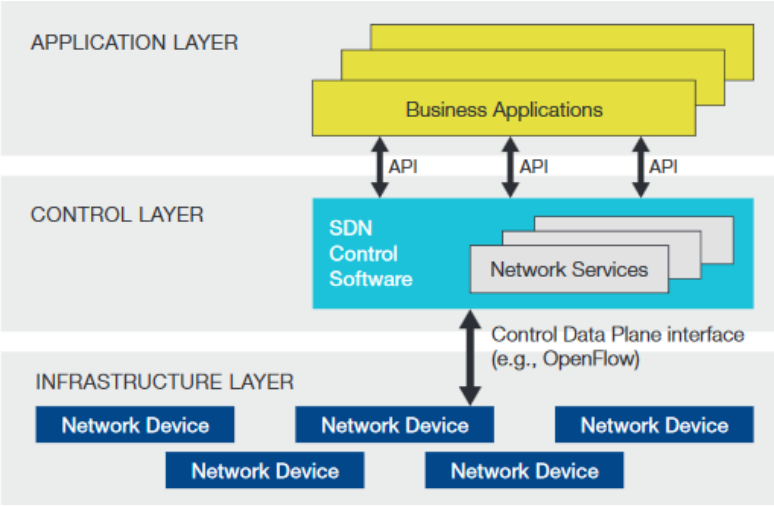
Υπάρχουν πολλές περιπτώσεις που το SDN είναι αναγκαίο και χρησιμοποιείται, που όλο και αυξάνονται εξαιτίας της αντίστοιχα αυξανόμενης απαίτησης που δημιουργούν νέες τεχνολογίες όπως, η επικράτηση των mobile συσκευών στην αγορά, υπηρεσίες cloud, big data. Αυτές είναι περιπτώσεις αναδιάταξης της τοπολογίας προώθησης εντός του δικτύου, μέσω αυτόματων διαδικασιών που ελέγχουν την κίνηση στο δίκτυο, και το διαμορφώνουν με τέτοιο τρόπο, ώστε να μειώνεται η συμφόρηση. Επίσης συμβάλει στο να γίνονται τα δίκτυα περισσότερο σταθερά και ανεκτικά σε βλάβες, έτσι ώστε όταν ένα μικρό κομμάτι του δικτύου παρουσιάσει πρόβλημα, το δίκτυο να μπορεί να λειτουργεί με τον καλύτερο δυνατό τρόπο απομονώνοντας το πρόβλημα, όσο γίνεται πιο κοντά στην ρίζα του. Ως παράδειγμα μπορεί να δοθεί μια συσκευή που για κάποιο λόγο τίθεται ανενεργή, ο ελεγκτής αντιλαμβάνεται ότι στο συγκεκριμένο σημείο κάποιος

κόμβος εξαφανίστηκε, και αναλαμβάνει να βρει νέο μονοπάτι ώστε η πληροφορία να φτάσει τελικά στον επιθυμητό προορισμό. Φυσικά, μεγάλη χρησιμότητα έχει και το να έχουμε την διαχείριση ενός δικτύου κάπου κεντρικοποιημένα, όπως στην περίπτωση ενός κτιρίου, ενός χώρου εργασίας, ενός πανεπιστημίου, μιας πόλης…

Ωστόσο, εκφράζονται και κάποιες ανησυχίες σχετικά με την επικράτηση του SDN. Πράγματι αξίζει να σημειωθεί, πως μια τόσο νέα τεχνολογία όπως το SDN, κρύβει κινδύνους ασφαλείας που δεν μπορούμε να φανταστούμε σε αυτό το σημείο που βρισκόμαστε. Το τοπίο όμως θα τείνει να ξεκαθαρίσει όσο καταλήγουμε σε μια πιο συμπαγή έννοια Software Defined Networking.

Από την άλλη, μπορούμε να πούμε και το μεγάλο πλεονέκτημα του SDN. Είναι κάτι που αξίζει να δοκιμαστεί, αφού σχεδιάζεται έτσι ώστε να ενσωματώνεται στην ήδη υπάρχουσα δικτύωση. Ένας ελεγκτής δηλαδή μπορεί εύκολα να ενσωματώνεται και να αποκόπτεται από το δίκτυο, χωρίς να χρειάζεται να γίνει από την αρχή η δικτύωση ή να αλλάξει η τοπολογία και ο τρόπος σύνδεσης των συσκευών. Ένας ελεγκτής θα πρέπει να μπορεί να συνδέεται και αποσυνδέεται από ένα δίκτυο, χωρίς να απαιτείται χρόνος όπου το δίκτυο θα παραμείνει ανενεργό.

Αν θέλουμε να κοιτάξουμε τώρα στο πως επικοινωνούνε τα επίπεδα του SDN μεταξύ τους, θα πρέπει πρώτα να έχουμε ορίσει τα layers των Southbound και Northbound API's. Το Northbound Interface είναι μια διεπαφή που επιτρέπει σε ένα στοιχείο του δικτύου, να επικοινωνεί με ένα στοιχείο υψηλότερου επιπέδου. Αντίστοιχα το Southbound Interface είναι μια διεπαφή που επιτρέπει σε ένα στοιχείο του δικτύου, να επικοινωνεί με ένα στοιχείο χαμηλότερου επιπέδου [2].



Εικόνα 1: Αρχιτεκτονική Software Defined Network (πηγή [9])

Ως παραδείγματα τέτοιων στοιχείων μπορούμε να θεωρήσουμε ένα router και τον ελεγκτή του δικτύου. Πιο αναλυτικά, το Southbound Interface είναι ένα πρότυπο που καθορίζει τον τρόπο

όπου ο ελεγκτής του SDN επικοινωνεί με το επίπεδο όπου τα δεδομένα κινούνται, για να κάνει αλλαγές στο δίκτυο ώστε να εξυπηρετεί καλύτερα τις ανάγκες του δικτύου. Ενώ από την άλλη το Northbound Intercace είναι ο τρόπος που ο ελεγκτής επικοινωνεί με εφαρμογές και υπηρεσίες που τρέχουν πάνω στο δίκτυο.

OpenFlow

Το Southbound Interface που χρησιμοποιείται συνήθως στα SDN είναι το OpenFlow. Βασική του λειτουργία είναι να ενεργοποιεί την επικοινωνία ώστε η συσκευή να μπορεί να ανακαλύψει την ευρύτερη τοπολογία του δικτύου, να ορίσει ροές καθώς και να υλοποιήσει αιτήματα που σχετίζονται με το πρωτόκολο OpenFlow μέσω Northbound API's. Είναι ο τρόπος με τον οποίο το SDN γίνεται πράγματικότητα.

Η βασική ιδέα είναι ότι τα περισσότερα σύγχρονα routers και switches χρησιμοποιούν για την επικοινωνία τους εντός του δικτύου flow tables για να υλοποιήσουν ουσιαστικά firewalls, NAT, QoS, και να συλλέξουν στατιστικά. Το OpenFlow εκμεταλλεύεται αυτές τις βασικές λειτουργίες. Προσφέρει το περιβάλλον που είναι απαραίτητο για να προγραμματιστούν αυτά τα flow tables σε διάφορες συσκευές. Ένας διαχειριστής δικτύου μπορεί να επιμερίσει την κίνηση στην τοπολογία σύμφωνα με τις ανάγκες του (είτε πρακτικές ή ακόμα και ερευνητικές). Μπορεί δηλαδή να καθορίσει ο ίδιος τις ροές επιλέγοντας κατάλληλα τις εγγραφές εντός των συσκευών. Με αυτό τον τρόπο μπορούν να εφαρμοστούν και να δοκιμαστούν πρωτόκολλα επικοινωνίας, μοντέλα ασφαλείας, σχήματα ροών και να επιλεχθούν τα επιθυμητά για μια τοπολογία [7] .

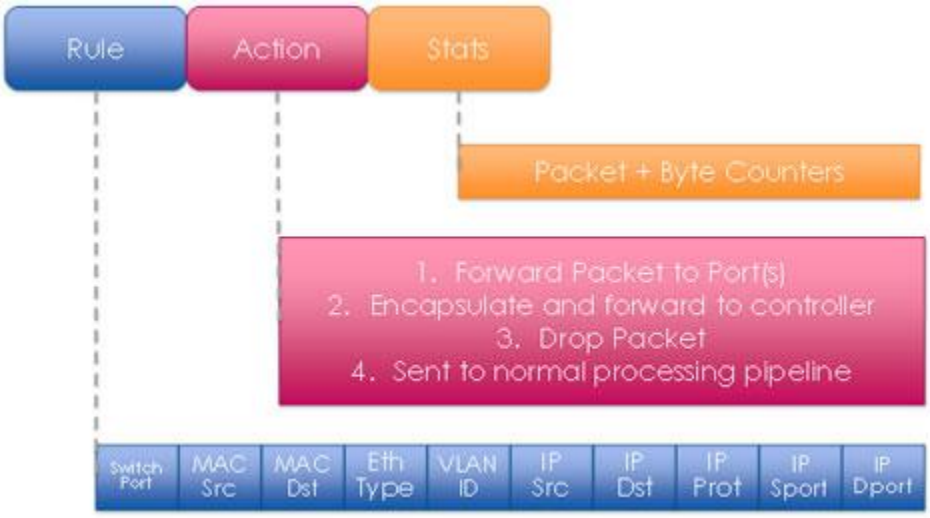
Ένα Flow Table στο OpenFlow ορίζεται ως ένας πίνακας ο οποίος καθορίζει μια ενέργεια για κάθε είσοδο. Μάλιστα, οι δυνατότητες ενός τέτοιου Flow table είναι επεκτάσιμες. Ένα OpenFlow Switch απαρτίζεται από τρία χαρακτηριστικά:

- Το Flow Table, με μία σχετική ενέργεια για κάθε είσοδο, ώστε να ξέρει το switch πως θα την διαχειριστεί
- Μία ασφαλή σύνδεση (Secure Channel) η οποία διατηρεί την σύνδεση του switch με τον ελεγκτή
- Το OpenFlow πρωτόκολλο το οποίο προσφέρει έναν πρότυπο τρόπο με τον οποίο ο ελεγκτής και το switch θα επικοινωνούν.

Οι ενέργειες σε ένα flow entry μπορεί να είναι η προώθηση ενός πακέτου στο δοθέν port (ή ports). Αυτό επιτρέπει ουσιαστικά στα πακέτα να διακινούνται εντός του δικτύου και λαμβάνει χώρα στο επίπεδο των δεδομένων. Μια άλλη βασική ενέργεια είναι το πακέτο να διαμορφώνεται κατάλληλα και να προωθείται στον ελεγκτή. Αυτό επιτυγχάνεται μέσω του Secure Channel. Τέλος μπορεί να αγνοηθεί κάποιο πακέτο. Μια τέτοια ενέργεια εξυπηρετεί ζητήματα ασφάλειας όπως Denial of Service επιθέσεις.

Το flow entry έχει τρία πεδία:

- Μια επικεφαλίδα πακέτου που ορίζει την ροή
- Την ενέργεια που καθορίζει πως θα έπρεπε να διαχειρίζονται τα πακέτα
- Στατιστικά στοιχεία, όπου μελετάνε αριθμό πακέτων και bytes για κάθε ροή, καθώς και τον χρόνο που το τελευταίο πακέτο πέρασε από την συγκεκριμένη εγγραφή.



Εικόνα 2: Flow-Table Οντότητες ενός OpenFlow switch (πηγή [8])

Όπως γίνεται αντιληπτό, οι δυνατότητες του OpenFlow το καθιστούν απαραίτητο στην οικογένεια του Software Defined Networking, αφού αυξάνει την απόδοση του δικτύου από πλευράς ταχύτητας και ασφάλειας, με αλλαγές που γίνονται real-time στο δίκτυο, σύμφωνα με τις ανάγκες του. Το feedback που έχει ο ελεγκτής, οι στατιστικές αναλύσεις και η συνεχής γνώση των λεπτομερειών της τοπολογίας του δικτύου, μεταφράζεται τελικά σε αλλαγές στα flow tables μέσω του πρωτοκόλλου Openflow, έτσι ώστε τελικά μια τοπολογία να γίνει γρηγορότερη, ασφαλέστερη και περισσότερο ευσταθής [8].

2. Το Πρωτόκολλο Netconf

2.1 Εισαγωγή στο Netconf

Το πρωτόκολλο Netconf περιγράφει έναν απλό μηχανισμό μέσω του οποίου μπορούμε να διαχειριστούμε συσκευές ενός δικτύου, να πάρουμε δεδομένα σχετικά με τις ρυθμίσεις τους ή να ορίσουμε και να τις ενημερώσουμε με νέες. Το πρωτόκολλο επιτρέπει στις συσκευές να εκθέτουν ένα ολοκληρωμένο πρότυπο API. Οι διάφορες εφαρμογές μπορούν να χρησιμοποιούν αυτό το API για να στείλουν και να λάβουν ολόκληρο ή μέρος από ένα σύνολο δεδομένων.

Το Netconf χρησιμοποιεί κλήσεις απομακρυσμένων διαδικασιών (remote procedure calls, RPC). Ένας client, κωδικοποιεί ένα τέτοιο RPC σε XML και το στέλνει στον server χρησιμοποιώντας μια ασφαλή σύνδεση. Ο server ανταποκρίνεται με μια απάντηση κωδικοποιημένη επίσης σε XML. Το περιεχόμενο αμφότερων κλήσης και απάντησης είναι καλά ορισμένο σε XML DTDs ή σε XML schemas, ή και στα δύο, έτσι ώστε και οι δύο πλευρές να αναγνωρίζουν το συντακτικό και τους περιορισμούς που επιβάλλονται κατά την ανταλλαγή.

Μια βασική πτυχή του πρωτοκόλλου Netconf είναι ότι επιτρέπει στην λειτουργικότητα του πρωτοκόλλου διαχείρισης να αναπαριστά πιστά τις λειτουργίες της συσκευής. Αυτό μειώνει το κόστος της υλοποίησης και επιτρέπει την έγκαιρη πρόσβαση σε νέα χαρακτηριστικά. Επιπλέον, οι εφαρμογές μπορούν να έχουν πρόσβαση τόσο στο συντακτικό όσο και στο εννοιολογικό περιεχόμενο του user interface της συσκευής.

Το Netconf επιτρέπει στον client να ανακαλύψει το σύνολο των επεκτάσεων πρωτοκόλλων που υποστηρίζει ο server. Αυτά είναι τα “capabilities” και επιτρέπουν στον client να προσαρμόσει την συμπεριφορά του ώστε να εκμεταλλευτεί τα πλεονεκτήματα που του δίνονται από τα χαρακτηριστικά της συσκευής. Τα ορισμένα αυτά capabilities μπορούν εύκολα να επεκταθούν, και έξω από τα πρότυπα αρκεί να ορίζονται με συντακτική και εννοιολογική αυστηρότητα.

Το πρωτόκολλο αυτό, είναι ο θεμέλιος λίθος ενός συστήματος αυτόματης διαμόρφωσης. Η XML είναι η κοινή γλώσσα των εκατέρωθεν επικοινωνούντων στοιχείων, προσφέροντας ένα ευέλικτο αλλά καλά ορισμένο μηχανισμό κωδικοποίησης περιεχομένων με ιεραρχική δομή. Το Netconf μπορεί να χρησιμοποιηθεί ταυτόχρονα με XML-based τεχνολογίες μετατροπής, όπως για παράδειγμα XSLT, παρέχοντας ένα σύστημα για αυτόματη παραγωγή των ρυθμίσεων. Το σύστημα αυτό μπορεί να κάνει ερωτήματα σε μία ή περισσότερες βάσεις δεδομένων για πληροφορίες σχετικά με τοπολογίες δικτύων, συνδέσεις, πολιτικές, πελάτες και υπηρεσίες. Αυτές οι πληροφορίες μπορούν να μετατραπούν χρησιμοποιώντας ένα η περισσότερα XSLT scripts από ένα data schema, σε μια συγκεκριμένη μορφή ανάλογα με τον πάροχο, το λειτουργικό σύστημα και το λογισμικό που χρησιμοποιείται.

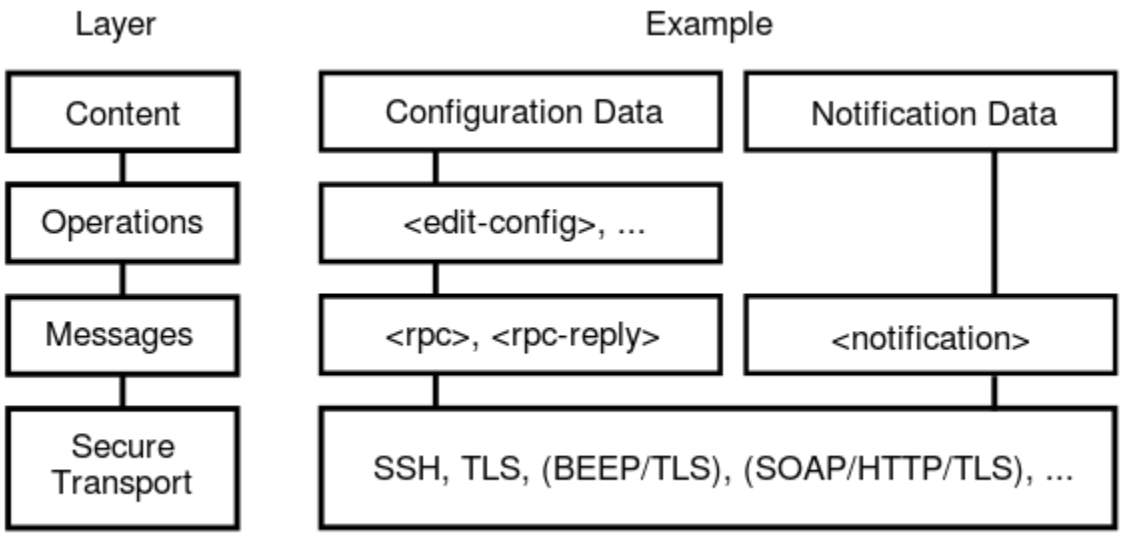
Οι επόμενες ενότητες είναι βασισμένες στις αναφορές [4] και [5].

Επισκόπηση του πρωτοκόλλου

Το πρωτόκολλο Netconf χρησιμοποιεί έναν απλό μηχανισμό βασισμένο στα RPC's για να πραγματοποιήσει την επικοινωνία μεταξύ του client και του server. Ο client μπορεί να είναι ένα script ή μία εφαρμογή που τρέχει ως μέρος του διαχειριστή του δικτύου. Ο server συνήθως είναι μια συσκευή.

Μια συνεδρία Netconf είναι ο λογικός σύνδεσμος μεταξύ του διαχειριστή του δικτύου και μιας συσκευής. Κάθε τέτοια συσκευή θα πρέπει να μπορεί να υποστηρίξει τουλάχιστον μια τέτοια συνεδρία Netconf. Στην περίπτωση πολλών συνεδριών, αλλαγές στις ρυθμίσεις της συσκευής γίνονται ορατές από όλες τις συνεδρίες.

Το πρωτόκολλο μπορεί να χωριστεί εννοιολογικά σε τέσσερα στρώματα.



Εικόνα 3: Επίπεδα του πρωτοκόλλου NETCONF (πηγή [23])

1. Το Secure Transport layer, προσφέρει ένα μονοπάτι επικοινωνίας μεταξύ του client και του server. Το Netconf μπορεί να βασιστεί σε οποιοδήποτε πρωτόκολλο αυτού του επιπέδου. Περισσότερα για τις συνδέσεις Netconf θα παρουσιαστούν παρακάτω.
2. Το Messages layer, παρέχει έναν απλό, ανεξάρτητο του τρόπου μεταφοράς, μηχανισμό για κωδικοποίηση και εμφώλευση των RPC's και των ειδοποιήσεων.
3. Το operations layer, ορίζει ένα σύνολο βασικών λειτουργιών με βάση το πρωτόκολλο που καλούνται ως RPC μέθοδοι, με παραμέτρους κωδικοποιημένες σε XML
4. Το content layer είναι έξω από την σκοπιά που μελετάμε. Γενικά αναμένεται να γίνουν προσπάθειες για να καθοριστούν τα Netconf μοντέλα δεδομένων.

Capabilities

Τα Netconf capabilities είναι ένα σύνολο λειτουργιών τα οποία συμπληρώνουν τα βασικά χαρακτηριστικά του πρωτοκόλλου. Κάθε capability είναι αναγνωρίσιμο από μία ταυτότητα URI(Uniform Resource Identifier).

Τα capabilities λοιπόν, αυξάνουν των αριθμό των βασικών λειτουργιών της συσκευής, περιγράφοντας τόσο τις πρόσθετες ικανότητες όσο και το περιεχόμενο που επιτρέπεται σύμφωνα με αυτές. Ο client ζητάει τις ικανότητες αυτές από τον server, και χρησιμοποιεί όποιες πρόσθετες υπάρχουν, παραμέτρους αλλά και περιεχόμενο που περιγράφεται από αυτές.

Επιπλέον, τα capabilities μπορούν να οριστούν οποιαδήποτε στιγμή, σε εξωτερικά αρχεία, κάνοντας εφικτό έτσι, να αυξάνονται στον χρόνο. Υπάρχουν capabilities από διάφορα πρότυπα ενώ ανάλογα με τις υλοποιήσεις μπορεί να υπάρχουν και ιδιόκτητα. Για αυτό τον λόγο όμως κιόλας, πρέπει τα URI των capabilities να είναι μοναδικά ώστε να αποφεύγονται αντιφάσεις.

Διαχωρισμός δεδομένων ρυθμίσεων-κατάστασης

Η πληροφορία που μπορεί να ανακτηθεί από ένα τρέχον σύστημα, χωρίζεται σε δύο κλάσεις. Τα δεδομένα σχετικά με ρυθμίσεις του συστήματος και τα δεδομένα που αφορούν την κατάσταση του συστήματος. Τα δεδομένα αυτά που είναι σχετικά με τις ρυθμίσεις, είναι το σύνολο αυτών των δεδομένων που είναι απαραίτητα για να φέρουν το σύστημα από την προεπιλεγμένη κατάσταση στην επιθυμητή. Αντίθετα τα δεδομένα κατάστασης είναι τα δεδομένα που δεν αφορούν ρυθμίσεις αλλά πληροφορίες μόνο ανάγνωσης σχετικά με την κατάσταση ή στατιστικά.

Όταν μια συσκευή επιχειρεί λειτουργίες ρύθμισης, μπορεί να προκύψουν κάποια προβλήματα.

- Σύγκριση δεδομένων ρυθμίσεων οι οποίες όμως έχουν άσχετες μεταξύ τους εγγραφές
- Κάποιο αίτημα που έρχεται προσπαθεί να αλλάξει περιεχόμενο μόνο ανάγνωσης
- Τεράστιο σύνολο δεδομένων

Για να λύσει τέτοια προβλήματα, το πρωτόκολλο Netconf αναγνωρίζει την διαφορά μεταξύ των δύο κλάσεων δεδομένων και παρέχει λειτουργίες για τη κάθε μία ξεχωριστά. Ως παράδειγμα μπορούμε να αναφέρουμε ότι η αίτηση <get-config> ζητάει δεδομένα ρυθμίσεων ενώ από την άλλη ένα αίτημα <get> μπορεί να ζητήσει είτε δεδομένα ρυθμίσεων είτε κατάστασης.

Επισημαίνεται ότι το Netconf εστιάζεται στο πως μια συσκευή θα πάρει τις ρυθμίσεις που πρέπει για να φτάσει στην επιθυμητή κατάσταση. Λοιπά δεδομένα που μπορεί να χρειάζονται για τις συσκευές εξαρτώνται από την εκάστοτε υλοποίηση. Για παράδειγμα , τα αρχεία κάποιου χρήστη ή μια βάση δεδομένων δεν εξετάζονται ως δεδομένα ρυθμίσεων από το πρωτόκολλο. Δηλαδή αν σε μια συσκευή υπάρχει μια βάση δεδομένων με πληροφορίες για την αναγνώριση των χρηστών, δεν είναι απαραίτητο ότι θα εμφανίζεται αυτή η πληροφορία ως ρύθμιση, αλλά εξαρτάται από την υλοποίηση που έχει γίνει.

2.2 Transport protocol requirements

Από αυτή την οπτική σκοπιά, το Netconf πρωτόκολλο χρησιμοποιεί ένα παράδειγμα επικοινωνίας βασισμένο σε RPC μηνύματα. Ένας client στέλνει μια σειρά από μηνύματα RPC που καλούν τον server να απαντήσει εν συνεχεία με τα αντίστοιχα reply μηνύματα.

Το Netconf μπορεί να βασιστεί σε οποιοδήποτε πρωτόκολλο μεταφοράς που παρέχει ένα τέτοιο σύνολο λειτουργιών. Δεν περιορίζεται σε κάποιο συγκεκριμένο, αλλά προσφέρει τους ορισμούς εκείνους σύμφωνα με τους οποίους ένα πρωτόκολλο μπορεί να ικανοποιήσει την απαιτούμενη επικοινωνία όπως έχει περιγραφεί μέχρι τώρα.

Ακόμη, πρέπει να παρέχεται ο μηχανισμός εκείνος που θα υποδεικνύει τα χαρακτηριστικά της συνεδρίας, δηλαδή ποιος είναι ο client και ο server, στο πρωτόκολλο Netconf. Παρακάτω περιγράφονται πιο αναλυτικά τα χαρακτηριστικά αυτά.

Λειτουργία βασισμένη στην σύνδεση

Το Netconf είναι ένα πρωτόκολλο που βασίζεται σε μια επίμονη, σταθερή σύνδεση μεταξύ των μερών της. Αυτή η σύνδεση προσφέρει αξιόπιστη, ακολουθιακή ροή δεδομένων. Αυτές οι συνδέσεις έχουν μεγάλο χρόνο ζωής και είναι επίμονες σχετικά με τις λειτουργίες που πρέπει να εκτελεστούν.

Επιπλέον, οι πόροι που ζητάει ο server για μια συγκεκριμένη σύνδεση, αυτόματα ελευθερώνονται μόλις η σύνδεση κλείσει για λόγους σωστής διαχείρισης αποτυχιών. Για παράδειγμα όταν υπάρχει μια απαίτηση για lock, τότε αυτό παραμένει μέχρι να αλλάξει η απαίτηση ή ο server να αποφασίσει ότι η σύνδεση έχει κλείσει. Αν η σύνδεση τερματιστεί ενώ ο client έχει στην κατοχή του μια λειτουργία lock, ο server μπορεί να εκτελέσει ότι χρειάζεται για να επανέλθει στην σωστή λειτουργία του.

Έλεγχος ταυτότητας, ακεραιότητα, εμπιστευτικότητα

Οι συνδέσεις Netconf, προσφέρουν έλεγχο ταυτότητας, ακεραιότητα στη μεταφορά των δεδομένων, εμπιστευτικότητα αλλά και ευστάθεια σε περίπτωση επαναλήψεων. Για αυτούς τους σκοπούς βασίζεται στο πρωτόκολο μεταφοράς. Το Netconf δεν ενδιαφέρεται για το πώς επιτυγχάνεται αυτό στο χαμηλότερο επίπεδο. Μπορεί να είναι δηλαδή μια σύνδεση Secure Shell (SSH) ή Transport Layer Security (TLS). Το πρωτόκολλο αυτό που θα επιλεχθεί είναι και υπεύθυνο για αναγνώριση του client από τον server και το ανάποδο. Το Netconf θεωρεί ότι αυτός ο έλεγχος γίνεται με έναν ασφαλή κρυπτογραφημένο τρόπο χρησιμοποιώντας αξιόπιστους μηχανισμούς.

Ένας στόχος του Netconf είναι να παρέχει ένα προγραμματιστικό περιβάλλον στην συσκευή που εκμεταλλεύεται ολοκληρωμένα όλες τις τοπικές λειτουργίες τις συσκευής. Έτσι για παράδειγμα σε μία συσκευή που υποστηρίζει έλεγχο RADIUS (Remote Authentication Dial-In User Service), ο έλεγχος θα γίνει σύμφωνα με αυτό από τον server. Ο έλεγχος τελικά, οδηγεί σε μία

σύνδεση από την οποία ο server γνωρίζει ότι ο client είναι ταυτοποιημένος. Μία τέτοια ταυτότητα είναι γνωστή ως το όνομα χρήστη του client. Ο αλγόριθμος που εκτελείται για να προκύψει και να αντληθεί αυτό το όνομα χρήστη εξαρτάται από το transport πρωτόκολλο που χρησιμοποιείται, έτσι ώστε τελικά να μπορεί να χρησιμοποιηθεί και από τα υπόλοιπα layers του Netconf. Τέλος, η άδειες πρόσβασης που έχει ο client, και επιβεβαιώνονται από το όνομα χρήστη του, είναι μέρος των ρυθμίσεων του Netconf server.

2.3 XML

Η XML εξυπηρετεί το κομμάτι του Netconf που αφορά την κωδικοποίηση. Με την XML επιτυγχάνεται η έκφραση σύνθετων ιεραρχικών κομματιών δεδομένων, να αναπαρίστανται ως κείμενα τα οποία μπορούν να διαβαστούν, αποθηκευτούν και χρησιμοποιηθούν τόσο από κλασσικά εργαλεία κειμένων, όσο και από εργαλεία ειδικά για χρήση XML. Όλα τα μηνύματα οφείλουν να είναι καλά ορισμένα και γραμμένα σε XML και κωδικοποιημένα με το πρότυπο UTF-8. Αν ένα <rpc> μήνυμα δεν είναι ορθώς γραμμένο σε XML ή δεν είναι κωδικοποιημένο στο ορισμένο πρότυπο UTF-8, τότε λαμβάνεται ένα rpc error (“malformer-message”). Αν για κάποιο λόγο δεν μπορεί να δοθεί απάντηση, η συνεδρία κλείνει.

