Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Αυτοματοποιημένη παραμετροποίηση δικτύου και παροχή υπηρεσίας με NETCONF/YANG

Διπλωματική εργασία

Ελευθέριου Πουλακάκη

Επιβλέπων καθηγητής: Συκάς Ευστάθιος

Αθήνα,
Μάρτιος 2017

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Αυτοματοποιημένη παραμετροποίηση δικτύου και παροχή υπηρεσίας με NETCONF/YANG

Διπλωματική εργασία

Ελευθέριου Πουλακάκη

Επιβλέπων καθηγητής: Συκάς Ευστάθιος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ... Μαρτίου 2017.

_____	_____	_____
Συκάς Ευστάθιος	Στασινόπουλος Γεώργιος	Ρουσσάκη Ιωάννα
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρη Καθηγήτρια Ε.Μ.Π.

Αθήνα,
Μάρτιος 2017

Πουλακάκης Ελευθέριος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η εισαγωγή των Προγραμματιζόμενων/Ευφυών Δικτύων (Software-Defined-Networking) φέρνει μεγάλες αλλαγές στις παρεχόμενες υπηρεσίες τηλεπικοινωνιών. Η ευρύτερη χρήση λογισμικού στον τομέα των Δικτύων Υπολογιστών και επομένως, η καταγραφής της ”λογικής” μέσα σε προγράμματα επιταχύνει και κάνει συνεπέστερες μια σειρά διαδικασιών. Η λογική αφαίρεση της λειτουργίας του δικτύου προσδίδει δυναμικότητα ακόμα και σε υποδομές με συμβατικές συκευές δικτύου. Η αυτοματοποίηση της διαδικασίας ρύθμισης μιας υποδομής δίνει τη δυνατότητα γρήγορης, άμεσης, και κατά τη ζήτηση η προσφοράς (on-demand) υπηρεσιών.

Στα πλαίσια της παρούσας διπλωματικής εργασίας σχεδιάστηκε μια υπηρεσία Quality of Service, που παρέχει σε δύο πελάτες της ένα ζητούμενο εύρος ζώνης (bandwidth on demand) με χρήση της τεχνικής διαφοροποιημένων υπηρεσιών (DSCP (Differentiated Services Domain Service) marking). Η διαδικασία ρύθμισης της δικτυακής υποδομής για να διατεθεί υπηρεσία αυτοματοποιήθηκε με τη χρήση του Ansible, και ο έλεγχος της υπηρεσίας γίνεται μέσω μια διαδικτυακής εφαρμογής (Web User Interface). Το πρωτόκολλο NETCONF χρησιμοποιείται για την ανάπτυξη της προγραμματιστικής διεπαφής με τις δικτυακές συσκευές.

Λέξεις Κλειδιά: δίκτυα υπολογιστών, ευφυή δίκτυα, προγραμματιζόμενα δίκτυα, διαχείριση παραμετροποίησης, αυτοματοποίηση, παροχή υπηρεσίας, διαχείριση δικτύου, quality of service, διαφοροποιημένες υπηρεσίες, NETCONF/YANG.

Abstract

Software-Defined networking (SDN) is a modern approach to computer networking, and is changing the way Telecommunication Services are provided. The wider usage of software on the domain of Computer Networks, makes the initialization, management and configuration of the network devices much faster and more consistent. The abstraction of the low-level operation of the network devices, makes even infrastractures consisting of conventional devices, more dynamic. The automation of the configuration and management procedures adds the capability of fast, immediate and on-demand service provisioning.

For the purposes of this diploma thesis, a Quality of Service function is designed and used to compose a bandwidth on-demand application. The network device operating level based on the differantiated services design with DSCP (Differentiated Services Code Point) marking. The automation layer of the application implemented using Ansible, and as a user interface, for control and access to the application purposes, a web application developed with the Django Web Framework. The NETCONF protocol is utilized to develop an API to configure the network devices.

Keywords: computer networks, software-defined networking, configuration management, network configuration, automation, service provisioning, quality of service, differentiated services, NETCONF/YANG.

Ευχαριστίες

Για αρχή θέλω να ευχαριστήσω τον καθηγητή κ.Ευστάθιο Συκά για την ανάθεση της διπλωματικής εργασίας και τον υποψήφιο διδάκτορα Πάρη Χαραλάμπου για την καλή συνεργασία αλλά και την ευελιξία που με άφησε να έχω στην εκπόνηση της εργασίας.

Ιδιαίτερες ευχαριστίες οφείλω στο Νίκο Κορμπάκη, που έχοντας εμπειρία από μια απαιτητική υποδομή, μου έδωσε συμβουλές και ιδέες που συνεισέφεραν στη διαμόρφωση του προγραμματιστικού κομματιού της εργασίας.

Ευχαριστώ τους "θαμώνες" του εργαστηρίου ΜΟΠ, όπου έλαβε χώρα το μεγαλύτερο κομμάτι της εκπόνησης της εργασίας αυτής, για τα ευχάριστα και ποιοτικά διαλείμματα.

Ένα μεγάλο ευχαριστώ στους φίλους και όσους ήταν και είναι κοντά μου για την υπομονή και υποστήριξη τους.

Ευχαριστώ όλους εκείνους τους συναδέλφους με τους οποίους περάσαμε άκρως ενδιαφέροντα φοιτητικά χρόνια, με την ευχή τα επόμενα μας να είναι ακόμα πιο "ενδιαφέροντα".

Τέλος, θέλω να ευχαριστήσω τους γονείς μου για τη στήριξή και την κατανόηση τους όλα αυτά τα χρόνια που ήμουν "στα θρανία" όπως και τις αδερφές μου, ειδικά για τη διαρκή, αλλά διακριτική, υπενθύμιση ότι βρίσκονται στο δίπλανο δωμάτιο.

Λευτέρης Πουλακάκης

Περιεχόμενα

Κατάλογος σχημάτων

Εισαγωγή

Τα Προγραμματιζόμενα Δίκτυα

Τα προγραμματιζόμενα/ευφυή δίκτυα (Software Defined Networking) αποτελούν μια νέα τάση που βρίσκει όλο και περισσότερο εφαρμογή στη λειτουργία και τη διαχείριση των δικτυακών υποδομών. Βασική ιδέα είναι ότι σε οποιοδήποτε επίπεδο της λειτουργίας μιας δικτυακής συσκεύης, μπορεί να εισαχθεί ο ορισμός λειτουργιών μέσω του λογισμισμικού (software). Η ιδέα βρήκε εφαρμογή από το επίπεδο αποφάσεων για τη διακίνηση των πακέτων δικτύου (Control Plane), μέχρι το επίπεδο διαχείρισης (Management Plane) μιας "παραδοσιακής" δικτυακής συσκεύης. Μπορεί ακόμα να προστεθεί, με ό,τι ιδιαιτερότητα έχει το συγκεκριμένο πεδίο, η κατηγορία των Network Function Virtualization[1] (NFV) που αποτελεί την υλοποίηση αποκλειστικά σε επίπεδο software, εικονικών δικτυακών συσκευών ή και εξειδικευμένων λειτουργιών διαχείρισης δικτυακής κίνησης.