Όλα τα περιεχόμενα των Netconf μηνυμάτων ορίζονται από το παρακάτω namespace:

```
urn:ietf:params:xml:ns:netconf:base:1.0
```

Ενώ σημειώνεται ότι τα ονόματα των capabilities πρέπει να είναι URIs.

Το Netconf πρωτόκολλο όπως ήδη έχει αναφερθεί χρησιμοποιεί ένα μοντέλο επικοινωνίας βασισμένο σε rpc μηνύματα. Τα μηνύματα αποστολής και λήψης πλαισιώνονται από <rpc> και <rpc-reply> φράσεις, για να ολοκληρωθεί έτσι η ανεξάρτητη του πρωτοκόλλου μεταφοράς επικοινωνία. Παρακάτω θα δοθούν παραδείγματα μηνυμάτων για να γίνει πλήρως κατανοητή η μορφή των μηνυμάτων.

<rpc> και <rpc-reply> elements

Το στοιχείο <rpc> χρησιμοποιείται για να πλαισιώσει μία αίτηση Netconf από τον client στον server. Μέσα σε αυτό το στοιχείο εσωκλείεται μία παράμετρος “message-id”, το οποίο είναι ένα string επιλεγμένο από τον αποστολέα, ώστε να διαχωρίζονται τα μηνύματα μεταξύ τους, και είναι συνήθως ένας γραμμικά αύξων αριθμός. Ο λήπτης δεν εκτελεί κάποια λειτουργία σύμφωνα με αυτή τη παράμετρο, αλλά απλά το αποθηκεύει για να στείλει την απάντηση <rpc-reply> στον αποστολέα σύμφωνα με το πρώτο. Ακόμη, όποια άλλη παράμετρος περιλαμβάνεται στο πλαίσιο <rpc> πρέπει να υπάρχει αυτούσιο και στην απάντηση.

<rpc-error>

Ένα <rpc-error> είναι ένα rpc-reply το οποίο αφορά λάθη τα οποία έχουν συμβεί κατά την διάρκεια της εξέτασης ή εκτέλεσης ενός rpc αιτήματος. Αν ένας server συναντήσει περισσότερα από ένα λάθη, δεν είναι απαραίτητο ότι θα αποτυπωθούν όλα τα λάθη και στο rpc-error που θα δοθεί ως απάντηση. Είναι όμως απαραίτητο να αποτυπωθεί ότι υπάρχει λάθος, αν έστω και ένα εντοπιστεί. Επίσης δεν επιστρέφονται λάθη τα οποία δεν αφορούν τον client ή δεν έχει τα κατάλληλα δικαιώματα για τις λειτουργίες που τα προκάλεσαν.

Ένα rpc-error περιλαμβάνει τις επόμενες πληροφορίες

error-type: Ορίζει το συγκεκριμένο layer από το οποίο το λάθος προέκυψε

- transport (layer: Secure Transport)
- rpc (layer: Messages)
- protocol (layer: Operations)
- application (layer: Content)

error-tag: Περιλαμβάνει ένα string το οποίο καθορίζει την κατάσταση του error.

error-severity: Περιλαμβάνει ένα string που ξεκαθαρίζει αν το error είναι τύπου error ή τύπου warning. Ωστόσο ακόμα δεν υπάρχει η υλοποίηση για πρόκληση warning αλλά δημιουργείται για μελλοντική υλοποίηση.

error-app-tag: Περιλαμβάνει ένα string το οποίο ξεκαθαρίζει την κατάσταση λαθών που προέκυψαν λόγω υλοποίησης ή λόγω μοντέλου δεδομένων, αν υπάρχουν τέτοια λάθη. Αν υπάρχουν και τα δύο είδη λαθών, τότε στέλνεται μήνυμα σχετικά με το μοντέλο δεδομένων.

error-path: Περιλαμβάνει το απόλυτο XPath που καθορίζει το μονοπάτι που σχετίζεται με το λάθος που προέκυψε. Μπορεί βέβαια να απουσιάζει από το πλαίσιο rpc-error αν δεν μπορεί να οριστεί.

error-message: Περιλαμβάνει κείμενο με την περιγραφή του λάθους, σε μορφή αναγνωρίσιμη από άνθρωπο. Ομοίως μπορεί να απουσιάζει αν δεν υπάρχει σχετικό μήνυμα να αναγραφεί.

error-info: Περιλαμβάνει περιεχόμενο λάθους σχετικό με το πρωτόκολλο ή του μοντέλου δεδομένων που χρησιμοποιείται. Είναι και αυτό ένα προαιρετικό περιεχόμενο το οποίο μπορεί να μην αναγράφεται αν δεν μπορεί να εντοπιστεί η ανάλογη πληροφορία.

Παρακάτω φαίνονται μερικά χαρακτηριστικά παραδείγματα λαθών για να γίνει σαφές το πώς διαμορφώνεται ένα rpc-error.

```
error-tag:      access-denied
error-type:     protocol, application
error-severity: error
error-info:     none
Description:    Access to the requested protocol operation or
                data model is denied because authorization failed.
```

Εικόνα 4: Netconf Error List - Access denied (πηγή [5])

```
 error-tag:      data-missing
 error-type:     application
 error-severity: error
 error-info:     none
 Description:    Request could not be completed because the relevant
                 data model content does not exist.  For example,
                 a "delete" operation was attempted on
                 data that does not exist.
```

Εικόνα 5: Netconf Error List - Data missing (πηγή [5])

```
error-tag:      unknown-element
error-type:     protocol, application
error-severity: error
error-info:     <bad-element> : name of the unexpected element
Description:    An unexpected element is present.
```

Εικόνα 6: Netconf Error List - Unknown attribute (πηγή [5])

```
error-tag:      operation-failed
error-type:     rpc, protocol, application
error-severity: error
error-info:     none
Description:    Request could not be completed because the requested
                operation failed for some reason not covered by
                any other error condition.
```

Εικόνα 7: Netconf Error List - Operation failed (πηγή [5])

Σημειώνεται σε αυτό το σημείο ότι τα rpc αιτήματα εξετάζονται και εκτελούνται σειριακά. Μπορούν να διατυπωθούν νέα αιτήματα πριν ολοκληρωθούν και ληφθούν οι απαντήσεις από τα προηγούμενα, ωστόσο οι απαντήσεις δίνονται μόνο με την σειρά που τέθηκαν.

2.4 Configuration model

Το Netconf προσφέρει ένα αρχικό σύνολο λειτουργιών και έναν αριθμό από capabilities τα οποία μπορούν να χρησιμοποιηθούν για να αυξήσουν αυτό το αρχικό σύνολο. Οι επικοινωνούντες ανταλλάσουν τα capabilities της συσκευής κατά την έναρξη της συνεδρίας.

Αποθήκες δεδομένων διαμόρφωσης

Το Netconf ορίζει την ύπαρξη ενός η περισσότερων αποθηκών δεδομένων σχετικά με της ρυθμίσεις διαμόρφωσης της συσκευής, και επιτρέπει λειτουργίες ρύθμισης σε αυτές. Μια τέτοια αποθήκη δεδομένων ορίζεται ως το ακριβές σύνολο των ρυθμίσεων εκείνων που έγιναν για να φτάσει η συσκευή από την αρχική προεπιλεγμένη κατάστασή της, σε μια επιθυμητή κατάσταση. Ωστόσο, αυτή η αποθήκη δεν περιλαμβάνει τα δεδομένα κατάστασης.

Η αποθήκη τρέχουσας διαμόρφωσης κρατάει όλες τις ρυθμίσεις που είναι ενεργές στην συσκευή. Φυσικά μόνο μια τέτοια διαμόρφωση μπορεί να είναι τρέχουσα κάθε φορά, οπότε αντίστοιχα υπάρχει και μόνο μια τέτοια αποθήκη της τρέχουσας διαμόρφωσης. Στο Netconf πρωτόκολλο η διαμόρφωση αυτή είναι γνωστή από τις λειτουργίες ως <running>.

Μοντελοποίηση δεδομένων

Η μοντελοποίηση των δεδομένων και θέματα σχετικά με το περιεχόμενο είναι έξω από την σκοπιά του Netconf. Ωστόσο αξίζει να κάνουμε μερικές αναφορές. Σε αυτό το επίπεδο, γίνεται η υπόθεση ότι και τα δύο μέρη της επικοινωνίας χρησιμοποιούν ένα από κοινού γνωστό μοντέλο δεδομένων. Αυτό σημαίνει ότι γνωρίζουν καλά τα χαρακτηριστικά του μοντέλου και ο τρόπος που στέλνονται τα δεδομένα είναι αυστηρά ορισμένος.

Το πρωτόκολλο Netconf κουβαλάει τις ρυθμίσεις διαμόρφωσης μέσα σε ένα στοιχείο <config> το οποίο είναι ασφαλώς σύμφωνο με το μοντέλο δεδομένων της συσκευής. Το πρωτόκολλο αντιμετωπίζει αυτό το στοιχείο ως κάτι ενιαίο και όχι απολύτως ξεκάθαρο. Η συσκευή χρησιμοποιεί τα capabilities για να ανακοινώσει τα μοντέλα δεδομένων που υποστηρίζει. Τόσο η συσκευή όσο και οι διαχειριστές μπορεί να υποστηρίζουν παραπάνω από ένα μοντέλα δεδομένων, που πέρα από τα βασικά, μπορεί να είναι και ιδιόκτητα.

2.5 Λειτουργίες του πρωτοκόλλου

Το netconf προσφέρει ένα μικρό σύνολο χαμηλού επιπέδου λειτουργιών για να επιτευχθεί η διαχείριση των συσκευών και η ανάκτηση πληροφοριών σχετικά με αυτήν. Λειτουργίες

ανάκτησης, διαμόρφωσης, αντιγραφής και διαγραφής αποθηκών δεδομένων. Επιπλέον λειτουργίες παρέχονται σύμφωνα με τα capabilities της συσκευής.

Οι βασικές λειτουργίες είναι αυτές που φαίνονται παρακάτω:

- get
- get-config
- edit-config
- copy-config
- delete-config
- lock
- unlock
- close-session
- kill-session

Μια λειτουργία σαν τις παραπάνω μπορεί να αποτύχει να εκτελεστεί για πολλούς λόγους, ενώ αν ζητηθεί μια λειτουργία που δεν υποστηρίζεται επιστρέφεται το αντίστοιχο μήνυμα. Οφείλει όμως το πρωτόκολλο να μην θεωρεί ότι μια λειτουργία εκτελείται πάντα, αλλά να ελέγχει για λάθη και να δίνει τις ανάλογες απαντήσεις.

Θα γίνει μια σύντομη περιγραφή των λειτουργιών.

<get-config> Ανακτώνται όλα ή μέρος από μία συγκεκριμένη αποθήκη δεδομένων διαμόρφωσης. Ως παράμετρος μπορεί να δοθεί το <running> που όπως εξηγήσαμε νωρίτερα, επιστρέφει την τρέχουσα διαμόρφωση. Ακόμη μπορεί να έχουμε και κάποια φίλτρα τα οποία όταν υπάρχουν μπορούν να καθορίσουν ποιο μέρος της διαμόρφωσης θα επιστρέψουν, ενώ αν δεν υπάρχει φίλτρο, προφανώς επιστρέφεται όλη η διαμόρφωση. Αν μπορεί να ικανοποιηθεί λοιπόν το αίτημα, ο server στέλνει ένα <rpc-reply> στοιχείο που περιέχει ένα <data> στοιχείο με τα αποτελέσματα του αιτήματος.

<edit-config> Η λειτουργία <edit-config> φορτώνει ολόκληρο ή μέρος από μια συγκεκριμένη διαμόρφωση σε μια συσκευή προορισμό, και μία συγκεκριμένη αποθήκη. Αυτή η λειτουργία επιτρέπει στην νέα διαμόρφωση να πραγματοποιηθεί με διάφορους τρόπους όπως χρησιμοποιώντας ένα τοπικό αρχείο, ή ένα απομακρυσμένο αρχείο είτε απευθείας με κείμενο. Αν η αποθήκη στην οποία προορίζεται η διαμόρφωση δεν υπάρχει θα δημιουργηθεί. Η συσκευή αναλύει την πηγή και τον προορισμό και κάνει τις απαραίτητες αλλαγές.

<copy-config> Δημιουργεί ή αντικαθιστά μια ολόκληρη αποθήκη δεδομένων διαμόρφωσης με το περιεχόμενο μιας άλλης ολοκληρωμένης. Αν η αποθήκη προορισμού υπάρχει ήδη, γράφεται πάνω σε αυτήν. Εναλλακτικά, αν επιτρέπεται από την συσκευή μια νέα δημιουργείται. Ως

παραμέτρους η αίτηση, παίρνει την πηγή και τον προορισμό. Δηλαδή το αρχείο με την διαμόρφωση και την αποθήκη που ζητάται να αλλαχτεί.

<delete-config> Η λειτουργία αυτή διαγράφει μια ολόκληρη διαμόρφωση. Σημειώνεται όμως ότι η τρέχουσα διαμόρφωση της συσκευής δεν μπορεί να διαγραφεί. Ως παράμετρος δίνεται ο προορισμός της διαμόρφωσης.

<lock> Η λειτουργία αυτή, επιτρέπει στον χρήστη να κλειδώσει ολόκληρη τη διαμόρφωση μιας συσκευής. Συνήθως αυτά τα lock γίνονται για μικρό χρόνο, και έχουν αξία γιατί ουσιαστικά επιτρέπουν στον χρήστη να κάνει αλλαγές στη διαμόρφωση χωρίς να φοβάται ότι κάποια άλλη συνεδρία θα επηρεαστεί. Έτσι για παράδειγμα, όταν υπάρχει lock από έναν client, και προσπαθήσει κάποιος άλλος να το κλειδώσει, η προσπάθεια αποτυγχάνει.

<unlock> Είναι προφανές ότι αυτή η λειτουργία απελευθερώνει ένα lock για να επαναφέρει την συσκευή σε καθεστώς ελευθερίας, και να δώσει ξανά την δυνατότητα σε άλλους χρήστες και συνεδρίες, να κάνουν αλλαγές αν το θελήσουν.

<get> Με την λειτουργία αυτή, ανακτούμε την τρέχουσα διαμόρφωση και πληροφορίες σχετικά με την κατάσταση της συσκευής. Μπορούμε να δώσουμε διάφορα φίλτρα για να ανακτήσουμε μόνο την πληροφορία που μας ενδιαφέρει.

<close-session> Η λειτουργία αυτή είναι ένα αίτημα για τερματισμό της Netconf συνεδρίας. Αν δοθεί λοιπόν ένα τέτοιο αίτημα, τερματίζεται η συνεδρία, αφού πρώτα η συσκευή απελευθερώσει όποιο lock έχει, αλλά και πόρους που έχουν σχέση με την συγκεκριμένη συνεδρία από την οποία προήλθε το close-session αίτημα. Αιτήματα που προκύπτουν μετά από ένα αίτημα close-session απορρίπτονται.

<kill-session> Η λειτουργία kill-session είναι μια βίαιη εκδοχή της close-session. Απαιτεί θα λέγαμε τον βίαιο τερματισμό της Netconf συνεδρίας. Η λέξη «βίαιος» σημαίνει ότι οποιαδήποτε λειτουργία λαμβάνει χώρα την στιγμή του αιτήματος αυτού, θα τερματιστεί αμέσως, τα locks απελευθερώνονται όπως και οι πόροι που σχετίζονται με την συνεδρία από όπου προήλθε το αίτημα kill.

3. Open Network Operating System (ONOS)

3.1 Επισκόπηση του ONOS

Το Open Network Operating System ή ως συντομογραφία ONOS, είναι ένα open source SDN λειτουργικό σύστημα. Προσφέρει σε παρόχους υπηρεσιών, πραγματικές SDN αλλά και NFV λύσεις. Στόχος αυτού του εγχειρήματος, είναι να γίνει αυτή η πλατφόρμα που θα ελέγχει τα δίκτυα των παρόχων, χωρίς αυτό να σημαίνει ότι δεν θα μπορεί να εξυπηρετεί και datacenters ή campus networks. Αυτό που σίγουρα σημαίνει όμως, είναι ότι πρέπει να τηρεί αυστηρά κάποια κριτήρια.

Πρώτα απ' όλα, οι απαιτήσεις κάνουν απαραίτητο μια πλατφόρμα σαν αυτή, να προσφέρει αποδοτικότητα, ευρεία διαθεσιμότητα και ανταπόκριση στον τομέα των δικτύων ενώ ακόμα να έχει και καλή προοπτική κλιμάκωσης και εξέλιξης με υποδομές που θα μπορούν μελλοντικά να ανταποκριθούν σε αλλαγές. Δεύτερον, για την διευκόλυνση αλλά και προσέλκυση παρόχων υπηρεσιών, η πλατφόρμα προσφέρει διεπαφές για υλοποίηση northbound API's, ενώ όσον αφορά το southbound κομμάτι, είναι και αυτό εξίσου αναγκαίο, για να επιτυγχάνεται διαχείριση πολλών συσκευών, από μια ευρεία γκάμα πρωτοκόλλων με συμπεριφορές ανεξάρτητες, που δεν επηρεάζουν η μία την άλλη. Ακόμη, προκειμένου να περιοριστεί η πολυπλοκότητα της πλατφόρμας και να διασφαλιστεί η εξέλιξη του, το ONOS έχει αναπτυχθεί με τέτοιο τρόπο, όπου ορίζεται αυστηρά η ανεξαρτησία μεταξύ των διεπαφών βορρά, νότου, ανατολής και δύσης.

Στο κομμάτι της ασφάλειας, το ONOS καταφέρνει να παραμένει ελαστικό και ανεκτικό σε διάφορες αποτυχίες, λόγω δόμησης του, ως ένα φυσικό κατανεμημένο σύστημα. Παρόλα αυτά, για να παραμείνει απλό αλλά και να είναι κεντρικοποιημένο όσον αφορά την ελεγκτική του δυνατότητα, το ONOS φαίνεται εικονικά ως πράγματι κεντρικοποιημένο. Αλλά επισημαίνεται ότι στην ουσία είναι ένα σύμπλεγμα από ανεξάρτητες, ατομικές περιπτώσεις στοιχείων που το δομούν. Κάθε τέτοιο στοιχείο ασφαλώς επικοινωνεί με τα άλλα δομικά στοιχεία του ONOS, και μπορούν να ανταλλάσουν φόρτο εργασίας όπου αυτό κρίνεται απαραίτητο από τον ελεγκτή. Έτσι τελικά, έχουμε πολλά επίπεδα δόμησης όπου κάθε ένα είναι αφαιρετική αναπαράσταση του κατώτερου επιπέδου ενώ το βασικό επίπεδο είναι αυτό του πυρήνα του ONOS, που κάνει και τον εικονικό συγκεντρωτισμό όλων των άλλων επιπέδων, λειτουργιών και δομικών στοιχείων του [6].

Southbound/northbound API

Ο πυρήνας διαχωρίζεται από τις υπόλοιπες βαθμίδες μέσω δύο σαφών λογικών ορίων. Το southbound interface, είναι ένα υψηλού επιπέδου API μέσω του οποίου ο πυρήνας αλληλεπιδρά με το περιβάλλον του δικτύου. Επίσης, εκτός από το να προσφέρει αυτή την άμεση επικοινωνία με το δίκτυο, το ONOS παρέχει και τις κατάλληλες υποδομές για να μπορεί να συνεργαστεί με οποιοδήποτε πρωτόκολλο επιλέξει ο χρήστης/προγραμματιστής είτε αυτό είναι OpenFlow, είτε

Netconf, OVSDB, TL1, είτε ακόμα και Command Line Interface (CLI). Έτσι το southbound API συμπεριφέρεται σαν μια γέφυρα για να επικοινωνεί ο πυρήνας με τα διάφορα πρωτόκολλα. Αυτό το ανεξάρτητο από το πρωτόκολλο API, εγγυάται ότι δεν θα υπάρχουν προβλήματα και αστοχίες λόγω του εκάστοτε πρωτοκόλλου στον πυρήνα.

Από την άλλη πλευρά, ο ONOS εκθέτει και εκφράζει ένα σύνολο αφαιρέσεων σε εφαρμογές και επιπλέον υπηρεσίες μέσω του northbound API. Αυτές οι αφαιρέσεις προσφέρουν μία γκάμα τρόπων πρόσβασης στις δικτυακές πληροφορίες, ξεκινώντας από τα χαμηλότερα επίπεδα, όπως οι συσκευές, τα links, οι hosts και φτάνοντας μέχρι τα υψηλότερα επίπεδα, όπως για παράδειγμα το γράφημα της δικτυακής τοπολογίας. Ακόμη προσφέρουν μια σειρά από αφαιρέσεις για να διαχειρίζονται την κατάσταση του δικτύου για παράδειγμα μέσω προγραμματισμού των ροών.

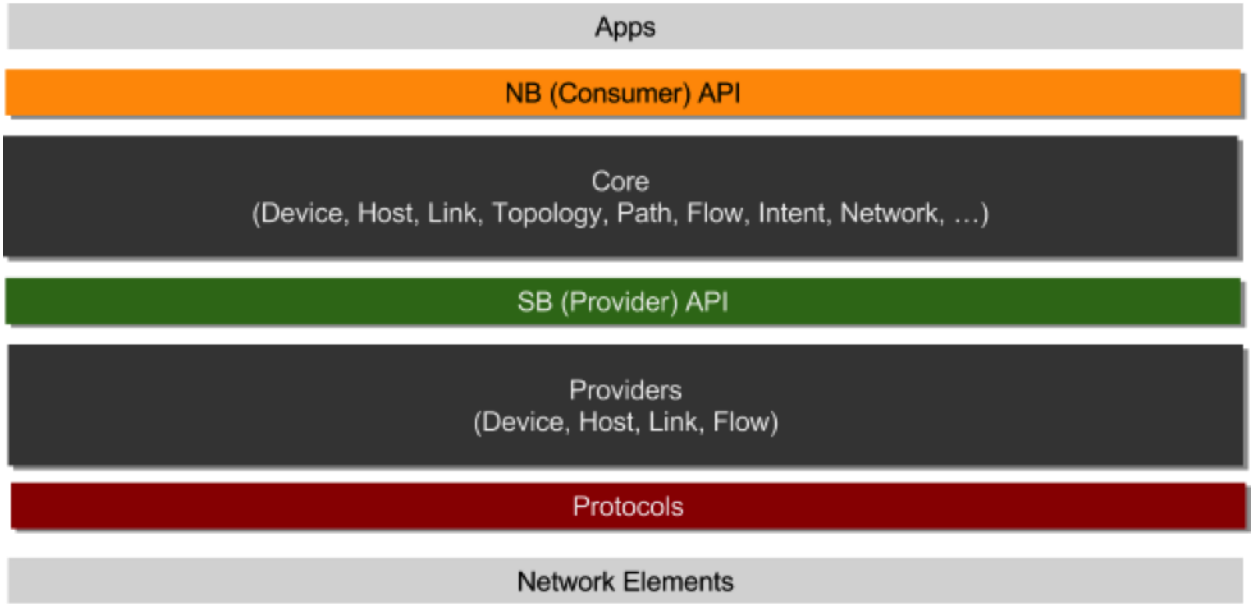
Ο πυρήνας του ONOS

Παρόλο που ο πυρήνας του ONOS ορισμένες φορές απεικονίζεται ως ένα και μόνο στοιχείο, στην πραγματικότητα πρόκειται για έναν μηχανισμό ανεξάρτητων υποσυστημάτων, καθένα υπεύθυνο για διαχείριση της δικής του κλάσης και κατάστασης. Αυτή η δομή, έχει εσωτερική επικοινωνία έτσι ώστε να εκμεταλλεύεται κάθε στοιχείο τις υπηρεσίες των άλλων υποσυστημάτων και να διαμορφώνουν τελικά το northbound API του πυρήνα του ONOS. Πολλές από αυτές τις υπηρεσίες ωστόσο, δεν έχουν καμία σχέση με τον έλεγχο του δικτύου αλλά εξυπηρετούν λόγους ολοκλήρωσης του ONOS ως λειτουργικού συστήματος.

Για να πραγματοποιηθεί ένα απλό και οικείο μοντέλο προγραμματισμού, κάθε υποσύστημα πυρήνα ακολουθεί το ίδιο μοτίβο σε υψηλό επίπεδο κατασκευής και επίσης τηρείται το ίδιο μοντέλο σχεδιασμού τόσο για το northbound όσο και για το southbound αν είναι δυνατόν interface. Ο σχεδιασμός έχει να κάνει κυρίως με δύο συνιστώσες οι οποίες συνεργάζονται μεταξύ τους. Η μία τέτοια συνιστώσα είναι αυτή της διαχείρισης η οποία ολοκληρώνει τις υπηρεσίες που μπορούν να χρησιμοποιηθούν από τον χρήστη και πρωταρχικός της στόχος είναι ο έλεγχος των northbound και southbound λειτουργιών. Η άλλη είναι η αποθήκευση, στην οποία το κομμάτι της διαχείρισης αναθέτει την ευθύνη της παρακολούθησης των ροών ανατολικά και δυτικά, ακόμη και αυτών που αφορούν επικοινωνίες με άλλα συμπλέγματα. Συμβάντα που προκύπτουν οπουδήποτε αλλού, μεταφέρονται στον διαχειριστή ο οποίος αναλαμβάνει είτε να τα αναθέσει στα κατώτερα επίπεδα, είτε εκτελεί κάποια ενέργεια ανάλογα με την φύση του συμβάντος. Τέλος λειτουργίες που απαιτούν περισσότερα δικαιώματα, τις διαχειρίζονται συνήθως ξεχωριστές υπηρεσίες που έχουν ειδικά τέτοια δικαιώματα [10].

3.2 Εξαρτήματα του συστήματος

Όπως εξηγήθηκε νωρίτερα, το ONOS είναι αρχιτεκτονικά δομημένο σε επίπεδα λειτουργιών.