Σκοπός της ανάπτυξης και εφαρμογής των παραπάνω είναι: α) η αυτοματοποίηση και άρα η ελαχιστοποίηση του χρόνου/κόστους ανάπτυξης, σχεδιασμού και εφαρμογής μια υπηρεσίας δικτύου, β) ο σχεδιασμός πιο εξειδικευμένων λειτουργιων στις συσκευές δικτύου που επεκτείνουν τις υπάρχουσες γ) η αποτελεσματικότερη προσφορά υπηρεσιών, παρακολούθηση και ανταπόκριση της υποδομής σε έκτακτες καταστάσεις (orchestration, healing, on-demand provisioning).

Σε μια αρχιτεκτονική Προγραμματιζόμενου Δικτύου κέντρικό ρόλο έχει ο ελεγκτής, ο οποίος αναλαμβάνει όλη τη λογική της εφαρμογής. Βασικό του καθήκον είναι να γνωρίζει ανά πάσα στιγμή την κατάσταση που βρίσκεται το δίκτυο (ποιές εφαρμογές υλοποιεί, μετρικές, έλεγχος συμφώρησης, κανόνες δρομολόγησης, κατάσταση δικτυακών κόμβων), αλλά να γνωρίζει και την επιθυμητή κατάσταση του δικτύου. Με βάση λοιπόν αυτά θα πρέπει να μπορεί να υπολογίζει τις αλλαγές που πρέπει να γίνουν στο δίκτυο ώστε να έρθει στην επιθυμητή κατάσταση και να τις φέρει εις πέρας.

Οι εφαρμογές και τα πρωτόκολλα που εντάσσονται στα Προγραμματιζόμενα/Ευφυή Δίκτυα μπορούν να χωριστούν με βάση το ρόλο που αναλαμβάνουν στη λειτουργία του δικτύου. Το OpenFlow είναι ένα παράδειγμα πρωτο-

κόλλου που αναλαμβάνει τη λειτουργία του Control Plane καθώς είναι εκείνο που παρέχει στις δικτυακές συσκεύες που υλοποιούν το επίπεδο διακίνησης δεδομένων την πληροφορία για το πώς αυτά θα προωθηθούν. Για την υλοποίηση του πρωτοκόλλου απαιτείται και εξειδικευμένος εξοπλισμός.

Η εργασία ασχολείται με το πρωτόκολλο NETCONF [2]. Το NETCONF δρα στο Management Plane των δικτυακών συσκευών. Η ανάπτυξη του NETCONF προσφέρει μηχανισμούς για την αξιόπιστη και ασφαλή παραμετροποίηση των δικτυακών συσκευών μέσω μιας προγραμματιστικής διεπαφής (API, Application Programming Interface). Παρέχεται έτσι ένα μέσο ώστε οι δικτυακές υπηρεσίες να γίνονται γρηγορότερα διαθέσιμες στον χρήστη.

Το σενάριο το οποίο υλοποιήθηκε για τους σκοπούς της εργασίας, ήταν αυτό της παροχής εύρους ζώνης, αναλόγως ζήτησης (bandwidth on demand), με χρήση της εφαρμογής Quality of Service με την τεχνική DSCP [3] (Differantiatd Services Code Point) marking. Η υπήρεσία απαιτεί τη ρύθμιση του δικτύου από άκρο σε άκρο (end-to-end) για να μπορέσει να λειτουργήσει αποτελεσματικά. Η αυτοματοποίηση της διαδικασίας ρύθμισης των συσκευών θα βελτιστοποιεί το χρόνο εφαρμογής της υπηρεσίας, και μειώνει το σφάλμα.

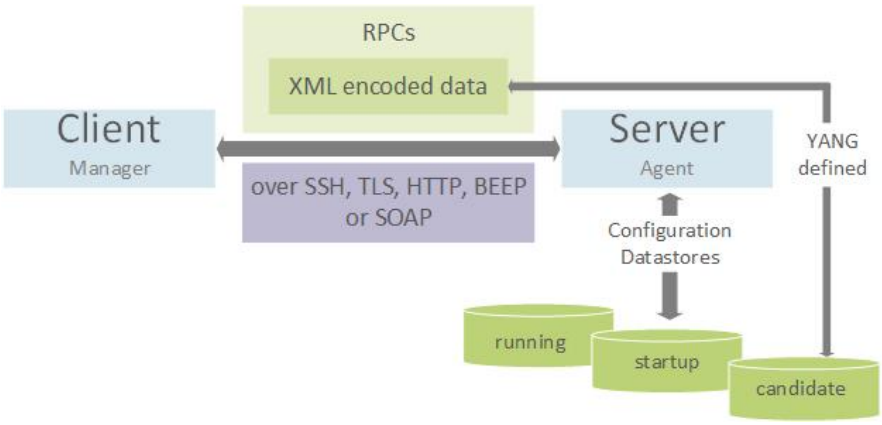
Στο ρόλο του ελεγκτή χρησιμοποιείται το Ansible [4], εργαλείο διαχείρισης ρυθμίσεων (configuration management) και αυτοματισμού (automation), σαν γραφική διεπαφή με το χρήστη της εφαρμογής (Graphical User Interface) αναπτύχθηκε μια απλή διαδικτυακή εφαρμογή με χρήση του Django Framework [5]. Η υλοποίηση του πρωτοκόλλου NETCONF έγινε μέσα στον πελάτη (client) cisco_client που εξειδικεύτηκε για τις ανάγκες του εξοπλισμού του εργαστηρίου.