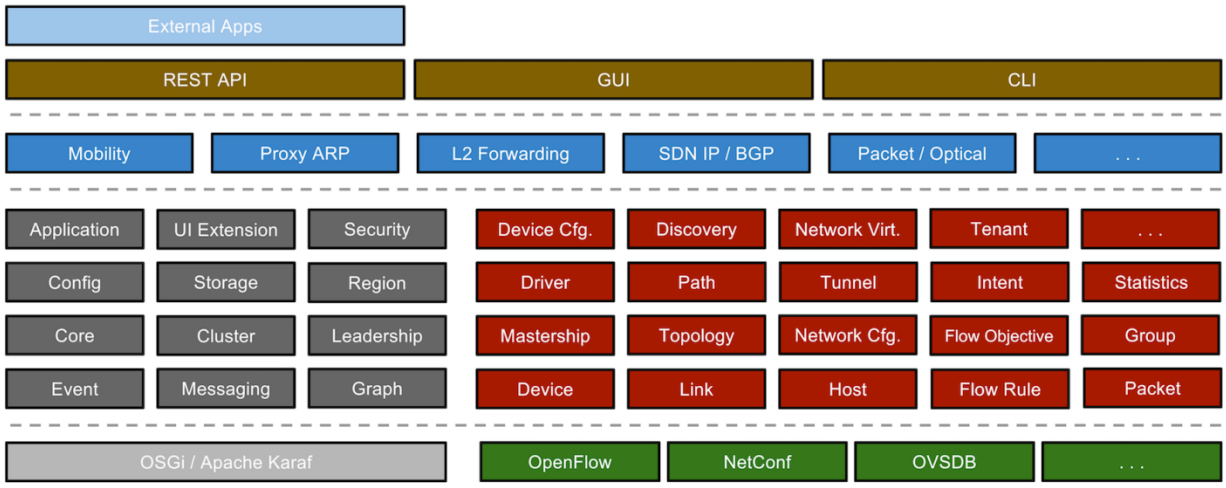


Εικόνα 8: Τα επίπεδα λειτουργιών του ONOS (πηγή [14])

Υποσυστήματα

Υποσύστημα είναι μία μονάδα η οποία αποτελείται από διάφορα στοιχεία, θα λέγαμε κάθετα χωρισμένα στην στοίβα των επιπέδων. Το ONOS ορίζει διάφορα θεμελιώδη τέτοια υποσυστήματα:

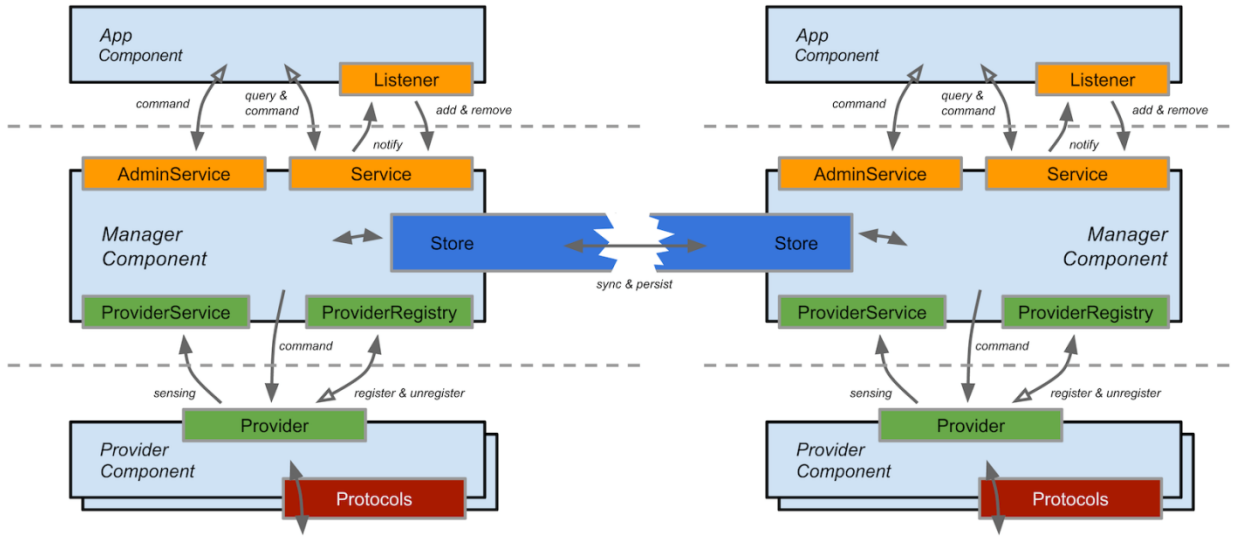
- **Υποσύστημα συσκευών:** Διαχειρίζεται την απογραφή των συσκευών
- **Υποσύστημα συνδέσμων:** Διαχειρίζεται την απογραφή των συνδέσμων
- **Υποσύστημα host:** Διαχειρίζεται την απογραφή των hosts και της τοποθεσίας τους στο δίκτυο
- **Υποσύστημα τοπολογιών:** Διαχειρίζεται στιγμιότυπα δικτυακών τοπολογιών βασισμένα στο χρόνο
- **Υπηρεσία μονοπατιού:** Υπολογίζει/βρίσκει μονοπάτια μεταξύ συσκευών ή hosts βασιζόμενη στο πιο πρόσφατο στιγμιότυπο τοπολογίας
- **Υποσύστημα κανόνων ροών:** Διαχειρίζεται την απογραφή καταγραφών ροών που είναι εγκατεστημένες στις συσκευές και προσφέρει μετρικές
- **Υποσύστημα πακέτων:** Επιτρέπει στις εφαρμογές να δέχονται πακέτα δεδομένων που προέρχονται από τις δικτυακές συσκευές και αντίστοιχα να στέλνουν προς αυτές άλλα πακέτα δεδομένων



Εικόνα 9: Υποσυστήματα που υπάρχουν στο ONOS ή σκοπεύουν να υλοποιηθούν στο μέλλον (πηγή [14])

Δομή υποσυστήματος

Κάθε ένα από τα στοιχεία ενός υποσυστήματος κατοικεί ουσιαστικά σε ένα από τα τρία βασικά επίπεδα (REST API, GUI, CLI) και μπορεί να ταυτοποιηθεί από μία ή περισσότερες διεπαφές java που υλοποιεί.



Εικόνα 10: Σχέσεις στοιχείων υποσυστήματος (πηγή [14])

Το χαμηλότερο επίπεδο του ONOS στη στοίβα επικοινωνεί με το δίκτυο μέσω βιβλιοθηκών σχετισμένα με πρωτόκολλα επικοινωνίας ενώ με τον πυρήνα μέσω του ProviderService interface. Ο εκάστοτε πάροχος και σε συμφωνία με το πρωτόκολλο επικοινωνίας είναι υπεύθυνος να αλληλεπιδρά με το περιβάλλον του δικτύου χρησιμοποιώντας διάφορα πρωτόκολλα διαχείρισης και ελέγχου και να υποστηρίζει υπηρεσίες μεταφοράς δεδομένων στον πυρήνα.

Κάθε πάροχος έχει και ένα δικό του ProviderId. Ο λόγος ύπαρξης αυτού του ProviderId είναι για να διαχωρίζονται οι πάροχοι μεταξύ τους και να καθίσταται σαφές σε συσκευές και άλλα τοπολογικά στοιχεία, από πού εξαρτάται η ύπαρξη τους ακόμα και μετά την αποδέσμευση του παρόχου.Αυτό το ProviderId, εμφωλεύει ένα σχήμα URI ώστε να επιτρέπει στις συσκευές να επικοινωνούν και να αποτελούν ζευγάρι παρόχου-υπηρεσίας ακόμα και αν ανήκει σε άλλη οικογένεια παρόχων.

Ακόμη, ένα υποσύστημα μπορεί να σχετίζεται με περισσότερους από έναν παρόχους. Σε αυτή τη περίπτωση ορίζεται αν ο πάροχος είναι πρωτεύων ή βοηθητικός. Έτσι ο πρωτεύων πάροχος μίας υπηρεσίας έχει στην κυριότητα του τις οντότητες που σχετίζονται με την συγκεκριμένη υπηρεσία, ενώ ο βοηθητικός προσφέρει κάποιες πρόσθετες πληροφορίες. Δίνεται έτσι προτεραιότητα στον κύριο πάροχο και αποφεύγονται συγκρούσεις και ασυμφωνίες.

Για ένα στοιχείο που βρίσκεται στον πυρήνα τώρα, ο Διαχειριστής, συλλέγει πληροφορίες από τους παρόχους και τις μοιράζει στις εφαρμογές και τις άλλες υπηρεσίες. Συνίσταται από:

- Μια northbound διεπαφή υπηρεσίας μέσω της οποίας άλλα συστατικά εξαρτήματα ή εφαρμογές μπορούν να πληροφορηθούν για κάτι συγκεκριμένο σχετικά με την κατάσταση του δικτύου
- Μια διεπαφή AdminService η οποία δέχεται εντολές και τις εφαρμόζει στη κατάσταση του δικτύου ή το σύστημα
- Μια southbound διεπαφή ProviderRegistry μέσω της οποία οι πάροχοι συνδέονται και επικοινωνούν με τον διαχειριστή
- Μια soutbound διεπαφή ProviderService που παρουσιάζεται σε έναν πάροχο , και μέσω της οποίας ο πάροχος μπορεί να στείλει και λάβει πληροφορία από τον διαχειριστή.

Η αναφερθείσα αυτή επικοινωνία μπορεί να γίνεται είτε σύγχρονα με ερωτήσεις προς την υπηρεσία, είτε ασύγχρονα εκμεταλλευόμενη την διεπαφή ListenerService η οποία σχετίζεται με τον κάθε πάροχο και αναμένει για κάποιο συμβάν να έρθει μέσω της EventListener υλοποίησης.

Επιπλέον, μέσα στον πυρήνα και στενά συνδεδεμένο με τον διαχειριστή, είναι και το Κατάστημα που βοηθάει τον διαχειριστή με ευρετήρια αλλά και συγχρονισμό στο να επεξεργάζεται την πληροφορία αλλά ακόμη και να είναι εύκολα προσβάσιμη αυτή η πληροφορία από άλλες οντότητες του συστήματος.

Οι διάφορες εφαρμογές «καταναλώνουν» τις πληροφορίες που προκύπτουν από τον διαχειριστή μέσω των AdminService και Service interfaces. Είναι προφανές ότι οι εφαρμογές έχουν ένα μεγάλο εύρος λειτουργιών από την δημιουργία μονοπατιών μέχρι την αναπαράσταση της τοπολογίας σε έναν φυλλομετρητή.

Αντίστοιχα με τον πάροχο και για αντίστοιχους λόγους, οι εφαρμογές ταυτοποιούνται από ένα ApplicationId. Η ταυτότητα χρησιμοποιείται από το ONOS για να ελέγχει πληροφορία που σχετίζεται με την συγκεκριμένη εφαρμογή. Για να αποκτήσουν ένα τέτοιο ID, οι εφαρμογές

εγγράφονται στο μητρώο του CoreService, δίνοντας το όνομα τους που τελικά ακολουθείται από αντίστροφη DNS σημειογραφία (πχ *org.onlab.onos.fwd*).

Συμβάντα και περιγραφές

Δύο θεμελιώδεις μονάδες κατανομής πληροφορίας στο ONOS είναι τα συμβάντα και οι περιγραφές. Αφότου δημιουργούνται είναι αμετάβλητα και σχετίζονται με συγκεκριμένα δικτυακά στοιχεία και ιδέες.

Οι περιγραφές χρησιμοποιούνται για να μεταδώσουν πληροφορία σχετικά με ένα στοιχείο μέσω του Southbound API. Για παράδειγμα, μια περιγραφή HostDescription περιέχει πληροφορία σχετικά με την MAC και την IP address του host, και της τοποθεσίας του στο δίκτυο (VLAN ID, ports…).

Τα συμβάντα τα χρησιμοποιεί ο διαχειριστής για να ειδοποιήσει τους listeners του σχετικά με αλλαγές στο δίκτυο, και το κατάστημα για να ειδοποιήσει τους συμβαλλόμενους με αυτό για ρυθμίσεις. Ένα συμβάν αποτελείται από τον τύπο του συμβάντος και ένα θέμα που δημιουργείται από την οντότητα που το προκαλεί. Για παράδειγμα, ένα DeviceEvent μπορεί να χρησιμοποιηθεί μεταξύ άλλων για να ειδοποιήσει τους Devicelisteners ότι μια συσκευή (θέμα) έχει εντοπιστεί (DEVICE_ADDED), χάθηκε (DEVICE_REMOVED), ή κάποιο συστατικό του έχει αλλαχτεί (DEVICE_UPDATE).

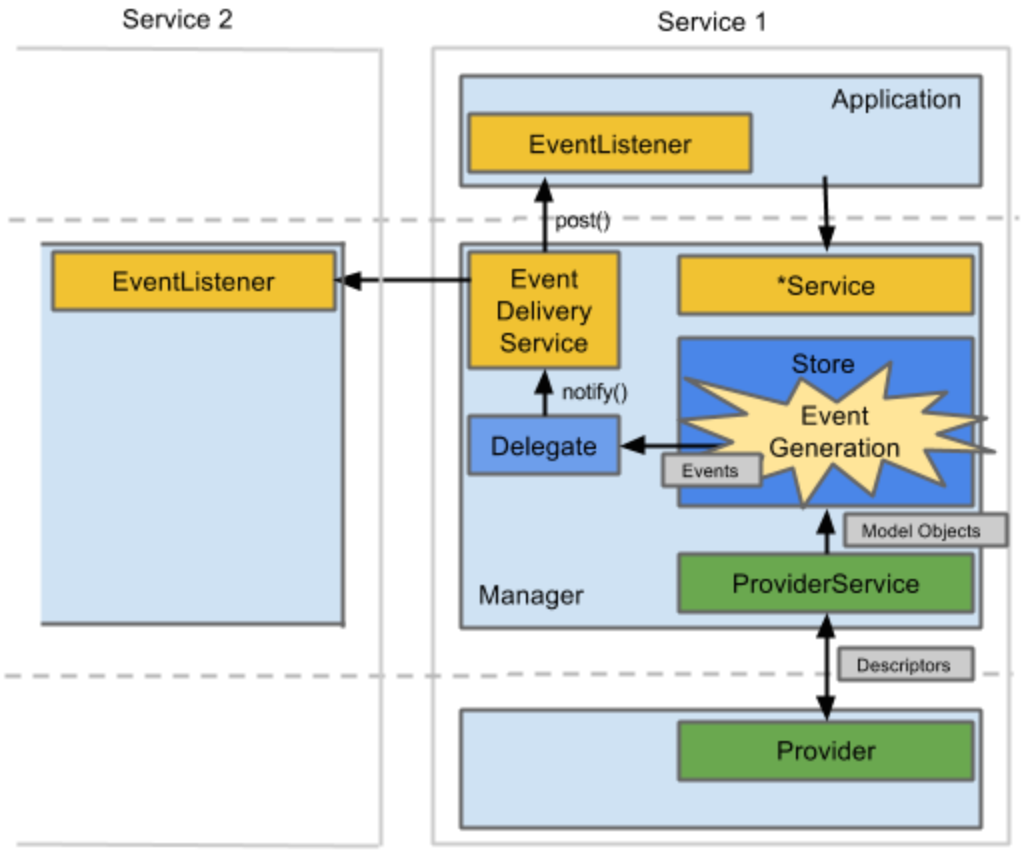
Τα συμβάντα δημιουργούνται από το κατάστημα με πληροφορία που έρχεται από τον διαχειριστή. Μόλις δημιουργηθεί ένα συμβάν, αυτό στέλνεται στους ενδιαφερόμενους listeners μέσω του StoreDelegate interface, το οποίο τελικά μεταφέρεται μέσω του EventDeliveryService. Ουσιαστικά το StoreDelegate μεταφέρει το συμβάν έξω από το κατάστημα ενώ το EventDeliveryService διασφαλίζει ότι θα φτάσει οπωσδήποτε, και μόνο σε αυτούς, του listeners που ενδιαφέρονται.

Οι listeners συμβάντων είναι οποιαδήποτε στοιχεία τα οποία υλοποιούν το EventListener interface. Τα παιδιά της κλάσης ταξινομούνται με βάση τον τύπο του Event στα οποία αυτά ακούν. Η τυπική υλοποίηση ενός listener συμβάντος είναι αυτός να βρίσκεται εσωτερικά μιας κλάσης ενός διαχειριστή ή μιας συσκευής, από τα οποία οι κατάλληλες υπηρεσίες καλούνται βασιζόμενα στο συμβάν [14].

3.3 Οδηγοί συσκευών

Επισκόπηση

Ο βασικός στόχος των οδηγών είναι να απομονώνουν κώδικα που αφορά τις συσκευές έτσι ώστε να μην ξεχύνεται στο υπόλοιπο σύστημα και να το επηρεάζει όσο το δυνατόν λιγότερο. Εφόσον αυτός ο κώδικας θα υπάρχει και στο μέλλον και ενώ το σύστημα του ONOS συνεχώς θα αναπτύσσεται, πρέπει οι οδηγοί να προσφέρουν



Εικόνα 11: Σχέση μεταξύ Συμβάντων, Περιγραφών και στοιχείων (πηγή [14])

τα μέσα που θα επιτρέπουν στις συσκευές να επικοινωνούν με τα υπόλοιπα εξαρτήματα του συστήματος διασφαλίζοντας ότι πάντοτε θα τηρείται ανεξαρτησία από τα πρωτόκολλα σε ανώτερο επίπεδο αλλά και από τα χαρακτηριστικά των συσκευών. Ακόμη, πρέπει να διασφαλίζεται ότι οι οδηγοί θα φορτώνονται σωστά και ασφαλώς για την ευστάθεια του συστήματος, κατά τις διαφορετικές φάσεις που αυτοί χρειάζονται και καλούνται. Τέτοιοι οδηγοί μπορεί να δημιουργούνται τόσο από το ON.Lab και τους συνεργάτες του, όσο και από τους προμηθευτές σύμφωνα με τις ανάγκες τους, όπως ακριβώς δηλαδή πράττουμε σε αυτή την εργασία για την επικοινωνία του ONOS με switches Cisco Ios.

Αναγνωρίζοντας τις εκάστοτε συσκευές παρατηρείται ότι είναι δυνατό και ασφαλώς θεμιτό, οι συσκευές να μοιράζονται υπηρεσίες της ίδιας οικογένειας αλλά και να υποστηρίζουν άλλες που είναι μοναδικές και απευθύνονται μόνο στις συγκεκριμένες συσκευές. Για το λόγο αυτό, η αρχιτεκτονική του ONOS είναι τέτοια ώστε να προωθεί την τμηματοποίηση αυτής της μορφής προσφέροντας στους προγραμματιστές και τους πελάτες να επιλέγουν από μια γκάμα διακριτών

συμπεριφορών που θα χρειαστούν που προκύπτει από σχέσεις κληρονομικότητας, κοινού διαμοιρασμού κλπ.

Δομή

Στο ONOS ένας Driver είναι μια αναπαράσταση μιας συγκεκριμένης οικογένειας συσκευών ή μιας συγκεκριμένης συσκευής. Για το λόγο αυτό έχει ένα μοναδικό όνομα πχ com.ciena.foo, υποστηρίζει ένα σύνολο από συμπεριφορές Behaviour classes, μπορεί να κληρονομεί συμπεριφορές από άλλον οδηγό ή ακόμα και να είναι ένας abstract οδηγός.

Το DriverProvider είναι η οντότητα εκείνη που προσφέρει και κοινωνικοποιεί στα λοιπά στοιχεία του συστήματος τον οδηγό της συσκευής και τις συμπεριφορές της. Από την άλλη το DriverAdminService είναι μια υπηρεσία που παρακολουθεί και διαχειρίζεται έμμεσα τους οδηγούς των συσκευών, κάνοντας ουσιαστικά διαχείριση των παρόχων οδηγών όπως φαίνεται και παρακάτω:

- Set<DriverProvider> getProviders()
- registerProvider(DriverProvider)
- unregisterProvider(DriverProvider)

Σημειώνεται ότι διαφορετικοί πάροχοι μπορούν να συνεισφέρουν με συμπεριφορές στον ίδιο οδηγό. Με αυτό τον τρόπο δηλαδή μπορεί οδηγοί που υποστηρίζουν OpenFlow να προέρχονται από μια πηγή, ενώ για το PoE configuration μια συσκευής να προέρχονται από άλλη.

Οι διάφοροι οδηγοί εντός του συστήματος είναι διακριτοί μέσω χαρακτηριστικών όπως το όνομα του οδηγού, ο κατασκευαστής η έκδοση του υλικού και του λογισμικού της συσκευής, οι συμπεριφορές τη συσκευής, το χαρακατηριστικό ID της συσκευής.

Για την μοντελοποίηση τώρα, διαφορετικές όψεις συμπεριφορών είτε για να περιγράφουν τους οδηγούς είτε να επικοινωνούν απευθείας με αυτούς, μοντελοποιούνται ως παράγωγα των Behaviour και HandlerBehaviour διεπαφών αντίστοιχα. Ο λόγος αυτού του διαχωρισμού είναι εξαιτίας της μικρής αλλά υπαρκτής διαφοράς αυτών των δύο περιεχομένων. Υπάρχουν οι αφαιρέσεις AbstractBehaviour και η AbstractHandlerBehaviour των οποίων οι υλοποιήσεις είναι και το ζητούμενο.

Ο ορισμός του Behaviour χαρακτηρίζεται ουσιαστικά από τις δυνατότητες μιας συσκευής. Οι συμπεριφορές αυτές πρέπει να είναι καλά ορισμένες μέσα στα πλαίσια του θέματος στο οποίο εστιάζουν. Οι υπηρεσίες οδήγησης έχουν την δυνατότητα να βλέπουν τι συμπεριφορές υποστηρίζονται από μια συσκευή. Σημειώνεται κάπου εδώ, ότι παρόλο που η βασική γλώσσα μοντελοποίησης που χρησιμοποιείται στο ONOS είναι η Java, μπορούν να χρησιμοποιηθούν εναλλακτικά άλλες όπως YAML, YANG, SNMP-MIB schema, Loxi, Lua.

Τέλος υπάρχουν δύο ακόμα δομές οι οποίες ολοκληρώνουν τους οδηγούς. Αυτές είναι το DriverData, το οποίο πρόκειται για ένα container όπου περιέχει δεδομένα που έχουν συλλεχθεί

για μία συσκευή μέσα από την αλληλεπίδρασή της με τον οδηγό. Προσφέρει συμπεριφορές για να επικοινωνεί με μία συσκευή και έχει ως γονιό μια κλάση Driver. Η άλλη δομή είναι η DriverHandler το οποίο αποτελεί ένα πλαίσιο επικοινωνίας με την συσκευή. Προσφέρει αντίστοιχα τις κατάλληλες συμπεριφορές για ολοκλήρωση αυτής της επικοινωνίας, εμφωλεύει την προηγούμενη δομή DriverData και επίσης έχει γονιό μια κλάση Driver. Γίνεται αντιληπτό από τα παραπάνω ότι για να λειτουργήσει η επικοινωνία με την συσκευή πρέπει να υπάρχει τουλάχιστον η δομή των DriverData, για να ολοκληρωθεί τελικά μέσα από ένα σύνολο αφαιρέσεων, ενώ μπορεί να γίνει πιο περιεκτική μέσω του DriverHandler ο οποίος πέρα από την πληροφορία που συλλέγει από το DriverData, μπορεί να χρειαστεί επιπλέον για να στήσει επικοινωνία με τις άλλες υπηρεσίες και λειτουργίες μεταξύ του ONOS και της συσκευής [15].

3.4 Southbound: Πρωτόκολλα, πάροχοι, οδηγοί

Νωρίτερα αναφέραμε μερικά πράγματα για τους οδηγούς και τους παρόχους. Θα δούμε στην συνέχεια πως αυτά συνδυάζονται και συνυπάρχουν κάτω από την ομπρέλα ενός πρωτοκόλλου Southbound.

Το ONOS αλληλεπιδρά με το κατώτερο επίπεδο με την βοήθεια των παρόχων. Οι πάροχοι, είναι αυτόνομες εφαρμογές βασισμένες στο OSGi οι οποίες μπορούν να ενεργοποιούνται και να απενεργοποιούνται δυναμικά κατά τον χρόνο εκτέλεσης. Ο βασικός τους στόχος είναι να δημιουργούν ένα επίπεδο αφαίρεσης για την ρύθμιση, τον έλεγχο και την διαχείριση των λειτουργιών μιας οικογένειας συσκευών (πχ Openflow, SNMP, Netconf).

Προσφέρουν τα μέσα για να εκτελούνται αιτήματα που πηγάζουν από τον πυρήνα, όπως install/get κανόνων ροής στο OpenFlow, set/get MIB παράμετρους στο SNMP, set/get για την κατάσταση των ports στο Netconf. Επίσης τα μέσα εκείνα για να γίνεται η διεργασία και να ειδοποιείται ο πυρήνας σχετικά με συμβάντα προερχόμενα από συσκευές.

Αιτήματα από τον πυρήνα πηγάζουν συνήθως από τον διαχειριστή (DeviceManager, PacketManager, LinkManager) ενώ οι πάροχοι ειδοποιούν για διάφορα συμβάντα τον πυρήνα χρησιμοποιώντας το ProviderService API.

Οι υλοποιήσεις των παρόχων χρησιμοποιούνται ουσιαστικά για να γενικεύσουν την λογική των λειτουργιών σε μια οικογένεια συσκευών. Όταν όντως κάτι είναι τόσο συγκεκριμένο που αφορά μόνο μια συσκευή, τότε αυτό υλοποιείται στους οδηγούς και καλείται ξεχωριστά από την ενδιαφερόμενη συσκευή. Από την άλλη, όταν έχουμε αλληλεπίδραση με εξωτερικά συμβάντα, μπορεί να υλοποιείται ένα εξάρτημα στο μόντουλο του πρωτοκόλλου που συνδέει τον πάροχο με κάποιο event listener. Ακόμη μπορούν να υποστηρίζονται και net-cfg συμβάντα, για να μπορούν εξωτερικοί χρήστες να ανακτήσουν πληροφορία χρησιμοποιώντας μηνύματα JSON μορφής όπως port και Ip address του ελεγκτή των συσκευών, των links, και άλλων στοιχείων.

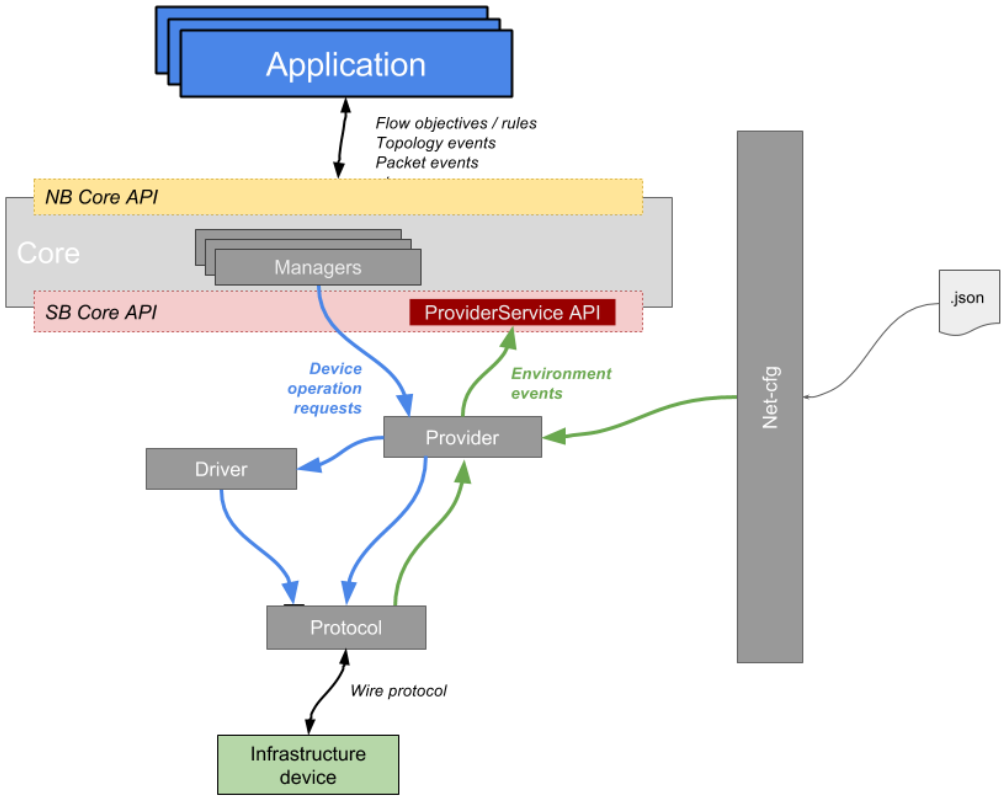
Συνήθως υποστηρίζονται περισσότεροι από ένας πάροχοι για μια οικογένεια συσκευών, κάθε ένας αφιερωμένος να προσφέρει αφαίρεση στον έλεγχο και στην αίσθηση του ευρύτερου περιβάλλοντος πάνω από φυσικές και λογικές δομικές οντότητες όπως συσκευές, links, κανόνες ροών, πακέτα κλπ. Παρατίθενται μερικοί πάροχοι που αφορούν την ίδια οικογένεια συσκευών:

- DeviceProvider: Υπεύθυνος να ειδοποιεί τον πυρήνα μέσω του DeviceProviderService API σχετικά με την ανακάλυψη νέων συσκευών και την περιγραφή τους (όπως κατασκευαστής, μοντέλο, έκδοση, ports κλπ) όσο αυτοί εκθέτουν μεθόδους στο DeviceManager με σκοπό να εκτελέσουν λειτουργίες όπως το να παίρνουν την κατάσταση των ports μια συσκευής (up/down).
- PacketProvider: Υπεύθυνος να ειδοποιεί τον πυρήνα μέσω του PacketProviderService API σχετικά με packet συμβάντα προερχόμενα από το επίπεδο των δεδομένων όταν αυτά εκτίθενται μέσω μεθόδων προς το Packetmanager με σκοπό να παράγουν packet-outs στο επίπεδο των δεδομένων.
- LinkProvider: Υπεύθυνο να ειδοποιεί τον πυρήνα μέσω του LinkProviderService API σχετικά με την ανακάλυψη νέων συνδέσμων χωρίς να απαιτείται κάποια έκθεση μεθόδων αφού δεν θα είχε νόημα να έχουμε μεθόδους για κενές οντότητες όπως αυτές των συνδέσμων.

Τα πρωτόκολλα από την άλλη είναι τμήματα που περιέχονται στον υποφάκελο /protocols του κώδικα του ONOS και μπορούν να εντοπιστούν από το συνήθως χαρακτηριστικό τους όνομα onos-<protocol_name>. Ένα τέτοιο τμήμα περιέχει όλα εκείνα τα χαρακτηριστικά που το καθιστούν ικανό να δομεί την επικοινωνία μεταξύ του ONOS και των συσκευών που υποστηρίζουν και επιλέγουν το συγκεκριμένο πρωτόκολλο. Παραδείγματα του τι θεωρείται πρωτόκολλο στο ONOS είναι τα OpenFlow, Netconf, SNMP, OVSDB κλπ.

Μία συνήθης πρακτική είναι κάθε τέτοιο τμήμα να χωρίζεται στα επιμέρους υποτμήματα του API (με την ονομασία onos-<protocol_name>-api) και του CTL (onos-<protocol_name>-ctl). Το API περιέχει ως επί το πλείστον διεπαφές γραμμένες σε Java και απλές κλάσεις που αναπαριστούν τις θεμελιώδεις οντότητες του πρωτοκόλλου ενώ από την άλλη το CTL περιέχει την συγκεκριμένη υλοποίηση του API. Ο διαχωρισμός API-CTL γίνεται για καθαρατότητα των λειτουργιών και ανεξαρτησία των τμημάτων έτσι ώστε όποιος το απαιτεί, να μπορεί να βασιστεί στο API χωρίς να είναι απαραίτητο να έρθουν και όλες οι υλοποιήσεις CTL του τμήματος.

Το κομμάτι του API, συνήθως περιέχει διεπαφές που αναγάγει σε ένα περισσότερο αφαιρετικό επίπεδο τις λειτουργίες και τα μηνύματα που είναι απαραίτητα για να είναι διαχειρίσιμη μια συσκευή. Για παράδειγμα για ένα δοθέν πρωτόκολλο “Foo” μπορούμε να φανταστούμε ότι έχουμε έναν ελεγκτή FooController και την διεπαφή του, που εκθέτει πληροφορίες σχετικά με την συσκευή στους ενδιαφερόμενους. Μια άλλη διεπαφή μπορεί να είναι το FooSession η οποία φροντίζει για την κατάσταση του κατώτερου επιπέδου μεταφοράς πληροφορίας μέσω μια συνεδρίας με την συσκευή.



Εικόνα 12: Επισκόπηση αρχιτεκτονικής (πάροχοι, οδηγοί, πρωτόκολλα) (πηγή [18])

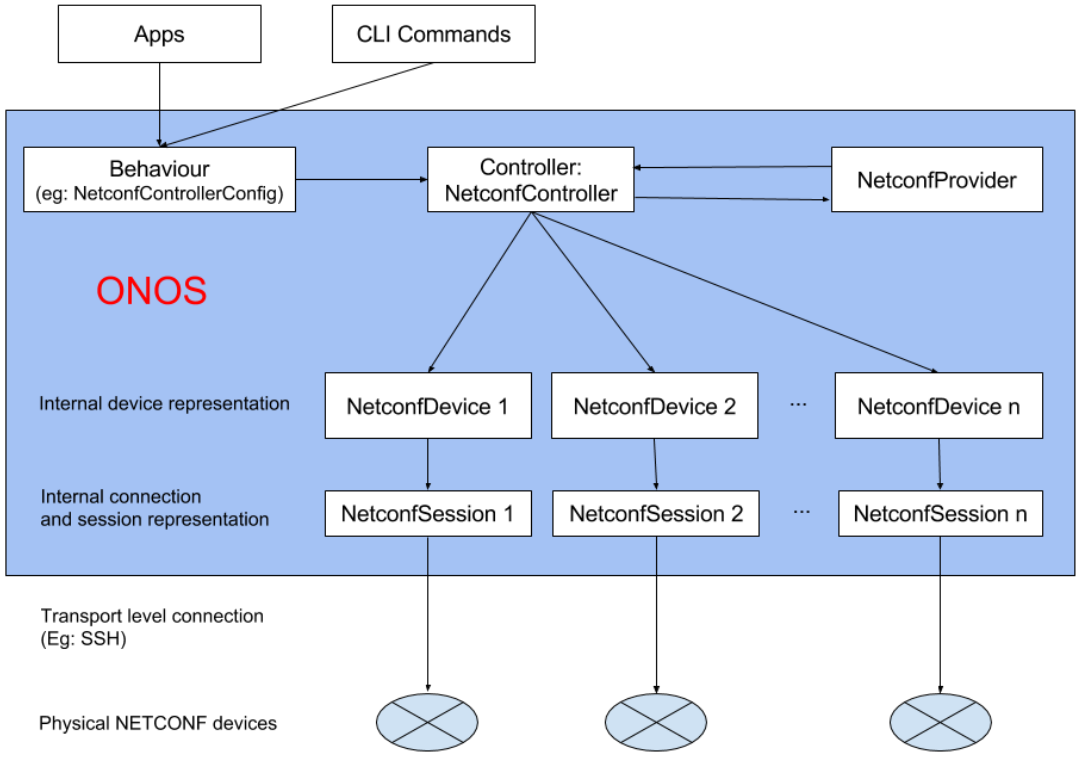
Όπως είπαμε το κομμάτι του CTL, περιέχει τις υλοποιήσεις των API's. Μπορούν να υπάρχουν πολλές υλοποιήσεις για το ίδιο API για να ολοκληρωθούν διαφορετικές λογικές ανάλογα με την χρήση (πχ επίμονες ή μη επίμονες συνεδρίες). Οι κλάσεις που κάνουν implementing τα API's, κάνουν την μετάφραση από το υψηλότερο επίπεδο όπου γίνονται οι κλήσεις σε Java, σε μια ισοδύναμη μορφή από μηνύματα καλωδίων και κλήσεις συστήματος για να επικοινωνήσουν με την συσκευή. Παραδείγματα λειτουργιών που λαμβάνουν χώρα στο τμήμα του CTL είναι η χειραψία, η εγκαθίδρυση επικοινωνίας και η συντήρηση της συσκευής. Αν η συνεδρία είναι επίμονη, για παράδειγμα το TCP στο OpenFlow ή το NETCONF, τότε όλη η πληροφορία για την κατάσταση της σύνδεσης διατηρείται στο υποσύστημα του πρωτοκόλλου, με αυτό να συνεπάγεται ότι στα ανώτερα επίπεδα όπως αυτά των οδηγών ή των παρόχων, δεν χρειάζεται να χάνεται χρόνος για λειτουργίες open/close της σύνδεσης, είτε επανασύνδεσης σε περίπτωση κατάρρευσης κλπ.

Ένα πρωτόκολλο στο ONOS δεν είναι μια εφαρμογή αλλά ένα απλό σύνολο από συστατικά OSGi τα οποία μπορούν να φορτώνονται ή να ξεφορτώνονται μέσω του KARAF όταν απαιτείται. Ο ελεγκτής του πρωτοκόλλου είναι ένα συστατικό OSGi στο οποίο γίνεται αναφορά εξωτερικά του τμήματος πρωτοκόλλου συνήθως από τον οδηγό ή τον πάροχο. Για μια καινούρια συσκευή οι πληροφορίες επαφής και όποια άλλη πληροφορία χρειάζεται, παρέχεται από τον

πάροχο ή προέρχεται από τον πυρήνα. Αλλιώς αν δεν υπάρχει στον πυρήνα προέρχεται από την ίδια την συσκευή [18].

3.5 Το πρωτόκολλο Netconf στο ONOS

Θα εστιάσουμε τώρα στο πρωτόκολλο Netconf στο ONOS για να εντοπίσουμε και να επαληθεύσουμε όσα είδαμε νωρίτερα αλλά και να εξοικειωθούμε με τις λειτουργίες του. Από το επόμενο σχήμα παίρνουμε την γενική ιδέα για την εφαρμογή της θεωρίας που είδαμε προηγουμένως.



Εικόνα 13: Υλοποίηση πρωτοκόλλου Netconf στο ONOS (πηγή [17])

Μπορούμε γρήγορα να παρατηρήσουμε την πρώτη εφαρμογή, ότι κάθε πρωτόκολλο είναι χωρισμένο στο κομμάτι του API και σε αυτό του CTL από τον κώδικα που είναι αποθηκευμένος και ελεύθερος προς τους προγραμματιστές και τους χρήστες, στην τοποθεσία που φιλοξενεί το github *https://github.com/opennetworkinglab/onos*. Πράγματι αν ανοίξουμε τον φάκελο με όνομα protocols και από αυτά επιλέξουμε ένα οποιοδήποτε πρωτόκολλο , αλλά ας πάρουμε απλά το Netconf που μας αφορά κιόλας, θα δούμε ότι σε εκείνο το σημείο υπάρχουν δύο υποφάκελοι.

Ένας αφορά το API και ένας αφορά το CTL. Αν περιηγηθούμε στο API θα δούμε κλάσεις όπως NetconfSession και NetconfController οι οποίες είναι υψηλού αφαιρετικού επιπέδου και ουσιαστικά εμφωλεύουν γενική πληροφορία, ή καλύτερα την βάση πάνω στην οποία θα υλοποιηθεί το ζητούμενο (δηλαδή αντίστοιχα στο NetconfSession και NetconfController). Από την άλλη μεριά, αν περιηγηθούμε στον φάκελο του ctl, εκεί θα δούμε τα ονόματα των αρχείων να έχουν την κατάληξη Impl που προέρχεται από το Implementation και είναι ουσιαστικά οι κλάσεις που υλοποιούν το api. Έτσι για παράδειγμα η κλάση NetconfSessionImpl του ctl υλοποιεί την κλάση NetconfSession του api. Μπορούμε με μια γρήγορα ματία να δούμε τις ίδιες μεθόδους που στο api είναι abstract, εδώ να περιέχουν κώδικα και να κάνουν Override τις μεθόδους αυτές που ορίζονται ρητά μάλιστα από την δήλωση: *public class NetconfSessionImpl implements NetconfSession { body } [17]*.

Μερικές από τις πιο βασικές κλάσεις θα εξηγήσουμε συνοπτικά. Αρχικά ας δούμε την κλάση NetconfCotnroller του api που υλοποιείται από την NetconfControllerImpl του ctl. Σε αυτές τις κλάσεις εντοπίζονται όλες οι συσκευές που υποστηρίζουν το πρωτόκολλο Netconf. Χρησιμεύει ως το μέσο σύνδεσης και λήψης των απαραίτητων διεπαφών για τα διάφορα συμβάντα των συσκευών. Έχει μεθόδους για σύνδεση/αποσύνδεση συσκευής (*NetconfDevice connectDevice(DeviceId deviceId)*, μεθόδους για εντοπισμό συσκευής στον χάρτη της τοπολογίας (*Map<DeviceId, NetconfDevice> getDevicesMap()*), μέθοδο για εύρεση συσκευής μέσω της ταυτότητας του ως κόμβο (*NetconfDevice getNetconfDevice(IpAddress ip, int port)*).

Ακόμη υπάρχει η κλάση NetconfDevice στο api και μια κλάση DefaultNetconfDevice στο ctl. Εδώ αναπαρίσταται μια συσκευή ικανή να επικοινωνήσει μέσω του πρωτοκόλλου Netconf συνδεδεμένη με τον πυρήνα ONOS μέσω ενός δικού του NetconfSession και τις πληροφορίες της αποθηκευμένες σε ένα αντικείμενο NetconfDeviceInfo. Υπάρχει μέθοδος που δείχνει την κατάσταση της συσκευής (*isActive()*) και άλλη που επιστρέφει την συνεδρία (*getSession()*).

Ασφαλώς έχουμε την κλάση NetconfSession στο api και την NetconfSessionImpl στο ctl. Η κλάση NetconfSession αναπαριστά και δείχνει το σημείο επικοινωνίας ανάμεσα στην συσκευή και το ONOS. Ως παράδειγμα φέρουμε την NetconfSessionImpl η οποία χρησιμοποιεί μια SSH2 σύνδεση και συνεδρία για ανταλλαγή πληροφορίας και εκτέλεση λειτουργιών όπως get/set-config με την φυσική συσκευή. Έχουμε πολλές μεθόδους οι οποίες έχουν αντιστοιχία με τις λειτουργίες που πρέπει να υπηρετεί το πρωτόκολλο όπως αυτές αναφέρθηκαν στο προηγούμενο κεφάλαιο. Χαρακτηριστικά έχουμε get(), get-config(), edit(), lock() και άλλες. Στην κλάση NetconfSessionImpl μπορούμε να δούμε τα rpc μηνύματα να διαμορφώνονται κατάλληλα πριν σταλούν στη συσκευή (XML_HEADER ,message-id, get-config, END_PATTERN).

Η κλάση NetconfDeviceInfo περιέχει βασικές πληροφορίες μιας συσκευής όπως Ip address, ports, πρωτόκολλο, όνομα χρήστη και κωδικό, καθώς και το χαρακτηριστικό DeviceId της συσκευής. Βοηθάει στο να επιτυγχάνεται η επικοινωνία χωρίς να χρειάζεται να γίνεται μεταφορά ολόκληρης οντότητας [16].

4. Συσκευές Cisco Ios

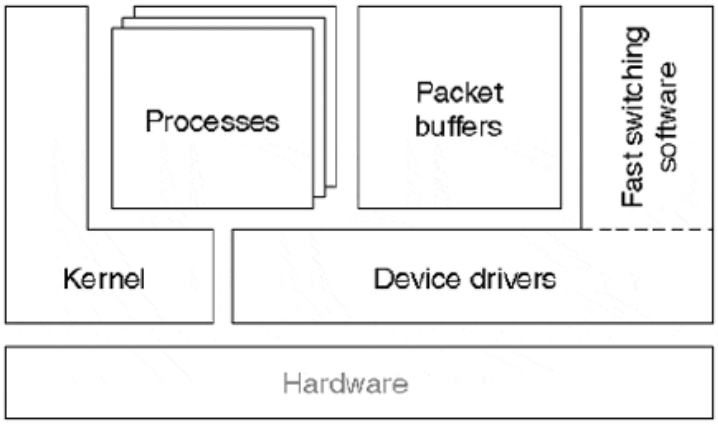
4.1 Γενικά

Επισκόπηση

Το Internetwork Operating System (IOS) είναι ένα multitasking λειτουργικό σύστημα το οποίο χρησιμοποιείται στις περισσότερες Cisco συσκευές (routers/switches). Έχει ένα σύνολο με πεπερασμένο αριθμό εντολών πολλών λέξεων. Το IOS πέρα από τις κλασσικές λειτουργίες που πρέπει να επιτελεί ένα λειτουργικό σύστημα (ανάθεση πόρων, διαχείριση μνήμης, ευελιξία, γρήγορη επικοινωνία, ευστάθεια) έχει και το βάρος να φέρει εις πέρας την μεταγωγή πακέτων γρήγορα και αποδοτικά [22].

Τα διάφορα λειτουργικά IOS παρόλο που στεγάζονται κάτω από την ίδια ονομασία, μπορεί να διαφέρουν από συσκευή σε συσκευή ανάλογα με τις απαιτήσεις και τις ανάγκες που έχει ο χρήστης. Ωστόσο εδώ θα αναφέρουμε μερικά χαρακτηριστικά που συναντάμε κατά κύριο λόγο σε τέτοιες συσκευές για να κατανοήσουμε την δομή, τον τρόπο λειτουργίας, και τις δυνατότητες τους.

Αρχικά το IOS είχε σχεδιαστεί να είναι ένα μικρό εμφωλευμένο σύστημα στους δρομολογητές της Cisco, γιατί εκείνη την εποχή, ο δρομολογητής ήταν περισσότερο το υλικό του ενώ αργότερα η τάση ήταν υλικό και λογισμικό να συνθέτουν μια μοναδική δομή που προσφέρει μεγάλη αποδοτικότητα σε όλα τα επίπεδα λειτουργιών του. Μαζί με τις ανάγκες ήρθε και η υλοποίηση, όταν έπρεπε πλέον οι συσκευές να υποστηρίζουν μια πληθώρα πρωτοκόλλων και άλλων λειτουργιών (πχ bridging).



Εικόνα 14: Αρχιτεκτονική του IOS (πηγή[20])

Πλέον θα λέγαμε ότι το IOS είναι σχεδιασμένο έτσι ώστε να προσφέρει γρήγορη, ασφαλή, αποδοτική μεταγωγή πακέτων μέσα από σύνθετες συσκευές που λειτουργούν ακόμα και με multitasking.

Η αρχιτεκτονική του IOS προκύπτει από πέντε συστατικά στοιχεία όπως φαίνονται και από την Εικόνα 14.

Διεργασίες: Υπάρχουν ατομικές εκχωρήσεις έργων στις διεργασίες για συντήρηση του συστήματος της συσκευής, μεταγωγή πακέτων και υλοποίησης πρωτοκόλλων δρομολόγησης.

Πυρήνας: Είναι ένας κλασσικός πυρήνας λειτουργικού συστήματος που πρέπει να ασχοληθεί και να διευθετεί ζητήματα διαχείρισης της μνήμης όπως ακόμη και να προγραμματίζει τις διεργασίες. Ακόμα είναι υπεύθυνο να προσφέρει δίκαιη κατανομή πόρων στις διεργασίες που τους έχουν ανάγκη.

Buffers Πακέτων: Εδώ υπάρχουν οι buffers που φυλάνε τα πακέτα μεταγωγής.

Οδηγοί συσκευής: Είναι συναρτήσεις που ελέγχουν τις διεπαφές του υλικού.

Λογισμικό Γρήγορης Μεταγωγής: Συναρτήσεις για βελτιστοποιημένη μεταγωγή πακέτων.

Διαμόρφωση συσκευής

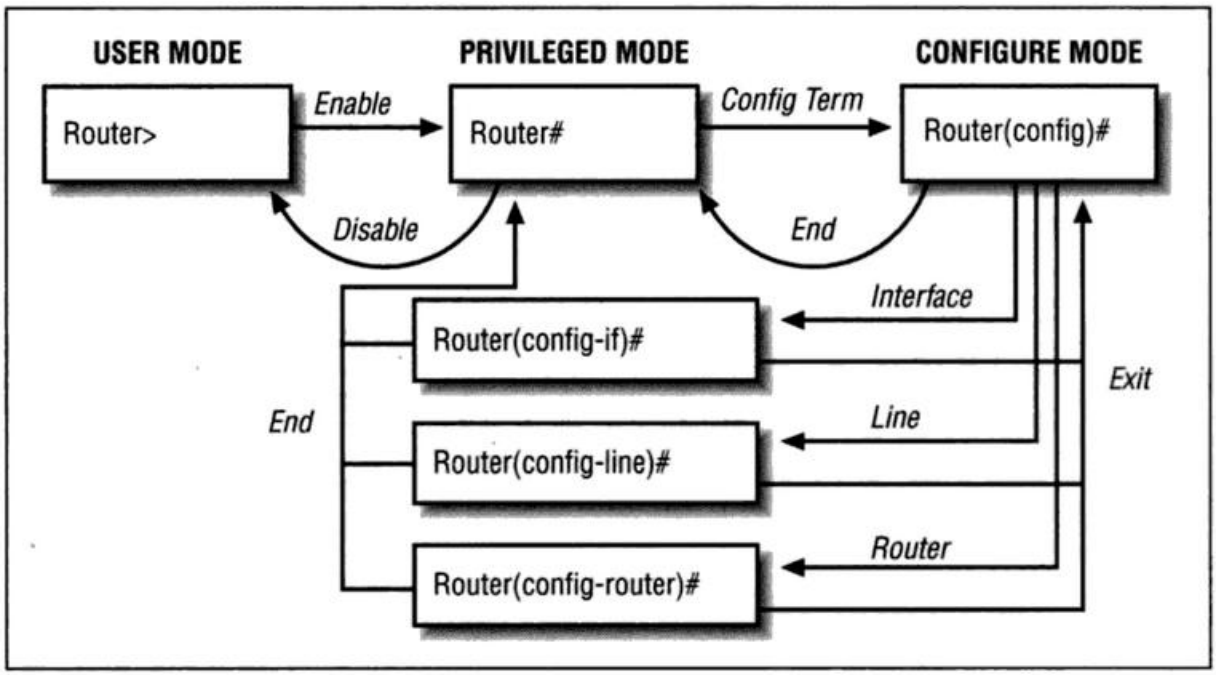
Για την πρόσβαση σε μια συσκευή που υποστηρίζει IOS υπάρχουν τρεις τρόποι.

Ο πρώτος τρόπος είναι μέσω της κονσόλας. Συνήθως χρησιμοποιείται σε καινούριες συσκευές που πρέπει να υποστούν τις αρχικές τους ρυθμίσεις. Σε συσκευές που δεν έχουν ούτε IP address, αλλά ούτε μια υπηρεσία dhcp που θα δώσει στη συσκευή μια IP address και επομένως δεν μπορούμε να τις διαχειριστούμε μέσω του δικτύου. Οι περισσότερες συσκευές Cisco έχουν μια φυσική θύρα για κονσόλα. Μέσω αυτής της θύρας μπορούμε να αποκτήσουμε πρόσβαση στη συγκεκριμένη συσκευή από έναν υπολογιστή μέσω ενός καλωδίου που στο άκρο του πρέπει να έχει κατάληξη RJ-45.

Ο άλλος τρόπος είναι μέσω της υπηρεσίας Telnet. Αυτή η μέθοδος χρησιμοποιείται αρκετά συχνά για πρόσβαση σε δικτυακές συσκευές. Η υπηρεσία Telnet είναι μια υπηρεσία που παρουσιάζει σε μια απομακρυσμένη κονσόλα την κονσόλα της συσκευής και μπορεί να γίνει με αυτό τον τρόπο η ρύθμιση και η απαραίτητη συντήρηση των συσκευών. Για να επιτευχθεί αυτό είναι απαραίτητο η συσκευή να είναι ρυθμισμένη κατάλληλα ώστε να τρέχει ένα Telnet server και να έχει ασφαλώς και μια διεύθυνση IP για επικοινωνία με το υπόλοιπο δίκτυο. Έτσι μέσω της προεπιλεγμένης (αλλά όχι μοναδικής που μπορεί να χρησιμοποιηθεί) θύρας 23, μπορούμε να αποκτήσουμε πρόσβαση και να χειριστούμε την συσκευή, με αυτό όμως να ελλοχεύει κινδύνους γιατί τα δεδομένα στέλνονται μέσω της υπηρεσίας αυτής με τη μορφή απλού κειμένου, συμπεριλαμβανομένων των κωδικών με ό,τι αυτό συνεπάγεται!

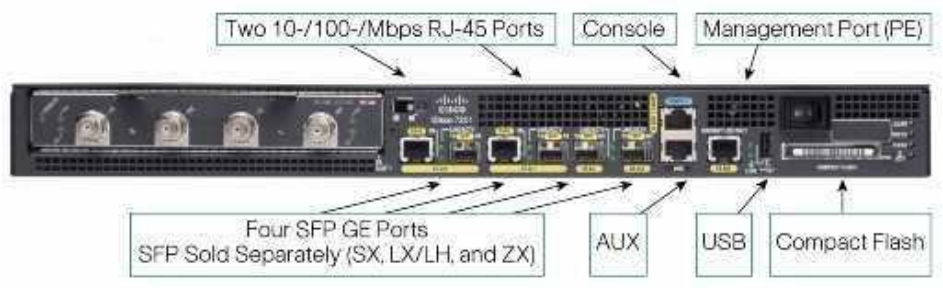
Αντί αυτής, η μέθοδος που χρησιμοποιείται κατά κόρον είναι η τελευταία και είναι η γνωστή υπηρεσία (η πρωτόκολλο αν θέλετε) SSH, η οποία προσφέρει τα πλεονεκτήματα της υπηρεσίας Telnet με επιπλέον προστασία αφού τα δεδομένα περνάνε μέσα από ένα επίπεδο ασφαλείας κωδικοποίησης των πληροφοριών και τελικά μεταδίδονται κρυπτογραφημένα.

Αφού αποκτήσουμε πρόσβαση σε μια συσκευή, τότε έχουμε τρεις κλάσεις λειτουργιών. Έχουμε το user EXEC mode η οποία είναι η προεπιλεγμένη κλάση για εντολές. Είναι το περιβάλλον που βρισκόμαστε μόλις αποκτήσουμε πρόσβαση στην συσκευή και υποστηρίζει μόνο τις πολύ βασικές εντολές (ping,telnet). Στην δεύτερη κλάση, privileged EXEC mode, εισερχόμαστε εκτελόντας την εντολή "*enable"* από το περιβάλλον του χρήστη. Ανάλογα με την ρύθμιση της συσκευής μπορεί να απαιτείται κωδικός και αυτό γιατί με πρόσβαση σε αυτή την κλάση, υπάρχει η δυνατότητα για ρύθμιση της συσκευής. Η τελευταία κλάση είναι η global configuration mode, που είναι μια κλάση που μπορείς να αποκτήσεις πρόσβαση με την εντολή "*configure terminal"*. Χρησιμοποιείται ομοίως για αλλαγή ρυθμίσεων της συσκευής και υποστηρίζει υποκλάσεις για διάφορες άλλες λειτουργίες.



Εικόνα 15: Σχέση μεταξύ Ios command modes (πηγή[20])

Οι συσκευές τύπου Cisco Ios αποθηκεύουν σε ένα αρχείο τις ρυθμίσεις. Μόλις εκτελεστεί μια εντολή τότε εκείνη αυτόματα αποθηκεύεται στο αρχείο που αφορά την τρέχουσα (running) διαμόρφωση. Το αρχείο με την τρέχουσα ρύθμιση είναι αποθηκευμένο στην RAM της συσκευής και για αυτό αν διακοπεί η λειτουργία της, οι ρυθμίσεις χάνονται. Για την αποφυγή αυτού του σεναρίου, και εφόσον είναι το ζητούμενο, η διαμόρφωση πρέπει να αποθηκεύεται και στο αρχείο που κρατάει τις ρυθμίσεις έναρξης (startup). Αυτό το αρχείο είναι ανεξάρτητο με την λειτουργία της συσκευής και είναι η ρύθμιση που έχει αυτή, όταν τίθεται σε λειτουργία.



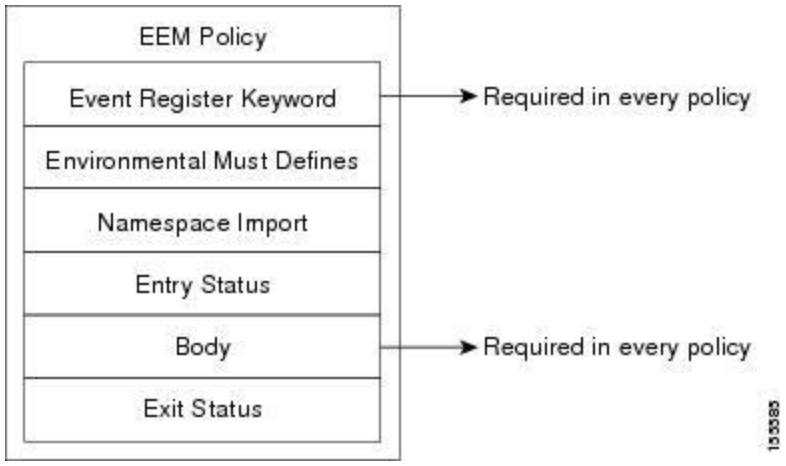
Εικόνα 16: Σασί της συσκευής Cisco 7201 (πηγή[19])

Για τις διάφορες χρήσεις της συσκευής οφείλουμε να της έχουμε δώσει ένα όνομα (hostname). Αφού δώσουμε μια επιθυμητή ονομασία ενεργοποιούμε τις υποστηριζόμενες διεπαφές που από προεπιλογή είναι απενεργοποιημένες με την εντολή "no shutdown" από το περιβάλλον της διεπαφής αυτής που θέλουμε να ενεργοποιήσουμε. Ακόμη αναγκαίο, όπως προαναφέρθηκε, για την επικοινωνία με το δίκτυο, είναι να υπάρχει και μία διεύθυνση IP της συσκευής η οποία μπορεί να δοθεί είτε χειροκίνητα είτε αφού ορίσουμε την κατάλληλη υπηρεσία (dhcp) που δίνει αυτόματα διευθύνσεις. Επίσης μπορούμε εύκολα να δώσουμε κωδικούς για τα διάφορα περιβάλλοντα της συσκευής αλλά και κωδικούς πρόσβασης για απομακρυσμένη υποστήριξη. Ωστόσο σημειώνεται πως για να μπορούμε να συνδεθούμε στην συσκευή απομακρυσμένα, έστω μέσω SSH, πρέπει να έχουμε ενεργοποιήσει την υπηρεσία στην συσκευή για να μπορεί να δεχτεί συνδέσεις [19].

Netconf over SSHv2

Μπορούμε να χρησιμοποιήσουμε NETCONF over SSHv2 για να εκτελέσουμε λειτουργίες μέσω του cisco command-line μέσω ενός κωδικοποιημένου επιπέδου μεταφοράς. Οι υποστηριζόμενες συνεδρίες μπορούν να φτάσουν μέχρι τις 16 ενώ επισημαίνεται ότι υποστηρίζεται μόνο η δεύτερη έκδοση του ssh.

Για να τρέξουμε Netconf over SSHv2, ο client (μια συσκευή Cisco Ios) εγκαθιδρύει μια σύνδεση SSH με τον server (τον διαχειριστή δικτύου Netconf). Ο client και ο server ανταλλάσουν κλειδιά για ασφάλεια και κρυπτογράφηση κωδικών. Ο χαρακτηριστικός αριθμός χρήστη ID και ο κωδικός της SSHv2 συνεδρίας χρησιμοποιούνται για αναγνώριση και ταυτοποίηση. Τότε εισερχόμαστε στο πρώτο επίπεδο της κλάσης του χρήστη με περιορισμένο εύρος δυνατοτήτων. Εάν υπάρχει ρυθμισμένη ταυτοποίηση μέσω του AAA (Authentication, Authorization, Accounting), τότε η υπηρεσία αυτή χρησιμοποιείται όπως θα ήταν αν ο χρήστης συνδεόταν απευθείας στην συσκευή. Αυτός ο τρόπος ασφαλούς σύνδεσης καθιστά την μετάβαση στο πρωτόκολλο Netconf πολύ ομαλή. Από την στιγμή που ο client έχει ταυτοποιηθεί, εγκαθιδρύεται η συνεδρία SSH μεταξύ client και server.



Εικόνα 17: Netconf over SSHv2 (πηγή[20])

Το SSHv2 τρέχει πάνω από ένα αξιόπιστο επίπεδο μεταφοράς και προσφέρει ένα δυνατό πλαίσιο αναγνώρισης και κρυπτογράφησης πληροφοριών. Παρέχει το μέσο για ασφαλή απομακρυσμένη σύνδεση και εκτέλεση εντολών σε μια άλλη συσκευή στο δίκτυο. Με ένα ζεύγος κλειδιών public-private τύπου RSA επιτυγχάνεται η ζητούμενη ασφάλεια.