1

Θεωρητικό Υπόβαθρο

1.1 Το πρωτόκολλο NETCONF

Το πρωτόκολλο NETCONF[2] αναπτύχθηκε από IETF[6] (Internet Engineering Task Force) στην προσπάθεια να βρεθεί ένας μηχανισμός διαχείρισης, εγκατάστασης ρυθμίσεων δικτυακών συσκευών και παρακολούθησης της λειτουργίας τους. Βασισμένο στο SNMP (Simple Network Management Protocol), ακολουθεί παρόμοια διαστρωμάτωση στα περιεχόμενά του. Διατηρεί μια ανεξάρτητη περιγραφή/μοντελοποίηση δεδομένων που βασίζεται στη γλώσσα YANG (Yet Another Next Generation), χρησιμοποιεί ασφαλές υπόστρωμα μεταφοράς (TLS, SSH, HTTPS) και τα μηνύματα του κωδικοποιούνται με χρήση της γλώσσας XML (Extensive Markup Language). Οι κλήσεις του NETCONF είναι RPCs (Remote Procedure Calls), πράγμα που σημαίνει ότι το πρωτόκολλο παρέχει το μηχανισμό ώστε οι κλήσεις διαδικασιών να εκτελούνται από τη δικτυακή συσκευή.

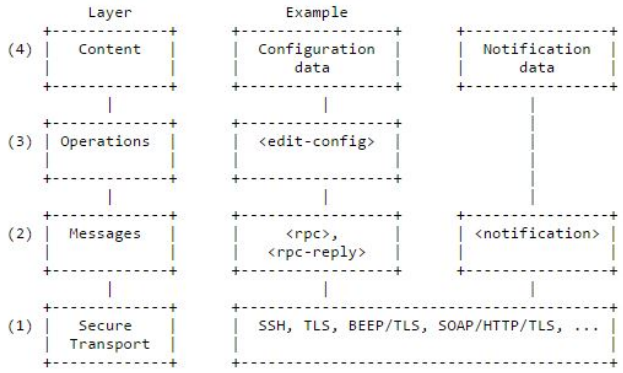


Σχήμα 1.1: Επισκόπηση Λειτουργίας NETCONF

Το μοντέλο λειτουργίας του πρωτοκόλλου είναι αυτό του πελάτη-εξυπηρετητή (client-server). Ο εξυπηρετητής διατηρεί τα μοντέλα των δεδομένων, τις τιμές τους μέσα στα σύνολα δεδομένων, τις διεργασίες (RPCs), και

τον τρόπο που αυτές επιδρούν στα δεδομένα. Ο πελάτης απλά καλεί τις διεργασίες αυτές με τα ζητούμενα ορίσματα για να επιδράση πάνω στα δεδομένα και επομένως πάνω στη συσκευή. Το σύνολο των κλήσεων που μπορεί να φέρει εις πέρας ο εξυηρετητής, αλλά και το μοντέλο που ακολουθούν τα δεδομένα του, διαφημίζονται στον πελάτη κατά την εγκατάσταση της σύνδεσης.

1.1.1 Διαστρωμάτωση



Σχήμα 1.2: Τα στρώματα του πρωτοκόλλου NETCONF

- Το στρώμα ασφαλούς μετάδοσης παρέχει επικοινωνία ανάμεσα στον εξυπηρετητή και τον πελάτη και μπορεί να επιλεγεί οποιοδήποτε πρωτόκολλο μπορεί να προσφέρει τις προϋποθέσεις.

- Το στρώμα των διεργασιών είναι οι ίδιες διεργασίες που καλούνται στον εξυπηρετητή μέσω των RPCs.

- Το στρωμα περιεχομένου περιγράφει τα δεδομένα που κωδικοποιούνται μέσα στα μηνύματα NETCONF. Η γλώσσα YANG προσφέρει την περιγραφή των δεδομένων του εξυπηρετητή και η γλώσσα XML την κωδικοποίηση των μηνυμάτων που μεταδίδονται ανάμεσα σε πελάτη-εξυπηρετητή.

1.1.2 Στρώμα Περιεχομένου

Η πληροφορία που μπορεί να ανακτηθεί από ένα σύστημα σε λειτουργία είναι διαχωρισμένη σε δύο κατηγορίες. Τα δεδομένα ρυθμίσεων (Configuration Data) είναι δεδομένα που μπορούν να γραφτούν από τον πελάτη και αλλάζουν την κατάσταση του συστήματος από την τρέχουσα σε μια άλλη επιθυμητή (state data). Τα δεδομένα κατάστασης είναι μόνο για ανάγνωση, και περιέχουν πληροφορίες και στατιστικά για την κατάσταση του εξυπηρετητή.

Ο διαχωρισμός γίνεται κυρίως για την ευκολία που προσφέρει στον ορισμό των δεδομέων, αλλά και για να αντιμετωπιστούν μια σειρά προβλημάτων, όπως για παράδειγμα ένας πελάτης που προσπαθήσει να γράψει σε μια μη εγγράψιμη θέση, τα στατιστικά που θα προσπαθήσει να ανακτήσει κάποιος πελάτης να μπερδευτούν με τα δεδομένα για τις ρυθμίσεις κ.ο.κ. Όπως θα δούμε παρακάτω, ο διαχωρισμός αυτός διαπερνά και το στρώμα στρώμα διεργασιών, καθώς προσφέρονται δύο διαφορετικες διεργασίες για την ανάκτηση των δεδομένων. Η <get-config> αναφερεται μόνο σε δεδομένα ρυθμίσεων, ενώ η <get> και στις δύο κατηγορίες δεδομένων.

Μοντελοποίηση Δεδομένων

Το NETCONF ορίζει μια σειρά από σύνολα δεδομένων(datastores) στα οποία επιτρέπεται η επεξεργασία. Το σύνολο των δεδομένων ρυθμίσεων (Configuration Datastore) ορίζεται ως το σύνολο των δεδομένων ρυθμίσεων που μπορούν να τροποποιηθούν από το διαχειριστή ώστε να μεταφερθεί μια συσκευή δικτύου, και γενικότερα ο εξυπηρετητής NETCONF, από την τρέχουσα κατάσταση του, σε μια άλλη επιθυμητή. Τα σύνολα αυτά μπορεί να είναι πολλάπλά. Για παράδειγμα ένας εξυπηρετητής μπορεί να διατηρεί τις τρέχουσες ρυθμίσεις (running), τις υποψήφιες (candidate), αντίγραφα ασφαλείας (back-up config) και ότι άλλο επιθυμεί να ορίσει ο διαχειριστής.

Τα δεδομένα που περιγράφουν την τρέχουσα κατάσταση μιας συσκευής, τις μετρικές της (throuhgput, packet countes, κ.ο.κ) ονομάζονται State Data(δεδομένα κατάστασης) και δεν δέχονται τροποποίησης, παρά μόνο ανάκτησης από το διαχειριστή για παρακολούθηση (monitoring).

Τα δεδομένα τρεχουσών ρυθμίσεων (Running Configuration Datastore) περιλαμβάνουν τις πλήρεις ρυθμίσεις που είναι ενεργές τη στιγμή εκείνη στην δικτυακή συσκευή. Το σύνολο των ρυθμίσεων αυτό είναι μοναδικό σε κάθε συσκευή και υπάρχει πάντα σε αυτή και καλείται με το στοιχείο <running> από την XML.

Η μοντελοποίηση των δεδομένων (data models) θα αναπτυχθεί εκτενώς

στο κεφάλαιο που αναφέρεται στη γλώσσα YANG (Yet Another Next Generation). Προϋποτίθεται κατά την επικοινωνία των δύο πλευρών (εξυπηρετητής, πελάτης) ότι η περιγραφή των δεδομένων είναι γνωστή κατά την εκκίνηση και στα δύο μερη. Το NETCONF μεταφέρει τα δεδομένα για τις ρυθμίσεις μέσα στο στοιχείο <config> και τα στοιχεία που χρησιμοποιούνται είναι εξειδικευμένα για την κάθε συσκευή. Ο εξυπηρετητής ανακοινώνει κατά την εκκίνηση της σύνδεσης, στις δυνατότητες του (capabilities) και το μοντέλο (data model) που χρησιμοποιεί.

1.1.3 Στρώμα διεργασιών

Το NETCONF χρησιμοποιεί τις RPC (Remote Procedure Calls), σαν το μηχανισμό κλήσης διεργασιών στον εξυπηρετητή από τον πελάτη. Αυτό σημαίνει πως οι διεργασίες αυτές υλοποιούνται και εφαρμόζονται στον εξυπηρετητή, και ο πελάτης τις καλεί με ένα όνομα και τις μεταβλητές που απαιτεί αυτή ως εισόδους.

Ο εξυπηρετητής είναι υποχρεωμένος να ελέγξει την ορθότητα της κλήσης, την εφαρμογή της αναλόγως, αλλά και την ενημέρωση του πελάτη για οποιοδήποτε σφάλμα υπήρξε, και αν όλα κύλησαν ομαλά να αποστείλει ένα μήνυμα θετικής έκβασης.

Όπως θα αναλυθεί και στο αντίστοιχο κεφάλαιο, το περιεχόμενο των μηνυμάτων είναι δομημένο. Για να γίνει αυτό χρησιμοποιείται η γλώσσα XML[7] (eXtensive Markup Language). Για την κλήση μιας RPC, πρέπει ένα μήνυμα NETCONF να έχει ένα στοιχείο <rpc> που θα περιέχει το όνομα της κλήσεις, και κάτω από αυτό το στοιχείο τις μεταβλητές. Εδώ παρουσιάζονται οι κλήσεις. Στις μεταβλητές και στον τρόπο που εισάγονται μέσα σε ένα μήνυμα αναλώνεται περισσότερο το κεφάλαιο για τη γλώσσα YANG.

<rpc>, <rcp-reply

Το στοιχείο <rpc> χρησιμοποιείται για να συμπεριλάβει την κλήση του πελάτη προς τον εξυπηρετητή. Το <rpc-reply> είναι το στοιχείο που περιλαμβάνει την απάντηση του εξυπηρετητή. Το ”message-id” είναι ένας τυχαία επιλεγμένος αύξοντας ακέραιος, που παίζει το ρόλο αναγνωριστικού για τα δύο μέρη της σύνδεσης. Ο εξυπηρετητής τον αποθηκεύει και τον χρησιμοποιεί ως το αντίστοιχο ”message-id” της απάντησης του στο στοιχείο <rpc-reply>. Για παράδειγμα:

```
<rpc message-id=""101
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <some-method>
      <!-- method parameters here... -->
      </some-method>
</rpc>
```

Το όνομα και οι παράμετροι της RPC κωδικοποιούνται ως περιεχόμενα του στοιχείου <rpc>. Το όνομα είναι στοιχείο κάτω από το <rpc> και οι παράμετροι του με τη σειρά τους εντάσσονται κάτω από το στοιχείο αυτό. Το επόμενο παράδειγμα καλεί την διεργασία my-own-method με παραμέτρους my-first-parameter και another-parameter.

```
<rpc message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <my-own-method xmlns="http://example.net/me/my-own/1.0">
      <my-first-parameter>14</my-first-parameter>
      <another-parameter>fred</another-parameter>
   </my-own-method>
</rpc>
```

Και αυτό το παράδειγμα καλεί την διεργασία </get> χωρίς παραμέτρους.

```
<rpc message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get/>
</rpc>
```

Σύμφωνα με τα παραπάνω η απάντηση του εξυπηρετητή στην τελευταία κλήση θα είναι η εξής.

```
<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      <data>
      <!-- contents here... -->
   </data>
</rpc-reply>
```

<rpc-error>

Το στοιχείο <rpc-error> περιέχεται στο μήνυμα που αποστέλλεται από τον εξυπηρετητή στην περίπτωση που υπάρχει σφάλμα κατά την επεξεργασία του μηνύματος που περιέχει αίτημα με το στοιχείο <rpc>. Σε περίπτωση πολλαπλών σφαλμάτων μπορεί να συμπεριλάβει όλα σε ένα μήνυμα. Παρόλα αυτά δεν είναι υποχρεωμένος από το πρωτόκολλο να το κάνει. Περιπτώσεις σφαλμάτων μπορεί να είναι:

- ένα μήνυμα με παραμορφωμένη XML δομή,
- ένα μήνυμα με λάθος όνομα κλήσης, ή μεταβλητών,
- μη επιτρεπόμενη στον πελάτη ενέργεια,

Κάθε σφάλμα εντάσσεται σε ένα στρώμα του πρωτοκόλλου. Στο στοιχείο <rpc-error> μπορεί να συμπεριληφθεί λοιπόν η πληροφορία για το στρώμα που έχει σφάλει, στο error-type:

- transport (layer: secure transport)
- rpc (layer: messages)
- protocol (layer: operations)
- application (layer: content)

<ok>

Το στοιχείο <ok> περιέχεται στο μήνυμα που αποστέλλεται από τον εξυπηρετητή κάτω από το στοιχείο <rpc-reply> όταν δεν υπήρξε οποιοδήποτε σφάλμα ή οποιαδήποτε προειδοποίηση κατά την επεξεργασία του αιτήματος του πελάτη.