Διαπιστώνουμε τελικά ότι το Netconf λειτουργεί στις συσκευές Cisco Ios (όταν αυτό υποστηρίζεται) με τον τρίτο τρόπο που είδαμε νωρίτερα. Δηλαδή το Netconf είναι για το ssh ένα ανώτερο αφαιρετικά επίπεδο που στηρίζεται σε αυτό. Για να ενεργοποιήσουμε την λειτουργία σύνδεσης μέσω ssh και άρα και Netconf σε μια συσκευή, πρέπει να συνδεθούμε στην κλάση privileged, και global, να έχουμε ένα όνομα συσκευής hostname και να έχουμε παράξει και ένα ζεύγος κλειδιών. Απαραίτητο επίσης για σύνδεση Netconf είναι να έχει καθοριστεί ότι η συσκευή θα λειτουργήσει με ssh2 αλλιώς η σύνδεση μας θα είναι ασταθής (ανέφικτη).

Μόλις γίνει η κατάλληλη διαμόρφωση στην συσκευή και για να ξεκινήσει μια συνεδρία netconf, αρκεί από την κονσόλα ενός Unix ή Unix-like συστήματος να εκτελέσουμε μια εντολή τύπου «*ssh -2 -s user@router.example.com netconf*» όπου router.example.com είναι ένα domain που έχουμε δώσει στην συσκευή αλλιώς η διεύθυνση IP.

Πέρα από τις απαραίτητες ρυθμίσεις για την βασική σύνδεση, μπορούμε να κάνουμε και άλλες τροποποιήσεις για πιο ειδικές περιπτώσεις, έτσι ώστε να φέρουμε την συσκευή, και γενικότερα την σύνδεση στα μέτρα μας. Πχ μπορούμε να ελέγξουμε το μέγιστο πλήθος συνεδριών που θα μπορούν να υπάρχουν ταυτόχρονα στην συσκευή χωρίς αυτό βέβαια να υπερβαίνει τις 16 όπως είδαμε νωρίτερα σε θεωρητικό επίπεδο (4-16 συνεδρίες υποστηρίζονται).

Εντολές

Όλα τα παραπάνω, είναι εύκολα να διαμορφωθούν μέσω της γλώσσας που υποστηρίζεται στις συσκευές Cisco Ios. Μετά από μια σύντομη τριβή με αυτή, ο οποιοσδήποτε είναι σε θέση να κατανοήσει τον τρόπο που παράγονται εντολές και να δομήσει αυτές που θα του δώσουν την

πληροφορία που χρειάζεται μέσα από ένα εύχρηστο συντακτικό αλλά και ένα πολύ σαφές επεξηγηματικό πλαίσιο των εντολών.

Δεν θα μπορούσαμε να ξεκινήσουμε με τίποτα άλλο μια τέτοια περιγραφή, παρά με το ερωτηματικό «?» το οποίο μας βοηθάει ουσιαστικά σταδιακά να γνωρίσουμε την γλώσσα. Είναι ένα εργαλείο το οποίο μας βοηθάει να συντάξουμε την εντολή που χρειαζόμαστε. Πατώντας το ερωτηματικό στην κονσόλα, εμφανίζεται μια λίστα με τις εντολές που μπορούμε να εκτελέσουμε στο συγκεκριμένο περιβάλλον που ανήκουμε. Μάλιστα για κάθε εντολή της λίστας υπάρχει και μια συνοπτική αλλά περιεκτική επεξήγηση της εντολής. Αλλά δεν προσφέρει μόνο αυτό. Κατά την διάρκεια της συγγραφής της εντολής μπορούμε να χτυπήσουμε το ερωτηματικό για να μας δώσει ποιες εντολές μπορούν να συνδυαστούν με αυτή που έχει γραφτεί στην κονσόλα μέχρι εκείνη την ώρα, βοηθώντας μας έτσι να οπτικοποιούμε ένα δέντρο εντολών και υποεντολών. Ένα απλό παράδειγμα το κάνει πολύ ξεκάθαρο. Πατώντας τα δύο πρώτα γράμματα της εντολής "copy" ακολουθούμενα από ένα ερωτηματικό, εμφανίζεται μια λίστα με τις λέξεις που αρχίζουν με τα γράμμα "co" και την σχετική επεξήγηση δεξιά. Αφού συνεχίσουμε την συγγραφή με τα δύο επόμενα γράμματα και πατώντας ξανά το ερωτηματικό, τότε εμφανίζεται μια νέα λίστα με τις εντολές που μπορούν να συνδυαστούν με την λέξη "copy" και εκεί εμείς διαλέγουμε την "running-config", ενώ στην συνέχεια πατώντας ερωτηματικό μπορούμε να δούμε ότι εμφανίζεται η ίδια λίστα για να δοθεί η δεύτερη παράμετρος της εντολής copy. Σε αυτό το σημείο επιλέγοντας να ολοκληρώσουμε την εντολή με την εντολή "startup-config" και πατώντας το πλήκτρο καταχώρησης "Enter" δίνουμε σήμα ώστε να εκτελεστεί η συγκεκριμένη λειτουργία αντιγραφής της τρέχουσας διαμόρφωσης στην αρχική διαμόρφωση [22].

Έχοντας κάνει το πρώτο βήμα για την κατανόηση της γλώσσας, δεν μένει να συνεχίσουμε με την εντολή "show" η οποία σε συνδυασμό με το ερωτηματικό, μας δίνει την δυνατότητα να συλλέξουμε πληροφορία για την συσκευή και τις δυνατότητες της. Με την εντολή "show running-config" μπορούμε να δούμε το αρχείο της τρέχουσας διαμόρφωσης και τις ρυθμίσεις της, ενώ αν θέλουμε να πάρουμε πληροφορίες για κάποιο συγκεκριμένο interface της συσκευής μπορούμε να χτυπήσουμε την εντολή "show interface?" για να πάρουμε τα υπάρχοντα interfaces, και συνεχίζοντας την εντολή με αυτό που μας ενδιαφέρει, να πάρουμε τελικά την πληροφορία για αυτό. Εδώ σημειώνεται πως η Cisco χρησιμοποιεί τον όρο "interface" για να αναφερθεί στις θύρες μιας Ios συσκευής. Τα interfaces αυτά μπορούν να διαμορφωθούν με διαφορετικές ρυθμίσεις ανάλογα με τον τύπο του interface και ανάλογα με τον τύπο της συσκευής (router, switch). Με την εντολή "*show ip int brief"* μπορούμε να δούμε το σύνολο των interfaces καθώς και άλλες πληροφορίες όπως κατάσταση (up/down) ή διεύθυνση Ip address.

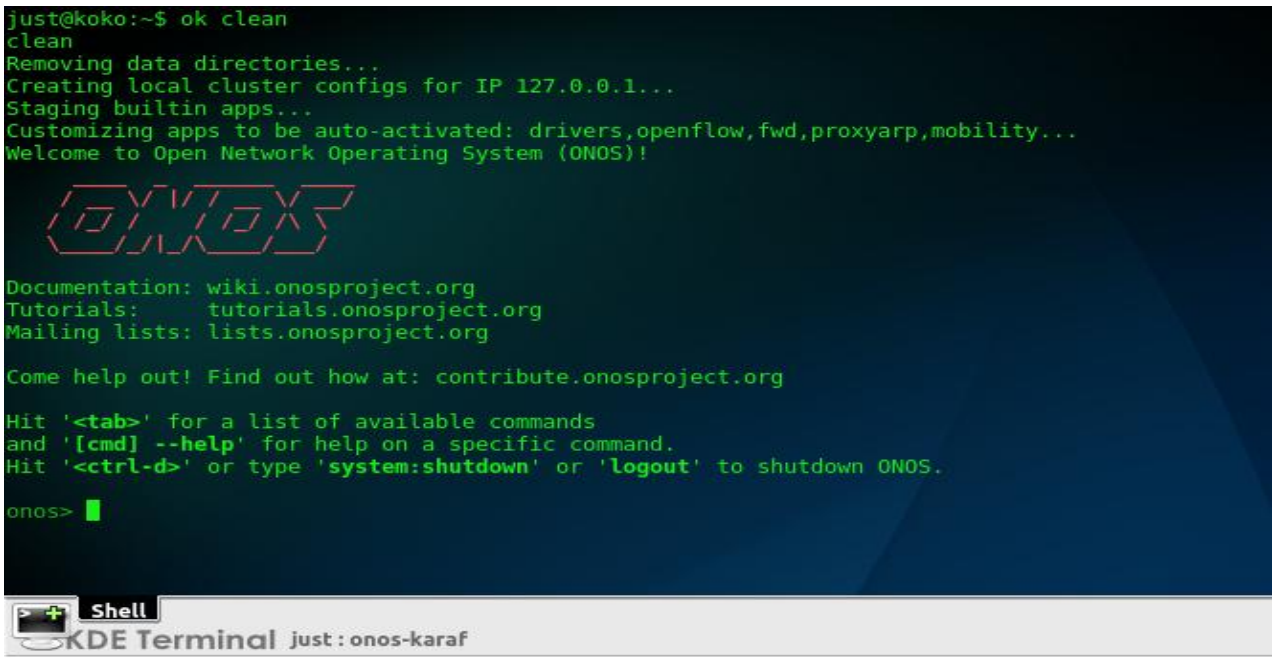
Με την εντολή "show version" παίρνουμε πληροφορία για το υλικό αλλά και το λογισμικό της συσκευής, ενώ αν συνδυαστεί με άλλες υποεντολές μπορούμε να δούμε τι είναι εγκατεστημένο "show version installed" ή τι τρέχει αυτή τη στιγμή "show version running". Ακόμη με την εντολή "show protocols" μπορούμε να δούμε ποια πρωτόκολλα υποστηρίζονται καθώς και την κατάστασή τους.

Σε αυτό το σημείο θα σταματήσουμε, γιατί είναι δύσκολο να αναφερθούμε παραπάνω στο Internetwork Operating System χωρίς να πλατιάσουμε. Άλλωστε εμείς μελετάμε το πώς επιτυγχάνεται η επικοινωνία του ONOS με συσκευές τύπου Cisco Ios με το πρωτόκολλο NETCONF χωρίς να μας ενδιαφέρει τι μπορεί να συμβαίνει μέσα στη συσκευή Ios. Μπορούμε να φανταστούμε τι λειτουργίες επιτελεί, και να βεβαιωθούμε για αυτές διαβάζοντας τα αντίστοιχα manuals των συσκευών που μας ενδιαφέρουν. Αφού πετύχουμε την συγκεκριμένη υλοποίηση τότε ο χρήστης θα πρέπει να ενδιαφερθεί και να μάθει τι συμβαίνει γύρω από αυτές τις συσκευές για να μπορέσει να τις διαχειριστεί κατάλληλα και αποδοτικά.

5. Υλοποίηση Επικοινωνίας Cisco Ios-ONOS

5.1 Εξοικείωση με το περιβάλλον εργασίας

Μετά από κάποια γενικά ζητήματα και αφού χτίστηκε το απαραίτητο θεωρητικό υπόβαθρο, είμαστε έτοιμοι για την ολοκλήρωση του στόχου μας. Συγκεκριμένα, θέλουμε να πετύχουμε την επικοινωνία του ONOS με συσκευές Cisco Ios σύμφωνα με το πρωτόκολλο Netconf, που λειτουργεί με σύνδεση SSHv2, και χρησιμοποιώντας τα απαραίτητα εργαλεία που μας προσφέρει το ONOS. Όταν ξεκίνησε η εκπόνηση της παρούσας εργασίας, το ONOS προσέφερε για την συγκεκριμένη υλοποίηση μια άλλη συμπεριφορά που στην πορεία λόγω αναβάθμισης και εξέλιξης του ONOS άλλαξε, οπότε έπρεπε να αλλαχτεί και να προσαρμοστεί στα νέα δεδομένα και η υλοποίηση που είχε γίνει μέχρι τότε. Για αυτό το λόγο θα γίνει και μια αναφορά σε αυτό το κομμάτι παρόλο που εντέλει δεν μας αφορά. Όλα τα αρχεία της υλοποίησης είναι προσβάσιμα στην διεύθυνση *https://gerrit.onosproject.org/#/c/10269* (ενώ για το το παλιό https://gerrit.onosproject.org/#/c/9499) αλλά και στους αντίστοιχους φακέλους του onos στο github (https://github.com/opennetworkinglab/onos) καθώς και στο Παράρτημα Α στο τέλος του βιβλίου.

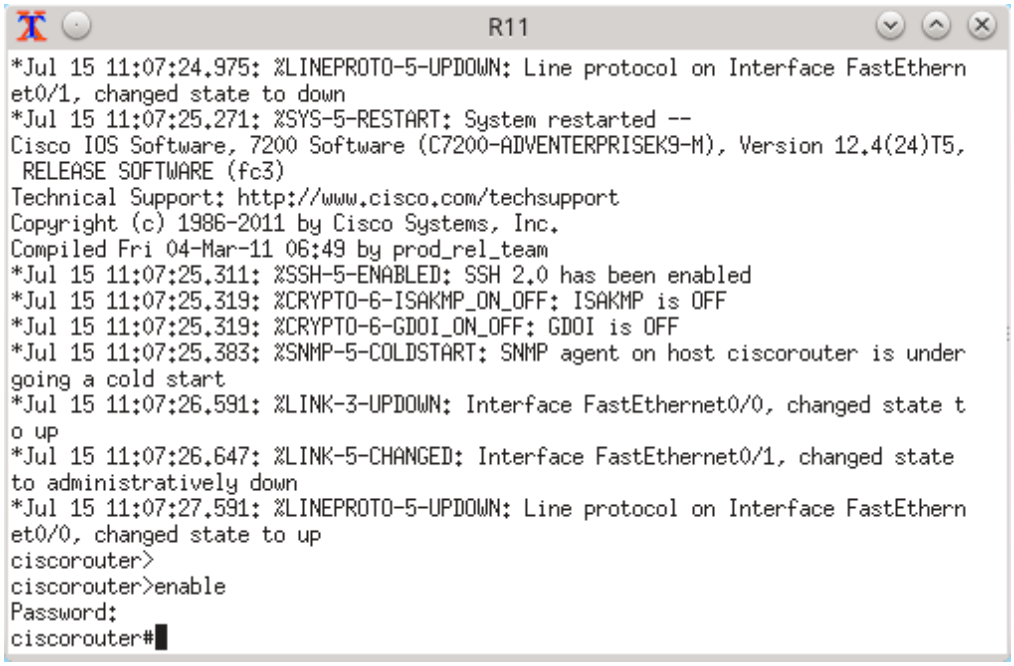


Εικόνα 18: ONOS cli

Για την εγκατάσταση του ONOS, ακολουθήσαμε τις οδηγίες που εμφανίζονται στον σύνδεσμο https://wiki.onosproject.org/display/ONOS/Installing+and+Running+ONOS με τις κατάλληλες

προσαρμογές ώστε τελικά να είναι λειτουργικό τοπικά σε ένα φυσικό (πραγματικό) μηχάνημα με λειτουργικό σύστημα Linux (Ubuntu 14.04). Έτσι τελικά ανοίγοντας terminal και καταχωρώντας την εντολή "onos-karaf clean" (ή συντομότερα "ok clean") και μετά από μερικά δευτερόλεπτα, έχουμε μπροστά μας ανοιχτή την κονσόλα του ONOS έτοιμη να δεχτεί εντολές από εμάς (Εικόνα 18).

Χτυπώντας την εντολή "devices" θα διαπιστώσουμε ότι δεν έχουμε καμία συσκευή υπό τον έλεγχο του Ελεγκτή του ONOS ενώ αν έχουμε αυτές εμφανίζονται στην κονσόλα. Επιπλέον μπορούμε να δοκιμάσουμε την λειτουργία του ONOS και με άλλες εντολές και να πάρουμε πληροφορίες, ενώ τονίζεται ότι για την εργασία μας οφείλεται να έχουμε ενεργοποιημένες τις κατάλληλες εφαρμογές. Μπορούμε να ελέγξουμε ποιες είναι ενεργοποιημένες με την εντολή "apps –a –s" η οποία εμφανίζει όλες τι εφαρμογές που είναι ενεργοποιημένες. Φροντίζουμε να ενεργοποιήσουμε το πρωτόκολλο Netconf με την εντολή "app activate org.onosproject.netconf" για να είμαστε όσο τον δυνατόν πιο έτοιμοι για την σύνδεση της υλοποίησης που θα γίνει με το API που προσφέρεται από το ONOS ενώ αργότερα θα χρειαστεί να ενεργοποιήσουμε και τον οδηγό της cisco που υλοποιήσαμε. Για περισσότερες εντολές στο περιβάλλον του ONOS ο χρήστης αρκεί να χτυπήσει το πλήκτρο TAB και τότε θα του εμφανιστούν στην κονσόλα όλες οι δυνατές επιλογές.

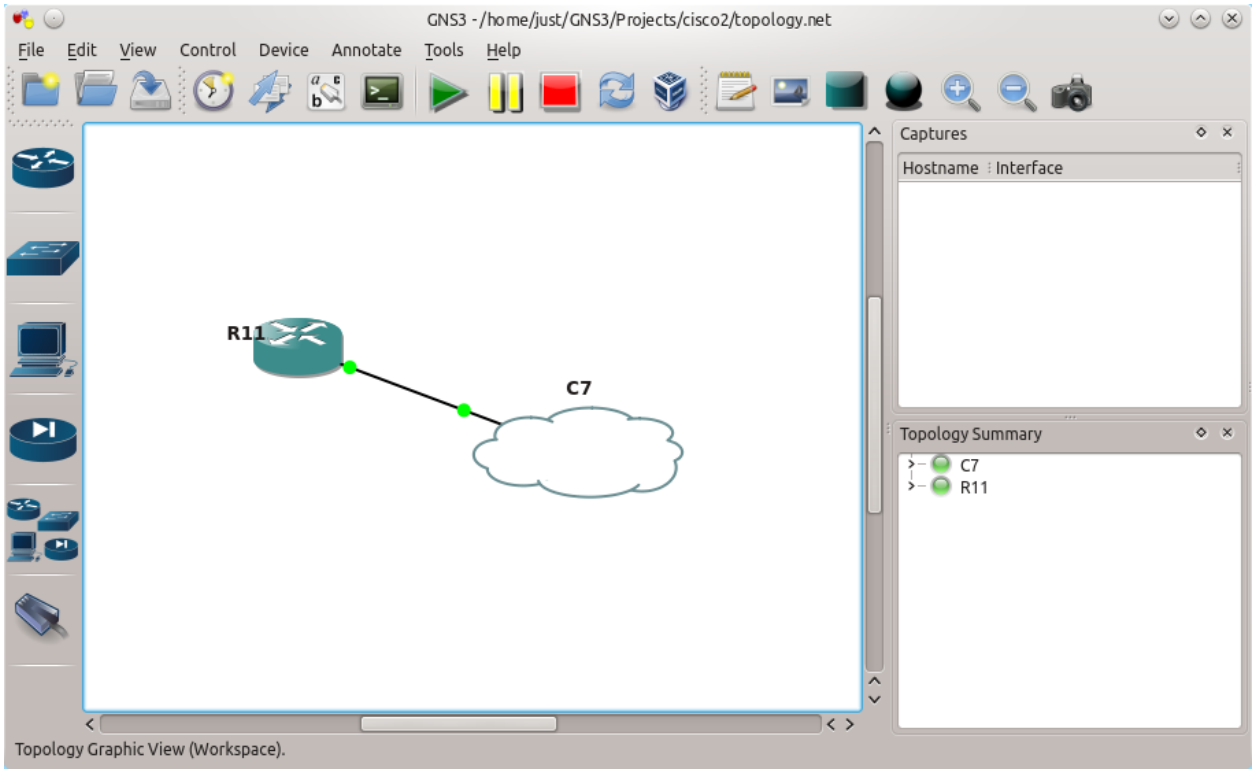


Εικόνα 19: Κονσόλα Cisco Ios συσκευής

Όσον αφορά τις συσκευές Cisco Ios, εργαστήκαμε με διάφορα Cisco Ios switches που διατίθενται δωρεάν από την επίσημη σελίδα της cisco(http://www.cisco.com/). Με το image της συσκευής C3560E κατασκευάστηκε ένα εικονικό switch των οποίων οι πληροφορίες φαίνονται στο αρχείο που χρησιμοποιείται και ως είσοδος για τα onos tests στο Παράρτημα Α με την ονομασία testShowVersion.xml. Για να το πετύχουμε αυτό χρησιμοποιήσαμε το εργαλείο gns3

το οποίο διανέμεται δωρεάν και εγκαθίστανται σύμφωνα με τις οδηγίες από την επίσημη ιστοσελίδα του εργαλείου https://www.gns3.com/. Αφού το δημιουργήσαμε εικονικά μπορούμε να επιβεβαιώσουμε την λειτουργία του χτυπώντας διπλό κλικ στην τοπολογία που φαίνεται στο user interface του gns3, και ανοίγοντας την κονσόλα της συσκευής για απευθείας επικοινωνία με αυτό. Σημειώνεται πως για την πρώτη λειτουργία χρειάζεται η εγκατάσταση που γίνεται εύκολα ακολουθώντας τις οδηγίες που παρουσιάζονται στην κονσόλα και καθορίζοντας κάποιους βασικούς παράγοντες (μνήμη, όνομα συσκευής, κωδικοί).

Ωστόσο, λόγω της δομής του gns3, η συσκευή για να είναι ορατή από το σύστημα μας πρέπει να γίνουν κάποιες επιπλέον ενέργειες ώστε να έχουμε μια εικονική τοπολογία (φυσικά με το πραγματικό μας μηχάνημα) των οποίων τα στοιχεία θα επικοινωνούν (υπολογιστής –συσκευή). Αυτό έγινε εφικτό αφού δημιουργήσαμε ένα εικονικό interface στο πραγματικό υπολογιστή το οποίο λειτούργησε σαν γέφυρα για την επικοινωνία με την συσκευή που στεγάζεται εντός του gns3. Στο gns3 για την ολοκλήρωση της επικοινωνίας υπολογιστή-συσκευής δημιουργήσαμε και μια συσκευή που αντιπροσωπεύει το πραγματικό μας μηχάνημα και συνδέεται με την συσκευή μέσω του εικονικού interface. Η τοπολογία τελικά είναι πολύ απλή και μπορεί να φανεί ξεκάθαρα στην Εικόνα 20.



Εικόνα 20: Εικονική τοπολογία υπολογιστή-συσκευής στο gns3

Συνοψίζουμε την διαδικασία που ακολουθείται για όλο το παραπάνω εγχείρημα στα εξής βήματα.

1. Δημιουργούμε την συσκευή από το image εντός του gns3.
2. Δημιουργούμε την εικόνα του μηχανήματος εντός του gns3.
3. Δημιουργούμε το interface που χρειάζεται για την επικοινωνία υπολογιστή-συσκευής.
4. Μέσω του user-interface του gns3 «τραβάμε» ένα καλώδιο από τον υπολογιστή μας στην συσκευή, από το εικονικό interface γέφυρα προς ένα interface(θύρα) της συσκευής.

Τελικά μπορούμε να επιβεβαιώσουμε την τοπολογία που μόλις χτίσαμε κάνοντας ping στην IP διεύθυνση της συσκευής από τον υπολογιστή (terminal) ή ανάποδα. Μπορούμε να προχωρήσουμε και παραπέρα, ορίζοντας τις κατάλληλες παραμέτρους ώστε το router να υποστηρίζει συνδέσεις Netconf. Φυσικά αυτό σημαίνει να έχουμε ορίσει και τις κατάλληλες λειτουργίες για να δέχεται συνδέσεις ssh2. Αυτό μπορεί να γίνει εκτελώντας τις παρακάτω εντολές εντός της κονσόλας της συσκευής (επεξήγηση εντολές δίπλα):

1. enable (ενεργοποίηση του privileged EXEC mode)
2. configure terminal (είσοδος στο global configuration mode)
3. hostname *hostname* (ρύθμιση ενός ονόματος host για τη συσκευή)
4. ip domain-name *name* (ρύθμιση domain name για τη συσκευή)
5. crypto key generate rsa (ενεργοποίηση του ssh server για τοπική απομακρυσμένη αναγνώριση)
6. ip ssh version 2 (Καθορίζει την έκδοση του ssh στην δεύτερη)

Τελικά είμαστε έτοιμοι για επιβεβαίωση και της Netconf σύνδεσης μέσω του υπολογιστή μας. Αφού έχουμε σε λειτουργία τη συσκευή Cisco εκτελούμε από το terminal του υπολογιστή μας την εντολή "ssh -2 -s user@router.example.com netconf" όπου user είναι το όνομα του χρήστη που θέλουμε να συνδεθούμε στην συσκευή ενώ όπου router.example.com είτε το domain name είτε απευθείας η IP διεύθυνση (αυτή που εμφανίζεται στο gns3). Τότε θα πρέπει γίνεται η σύνδεση μέσω του πρωτοκόλλου netconf από τον υπολογιστή μας στην συσκευή και στέλνοντας μηνύματα "hello" είτε άλλα όπως get-config θα πρέπει να παίρνουμε τις αντίστοιχες απαντήσεις. Πράγματι αν μετά από ένα μήνυμα hello με τα capabilities στείλουμε το get-config επιστρέφεται το configuration που έχει γίνει στην συσκευή από εμάς.

Τέλος, για την συγγραφή του κώδικα χρησιμοποιήθηκε το intelliJ IDE της JetBrains. Για πλήρη αποδοτικότητα των εργαλείων του intelliJ σε ένα τόσο μεγάλο project όπως το ONOS, φροντίσαμε να κάνουμε σωστή εγκατάσταση του κώδικα στο IDE σύμφωνα με τις οδηγίες που φαίνονται στον σύνδεσμο των wiki του onosproject: *https://wiki.onosproject.org/display/ONOS/Importing+ONOS+projects+into+IntelliJ+IDEA*.

5.2 Σχεδιασμός της υλοποίησης

Όσον αφορά τον σχεδιασμό της υλοποίησης έπρεπε πρώτα να γίνει κατανόηση της δομής του ONOS και ειδικότερα του κομματιού που αφορά την υλοποίηση γύρω από την επικοινωνία μέσω Netconf. Ξεκινώντας από τις κλάσεις που υλοποιούν τις συσκευές και κρατάνε αντικείμενα με τις πληροφορίες αυτών και λαμβάνοντας υπόψη ότι η επικοινωνία θα συντελείται μέσω Netconf, έγινε καταγραφή ορισμένων κλάσεων που ήταν απαραίτητες για μια υλοποίηση σαν την δική μας. Για την βοήθεια στην κατανόηση του κώδικα, δόθηκε σημασία στο Onos java api που φιλοξενείται στον ιστότοπο *http://api.onosproject.org/1.6.0/* (για την έκδοση 1.6.0) το οποίο προσφέρει σημαντικά στοιχεία για τις συναρτήσεις, τις κλάσεις, τις διεπαφές καθώς και σύντομο σχολιασμό αυτών. Ακόμη μια καλύτερη εικόνα παρέχεται και με την χρήση κάποιου IDE (intelliJ) το οποίο κάνει εύκολη και γρήγορη την αναζήτηση και την περιήγηση στις διάφορες κλάσεις του ONOS.

Ασφαλώς γρήγορα κατανοήθηκε η σημαντικότητα κλάσεων όπως η NetconfController και NetconfSession για αυτό μελετήθηκαν και κατανοήθηκε ο τρόπος που αυτές θα χρησιμοποιηθούν για να οριστεί η επικοινωνία με την συσκευή. Συγκεκριμένα, αφού εδραιωθούν αντικείμενα αυτών των δύο κλάσεων, τότε με ένα μήνυμα προς την συσκευή τύπου rpc θα γίνει αίτηση για κείμενα που περιέχουν την πληροφορία που μας ενδιαφέρει.

Από την άλλη, για το επόμενο βήμα, σημειώθηκε ότι το ONOS χρησιμοποιεί ως συμπεριφορά για την καταγραφή μιας συσκευής και την ταυτοποίηση της στο σύστημα, την DeviceDescriptionDiscovery που είναι μια κλάση που δημιουργεί το αντικείμενο αυτό με τις πληροφορίες τις συσκευής. Ποιες είναι αυτές οι πληροφορίες; Εύκολα μπορούμε να τις εντοπίσουμε και να τις κατηγοριοποιήσουμε σε δύο σκέλη. Αυτό που αφορά γενικές πληροφορίες της συσκευής (Κατασκευαστής, λειτουργικό, έκδοση συσκευής, έκδοση λειτουργικού) και σε αυτό που αφορά πληροφορίες σχετικά με τα interfaces (θύρες) της συσκευής (θύρα, ταχύτητα, κατάσταση). Εδώ πρέπει να αναφερθεί ότι ενώ αυτές στεγάζονται κάτω από την ίδια κλάση και τελικά μπορούν να ανήκουν στο ίδιο αντικείμενο java, παλιότερα δεν ίσχυε, και η αντίστοιχη συμπεριφορά υλοποιούνταν από δύο διαφορετικά αντικείμενα κάτω από την συμπεριφορά του PortDiscovery για τις θύρες και το DeviceDescription ξεχωριστά για τις γενικές λοιπές πληροφορίες.

Αργότερα, αφού έγινε αυτή η καταγραφή των πληροφοριών που μας ενδιαφέρουν, έγινε έρευνα στις συσκευές Cisco Ios, να παρατηρηθεί η δομή των πληροφοριών που προσφέρονται από κάθε μια και να γίνει μια προσπάθεια να βγει ένα μονοπάτι που μπορεί να μας προσφέρει πάντα τις πληροφορίες. Από το σύνολο των εντολών *show* της συσκευής, τελικά καταλήξαμε σε ένα πολύ μικρό υποσύνολο αυτών που περιέχουν σχεδόν όλη την απαραίτητη πληροφορία. Το «σχεδόν» αφορά δύο στοιχεία τα οποία αποφασίστηκε να μην ανήκουν στην υλοποίηση και τελικά να μην εντάσσονται στις λεπτομέρειες μια συσκευής που αποθηκεύει το ONOS για αυτή, γιατί εντοπίστηκαν εξαιρέσεις συσκευών που είτε δεν παρουσίαζαν την συγκεκριμένη πληροφορία

στην εντολή show, είτε την παρουσίαζε με τέτοιο τρόπο, που δεν άξιζε ούτε να ασχοληθούμε για την αναζήτηση της αλλά ούτε ήταν εύκολο να εντοπιστεί από συσκευή σε συσκευή. Καλύπτεται όμως η γενική περίπτωση αν θέλετε των Cisco Ios συσκευών.

Τελικά, έγινε ο σχεδιασμός για το πώς θα ανακτηθεί αυτή η πληροφορία από τις εντολές show οι οποίες θα επιστρέφουν σύμφωνα με τον σχεδιασμό κείμενο. Σημειώθηκαν τα σημεία εκείνα που εμφανίζουν την πληροφορία που χρειάζονται οι κλάσεις που ολοκληρώνουν την DeviceDescriptionDiscovery. Επειδή η ανάκτηση της πληροφορίας θα έπρεπε να γίνει από κείμενο, αποφασίστηκε σε αυτή τη φάση, ποιος θα ήταν ο πιο αξιόπιστος τρόπος να ανακτηθεί η πληροφορία στην γενική περίπτωση με έναν στιβαρό τρόπο που δεν θα οδηγεί ούτε σε σφάλματα αλλά θα είναι και λειτουργικός.

5.3 Υλοποίηση

Το πρώτο πράγμα που έγινε κατά την υλοποίηση ήταν να δημιουργηθεί η βασική κλάση που θα συγκεντρώνει την πληροφορία που χρειάζεται το ONOS μέσω κλήσεων των απαραίτητων συναρτήσεων. Αυτή η κλάση είναι η CiscoIosDeviceDescription και κάνει implementation την DeviceDescriptionDiscovery. Από εκεί και πέρα, δημιουργήθηκε μια ακόμη κλάση, η TextBlockParserCisco, η οποία παίρνοντας το κείμενο που θα επιστρέφει η συσκευή, θα ανακτά την πληροφορία που ενδιαφέρει το ONOS. Οι δύο κλάσεις αυτές φαίνονται στο Παράρτημα Α.

Ο πυρήνας

Για την συγγραφή της CiscoIosDeviceDescription, φτιάξαμε την κλάση, και καταχωρήθηκαν τα πρώτα βασικά imports. Ύστερα, σε αρμονία με την υπερκλάση DeviceDescriptionDiscovery που γίνεται implement φτιάξαμε δύο μεθόδους που κάνουν override αυτές που βρίσκονται στην abstract κλάση DeviceDescription. Αυτές είναι οι discoverDeviceDetails() που επιστρέφει ένα αντικείμενο DeviceDescription και η discoverPortDetails που επιστρέφει μια λίστα με τα ports της συσκευής. Αυτές οι μέθοδοι συλλέγουν όπως αναφέραμε νωρίτερα τις γενικές πληροφορίες της συσκευής και πληροφορίες για τα ports αντίστοιχα. Διαπιστώθηκε ότι από την εντολή “show version” σε μια Cisco Ios συσκευή μπορούμε να πάρουμε όλη τη χρήσιμη πληροφορία που αφορά τις γενικές πληροφορίες ενώ από την άλλη η εντολή “show interfaces” σε μια Cisco Ios συσκευή περιέχει όλη την πληροφορία που είναι απαραίτητη για τις πληροφορίες των ports. Για τον λόγο αυτό έχουμε δύο μεθόδους, την showVersionRequestBuilder και την showInterfacesRequestBuilder η οποίες μέσω της συνάρτησης StringBuilder (που γίνεται και import για να μπορεί να χρησιμοποιηθεί) δημιουργούν το string εκείνο που θα σταλεί προς την συσκευή για να πάρει τις απαντήσεις του “show version” και “show interfaces” μέσω του netconf πρωτοκόλλου. Οι μέθοδοι αυτοί, επιστρέφουν το string αυτό με την κλήση τους. Το μήνυμα προς αποστολή είναι ένα αίτημα get του version και του interfaces αντίστοιχα, όπως είδαμε στο κεφάλαιο 2 για το Netconf και αφορά το running configuration της συσκευής, ενώ ζητάται να επιστραφεί η απάντηση σε text block μορφή.

Η μέθοδος discoverDeviceDetails() αφού χτίσει πρώτα τα αντικείμενα NetconfController και NetconfSession που ολοκληρώνουν ουσιαστικά την Netconf επικοινωνία με την συσκευή (δηλαδή μηνύματα rpc εμφωλεύοντας xml), δοκιμάζει μέσω της εντολής *version = session.get(showVersionRequestBuilder());* να κάνει "get" του αποτελέσματος της show version, εντός ενός try-catch block, ενώ σε περίπτωση αποτυχίας (είτε λόγω μη προσβασιμότητας στην συσκευή, είτε λόγω δικαιωμάτων, είτε εξαιτίας άλλου λόγου) εμφανίζει μήνυμα αποτυχίας μέσω του IOException exception. Από εκεί και πέρα, στην παρούσα φάση, σχεδιάζουμε την κλήση μιας συνάρτησης της άλλης κλάσης που δημιουργήσαμε, και προς το παρόν δεν έχουμε γράψει ακόμη(της TextBlockParserCisco), και την οποία ονομάζουμε *parseCiscoIosDeviceDetails(version)* και όπως βλέπουμε παίρνει ως όρισμα το version δηλαδή το κείμενο εκείνο από όπου θα ανακτηθεί η πληροφορία για την συσκευή. Αυτή θα επιστρέφει έναν πίνακα με όλη την πληροφορία που μας ενδιαφέρει ως strings. Εδώ για να γίνει ξεκάθαρο επισημαίνεται ότι επειδή η πληροφορία του chassisId δεν παρουσιάζεται πάντα ορθά στις συσκευές Cisco Ios δεν την ανακτούμε και για αυτό μπορεί να γίνει η υλοποίηση που αναφέρθηκε παραπάνω με τον πίνακα των Strings. Σε περίπτωση που μπορούσαμε να ανακτήσουμε και το ChassisId τότε θα γινόταν μια διαφορετική υλοποίηση επιστροφής από την κλήση της συνάρτησης ενός αντικειμένου DeviceDescription που θα περιείχε όλη την πληροφορία γιατί δεν θα ήταν εφικτό να επιστραφεί πίνακας με strings από την στιγμή που το chassisId δεν είναι ένα τέτοιο αντικείμενο. Αφού έχουμε χτίσει όλο αυτό το πλαίσιο που θα επικοινωνεί με την συσκευή, θα ζητάει την απάντηση της show version, και θα την στέλνει για επεξεργασία στην parseCiscoIosDeviceDescription της TextBlockParserCisco κλάσης, θα συνεχίσουμε με την δημιουργία του αντικειμένου που πρέπει να επιστραφεί στο ONOS και εξαρτάται από την συμπεριφορά που έχουμε επιλέξει (δηλαδή το DeviceDescription). Αναζητώντας την συγκεκριμένη δομή γύρω από αυτή την κλάση στο javadoc του Onos βλέπουμε ότι η πληροφορία που πρέπει να επιστρέφεται είναι τα:

- chassisId το οποίο επιστρέφει το χαρακτηριστικό αναγνωριστικό του σασί της συσκευής
- deviceUri το Uri της συσκευής που αποτελεί και το χαρακτηριστικό της ανάμεσα στις άλλες συσκευές του Onos
- το hwVersion δηλαδή την έκδοση του υλικού της συσκευής
- το manufacter δηλαδή τον κατασκευαστή της συσκευής (cisco)
- τον serialNumber δηλαδή τον σειριακό αριθμό της συσκευής που δόθηκε σε αυτή από τον κατασκευαστή
- την έκδοση swVersion του λειτουργικού συστήματος (όλες πρόκειται για Ios εκδόσεις)
- τον τύπο Device.Type της συσκευής (εμείς εξετάσαμε switches).

Έτσι τελικά φτιάχνουμε το αντικείμενο αυτό με την τελευταία εντολή της μεθόδου. Σημειώνουμε ότι έχουμε κάνει την σύμβαση (έχουμε γράψει με αυτό τον τρόπο τον κώδικα) ώστε η details[0] να δίνει τον κατασκευαστή, το details[1] να δίνει το hwVersion, το details[2] να δίνει το swVersion και το details[3] να δίνει το serialNumber σύμφωνα με τις μεθόδους της κλάσης DefaultDeviceDescription που δημιουργεί το αντικείμενο που αντιπροσωπεύει την

συσκευή. Σημειώνεται πως για την παράμετρο του deviceUri, και επειδή αυτό δίνεται από το Onos, για την δημιουργία του αντικειμένου με DeviceDescription, και για να μην αλλάξουμε το Uri, προηγούνται της επιστροφής της συνάρτησης τρείς εντολές οι οποίες ουσιαστικά παίρνουν το Uri της συσκευής που έχει δοθεί από το Onos (πχ όταν αυτή δημιουργείται μέσω json).

Η μέθοδος discoverPortDetails() κατ' αντιστοιχία με την discoverDeviceDetails() χτίζει αρχικά τα αντικείμενα NetconfController και NetconfSession για να ολοκληρώσουν και να κάνουν εφικτή την επικοινωνία με την συσκευή κάτω από μια συνεδρία Netconf. Σε αυτή την μέθοδο μέσω της εντολής *interfaces = session.get(showInterfacesRequestBuilder());* Δοκιμάζεται να γίνει ένα rpc "get" του αποτελέσματος της εντολής "show interfaces" εντός (πάλι) ενός try-catch block που σε περίπτωση αποτυχίας επιστροφής της απάντησης εκτελείται η εξαίρεση επιστρέφοντας μήνυμα αποτυχίας. Στην τελευταία εντολή επιστρέφεται η λίστα List<PortDescription> που έχει οριστεί άλλωστε και στην δήλωση της συνάρτησης, αφού πρώτα όμως γίνεται μια κλήση στην μέθοδο parseCiscoIosPorts(interfaces) της κλάσης TextBlockParserCisco. Ως όρισμα αυτή η μέθοδος (που δεν έχουμε υλοποιήσει ακόμα) θέλουμε να παίρνει τα intarfaces που δεν είναι άλλο από ένα string με το κείμενο της απάντησης του "get" σε TextBlock μορφή. Αντίθετα με την μέθοδο discoverDeviceDetails που δημιουργούσε το αντικείμενο, εδώ το αντικείμενο περιμένουμε να έρθει έτοιμο ως επιστροφή από την μέθοδο της κλάσης TextBlockParserCisco, parseCiscoIosPorts(interfaces), ώστε να ολοκληρωθεί τελικά η χειραψία μεταξύ του Onos και της Cisco Ios συσκευής. Σημειώνουμε και πρέπει να το θυμόμαστε κατά την υλοποίηση των αντίστοιχων μεθόδων στην κλαση TextBlockParserCisco, ότι το αντικείμενο PortDescription, χρειάζεται την εξής πληροφορία:

- portNumbeR() τον αριθμό της θύρας
- portSpeed() την ταχύτητα της σύνδεσης υπο την συγκεκριμένη θύρα
- isEnabled() την κατάσταση της θύρας (up/down)
- Port.Type() τον τύπο της θύρας (Ethernet, FastEthernet, Serial)

Μετά από αυτή την επικοινωνία, και ενώ το Onos έχει την απαραίτητη πληροφορία για την συσκευή που θα του επιτρέψει να επικοινωνήσει μαζί του, μπορεί το Onos να εκμεταλλευτεί το πρωτόκολλο Netconf και να διαχειριστεί την συσκευή όπως είδαμε στο 2^{ο} κεφάλαιο.

Συλλογή πληροφορίας

Παρακάτω θα αναλύσουμε την συγγραφή της κλάσης TextBlockParserCisco. Είναι μια κλάση η οποία λειτουργεί διαδικαστικά. Η μια συνάρτηση καλεί κάποιες άλλες και συσσωρεύει πληροφορία. Εντός της κλάσης αυτής, υλοποιείται η ανάκτηση της απαραίτητης πληροφορίας τόσο της μεθόδου discoverPortDetails() όσο και της discoverDeviceDetails(). Θα ξεκινήσουμε με τις μεθόδους εκείνες που αφορούν πρώτα την discoverDeviceDetails(). Σημειώνουμε πως για λόγους αισθητικής, προγραμματιστικής αποδοτικότητας και καθαρότητας-σαφήνειας, ορίζουμε στην αρχή της κλάσης σταθερές συμβολοσειρές, με φράσεις/λέξεις/χαρακτήρες που χρησιμοποιούνται ως κλειδιά στην ανάκτηση της πληροφορίας.

Όπως αναφέραμε και νωρίτερα, στην κλάση CiscoIosDeviceDescription υλοποιήθηκε η κλήση μιας συνάρτησης parseCiscoIosDeviceDetails(version) της TextBlockParserCisco κλάσης που ήρθε η ώρα να γράψουμε. Αυτή η μέθοδος παίρνει ως όρισμα το κείμενο εκείνο που επιστρέφεται ως απάντηση στην εντολή show version στην συσκευή. Από αυτή πρέπει να φροντίσουμε να πάρουμε την απαραίτητη πληροφορία που αναφέρθηκε παραπάνω. Έτσι το πρώτο πράγμα που κάναμε κατά την υλοποίηση είναι να δημιουργήσουμε την συνάρτηση αυτή, και σε συμφωνία με την αντίστοιχη κλήση και τον σχεδιασμό που έγινε, την ορίζουμε έτσι ώστε να επιστρέφει έναν πίνακα. Αυτός ο πίνακας όπως είδαμε θα περιέχει τέσσερα στοιχεία τα οποία ήρθε η ώρα να τα ορίσουμε και αυτά. Αυτό το επιτυγχάνουμε με αντίστοιχο αριθμό κλήσεων μεθόδων που θα ανακτούν η καθεμία την απαραίτητη πληροφορία για την οποία είναι σχεδιασμένη. Πιο συγκεκριμένα, σε κάθε θέση του πίνακα θέτουμε το αποτέλεσμα μιας διαφορετικής συνάρτηση έτσι ώστε τελικά ο πίνακας να περιέχει όλη την πληροφορία με τον κατασκευαστή να περνάει σε αυτόν ως το αποτέλεσμα μιας μεθόδου getManufacturer (version), την έκδοση του υλικού ως αποτέλεσμα της getHwVersion(version), την έκδοση του λειτουργικού ως το αποτέλεσμα της μεθόδου getSwVersion(version) και του σειριακού αριθμού ως το αποτέλεσμα της μεθόδου serialNumber(version) με το version να είναι το ίδιο κείμενο που δόθηκε ως παράμετρος στον πατέρα-μέθοδο των συναρτήσεων αυτών. Τελικά επιστρέφουμε αυτόν τον πίνακα για να χρησιμοποιηθούν όπως είδαμε τα στοιχεία του πίνακα από την κλάση CiscoIosDeviceDescription και πιο συγκεκριμένα την μέθοδο discoverDeviceDetails().

Όσον αφορά τον κατασκευαστή, αυτό είναι εύκολο να το πάρουμε. Έτσι για την μέθοδο getManufacturer αρκεί να πάρουμε την πρώτη λέξη της πρώτης γραμμής ως τον κατασκευαστή, αφού εκεί πέρα σε όλες τις Cisco Ios συσκευές αναγράφεται η λέξη Cisco. Η μέθοδος trim() μας βοηθάει να αποφύγουμε τυχόν κενά που υπάρχουν πριν την λέξη Cisco που παρουσιάζονται λόγω block δομής.

Για την getHwVersion αυτό που βρέθηκε ασφαλές και ανταποκρίνεται στο σύνολο των Cisco Ios συσκευών είναι να αναζητείται η πληροφορία της έκδοσης του υλικού στην γραμμή που έχει την *μορφή cisco <version> (<cpu>) processor with <number of bytes>K bytes of memory*. Για ευκολία και χωρίς να χάνει την εφαρμογή της η καθολικότητα, επιλέγουμε ως φράση κλειδί την "bytes of memory" για τον εντοπισμό της γραμμής που περιέχει την πληροφορία που μας ενδιαφέρει, ενώ για να την εγκλωβίσουμε ακόμα περισσότερο εντός της γραμμής, την αναζητούμε πριν από την παρένθεση.

Για το getSwVersion, παρατηρούμε σε όλες τις Cisco Ios συσκευές ότι στην πρώτη γραμμή αναφέρεται η λέξη "software" καθώς και η λέξη "version" παραπάνω από μια φορές. Αποφασίστηκε λοιπόν η έκδοση του λειτουργικού να συντίθεται από τις φράσεις που συνοδεύουν τα "software", "version" και "RELEASE SOFTWARE". Ως παράδειγμα μιας συσκευής είναι ότι για την πρώτη γραμμή «Cisco IOS Software, C3560E Software (C3560E-UNIVERSALK9-M), Version 15.0(2)EJ, RELEASE SOFTWARE (fc1)» θα επιστραφεί ως έκδοση του λειτουργικού το "IOS C3560E 15.0(2)EJ". Ανάλογα με την λέξη κλειδί παίρνουμε την προηγούμενη ή την επόμενη λέξη.

Για τον σειριακό αριθμό τέλος, αναζητούμε την φράση κλειδί “Processor board ID” στο κείμενο και η λέξη που ακολουθεί είναι ο σειριακός αριθμός της συσκευής. Εδώ σημειώνεται ότι παρόλο που υπάρχει πάντα αυτό το σημείο στο “show version” των Ios συσκευών, μπορεί το μέγεθος του σειριακού αριθμού να διαφέρει. Αυτό υπολογίστηκε στην συγγραφή για την αποφυγή λαθών κατά την καταγραφή.

Σημειώνεται για άλλη μια φορά ότι το chassisId δεν αναφέρεται σε όλες τις συσκευές και για αυτό αποφεύχθηκε να αναζητηθεί. Ωστόσο σε άλλες συσκευές μπορεί να εμφανίζεται ως απάντηση στο “show version” είτε το “show diag” ενώ το deviceId δίνεται από το Onos.

Στην συνέχεια υλοποιήθηκε η μέθοδος parseCiscoIosPorts η οποία καλούμενη από την κλάση CiscoIosDeviceDescription, επιστρέφει μια λίστα με τα ports της συσκευής και πληροφορίες γύρω από αυτά. Οπότε σε αυτή την μέθοδο της TextBlockParserCisco κλάσης, φροντίζουμε να συλλέξουμε όλη αυτή την απαραίτητη πληροφορία. Η μέθοδος parseCiscoIosPorts παίρνει ως όρισμα το αποτέλεσμα της εντολής “show interfaces” ως κείμενο (rpc-reply), και αντίστοιχα με την προηγούμενη υλοποίηση εντοπίζει την πληροφορία εντός του κειμένου. Αυτή τη φορά όμως υπάρχει ένα στοιχείο που μας βοηθάει στην αναζήτηση. Κάθε interface ξεκινάει με το όνομα του interface από τον πρώτο χαρακτήρα της γραμμής ενώ πληροφορίες σχετικά με το συγκεκριμένο interface ξεκινούν με κενά ακέραιο πολλαπλάσιο αριθμό του 2. Βέβαια μπορεί να υπάρχει και κείμενο που ενώ έχει αυτά τα χαρακτηριστικά (πχ ξεκινάει από τον πρώτο χαρακτήρα της γραμμής με χαρακτήρες διάφορους του κενού) δεν αφορά interface (πχ VLAN). Λαμβάνοντας τα παραπάνω υπόψιν, αρχικά οφείλουμε να αφαιρέσουμε από την αναζήτηση την πρώτη γραμμή που περιέχει τα rpc στοιχεία του μηνύματος ενώ επίσης πρέπει να αφαιρέσουμε από την αναζήτηση φράσεις που δεν αφορούν interfaces. Στην παρούσα υλοποίηση θεωρήθηκε ένα σύνολο βασικών interfaces που μπορούν να υπάρχουν σε Cisco Ios switches. Αυτά είναι τα Ethernet, FastEthernet, GigabitEthernet, Serial, Pos, Fddi. Τελικά είμαστε έτοιμοι για την ανάκτηση της πληροοφρίας μέσω κλήσεων μεθόδων.

Αρχικά καλούμε μια μέθοδο interfacesCounterMethod η οποία με είσοδο το κείμενο που επιστρέφεται από την εντολή “show interfaces” που εκτελείται στην συσκευή, μετράει και υπολογίζει τα διαφορετικά interfaces που υπάρχουν σε αυτή. Αυτή η μέθοδος φροντίζει δηλαδή να μην επιστραφεί ως interface ένα VLAN αλλά μόνο αυτά που βρίσκονται στην λίστα INTERFACES που δηλώνονται ως σταθερές στο αρχείο. Ύστερα ξεκινάει μία διαδικασία όπου το κείμενο για επαναλήψεις όσες και ο αριθμός των interfaces, χωρίζεται στα κομμάτια μόνο που αφορούν interfaces και γίνεται επεξεργασία αυτών. Μέσα στην λούπα for, βλέπουμε την κλήση της μεθόδου parentInterfaceMethod με όρισμα το κείμενο επιστροφής από την show interfaces. Η μέθοδος αυτή, κάνει τους απαραίτητους ελέγχους για να αναγνωρίσει το κάθε interface ξεχωριστά, ανάλογα με τους πρώτους χαρακτήρες κάθε γραμμής και σύμφωνα με τον κανόνα ότι ένα interface ξεκινάει πάντα από τον πρώτο χαρακτήρα της γραμμής με το όνομα του ενώ πληροφορίες-παιδιά του interface βρίσκονται πιο μέσα στις επόμενες γραμμές. Τελικά η μέθοδος αυτός επιστρέφει εντός της επανάληψης for, το πρώτο interface που βρίσκει στο κείμενο ως ανεξάρτητο κομμάτι, και φροντίζει αργότερα στο ίδιο περιβάλλον να το ξεχωρίσει

βάζοντας κενά πριν το όνομα του interface έτσι ώστε να μην θεωρείται πλέον θύρα και να μην λαμβάνεται υπόψη. Ωστόσο για το κομμάτι του κειμένου που έχει κρατηθεί στην μεταβλητή parentInterface, ακολουθεί μια άλλη διαδρομή για την σύνθεση της πληροφορίας. Για κάθε κομμάτι που αφορά interface μέσα από την επανάληψη for δηλαδή, γίνεται η κλήση της μεθόδου findPortInfo. Αυτή είναι μια μέθοδος η οποία συγκεντρώνει τις κλήσεις των μεθόδων που θα δημιουργήσουν την πληροφορία γύρω από τα ports, και τελικά θα την επιστρέψει ως ένα αντικείμενο DefaultPortDescription.

Σε αυτό το σημείο καλείται η μέθοδος getPortType που με όρισμα τον πίνακα των γραμμών του κειμένου του κάθε interface αναζητά τον τύπο των Port του. Για το συγκεκριμένο κομμάτι, ύστερα από έρευνα διαπιστώθηκε ότι με Ethernet, FastEthernet, GigabitEtherner και Serial έχουμε σύνδεση καλωδίου χαλκού, οπότε στο Type θέτουμε την κατάσταση “COPPER” ενώ για τα interfaces Fddi και Pos έχουμε σύνδεση οπτικής ίνας και για αυτό θέτουμε την κατάσταση “FIBER”. Για να το πετύχουμε αυτό ελέγχουμε δηλαδή το όνομα του interface.

Ύστερα ελέγχεται η κατάσταση του port. Σημειώνεται εδώ, ότι τα ports αναγράφονται στην μορφή “InterfaceSlot/Port”. Αυτό συμβαίνει μέσω της μεθόδου getIsEnabled. Αυτό επαληθεύεται γρήγορα από την πρώτη γραμμή του interface στο οποίο αναγράφεται η φράση «"is up, line protocol is up» όταν η κατάσταση του είναι Up, αλλιώς θα είναι down, όπου συνήθως αναγράφεται “administratively down” (δεν το ελέγχουμε). Από την μέθοδο αυτή επιστρέφεται μια δυαδική μεταβλητή 0/1.

Σαν επόμενο βήμα υλοποιήθηκε η συνάρτηση getPort η οποία θα πρέπει να επιστρέφει το Port από το οποίο θα μπορεί να επικοινωνεί η συσκευή με το ONOS ή άλλες άκρες. Η πληροφορία αυτή δίνεται από τους αριθμούς που συνοδεύουν το όνομα του Interface στην μορφή InterfaceSlot/Port. Έτσι, μπορούμε στην πρώτη γραμμή, και ειδικότερα στην πρώτη λέξη, να πάρουμε τους χαρακτήρες που βρίσκονται μετά τον χαρακτήρα της καθέτου “/” μέχρι να εντοπιστεί κενό (που σημαίνει ότι ο οσωνδήποτε ψηφίων αριθμός έχει τελειώσει). Επιστρέφεται το αποτέλεσμα ως String για να μετατραπεί στην κατάλληλη μορφή. Για τις ανάγκες διατήρησης σωστής και ευσταθούς υλοποίησης χρησιμοποιούνται try-catch block, καθώς και περιπτώσεις όπου η πόρτα παίρνει αρνητικό αριθμό (-1).

Τέλος πρέπει να πάρουμε και την ταχύτητα με την οποία μπορούν να μεταφερθούν δεδομένα μέσω του συγκεκριμένου port. Για τον σκοπό αυτό επιστρατεύουμε μια μέθοδο getPortSpeed η οποία εντοπίζει αυτή την πληροφορία. Για την ακρίβεια γίνεται αναζήτηση σε όλο το κείμενο ενός interface για τις λέξεις “BW” και “Kbit/sec” και η πληροφορία ανακτάται από το ενδιάμεσο διάστημα αυτών. Επιστρέφεται ως long στην μέθοδο που την καλεί (findPortInfo).

Τελικά η μέθοδος findPortInfo έχει συγκεντρωμένη όλη την πληροφορία που χρειάζεται για να φτιάξει το αντικείμενο που θα επιστραφεί ως λίστα στην κλάση CiscoIosDeviceDescritpion. Ως σχόλιο στην κάθε πόρτα περνάει και το όνομα του interface. Δηλαδή αν για παράδειγμα το interface είναι το GigabitEthernet0/4 τότε στις πληροφορίες της πόρτας που είναι η τέσσερα, θα

υπάρχει και ως σχόλιο η λέξη GigabitEthernet. Νωρίτερα έχουμε μετατρέψει το string του τύπου της θύρας στο κατάλληλο Port.Type. Στην τελευταία εντολή συντίθεται το αντικείμενο DefaultPortDescription με παραμέτρους αυτά τα αποτελέσματα καθώς και το Annotation με το όνομα του interface ενώ σε περίπτωση που η θύρα έχει την τιμή "-1" τότε επιστρέφεται ένα Null αντικείμενο, και η findPortInfo ετοιμάζεται να ξανακληθεί για τα υπόλοιπα interfaces.

Για την ολοκλήρωση της υλοποίησης προστέθηκαν γραμμές κώδικα στο αρχείο των cisco-drivers.xml έτσι ώστε να καλείται η συμπεριφορά DeviceDescriptionDiscovery που γίνεται implement από την δική μας CiscoIosDeviceDescription. Αυτό είναι το σημείο που η υλοποίηση δένει με το υπόλοιπο ONOS ώστε να γίνονται οι απαραίτητες κλήσεις των μεθόδων που θα παράξουν τα αποτελέσματα και τελικά θα έχουμε μια συσκευή ορατή από το ONOS και έτοιμη για Netconf επικοινωνία.