1.1.4 Σημαντικές Διεργασίες

Το NETCONF προσφέρει ένα μικρό σύνολο από πολύ βασικές διεργασίες για τη διαχείριση και την ανάκτηση δεδομένων από μια δικτυακή συσκευή. Η βασική έκδοση του πρωτοκόλλου περιλαμβάνει διεργασίες για να ανακτηθεί (configuration και operational datastore), αντιγραφεί, διαγραφεί το σύνολο των δεδομένων ρυθμίσεων (configuration datastore μόνο). Οι διεργασίες αυτές είναι οι: get, get-config, edit-config, copy-config, delete-config, lock, unlock, close-session, kill-session.

<get-config>

Ανακτά όλο του επιλεγμένου συνόλου δεδομένων ρυθμίσεων (configuration datastore). Η κλήση με παράμετρο source ορίζει το είδος των δεδομένων που θα ανακτηθούν, και με την παράμετρο filter επιλέγεται ένα συγκεκριμένο κομμάτι του. Η ύπαρξη της πρώτης είναι υποχρεωτική. Αν η δεύτερη δεν ορίζεται ανασύρρονται όλα τα δεδομένα. Παρακάτω φαίνεται ένα μήνυμα NETCONF που καλέι την <get-config> με source τα δεδομένα τρέχουσων ρυθμίσεων (running-config), και συγκεκριμένα ανακτά μόνο το αντικείμενο users που είναι το όρισμα του <filter>.

```
<rpc message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get-config>
      <source>
            <running/>
            </source>
      <filter type="subtree">
            <top>
                  <users/>
            </top>
      </filter>
   </get-config>
</rpc>
```

Η απάντηση του εξυπηρετητή:

```
<rpc-reply message-id=""101
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <data>
      <top>
            <users>
            <user>
                  <name>root</name>
                  <type>superuser</type>
                  <full-name>Charlie Root</full-name>
                  <company-info>
                        <dept>1</dept>
                        <id>1</id>
                  </company-info>
            </user>
       <!-- additional <user> elements appear here -->
            </users>
      </top>
      </data>
</rpc-reply>
```

<edit-config>

Η διεργασία <edit-config> αλλάζει ένα κομμάτι των δεδομένων ενός στοχευμένου σύνολου ρυθμίσεων (target configuration datastore). Το σύνολο που στοχεύει η διεργασία ανανεώνεται αν υπάρχει, και αν δεν υπάρχει το δημιουργεί. Η διεργασία προσφέρει μια σειρά επιλογών για την εξειδίκευση της λειτουργίας της. Μέσω της ιδιότητας operation ορίζεται ο τρόπος με τον οποίο θα γίνει η επεξεργασία των δεδομένων. Η μόνη υποχρεωτική παράμετρος που δέχεται είναι η <target> που δηλώνει την datastore στην οποία θα γίνουν οι αλλαγές. Μπορεί να οριστεί και η προεπιλεγμένη λειτουργία μέσω της default-operation η οποία έχει ισχύ μόνο για την τρέχουσα σύνδεση NETCONF.

- merge: οι ρυθμίσεις που περιγράφονται στο μήνυμα συγχωνεύεται με το υπάρχον σύνολο ρυθμίσεων. Αυτή είναι και η προεπιλεγμένη λειτουργία,
- replace: οι ρυθμίσεις που περιγράφονται στο μήνυμα αντικαθιστούν οποιαδήποτε σχετική ρύθμιση βρεθεί στα δεδομένα ρυθμίσεων. Αν δεν βρεθεί κάτι τότε δημιουργείται,

- create: δημιουργία δεδομένων.

```
<rpc message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <edit-config operation="replace">
      <target>
            <running/>
      </target>
      <config>
            <top>
            <interface>
                  <name>Ethernet0/0</name>
                  <mtu>1500</mtu>
            </interface>
            </top>
      </config>
  </edit-config>
</rpc>
```

<lock> και <unlock>

Η διεργασία αυτή προσφέρει τη δυνατότητα στον πελάτη να "κλειδώσει" ολόκληρο το σύστημα που αποθηκεύει τα σύνολα των ρυθμίσεων στον εξυπηρετητή. Η λειτουργία του "κλειδώματος" είναι αυτή που περιγραφή και το όνομα. Δηλαδή για όσο ένας πελάτης διαπραγματεύται τις αλλαγές των ρυθμίσεων ενός εξυπηρετητή, δεν μπορεί οποιοσδήποτε άλλος να το κάνει.Ο σκοπός του "κλειδώματος" είναι να εμποδίσει την ανάδραση με άλλες συνδέσεις NETCONF από άλλους πελάτες ή ακόμα και φυσικά πρόσωπα που διαχειρίζονται μια συσκευή μέσω γραμμής εντολών (CLI). Δέχεται την παράμετρο <target> που λειτουργεί όπως ήδη περιγράφηκε.

```
<rpc message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <lock>
      <target>
            <running/>
```

```
      </target>
   </lock>
</rpc>
```

Η <unlock> αφαιρεί ένα κλείδωμα που έχει θέσει η διεργασία <lock>. Δέχεται την παράμετρο <target> και αποτυγχάνει αν το σύνολο ρυθμίσεων (configuration datastore) δεν ήταν κλειδωμένο.

Άλλες σημαντικές διεργασίες

Η <copy-config> αντιγράφει ολόκληρο το σύνολο των ρυθμίσεων και δέχεται τις ίδιες παραμέτρους με την προηγούμενη.

Η <delete-config> διαγράφει ένα σύνολο ρυθμίσεων. Δέχεται την παράμετρο target, στην οποία απαγορεύται η τιμή <running>.

Η <get> ανακτά το σύνολο δεδομένων, οποιουδήποτε είδους. Δέχεται την παράμετρο filter.

Η <close-session> καλεί σε αμοιβαίο τερματισμό της σύνδεσης.

Κλήση

```
      <rpc message-id="101" xmlns="
          urn:ietf:params:xml:ns:netconf:base:1.0">
      <close-session/>
   </rpc>
```

Απάντηση

```
<rpc-reply message-id="101"
            xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <ok/>
</rpc-reply>
```

Η σύνδεση NETCONF τερματίζει μετά και από την απάντηση του εξυπηρετητή.

Η <kill-session> κλείνει τη σύνδεση, αφαιρεί όλα τα κλειδώματα και σταματάει οποιαδήποτε διαδικασία έχει ξεκινήσει και επαναφέρει στην προηγούμενη κατάσταση τα σύνολα ρυθμίσεων.

1.1.5 Στρώμα Μετάδοσης NETCONF

Το NETCONF δεν υλοποιεί δικό του στρώμα μετάδοσης, αλλά χρησιμοποιεί υπάρχοντα πρωτόκολλα που δρουν στο επίπεδο μετάδοσης. Υπάρχουν διαφορετικές εκδοχές, ανάλογα με τις απαιτήσεις της εφαρμογής, αλλά όπως και να έχει το πρωτόκολλο που θα χρησιμοποιηθεί πρέπει να προσφέρει τα παρακάτω.

Connection Oriented

Το NETCONF ως πρωτόκολλο βασισμένο στην σύνδεση (connection-oriented) πρέπει να έχει εξασφαλισμένη από τη σύνδεση την αξιόπιστη και εν σειρά μεταφορά δεδομένων, πράγμα σημαντικό δεδομένου ότι οι συνδέσεις είναι μεγάλες σε διάρκεια. Επιπρόσθετα οταν μια δικτυακή συσκευή διαχειρίζεται μέσω NETCONF πρέπει να είναι σε θέση να διατηρεί την απομόνωση των δεδομένων ανάμεσα σε συνδέσεις που ανοίγουν και κλείνουν. Επομένως όταν μια σύνδεση κλείνει (είτε ηθελημένα, είτε χάνεται) ο εξυπηρετητές πρέπει αυτομάτως να σταματάει οποιαδήποτε διεργασία ήταν σε εξέλιξη. Για παράδειγμα εάν ένας πελάτης έχει ”κλειδώσει” το σύνολο των ρυθμίσεων ενός εξυπηρετητή για να τις αλλάξει με ασφάλεια, πρέπει αν η σύνδεση εκπέσει να αναιρέσει αυτό το κλείδωμα. Ο σκοπός όλου αυτού είναι να γίνει η ανάκτηση του συστήματος σε περίπτωση σφάλματος πιο εύκολη.

Πιστότητα, Ακεραιότητα, Εμπιστευτικότητα

Πρέπει να προσφέρεται η πιστοποίηση των συμμετεχόντων (πελάτη, εξυπηρετητή), η ακεραιότητα των δεδομένων, και η εμπιστευτικότητα. Το NETCONF βασίζεται αποκλειστικά στο πρωτόκολλο που υλοποιεί την μετάδοση για αυτό το σκοπό και δεν υλοποιεί κάποιον δικό του μηχανισμό. Κλασικά παραδείγματα υλοποιήσεων είναι η χρήση του Transport Layer Security (TLS) [RFC5246] και του Secure Shell (SSH) [RFC4251]. Αυτό απαλλάσσει τον εξυπηρετητή και τον πελάτη από τον να υλοποιήσουν πρόσθετη λειτουργικότητα για να χρησιμοποιησούν το NETCONF, φέρνοντας το πρωτόκολλο πιο κοντά

σε μια απλή λειτουργία.

Αποτέλεσμα της διαδικασίας πιστοποίησης πρέπει να είναι ένας πελάτης με γνωστή ταυτότητα και γνωστά δικαιώματα επί του εξυπηρετητή. Η ταυτότητα του πελάτη συχνά ονομάζεται NETCONF username, που είναι ένα σύνολο χαρακτήρων και η διαδικασία εξαγωγής τους αφορά ξανά το υφιστάμενο πρωτόκολλο μετάδοσης. Το SSH θεωρείται υποχρεωτικά υποστηριζόμενο από το NETCONF.

Σε μια σειρά πρωτοκόλλων μετάδοσης προϋπάρχουν οι μηχανισμοί για την ικανοποίηση των παραπάνω. Επομένως το NETCONF χρησιμοποιεί ένα υφιστάμενο στρώμα μετάδοσης υλοποιημένο από ένα πρωτόκολλο εκ των SSH, BEEP, SOAP, HTTPS.

1.1.6 Η υλοποίηση NETCONF over SSHv2

Πριν ξεκινήσει η σύνδεση NETCONF, εγκαθίσταται η σύνδεση SSH. Το username που χρησιμοποιεί το SSH για να εγκαταστήσει τη σύνδεση είναι αυτό που χρησιμοποιεί κα το NETCONF για τη δική του. Η σύνδεση SSH προσφέρει στο NETCONF ένα κανάλι τύπου session. Μόλις εγκατασταθεί η σύνδεση SSH ο πελάτης καλεί το υποσύστημα netconf του SSH στον εξυπηρετητή (netconf subsystem). Το SSH λειτουργεί στην προκαθορισμένη TCP θύρα 22. Για το NETCONF έχει δοθεί από την IANA (Internet Assigned Numbers Authority) η 830. Παρόλα αυτά αναλόγως τον κατασκευαστή συναντάται η χρήση για τη σύνδεση NETCONF και των δύο θυρών. Ένας χρήστης θα μπορούσε να καλέσει έναν εξυπηρετητή ως εξής από μια γραμμή εντολών.

```
[user@client]$ ssh -s server.example.org -p 830 netconf
```

Η παραπάνω εντολή θα εμφάνιζε το HELLO μήνυμα από την πλευρά του εξυπηρετητή. Όταν πλέον εγκατασταθεί η σύνδεση SSH ξεκινάει η εγκατάσταση της NETCONF σύνδεσης με HELLO μήνυμα όπως έχει περιγραφεί παραπάνω.

Πλαισιοποίηση

Η συμβολοσειρά που χρησιμοποιείται για το διαχωρισμό των μηνυμάτων NETCONF είναι η]]>]]>. Όλα τα μηνύματα, μαζί και το HELLO, πρέπει να φέρουν τους χαρακτήρες στο τέλους τους, αλλιώς ο εξυπηρετητής θα περιμένει μέχρι να το δεχτεί, οδηγώντας σε παραμόρφωση των μηνυμάτων και του περιεχομένου τους με αποτέλεσμα να αποτύχει μια διεργασία που κα-

λείται από τον πελάτη ή/και να κλείσει η σύνδεση. Αυτό εφαρμόζεται αν ο πελάτης και ο εξυπηρετητής χρησιμοποιούν την βασική έκδοση της υλοποίησης NETCONF (netconf:base:1.0). Αν χρησιμοποιούν την έκδοση 1.1 (netconf:base:1.1) τότε ακολουθείται ένας μηχανισμός κατάτμησης των μηνυμάτων με κανόνα ABNF(Augmented Back-Naur Form)[RFC5234]. Το πρώτο παράδειγμα ακολουθεί την έκδοση 1.0 και το δεύτερο την 1.1.