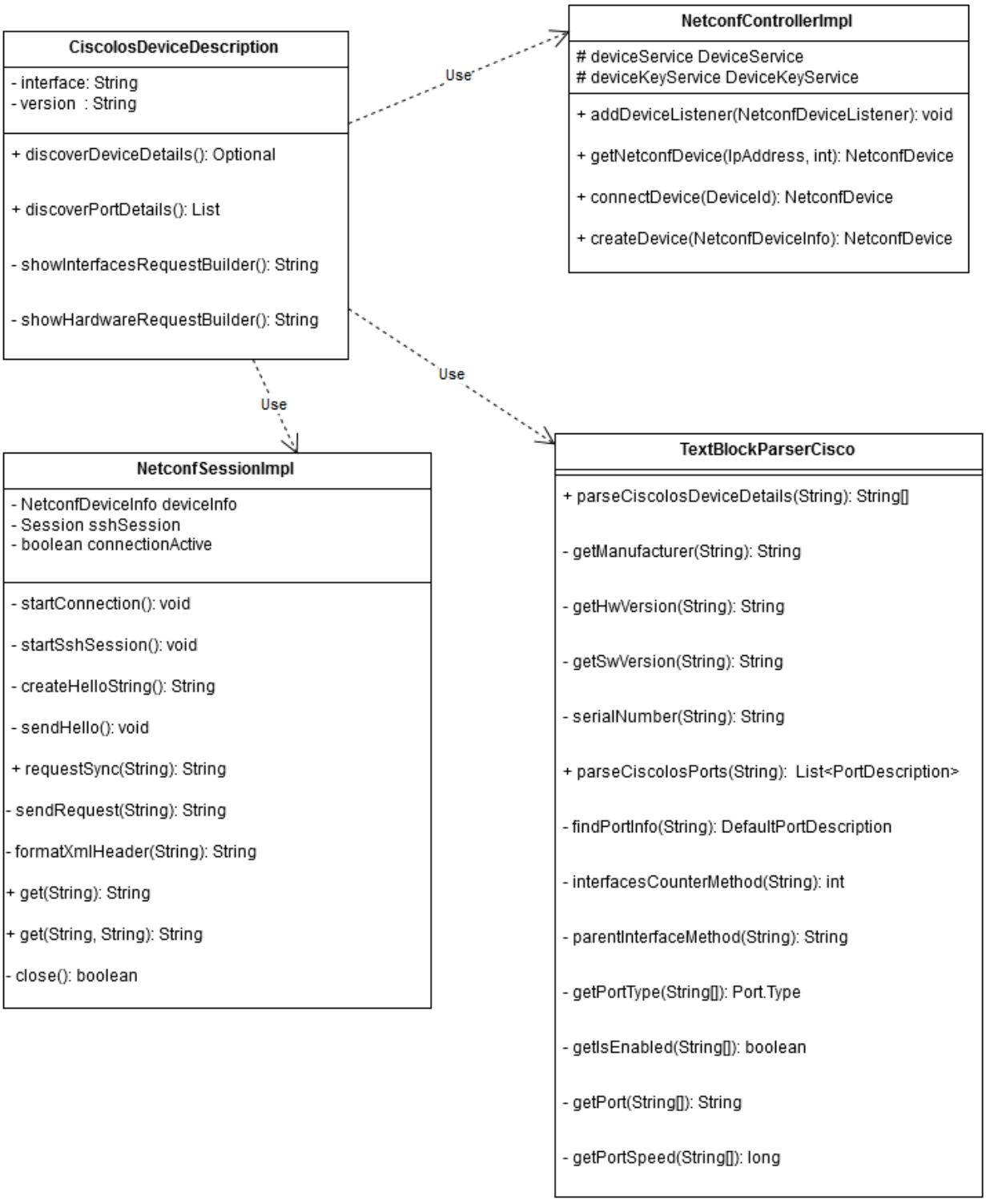
UML διαγράμματα

Παρατίθεται UML των κλάσεων που παρουσιάζει το πώς σχετίζονται οι κλάσεις μεταξύ τους καθώς και flow chart για να γίνει περισσότερο ξεκάθαρο πως γίνονται οι κλήσεις των συναρτήσεων στις Εικόνα 21 και Εικόνα 22 αντίστοιχα.

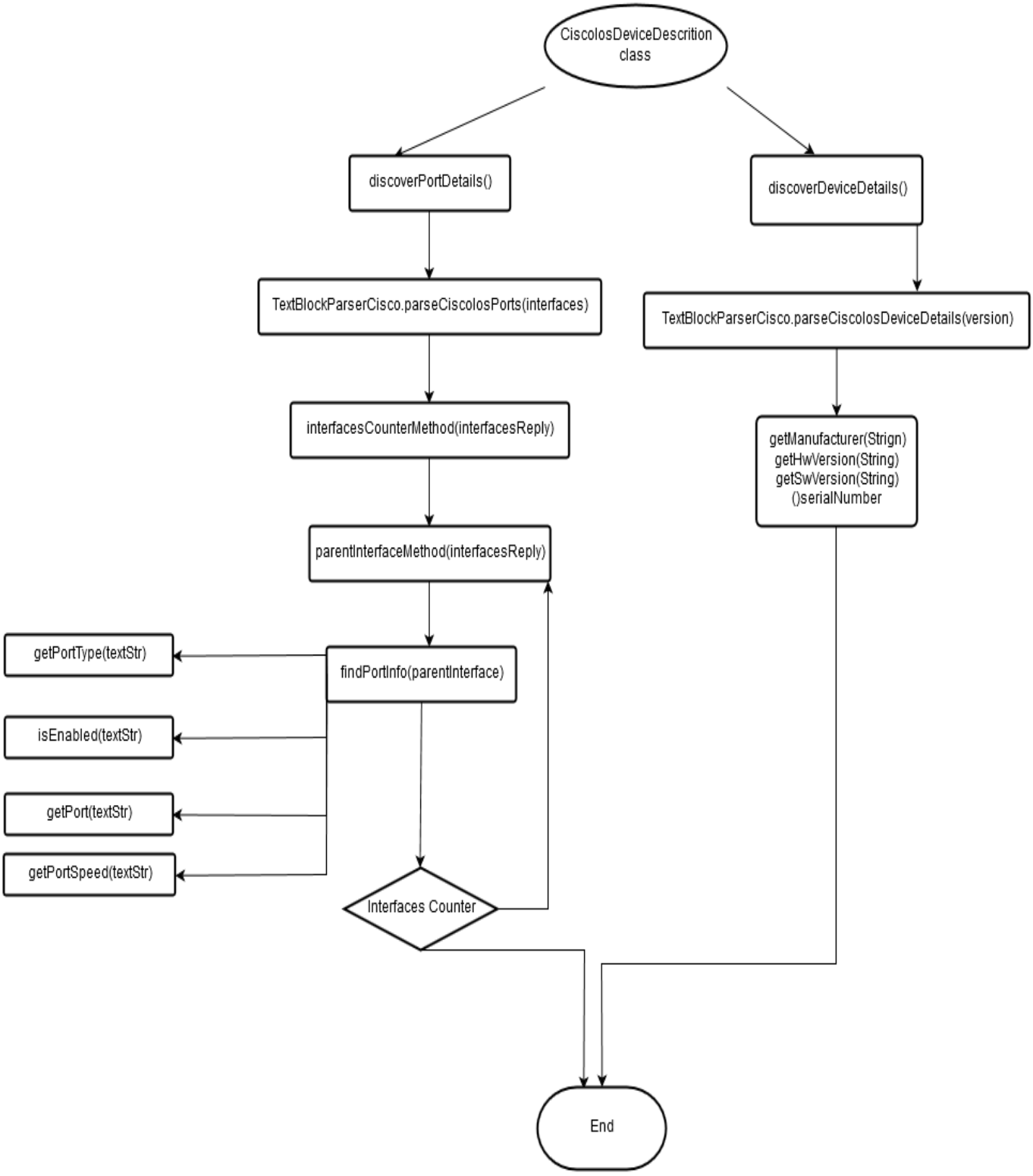
5.4 Έλεγχος λειτουργίας υλοποίησης

Αφού συγγράψαμε τον κώδικα πρέπει να τον ελέγξουμε για να επιβεβαιώσουμε την ορθή λειτουργία του. Το πρώτο επίπεδο άμυνας σε λάθη συντακτικά, λογικά, ή σε ζητήματα αστάθειας της λειτουργικότητας της υλοποίησης, είναι φυσικά να είμαστε σίγουροι για την συγγραφή και για κάθε κομμάτι αυτής. Κατά την συγγραφή γινόταν σε όλα τα βήματα σωστή κατανόηση των μεθόδων που χρησιμοποιήθηκαν, γίνονταν τα κατάλληλα imports, και έγινε προσπάθεια να υπάρχουν οι μέθοδοι εκείνοι που κάνουν την δουλεία για την οποία επιστρατεύονται με γνώμονα την ταχύτητα, την ευστάθεια καθώς και το μικρό βάρος προς το συνολικό έργο του ONOS. Έτσι σε όλη την διάρκεια της υλοποίησης γινόταν συνεχής έλεγχος για την ορθότητα της συγγραφής καθώς και αναζήτηση για καλύτερους τρόπους υλοποίησης των διάφορων λογικών του κώδικα. Φυσικά ελεγχόταν και κατά πόσο ο κώδικα περνούσε το compile, αλλά σε αυτό βοήθησε πολύ το IDE intelliJ το οποίο με τις ειδοποιήσεις του απέτρεπε σε μεγάλο βαθμό μικρά λάθη που οδηγούσαν σε compilation fail. Έτσι συχνά κατά την συγγραφή μετά από υλοποίηση μικρών κομματιών γινόταν "mvn clean install" και τρέχαμε το ONOS για να φαίνεται πληροφορία από τα logs του ONOS. Τόσο τα logs όσο και τα εργαλεία του IDE βοήθησαν πολύ στο debuggin. Με στόχο να δούμε την συσκευή και τις πραγματικές της πληροφορίες μέσα από το ONOS η υλοποίηση εξελισσόταν.

Για την επιβεβαίωση της ορθής λειτουργίας της υλοποίησης, και παράλληλα με την υπόλοιπη συγγραφή, γινόταν και η δημιουργία κλάσεων test ως θεμιτό στοιχείο για το project του ONOS και την προσφορά κώδικα σε αυτό. Φτιάχτηκε λοιπόν μία test κλάση TextBlockParserCiscoTest



Εικόνα 21: UML υλοποίησης



Εικόνα 22: Διάγραμμα ροής κλήσεων μεθόδων

με δύο test μέσα. Το κάθε ένα ελέγχει και μια συμπεριφορά του DeviceDescriptionDiscovery. Ένα για τα interfaces και ένα για τα details. Το αρχείο αυτό παρατίθεται επίσης στο τέλος του βιβλίου στο Παράρτημα Α με ομώνυμη ονομασία.

Για την ανάπτυξη της κλάσης αυτής, πήραμε πραγματική απάντηση από συσκευή Cisco Ios για την εντολή "show version" αυτής, ενώ επιπλέον συνθέσαμε μια δική μας απάντηση με έξι interfaces που αντιστοιχούν σε interfaces πραγματικών συσκευών που όμως δεν ήταν πραγματικά παρόντα στην δική μας εικονική gns3 συσκευή. Σε αυτά τα test έγινε έλεγχος μεταξύ των πληροφοριών που εμείς αναμέναμε να πάρουμε και ορίσαμε ρητά στην κλάση αυτή, και την πληροφορία που υλοποίηση δίνει. Αυτά τα test τρέχουν και κατά την διάρκεια εγκατάστασης του ONOS.

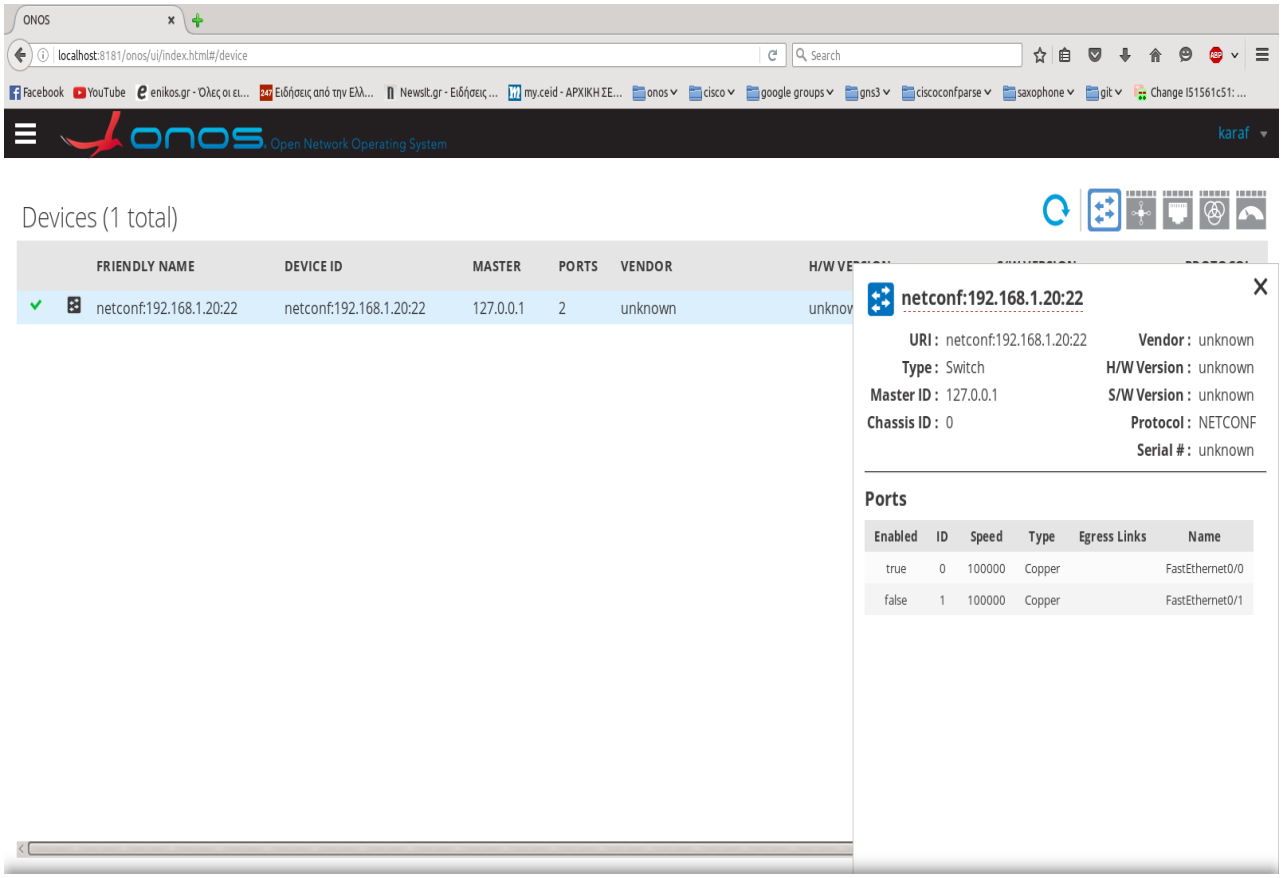
Τέλος για να συμβαδίζει με τις απαιτήσεις και την αξιολόγηση που περνάει ο κώδικας κατά την κατάθεση στο gerrit, τρέξαμε και buck tests (buck build onos, buck test) αφού είναι εργαλεία που επίσημα χρησιμοποιείται και από το project του ONOS στο gerrit.

Αφού όλα πλέον είχαν γίνει σωστά, μπορούσαμε να δούμε τις πληροφορίες της συσκευής τόσο στο User Interface του ONOS μαζί με της πληροφορίες του, όσο και στην κονσόλα χτυπώντας την εντολή "devices" μετά από «σπρώξιμο» της συσκευής σε αυτό μέσω json. Γράφτηκε λοιπόν το αρχείο εκείνο που έσπρωχνε την συσκευή στο ONOS με την κατάλληλη IP διεύθυνση, username και password και επιβεβαιώθηκε ότι ο κώδικας είναι πλήρως λειτουργικός.

Ακολουθώντας τα παρακάτω βήματα:

1. ok clean (terminal 1)
2. app activate org.onosproject.netconf (terminal 1)
3. app activate org.onosproject.drivers.cisco (terminal 1)
4. tl (terminal 2 για παρακολούθηση των logs)
5. onos-netcfg localhost myjson.json (terminal 3)

καταφέραμε για εικονική συσκευή να πάρουμε τις θύρες της και πληροφορίες κατάστασης, ταχύτητας καθώς και τον τύπο τους. Ωστόσο όπως φαίνεται δεν εμφανίζεται ο κατασκευαστής και οι λοιπές γενικές πληροφορίες. Αυτό οφείλεται σε έλλειψη (προς το παρόν) του ONOS.



Εικόνα 23: ONOS user interface - Συσκευή

5.5 Κατάθεση και αξιολόγηση του κώδικα

Ο κώδικας τελικά, αφού αποδείχτηκε ότι είναι λειτουργικός ακολουθήθηκαν οι απαραίτητες διαδικασίες για να κατατεθεί και να συγχωνευτεί με την υπόλοιπη δομή του ONOS. Συγκεκριμένα αφού δημιουργήθηκε λογαριασμός στο gerrit ακολουθήθηκαν οι οδηγίες που φαίνεται στον σύνδεσμο *https://git.eclipse.org/r/Documentation/user-upload.html* και ο κώδικας με την μορφή πρότασης για αλλαγή εμφανίστηκε στο gerrit με σκοπό την αξιολόγηση του. Εκεί προτάθηκαν κάποιες διορθώσεις, και αφού έγιναν, ο κώδικας θεωρήθηκε ότι μπορούσε να συγχωνευτεί με το υπόλοιπο project του ONOS, οπότε συγχωνεύτηκε με αυτό (*https://gerrit.onosproject.org/#/c/10269*). Το κομμάτι λοιπόν πλέον διατίθεται ως μέρος του συνόλου του ONOS και φυσικά είναι και στο project του ONOS στο github.

6. Συμπεράσματα

Στα πλαίσια αυτής της διπλωματικής εργασίας, υλοποιήσαμε την επικοινωνία μεταξύ του όλο και πιο γρήγορα αναπτυσσόμενου Open Networking Operating System (ONOS) και συσκευών Cisco οι οποίες χρησιμοποιούν λειτουργικό Ios. Η επικοινωνία γίνεται υπό το πρωτόκολλο Netconf που χρησιμοποιεί συνεδρίες SSHv2 για να εκμεταλλευτεί την κρυπτογράφηση του. Γράφτηκαν εκείνες οι κλάσεις οι οποίες αναζήτησαν πληροφορίες για την συσκευή, όπως interface,speed,status και τύπο σύνδεσης, από τις απαντήσεις στα κατάλληλα αιτήματα προς εκείνη μέσω μηνυμάτων XML τα οποία εμφωλεύονται σε rpc αιτήματα, και τελικά κοινωνικοποιήσαμε αυτή την πληροφορία προς το ONOS ώστε να εδραιώσει μια μέχρι πρότινος υποτυπώδη επικοινωνία. Η επικοινωνία πέτυχε και πλέον μπορούμε να βλέπουμε τουλάχιστον τις θύρες της συσκευής. Οι γενικές πληροφορίες όπως ο κατασκευαστής και οι εκδόσεις υλικού και λειτουργικού δεν είναι ζωτικής σημασίας, έτσι η συσκευή μπορεί να είναι λειτουργική και χωρίς αυτές.

Μέσω της επικοινωνίας αυτής, μπορούμε σε πρώτη φάση να επιβεβαιώσουμε ότι παίρνουμε τις ρυθμίσεις της συσκευής μέσω της εντολής “device-configuration” ακολουθούμενη από την συσκευή με την οποία έχει επιτευχθεί επικοινωνία. Προς το παρών κάποιες εντολές δεν λειτουργούν και αυτό οφείλεται στο ότι το ONOS δεν έχει ολοκληρώσει τις διεπαφές για επικοινωνία μέσω Netconf.

Οι συσκευές Cisco που χρησιμοποιούν το λειτουργικό Ios είναι πλέον εφικτό να θέτουν την βάση για επικοινωνία με το ONOS. Είναι ένα μεγάλο βήμα τόσο για το ONOS, το οποίο θα μπορεί να υποστηρίζει τις περισσότερες συσκευές ενός τόσο μεγάλου κατασκευαστή όπως της Cisco, για ένα τόσο διαδεδομένο και χρήσιμο πρωτόκολλο όπως το Netconf, όσο και για την ίδια την Cisco η οποία με αυτή την υλοποίηση παίρνει «πόντους», καθώς οι πελάτες της που χρησιμοποιούν συσκευές Cisco Ios θα μπορούν επίσης να επικοινωνήσουν με το ONOS και να εκμεταλλευτούν τα πλεονεκτήματα του και την προσφορά του Software Defined Networking για βελτίωση των υψηλής ταχύτητας δεδομένων δικτύων τους.

Μέσω του πρωτοκόλλου Netconf μπορεί το ONOS να διαχειρίζεται συσκευές Cisco Ios και να εγκαθιστά, διαχειρίζεται ή ακόμα και να διαγράφει τις διαμορφώσεις τους. Σε συνδυασμό με άλλες δυνατότητες ενός δικτύου SDN, χτίζεται τελικά το πλαίσιο μέσα στο οποίο αλγόριθμοι, στατιστικές αναλύσεις, έλεγχοι και αποφάσεις κάνουν ένα οποιοδήποτε δίκτυο περισσότερο αποδοτικό, ευσταθές, ασφαλές, γρήγορο, ευέλικτο και αξιόπιστο.

8. Μελλοντική Εργασία

Το ONOS, ως ένα project ανοιχτού κώδικα, βασίζει την επιτυχία του και στον εθελοντισμό. Είναι σημαντικό λοιπόν να υπάρχει ανταπόκριση ώστε να συνεχίσει να εξελίσσεται και να προσφέρει στον τομέα των δικτύων δεδομένων υψηλών ταχυτήτων όλο και περισσότερο. Πρέπει να υπάρχει συνεισφορά από προγραμματιστές και ερευνητές για να επιστρέφεται η εργασία ως καρπός μέσω πιο αξιόπιστων δικτύων, ασφαλών και γρήγορων.

Ένας προγραμματιστής μπορεί να ασχοληθεί με μία πληθώρα θεματολογιών που βρίσκονται σε εξέλιξη στο ONOS. Υπάρχουν πρωτόκολλα όπως το Netconf, pcep, bgp, snmp, openflow τα οποία χρειάζονται εξέλιξη ή ολοκλήρωση. Από την άλλη, ως ένα σχετικά νέο εγχείρημα, το ONOS είναι ανοιχτό προς νέα πρωτόκολλα τα οποία αν υλοποιηθούν θα ωφελήσουν την παγκόσμια κοινότητα. Πρωτόκολλα τα οποία χρησιμοποιούνται σε μεγάλο εύρος από παρόχους, από προγραμματιστές και από χρήστες. Μια αλλαγή δεν ωφελεί μεμονωμένα έναν κόμβο, αλλά το σύνολο του δικτύου.

Χαρακτηριστικά για το πρωτόκολλο Netconf με το οποίο ασχοληθήκαμε, είδαμε ότι δεν υπάρχει ολοκληρωμένη επικοινωνία, αφού πληροφορίες όπως αυτές του κατασκευαστή, της έκδοσης του υλικού και του λειτουργικού ή του σειριακού αριθμού δεν μεταφέρονται στο ONOS. Ο λόγος είναι ότι δεν υπάρχει η κατάλληλη διεπαφή που θα προκαλέσει τις κλήσεις εκείνες ώστε οι πληροφορίες να φτάσουν στο ONOS. Από την δική μας πλευρά φροντίσαμε να ανακτούμε αυτή τη πληροφορία. Επομένως ως μελλοντική εργασία θα προτεινόταν να γίνει η έρευνα και τελικά να υλοποιηθεί αυτό το κομμάτι, το οποίο θα δένει με την παρούσα υλοποίηση και θα ολοκληρώνει την συμπεριφορά που ζητάει βασικές πληροφορίες των συσκευών Cisco Ios.

Αλλά και πιο πέρα, είδαμε ότι εντολές δεν λειτουργούσαν μετά την επικοινωνία, εντολές του ONOS οι οποίες εμφάνιζαν μηνύματα έλλειψης των κατάλληλων διεπαφών για να ολοκληρωθούν. Ομοίως μπορεί να γίνει έρευνα και υλοποίηση. Φυσικά, το πρωτόκολλο Netconf έχει πολλές δυνατότητες οι οποίες δεν αντανακλώνται ακόμη στο ONOS. Είναι σημαντικό να έχουν αντίκρισμα όλες οι δυνατότητες του Netconf στο ONOS.

Ασφαλώς, δεν είναι μόνο οι Cisco συσκευές που μπορούν να επικοινωνούν μέσω Netconf. Οι περισσότερες συσκευές σήμερα, switches και δρομολογητές, υποστηρίζουν ένα τόσο διαδεδομένο και χρήσιμο πρωτόκολλο όπως το Netconf. Με βάση την παρούσα εργασία και ως παράδειγμα, θα μπορούσε αντίστοιχα να υλοποιηθεί το πρωτόκολλο και για άλλους κατασκευαστές. Χαρακτηριστικά αναφέρονται μερικοί οι οποίοι ήδη έχουν συνεργασία και χρησιμοποιούν το εργαλείο ONOS όπως ciena, fujitsu, lumentum,corsa.

Γενικότερα, το SDN είναι μια τεχνολογία η οποία θα αλλάξει ριζικά τα δίκτυα υπολογιστών όπως τα ξέρουμε μέχρι σήμερα. Βρίσκεται ακόμα σε αρχικό στάδιο και δεν έχει οριστεί ξεκάθαρα ποιος θα είναι ο ρόλος του. Έτσι υπάρχουν απαιτήσεις για ερευνητές που θα προχωρήσουν παραπέρα το SDN, ενώ ταυτόχρονα με την εξέλιξη του θα μεγαλώνουν και οι

ανάγκες για ασφάλεια, αξιοπιστία και ταχύτητα. Ακόμη θα προκύπτουν συνεχώς νέες ανάγκες ή και προβλήματα τα οποία θα πρέπει να ξεπεραστούν με την συμβολή και της κοινότητας ανοιχτού κώδικα.

ΠΑΡΑΡΤΗΜΑ Α

drivers/cisco/src/main/java/org/onosproject/drivers/cisco/CiscoIosDeviceDescription.java

```
1  /*
2   * Copyright 2016-present Open Networking Laboratory
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package org.onosproject.drivers.cisco;
18
19 import com.google.common.collect.ImmutableList;
20
21 import org.onosproject.net.Device;
22 import org.onosproject.net.DeviceId;
23 import org.onosproject.net.device.DefaultDeviceDescription;
24 import org.onosproject.net.device.DeviceDescription;
25 import org.onosproject.net.device.DeviceDescriptionDiscovery;
26 import org.onosproject.net.device.DeviceService;
27 import org.onosproject.net.device.PortDescription;
28 import org.onosproject.net.driver.AbstractHandlerBehaviour;
29 import org.onosproject.netconf.NetconfController;
30 import org.onosproject.netconf.NetconfException;
31 import org.onosproject.netconf.NetconfSession;
32 import org.slf4j.Logger;
33 import java.io.IOException;
34 import java.util.List;
35 import static com.google.common.base.Preconditions.checkNotNull;
36 import static org.slf4j.LoggerFactory.getLogger;
37
38 public class CiscoIosDeviceDescription extends AbstractHandlerBehaviour
39         implements DeviceDescriptionDiscovery {
40
41
42     private final Logger log = getLogger(getClass());
43     private String version;
44     private String interfaces;
45
46     @Override
47     public DeviceDescription discoverDeviceDetails() {
48         NetconfController controller =
49 checkNotNull(handler().get(NetconfController.class));
```

```
50          NetconfSession session =
51  controller.getDevicesMap().get(handler().data().deviceId()).getSession();
52          try {
53              version = session.get(showVersionRequestBuilder());
54          } catch (IOException e) {
55              throw new RuntimeException(new NetconfException("Failed to
56  retrieve version info.", e));
57          }
58
59          String[] details =
60  TextBlockParserCisco.parseCiscoIosDeviceDetails(version);
61
62          DeviceService deviceService =
63  checkNotNull(handler().get(DeviceService.class));
64          DeviceId deviceId = handler().data().deviceId();
65          Device device = deviceService.getDevice(deviceId);
66
67          return new DefaultDeviceDescription(device.id().uri(),
68  Device.Type.SWITCH,
69                                              details[0], details[1],
70                                              details[2], details[3],
71                                              device.chassisId());
72      }
73
74      @Override
75      public List<PortDescription> discoverPortDetails() {
76          NetconfController controller =
77  checkNotNull(handler().get(NetconfController.class));
78          NetconfSession session =
79  controller.getDevicesMap().get(handler().data().deviceId()).getSession();
80          try {
81              interfaces = session.get(showInterfacesRequestBuilder());
82          } catch (IOException e) {
83              log.error("Failed to retrieve Interfaces");
84              return ImmutableList.of();
85          }
86          return
87  ImmutableList.copyOf(TextBlockParserCisco.parseCiscoIosPorts(interfaces));
88      }
89
90      /**
91       * Builds a request crafted to get the configuration required to create
92       * details descriptions for the device.
93       *
94       * @return The request string.
95       */
96      private String showVersionRequestBuilder() {
97          StringBuilder rpc = new StringBuilder("<rpc
98  xmlns=\"urn:ietf:params:xml:ns:netconf:base:1.0\">");
99          rpc.append("<get>");
100         rpc.append("<filter>");
101         rpc.append("<config-format-text-block>");
102         rpc.append("<text-filter-spec> | include exp_to_match_run_conf
103 </text-filter-spec>");
104         rpc.append("</config-format-text-block>");
105         rpc.append("<oper-data-format-text-block>");
106         rpc.append("<show>version</show>");
```

```
107             rpc.append("</oper-data-format-text-block>");
108             rpc.append("</filter>");
109             rpc.append("</get>");
110             rpc.append("</rpc>]]>]]>");
111             return rpc.toString();
112         }
113 
114         /**
115          * Builds a request crafted to get the configuration required to create
116          * details descriptions for the device.
117          *
118          * @return The request string.
119          */
120         private String showInterfacesRequestBuilder() {
121             //Message ID is injected later.
122             StringBuilder rpc = new StringBuilder("<rpc
123     xmlns=\"urn:ietf:params:xml:ns:netconf:base:1.0\">");
124             rpc.append("<get>");
125             rpc.append("<filter>");
126             rpc.append("<config-format-text-block>");
127             rpc.append("<text-filter-spec> | include exp_to_match_run_conf
128     </text-filter-spec>");
129             rpc.append("</config-format-text-block>");
130             rpc.append("<oper-data-format-text-block>");
131             rpc.append("<show>interfaces</show>");
132             rpc.append("</oper-data-format-text-block>");
133             rpc.append("</filter>");
134             rpc.append("</get>");
135             rpc.append("</rpc>]]>]]>");
136             return rpc.toString();
137         }
138 
139     }
```