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <capabilities>
            <capability>
            urn:ietf:params:netconf:base:1.1
      </capability>
   </capabilities>
</hello>
]]>]]>
```

Είναι ευθύνη του πρωτοκόλλου SSH να κάνει την πιστοποίηση του πελάτη, αλλά και να μεταφέρει τα μηνύματα ακέραια και προστατευμένα. Ο μηχανισμός τέλους μηνυμάτων με τους χαρακτήρες "]]>]]>" μπορεί να φέρει σε περιπτώσεις ζητήμα ασφάλειας κυρίως από επιθέσεις έγχυσης (injection). Για το λόγο αυτό αναπτύχθηκε η μέθοδος της έκδοσης 1.1, αλλά λόγω της πολύ σπάνιας και μικρής απειλής χρησιμοποιείται ευραίως η μέθοδος της έκδοσης 1.0.

1.1.7 NETCONF Capabilities

Οι δυνατότητες που έχουν τα δύο μέρη της σύνδεσης NETCONF διαφημίζονται κατά την έναρξη της σύνδεσης. Κατά την έναρξη της διαδικασίας εγκατάστασης της σύνδεσης, αποστέλλεται ένα μήνυμα (HELLO message) και από τις δύο πλευρές, το οποίο φέρει σε μορφή XML, το σύνολο των δυνατοτήτων που μπορούν να ανταπεξέλθουν. Κατά τη διάρκεια της σύνδεσης τα δύο μέρη χρησιμοποιούν μόνο τις κοινές δυνατότητες, ενώ αν δεν βρεθεί καμία κοινή η σύνδεση δεν εγκαθίσταται. Ένα παράδειγμα μηνύματος που φέρει τα capabilities μπορεί να είναι:

```
<capabilities>
```

```
      <capability>urn:ietf:params:netconf:base:1.0</capability>
</capabilities>
```

Το μήνυμα HELLO

Το μήνυμα όπως και όλα τα υπόλοιπα κωδικοποιείται σε XML. Το εναρκτήριο στοιχείο είναι το <hello>. Κάτω από το στοιχείο αυτό τοποθετούνται κάτω από το στοιχείο <capabilities> η λίστα από τις δυνατότητες του πελάτη ή εξυπηρετητή με το όνομα της σε μορφή URN. Επίσης τοποθετείται και το <session-id> που είναι ένα αριθμητικό αναγνωριστικό της σύνδεσης το οποίο κρατάει ο εξυπηρετητής για να μπορεί να διαχωρίσει διαφορετικές συνδέσεις NETCONF. Ένα παράδειγμα:

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <capabilities>
     <capability>
      urn:ietf:params:netconf:base:1.1
     </capability>
     <capability>
      urn:ietf:params:netconf:capability:startup:1.0
     </capability>
     <capability>
      http://example.net/router/2.3/myfeature
     </capability>
    </capabilities>
    <session-id>4</session-id>
  </hello>
```

Το μήνυμα αποστέλλεται αμέσως μόλις το υφιστάμενο πρωτόκολλο μετάδοσης εγκαταστήσει τη δική του σύνδεση.

1.2 Η γλώσσα YANG

Η YANG[8] (Yet Another Next Generation) είναι μια γλώσσα περιγραφής/μοντελοποίησης δεδομένων (data modeling language). Σκοπός της είναι

η περιγραφή των δεδομένων ρυθμίσεων και κατάστασης (configuration και state data) του πρωτοκόλλου NETCONF. Παίζει για το NETCONF τον ίδιο ρόλο που παίζει η SMI (Structure of Managment Information) για το SNMP. Η ομάδα που ανάπτυξε την NG-SNMP SMI (Next Generation – Simple Network Managment Protocol SMI) ασχολήθηκε και με την ανάπτυξη της YANG, εξού και το όνομά της.

Σκοπός της ανάπτυξής της ήταν μια γλώσσα που θα μπορεί να ανταπεξέλθει στη διαχείριση των ρυθμίσεων ενός δικτύου (configuration management) καθώς καμία από τις υπάρχουσες αντίστοιχες γλώσσες (XML Schema, UML κ.α.) δεν στόχευε ειδικά σε αυτή τη χρήση. Η δυνατότητα να είναι ευκολοδιάβαστες για τον άνθρωπο και να προσφέρουν γρήγορη αξιολόγηση (validation) ήταν το ζητούμενο από τη γλώσσα αυτή.

Η περιγραφή των δεδομένων στη YANG είναι ιεραρχική. Η δομή λοιπόν είναι δενδρική και κάθε περιγραφή δεδομένου αποτελεί έναν κόμβο (node). Η σύνταξή της είναι ανάλογη με αυτή της JSON (JavaScript Object Notation).

1.2.1 Η οργάνωση αρχείων

YANG Data Models

Τα YANG Models είναι πλήρεις περιγραφές των δεδομένων των ρυθμίσεων ενός εξυπηρετητή NETCONF.

YANG Modules

Ένα αρχείο YANG το οποίο περιέχει ένα σύνολο από μοντέλα δεδομένων και μπορεί να περιγράψει ένα κομμάτι ή το σύνολο των δεδομένων ρύθμισης ενός εξυπηρετητή NETCONF. Η υπομονάδα περιγράφει ένα υποσύνολο των ρυθμίσεων. Μια μονάδα YANG μπορεί να αποτελείται από ένα σύνολο υπομονάδων.

Η οργάνωση αυτή των μονάδων YANG είναι ένα πολύ βασικό χαρακτηριστικό της. Οι μονάδες και υπομονάδες είνα αυτόνομες. Αυτό σημαίνει ότι μια δικτυακή συσκευή (ή οποιοσδήποτε NETCONF εξυπηρετητής) μπορεί να συγκροτήσει την σύνολη μονάδα YANG της (YANG module) ”συναρμολογώντας” το από υπάρχουσες υπομονάδες.

Αυτό σημαίνει καταρχήν ότι η YANG είναι μια γλώσσα σπονδυλωτή (modular). Αυτό σημαίνει πως κάθε κατασκευαστής δε χρειάζεται να γράφει εξαρχής τις μονάδες YANG για τους NETCONF εξυπηρετητές που αναπτύσ-

σουν. Βέβαια αυτό λόγο ανταγωνισμού δεν εφαρμόζεται και σε μεγάλο βαθμό σε αυτό το επίπεδο. Παρόλα αυτά υπάρχουν κινήσεις από παραγωγούς στο να υιοθετούν κοινά και "ανοιχτά" μοντέλα και μονάδες YANG για την περιγραφή των ρυθμίσεων των συσκευών τους.