drivers/cisco/src/main/java/org/onosproject/drivers/cisco/TextBlockParserCisco.java

```
1  /*
2  * Copyright 2016-present Open Networking Laboratory
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  *     http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package org.onosproject.drivers.cisco;
18
19 import com.google.common.collect.Lists;
20 import org.onosproject.net.AnnotationKeys;
21 import org.onosproject.net.DefaultAnnotations;
22 import org.onosproject.net.PortNumber;
23 import org.onosproject.net.device.DefaultPortDescription;
24 import org.onosproject.net.device.PortDescription;
25 import java.util.Arrays;
26 import java.util.List;
27
28 import static org.onosproject.net.Port.Type;
29
30 /**
31  *Parser for Netconf XML configurations and replys as plain text.
32  */
33 final class TextBlockParserCisco {
34
35     private static final String PHRASE = "bytes of memory.";
36     private static final String VERSION = "Version ";
37     private static final String EOF_VERSION1 = "Software";
38     private static final String EOF_VERSION2 = "Software,";
39     private static final String EOF_VERSION3 = "RELEASE";
40     private static final String PROCESSOR_BOARD = "Processor board ID ";
41     private static final String BANDWIDTH = "BW ";
42     private static final String SPEED = " Kbit/sec";
43     private static final String ETHERNET = "Eth";
44     private static final String FASTETHERNET = "Fas";
45     private static final String GIGABITETHERNET = "Gig";
46     private static final String FDDI = "Fdd";
47     private static final String POS = "POS";
48     private static final String SERIAL = "Ser";
49     private static final String NEWLINE_SPLITTER = "\n";
50     private static final String PORT_DELIMITER = "/";
51     private static final String SPACE = " ";
52     private static final String IS_UP = "is up, line protocol is up";
53     private static final List INTERFACES = Arrays.asList(ETHERNET,
54 FASTETHERNET, GIGABITETHERNET, SERIAL, FDDI, POS);
```

```
55      private static final List FIBERINTERFACES = Arrays.asList(FDDI, POS);
56
57
58      private TextBlockParserCisco() {
59          //not called
60      }
61
62      /**
63       * Adding information in an array for CiscoIosDeviceDescriptin call.
64       * @param version the return of show version command
65       * @return the array with the information
66       */
67      static String[] parseCiscoIosDeviceDetails(String version) {
68          String[] details = new String[4];
69          details[0] = getManufacturer(version);
70          details[1] = getHwVersion(version);
71          details[2] = getSwVersion(version);
72          details[3] = serialNumber(version);
73          return details;
74      }
75
76      /**
77       * Retrieving manufacturer of device.
78       * @param version the return of show version command
79       * @return the manufacturer of the device
80       */
81      private static String getManufacturer(String version) {
82          int i;
83          String[] textStr = version.split(NEWLINE_SPLITTER);
84          String[] lineStr = textStr[0].trim().split(SPACE);
85          return lineStr[0];
86      }
87
88      /**
89       * Retrieving hardware version of device.
90       * @param version the return of show version command
91       * @return the hardware version of the device
92       */
93      private static String getHwVersion(String version) {
94          String[] textStr = version.split(NEWLINE_SPLITTER);
95          String processor = SPACE;
96          int i;
97          for (i = 0; i < textStr.length; i++) {
98              if (textStr[i].indexOf(PHRASE) > 0) {
99                  String[] lineStr = textStr[i].trim().split(SPACE);
100                 processor = lineStr[1];
101                 break;
102             } else {
103                 processor = SPACE;
104             }
105         }
106         return processor;
107     }
108
109     /**
110      * Retrieving software version of device.
111      * @param version the return of show version command
```

```
112         * @return the software version of the device
113         */
114        private static String getSwVersion(String version) {
115            String[] textStr = version.split(NEWLINE_SPLITTER);
116            int i;
117            for (i = 0; i < textStr.length; i++) {
118                if (textStr[i].indexOf(VERSION) > 0) {
119                    break;
120                }
121            }
122            String[] lineStr = textStr[i].trim().split(SPACE);
123            StringBuilder sw = new StringBuilder();
124            for (int j = 0; j < lineStr.length; j++) {
125                if (lineStr[j].equals(EOF_VERSION1) ||
126    lineStr[j].equals(EOF_VERSION2)
127                        ) {
128                    sw.append(lineStr[j - 1]).append(SPACE);
129                } else if (lineStr[j].equals(EOF_VERSION3)) {
130                    sw.append(lineStr[j - 1]);
131                    sw.setLength(sw.length() - 1);
132                }
133            }
134            return sw.toString();
135        }
136
137        /**
138         * Retrieving serial number of device.
139         * @param version the return of show version command
140         * @return the serial number of the device
141         */
142        private static String serialNumber(String version) {
143            String[] textStr = version.split(NEWLINE_SPLITTER);
144            int i;
145            for (i = 0; i < textStr.length; i++) {
146                if (textStr[i].indexOf(PROCESSOR_BOARD) > 0) {
147                    break;
148                }
149            }
150            return textStr[i].substring(textStr[i].indexOf(PROCESSOR_BOARD) +
151    PROCESSOR_BOARD.length(),
152                                        textStr[i].length());
153        }
154
155        /**
156         * Calls methods to create information about Ports.
157         * @param interfacesReply the interfaces as plain text
158         * @return the Port description list
159         */
160        public static List<PortDescription> parseCiscoIosPorts(String
161    interfacesReply) {
162            String parentInterface;
163            String[] parentArray;
164            String tempString;
165            List<PortDescription> portDesc = Lists.newArrayList();
166            int interfacesCounter = interfacesCounterMethod(interfacesReply);
167            for (int i = 0; i < interfacesCounter; i++) {
168                parentInterface = parentInterfaceMethod(interfacesReply);
```

```
169             portDesc.add(findPortInfo(parentInterface));
170             parentArray = parentInterface.split(SPACE);
171             tempString = parentArray[0] + SPACE;
172             interfacesReply = interfacesReply.replace(tempString, SPACE +
173 tempString);
174         }
175         return portDesc;
176     }
177
178     /**
179      * Creates the port information object.
180      * @param interfaceTree the interfaces as plain text
181      * @return the Port description object
182      */
183     private static DefaultPortDescription findPortInfo(String interfaceTree)
184 {
185         String[] textStr = interfaceTree.split(NEWLINE_SPLITTER);
186         String[] firstLine = textStr[0].split(SPACE);
187         String firstWord = firstLine[0];
188         Type type = getPortType(textStr);
189         boolean isEnabled = getIsEnabled(textStr);
190         String port = getPort(textStr);
191         long portSpeed = getPortSpeed(textStr);
192         DefaultAnnotations.Builder annotations = DefaultAnnotations.builder()
193                 .set(AnnotationKeys.PORT_NAME, firstWord);
194         return port.equals("-1") ? null : new
195 DefaultPortDescription(PortNumber.portNumber(port),
196                                                                 isEnabled,
197 type, portSpeed, annotations.build());
198     }
199
200     /**
201      * Counts the number of existing interfaces.
202      * @param interfacesReply the interfaces as plain text
203      * @return interfaces counter
204      */
205     private static int interfacesCounterMethod(String interfacesReply) {
206         int counter;
207         String first3Characters;
208         String[] textStr = interfacesReply.split(NEWLINE_SPLITTER);
209         int lastLine = textStr.length - 1;
210         counter = 0;
211         for (int i = 1; i < lastLine; i++) {
212             first3Characters = textStr[i].substring(0, 3);
213             if (INTERFACES.contains(first3Characters)) {
214                 counter++;
215             }
216         }
217         return counter;
218     }
219
220     /**
221      * Parses the text and seperates to Parent Interfaces.
222      * @param interfacesReply the interfaces as plain text
223      * @return Parent interface
224      */
225     private static String parentInterfaceMethod(String interfacesReply) {
```

```
226             String firstCharacter;
227             String first3Characters;
228             boolean isChild = false;
229             StringBuilder anInterface = new StringBuilder("");
230             String[] textStr = interfacesReply.split("\\n");
231             int lastLine = textStr.length - 1;
232             for (int i = 1; i < lastLine; i++) {
233                 firstCharacter = textStr[i].substring(0, 1);
234                 first3Characters = textStr[i].substring(0, 3);
235                 if (!(firstCharacter.equals(SPACE)) && isChild) {
236                     break;
237                 } else if (firstCharacter.equals(SPACE) && isChild) {
238                     anInterface.append(textStr[i]).append(NEWLINE_SPLITTER);
239                 } else if (INTERFACES.contains(first3Characters)) {
240                     isChild = true;
241                     anInterface.append(textStr[i]).append(NEWLINE_SPLITTER);
242                 }
243             }
244             return anInterface.toString();
245         }
246 
247         /**
248          * Get the port type for an interface.
249          * @param textStr interface splitted as an array
250          * @return Port type
251          */
252         private static Type getPortType(String[] textStr) {
253             String first3Characters;
254             first3Characters = textStr[0].substring(0, 3);
255             return FIBERINTERFACES.contains(first3Characters) ? Type.FIBER :
256     Type.COPPER;
257         }
258 
259         /**
260          * Get the state for an interface.
261          * @param textStr interface splitted as an array
262          * @return isEnabled state
263          */
264         private static boolean getIsEnabled(String[] textStr) {
265             return textStr[0].contains(IS_UP);
266         }
267 
268         /**
269          * Get the port number for an interface.
270          * @param textStr interface splitted as an array
271          * @return port number
272          */
273         private static String getPort(String[] textStr) {
274             String port;
275             try {
276                 if (textStr[0].indexOf(PORT_DELIMITER) > 0) {
277                     port =
278     textStr[0].substring(textStr[0].lastIndexOf(PORT_DELIMITER) + 1,
279                                                 textStr[0].indexOf(SPACE));
280                 } else {
281                     port = "-1";
282                 }
```

```
283             } catch (RuntimeException e) {
284                 port = "-1";
285             }
286             return port;
287         }
288 
289         /**
290          * Get the port speed for an interface.
291          * @param textStr interface splitted as an array
292          * @return port speed
293          */
294         private static long getPortSpeed(String[] textStr) {
295             long portSpeed = 0;
296             String result;
297             int lastLine = textStr.length - 1;
298             for (int i = 0; i < lastLine; i++) {
299                 if ((textStr[i].indexOf(BANDWIDTH) > 0) &&
300     (textStr[i].indexOf(SPEED) > 0)) {
301                     result = textStr[i].substring(textStr[i].indexOf(BANDWIDTH) +
302     3, textStr[i].indexOf(SPEED));
303                     portSpeed = Long.valueOf(result);
304                     break;
305                 }
306             }
307             return portSpeed;
308         }
309 
310     }
```

drivers/cisco/src/main/resources/**cisco-drivers.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  ~ Copyright 2016-present Open Networking Laboratory
  ~
  ~ Licensed under the Apache License, Version 2.0 (the "License");
  ~ you may not use this file except in compliance with the License.
  ~ You may obtain a copy of the License at
  ~
  ~     http://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
  -->
<drivers>
    <driver name="cisco-netconf" extends="netconf" manufacturer="Cisco"
            hwVersion="" swVersion="IOS">
        <behaviour api="org.onosproject.net.behaviour.InterfaceConfig"

impl="org.onosproject.drivers.cisco.InterfaceConfigCiscoIosImpl"/>
        <behaviour
api="org.onosproject.net.device.DeviceDescriptionDiscovery"

impl="org.onosproject.drivers.cisco.CiscoIosDeviceDescription"/>
    </driver>
</drivers>
```

drivers/cisco/src/test/java/org/onosproject/drivers/cisco/**TextBlockParserCiscoTest.java**

```
1  /*
2   * Copyright 2016-present Open Networking Laboratory
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *     http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package org.onosproject.drivers.cisco;
18
19
20 import org.junit.Test;
21 import org.onlab.packet.ChassisId;
22 import org.onosproject.net.AnnotationKeys;
23 import org.onosproject.net.DefaultAnnotations;
24 import org.onosproject.net.DefaultDevice;
25 import org.onosproject.net.DeviceId;
26 import org.onosproject.net.Port;
27 import org.onosproject.net.PortNumber;
28 import org.onosproject.net.device.DefaultPortDescription;
29 import org.onosproject.net.device.PortDescription;
30 import org.onosproject.net.provider.ProviderId;
31 import java.io.InputStream;
32 import java.util.ArrayList;
33 import java.util.List;
34 import java.util.Scanner;
35 import static org.junit.Assert.assertEquals;
36 import static org.onosproject.net.Device.Type.SWITCH;
37 import static org.onosproject.net.DeviceId.deviceId;
38
39
40 /**
41  * Tests the parser for Netconf TextBlock configurations and replies from
42 Cisco devices.
43  */
44 public class TextBlockParserCiscoTest {
45
46     private static final PortNumber INTF1_PORT = PortNumber.portNumber(0);
47     private static final String INTF1_NAME = "FastEthernet0/0";
48     private static final PortNumber INTF2_PORT = PortNumber.portNumber(0);
49     private static final String INTF2_NAME = "Ethernet1/0";
50     private static final PortNumber INTF3_PORT = PortNumber.portNumber(0);
51     private static final String INTF3_NAME = "GigabitEthernet2/0";
52     private static final PortNumber INTF4_PORT = PortNumber.portNumber(0);
53     private static final String INTF4_NAME = "Serial3/0";
```

```
54      private static final PortNumber INTF5_PORT = PortNumber.portNumber(0);
55      private static final String INTF5_NAME = "POS4/0";
56      private static final PortNumber INTF6_PORT = PortNumber.portNumber(0);
57      private static final String INTF6_NAME = "Fddi5/0";
58      private static final Port.Type COPPER = Port.Type.COPPER;
59      private static final Port.Type FIBER = Port.Type.FIBER;
60      private static final long CONNECTION_SPEED_ETHERNET = 100000;
61      private static final long CONNECTION_SPEED_SERIAL = 1544;
62      private static final long CONNECTION_SPEED_POS = 9952000;
63      private static final long CONNECTION_SPEED_FDDI = 100000;
64      private static final boolean IS_ENABLED = true;
65      private static final boolean IS_NOT_ENABLED = false;
66      private static final String SHOW_VERSION = "/testShowVersion.xml";
67      private static final String SHOW_INTFS = "/testShowInterfaces.xml";
68      private static final String SW = "IOS C3560E 15.0(2)EJ";
69      private static final String HW = "SM-X-ES3-24-P";
70      private static final String MFR = "Cisco";
71      private static final String SN = "FOC18401Z3R";
72      private static final ProviderId PROVIDERID = new ProviderId("of", "foo");
73      private static final DeviceId DEVICE = deviceId("of:foo");
74      private static final ChassisId CID = new ChassisId();
75
76      @Test
77      public void controllersVersion() {
78          InputStream streamOrig =
79  getClass().getResourceAsStream(SHOW_VERSION);
80          String version = new Scanner(streamOrig, "UTF-
81  8").useDelimiter("\\Z").next();
82          version = version.substring(version.indexOf('\n') + 1);
83          String[] actualDetails =
84  TextBlockParserCisco.parseCiscoIosDeviceDetails(version);
85
86          assertEquals("Information could not be retrieved",
87                       getExpectedInfo(), actualInfo(actualDetails));
88      }
89
90      @Test
91      public void controllersIntfs() {
92          InputStream streamOrig = getClass().getResourceAsStream(SHOW_INTFS);
93          String rpcReply = new Scanner(streamOrig, "UTF-
94  8").useDelimiter("\\Z").next();
95          List<PortDescription> actualIntfs =
96  TextBlockParserCisco.parseCiscoIosPorts(rpcReply);
97          assertEquals("Information could not be retrieved",
98                       getExpectedIntfs(), actualIntfs);
99      }
100
101     private DefaultDevice getExpectedInfo() {
102         return new DefaultDevice(PROVIDERID, DEVICE, SWITCH, MFR, HW, SW, SN,
103 CID);
104     }
105
106     private DefaultDevice actualInfo(String[] actualDetails) {
107
108         return new DefaultDevice(PROVIDERID, DEVICE, SWITCH,
109 actualDetails[0],
110                                  actualDetails[1], actualDetails[2],
```

```
111                                     actualDetails[3], CID);
112         }
113
114         private List<PortDescription> getExpectedIntfs() {
115             DefaultAnnotations.Builder int1Annotations =
116     DefaultAnnotations.builder()
117                     .set(AnnotationKeys.PORT_NAME, INTF1_NAME);
118             DefaultAnnotations.Builder int2Annotations =
119     DefaultAnnotations.builder()
120                     .set(AnnotationKeys.PORT_NAME, INTF2_NAME);
121             DefaultAnnotations.Builder int3Annotations =
122     DefaultAnnotations.builder()
123                     .set(AnnotationKeys.PORT_NAME, INTF3_NAME);
124             DefaultAnnotations.Builder int4Annotations =
125     DefaultAnnotations.builder()
126                     .set(AnnotationKeys.PORT_NAME, INTF4_NAME);
127             DefaultAnnotations.Builder int5Annotations =
128     DefaultAnnotations.builder()
129                     .set(AnnotationKeys.PORT_NAME, INTF5_NAME);
130             DefaultAnnotations.Builder int6Annotations =
131     DefaultAnnotations.builder()
132                     .set(AnnotationKeys.PORT_NAME, INTF6_NAME);
133
134             List<PortDescription> intfs = new ArrayList<>();
135             intfs.add(new DefaultPortDescription(INTF1_PORT, IS_ENABLED, COPPER,
136     CONNECTION_SPEED_ETHERNET,
137                                                  int1Annotations.build()));
138             intfs.add(new DefaultPortDescription(INTF2_PORT, IS_NOT_ENABLED,
139     COPPER, CONNECTION_SPEED_ETHERNET,
140                                                  int2Annotations.build()));
141             intfs.add(new DefaultPortDescription(INTF3_PORT, IS_NOT_ENABLED,
142     COPPER, CONNECTION_SPEED_ETHERNET,
143                                                  int3Annotations.build()));
144             intfs.add(new DefaultPortDescription(INTF4_PORT, IS_ENABLED, COPPER,
145     CONNECTION_SPEED_SERIAL,
146                                                  int4Annotations.build()));
147             intfs.add(new DefaultPortDescription(INTF5_PORT, IS_ENABLED, FIBER,
148     CONNECTION_SPEED_POS,
149                                                  int5Annotations.build()));
150             intfs.add(new DefaultPortDescription(INTF6_PORT, IS_ENABLED, FIBER,
151     CONNECTION_SPEED_FDDI,
152                                                  int6Annotations.build()));
153             return intfs;
154         }
155
156     }
```

drivers/cisco/src/test/resources/**testShowInterfaces.xml**

```
<?xml version="1.0" encoding="UTF-8"?><rpc-reply message-id="7"
xmlns="urn:ietf:params:netconf:base:1.0"><data><cli-oper-data-
block><item><show>interfaces</show><response>
FastEthernet0/0 is up, line protocol is up
  Hardware is i82543 (Livengood), address is ca00.12b5.0008 (bia
ca00.12b5.0008)
  Internet address is 192.168.1.20/24
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:00, output hang never
  Last clearing of &quot;show interface&quot; counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
    5 minute input rate 2000 bits/sec, 1 packets/sec
    5 minute output rate 0 bits/sec, 0 packets/sec
    3589 packets input, 681498 bytes
    Received 2459 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog
    0 input packets with dribble condition detected
    2518 packets output, 242991 bytes, 0 underruns
    0 output errors, 0 collisions, 2 interface resets
    149 unknown protocol drops
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier
    0 output buffer failures, 0 output buffers swapped out
Ethernet1/0 is administratively down, line protocol is down
  Hardware is i82543 (Livengood), address is ca00.12b5.0006 (bia
ca00.12b5.0006)
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output never, output hang never
  Last clearing of &quot;show interface&quot; counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
    Output queue: 0/40 (size/max)
    5 minute input rate 0 bits/sec, 0 packets/sec
    5 minute output rate 0 bits/sec, 0 packets/sec
    0 packets input, 0 bytes
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog
    0 input packets with dribble condition detected
    0 packets output, 0 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets
```

```
     0 unknown protocol drops
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
GigabitEthernet2/0 is administratively down, line protocol is down
  Hardware is i82543 (Livengood), address is ca00.12b5.0006 (bia
ca00.12b5.0006)
  MTU 1500 bytes, BW 100000 Kbit/sec, DLY 100 usec,
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output never, output hang never
  Last clearing of &quot;show interface&quot; counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
    5 minute input rate 0 bits/sec, 0 packets/sec
    5 minute output rate 0 bits/sec, 0 packets/sec
    0 packets input, 0 bytes
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog
    0 input packets with dribble condition detected
    0 packets output, 0 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets
    0 unknown protocol drops
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier
    0 output buffer failures, 0 output buffers swapped out
Serial3/0 is up, line protocol is up
  Hardware is MCI Serial
  Internet address is 192.168.10.203, subnet mask is 255.255.255.0
  MTU 1500 bytes, BW 1544 Kbit/sec, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation HDLC, loopback not set, keepalive set (10 sec)
  Last input 0:00:07, output 0:00:00, output hang never
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
    5 minute input rate 0 bits/sec, 0 packets/sec
    5 minute output rate 0 bits/sec, 0 packets/sec
    16263 packets input, 1347238 bytes, 0 no buffer
    Received 13983 broadcasts, 0 runts, 0 giants
    2 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 2 abort
    1 carrier transitions
    22146 packets output, 2383680 bytes, 0 underruns
    0 output errors, 0 collisions, 2 interface resets, 0 restarts
POS4/0 is up, line protocol is up
  Hardware is Packet over SONET
  Internet address is 10.41.41.2/24
  MTU 4470 bytes, BW 9952000 Kbit/sec, DLY 100 usec, rely 255/255, load 1/255
  Encapsulation HDLC, crc 32, loopback not set
  Keepalive not set
  Scramble enabled
  Last input 00:00:59, output 00:00:11, output hang never
  Last clearing of "show interface" counters 00:00:14
  Queueing strategy: fifo
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
```

```
  Available Bandwidth 9582482 kilobits/sec
    5 minute input rate 0 bits/sec, 0 packets/sec
    5 minute output rate 0 bits/sec, 0 packets/sec
    0 packets input, 0 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 parity
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    1 packets output, 314 bytes, 0 underruns
    0 output errors, 0 applique, 0 interface resets
    0 output buffer failures, 0 output buffers swapped out
    0 carrier transitions
Fddi5/0 is up, line protocol is up
  Hardware is cxBus Fddi, address is 0000.0c02.adf1 (bia 0000.0c02.adf1)
  Internet address is 10.108.33.14, subnet mask is 255.255.255.0
  MTU 4470 bytes, BW 100000 Kbit/sec, DLY 100 usec, rely 255/255, load 1/255
  Encapsulation SNAP, loopback not set, keepalive not set
  ARP type: SNAP, ARP Timeout 4:00:00
  Phy-A state is active, neighbor is   B, cmt signal bits 008/20C, status ILS
  Phy-B state is active, neighbor is   A, cmt signal bits 20C/008, status ILS
  ECM is in, CFM is thru, RMT is ring_op
  Token rotation 5000 usec, ring operational 21:32:34
  Upstream neighbor 0000.0c02.ba83, downstream neighbor 0000.0c02.ba83
  Last input 0:00:05, output 0:00:00, output hang never
  Last clearing of show interface counters 0:59:10
  Output queue 0/40, 0 drops; input queue 0/75, 0 drops
    5 minute input rate 69000 bits/sec, 44 packets/sec
    5 minute output rate 0 bits/sec, 1 packets/sec
    113157 packets input, 21622582 bytes, 0 no buffer
    Received 276 broadcasts, 0 runts, 0 giants
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    4740 packets output, 487346 bytes, 0 underruns
    0 output errors, 0 collisions, 0 interface resets, 0 restarts
    0 transitions, 2 traces, 3 claims, 2 beacons</response></item></cli-oper-
data-block></data></rpc-reply>]]>]]>
```

drivers/cisco/src/test/resources/**testShowVersion.xml**

```
<?xml version="1.0" encoding="UTF-8"?><rpc-reply message-id="7"
xmlns="urn:ietf:params:netconf:base:1.0"><data><cli-oper-data-
block><item><show>hardware</show><response>
    Cisco IOS Software, C3560E Software (C3560E-UNIVERSALK9-M), Version
15.0(2)EJ, RELEASE SOFTWARE (fc1)
    Technical Support: http://www.cisco.com/techsupport
    Copyright (c) 1986-2013 by Cisco Systems, Inc.
    Compiled Fri 13-Sep-13 12:09 by prod_rel_team
    ROM: Bootstrap program is C3560E boot loader
    BOOTLDR: C3560E Boot Loader (C3560X-HBOOT-M) Version 15.0(2r)EJ1, RELEASE
SOFTWARE (fc1)
    switch01 uptime is 1 week, 3 days, 21 hours, 39 minutes
    System returned to ROM by power-on
    System restarted at 08:15:31 UTC Thu Apr 14 2016
    System image file is &quot;flash:/c3560e-universalk9-mz.150-
2.EJ.bin&quot;
    This product contains cryptographic features and is subject to United
    States and local country laws governing import, export, transfer and
    use. Delivery of Cisco cryptographic products does not imply
    third-party authority to import, export, distribute or use encryption.
    Importers, exporters, distributors and users are responsible for
    compliance with U.S. and local country laws. By using this product you
    agree to comply with applicable laws and regulations. If you are unable
    to comply with U.S. and local laws, return this product immediately.
    A summary of U.S. laws governing Cisco cryptographic products may be
found at:
    http://www.cisco.com/wwl/export/crypto/tool/stqrg.html
    If you require further assistance please contact us by sending email to
    export@cisco.com.
    License Level: lanbase
    License Type: Permanent
    Next reload license Level: lanbase
    cisco SM-X-ES3-24-P (PowerPC405) processor with 262144K bytes of memory.
    Processor board ID FOC18401Z3R
    Last reset from power-on
    2 Virtual Ethernet interfaces
    26 Gigabit Ethernet interfaces
    The password-recovery mechanism is enabled.
    512K bytes of flash-simulated non-volatile configuration memory.
    Base ethernet MAC Address       : 68:99:CD:AA:2F:80
    Model number                    : SM-X-ES3-24-P
    System serial number            : FOC18401Z3R
    Hardware Board Revision Number  : 0x00
    Switch Ports Model              SW Version            SW Image
    ------ ----- -----              ----------            ----------
    *    1 26    SM-X-ES3-24-P      15.0(2)EJ             C3560E-UNIVERSALK9-
M
</response></item></cli-oper-data-block></data></rpc-reply>]]>]]>
```

Βιβλιογραφία

1. Overview of Software Defined Networking - http://datatrend.com/software-defined-networking/
2. Northbound/Southbound interface - http://whatis.techtarget.com/definition/northbound-interface-southbound-interface
3. Xu, H., Wang, C., Liu, W., & Chen, H. (2012). NETCONF-based integrated management for internet of things using RESTful web services. *International Journal of Future Generation Communication and Networking*, *5*(3), 73-82. NETCONF Configuration Protocol. RFC 6241- https://tools.ietf.org/html/rfc6241
4. Wasserman, M. (2011). Using the NETCONF Protocol over Secure Shell (SSH).
5. Enns, R., Bjorklund, M., & Schoenwaelder, J. (2011). Network configuration protocol (NETCONF). *Network*.
6. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T. & Parulkar, G. (2014, August). ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 1-6). ACM.
7. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*,*38*(2), 69-74.
8. https://www.sdxcentral.com/sdn/definitions/what-is-openflow/ - OpenFlow overview
9. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf - SDN: The New Norm for Networks
10. https://wiki.onosproject.org/display/ONOS/Overview+of+ONOS+architecture/ - Onos architecture
11. Mekky, H., Hao, F., Mukherjee, S., Zhang, Z. L., & Lakshman, T. V. (2014, August). Application-aware data plane processing in SDN. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 13-18). ACM.
12. Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14-76.
13. Χρήστος Ι. Μπούρας, Πανεπιστημιακές σημειώσεις,μάθημα: Ευρυζωνικές Τεχνολογίες, Πανεπιστήμιο Πατρών, 2015.
14. Onos system components - https://wiki.onosproject.org/display/ONOS/System+Components
15. https://wiki.onosproject.org/display/ONOS/Device+Driver+Subsystem - ONOS Device Driver Subsystem -
16. https://wiki.onosproject.org/display/ONOS/NETCONF - Onos - Netconf

17. https://wiki.onosproject.org/display/ONOS13/NETCONF#NETCONF-Testinginfrastructure – ONOS-NETCONF classes
18. https://wiki.onosproject.org/display/ONOS/Southbound:+Protocol,+Providers,+Drivers – Soutbound: Protocol, Providers, Drivers
19. http://www.cisco.com/ - Cisco official pages
20. Inside Cisco Ios Software Architecture. - V. Bollapragada, C. Murphy, R. White
21. https://en.wikipedia.org/wiki/Software-defined_networking - SDN
22. https://en.wikipedia.org/wiki/Cisco - Cisco Ios
23. NETCONF - https://en.wikipedia.org/wiki/NETCONF