Ο διακηρυγμένος στόχος της YANG άλλωστε ήταν να υπερβεί, χρησιμοποιώντας αυτό το βασικό χαρακτηριστικό της, την αδυναμία του SNMP και των MIB σε αυτό το κομμάτι. Οι διαφορετικές και αποκλειστικές MIB που παρήγηγαν οι κατασκευαστές (πέρα από τη δυσκολία στο διάβασμα και το γράψιμό τους) έκανε την ανάπτυξη εργαλείων με βάση το SNMP το ίδιο "κλειστή". Αποτέλεσμα η δυσχρησία σε ένα περιβάλλον ετερογενές ως προς τους κατασκευαστές (όπως είναι οι υποδομές παρόχων διαδικτύου κλπ).

Το OpenConfig[9] είναι μια κίνηση διαχειριστών δικτύων και εταιρείων στην κατεύθυνση ανάπτυξης κοινών μοντέλων YANG. Στόχευση του η ανάπτυξη ουδέτερων ως προς τον κατασκευαστή για τη διαχείριση και ρύθμιση δικτυακών συσκευών. Μια σειρά κατασκευαστών έχει υιοθετήσει μερικώς ή πλήρως το ανοιχτό αυτό πρότυπο.

1.2.2 Άλλες χρήσεις

Η YANG ως γλώσσα περιγραφής δεδομένων, πέρα από τη χρήση της εντός των δικτυακών συσκευών, χρησιμοποιείται και στο εσωτερικό πολλών εφαρμογών διαχείρισης δικτύων. Όπως είπώθηκε και στα προηγούμενα το NETCONF βασίζεται σε σύνολα δεδομένων (datastores) που περιγράφουν ρυθμίσεις και καταστάσεις λειτουργίας των δικτυακών συσκευών. Άρα η YANG μπορεί να χρησιμοποιηθεί για την περιγραφή των δεδομένων και στο εσωτερικό των εφαρμογών αυτών. Αυτό από τους σύγχρονους ελεγκτές ονομάζεται "στρώμα αφαίρεσης εφαρμογών" (Software Abstraction Layer, SAL), το οποίο επειδή ορίζεται από μοντέλα YANG ονομάζεται πλήρως MD-SAL (Model-Driven SAL).

Δεδομένου ότι ο ελεγκτής στο εσωτερικό του περιγράφει τις υπηρεσίες του με χρήση μοντέλων YANG τον κάνει να προσομοιάζει με ένα σε έναν εξυπηρετητή NETCONF. Έτσι δίνεται η δυνατότητα στον άνθρωπο πίσω από τον controller να επικοινωνήσει με αυτών χρησιμοποιώντας το NETCONF και το γνωστό YANG μοντέλο του controller και των εφαρμογών του.

Η μεταφορά του οικοσυστήματος του NETCONF και της YANG πέρα από την δικτυακή συσκεύη, πλέον μέσα στο λογισμικό του ελεγκτη προσφέρει τα θετικά του πρωτοκόλλου πλέον και στον ελεγκτή του δικτύου.

- ενιαία αποθήκευση των δεδομένων και των ρυθμίσεων
- περιγραφή με ένα ενιαίο τρόπο οποιαδήποτε εφαρμογής "τρέχει" στον

ελεγκτή

- ενιαιοποίηση των προγραμματιστικών εργαλείων που απαιτούνται από τον άνθρωπο πάνω από τον ελεγκτή

Αυτός είναι και ο λόγος οι ελεγκτές δικτύων, πέρα από αυτούς που ειδικεύονται στο πρωτόκολλο Openflow, ακολουθούν λιγότερο ή περισσότερο αυτή την αρχιτεκτονική. Πληρέστερο παράδειγμα της περιγραφής ο Opendaylight[10] (Linux Foundation), οι ConfD και NCS (της Tail-f, εταίρειας που ανέπτυξε και εμπορικά το NETCONF και πρόσφατα εξαγοράστηκε από τη Cisco) και ο OpenContrail (Juniper Networks).

1.2.3 Σύνταξη

Η δομή της YANG, όπως ειπώθηκε, είναι ιεραρχική και συγκεκριμένα ένα YANG module έχει τη μορφή δένδρου. Κάθε κόμβος του δέντρου έχει ένα όνομα και ένα σύνολο από "παιδιά". Κάθε κόμβος και κάθε παιδί περιγράφει τα περιεχόμενα του. Τα περιεχόμενα μπορούν να πάρουν τιμές μέσα από δομές ήδη ορισμένες από τη YANG (π.χ. string, integer, ip-address, mask), ή ορισμένες από το χρήστη. Βασικά στοιχεία της YANG είναι:

- leaf: Τα "φύλλα" περιέχουν ένα απλό δεδομένο, όπως είναι ένας ακέραιος ή ένα string.
- leaf-list: μια ακολουθία από φύλλα
- container: Τα container ομαδοποιούν σχετικούς μεταξύ τους κόμβους και δεν παίρνουν κάποια τιμή, αλλά περιέχουν μόνο παιδιά.
- list: περιγράφει τα στοιχείων που απαρτίζουν μια λίστα. Όλα τα στοιχεία της λίστα είναι όμοια.
- grouping: μια ομαδοποίηση που μπορεί να επαναχρησιμοποιηθέι μέσα σε κάποιο στοιχείο.

Ένα παράδειγμα ενός YANG module μπορεί να είναι το παρακάτω, μαζί με την αντίστοιχη έκφραση XML του.

```
grouping endpoint {
  leaf address {
    type ip-address;
```

```
  }
  leaf port {
    type port-number;
  }
}

container connection {
  container source {
    uses endpoint {
      refine port {
      default 161;
      }
    }
  }
  container destination {
    uses endpoint {
      refine port {
      default 161;
      }
    }
  }
}
```

Η χρήση του παραπάνω μοντέλου σε ένα μήνυμα NETCONF θα μπορούσε να είναι η παρακάτω

```
<connection>
      <source>
            <address>192.168.0.3</address>
            <port>8080</port>
      </source>
      <destination>
            <address>192.168.0.4</address>
            <port>8080</port>
      </destination>
</connection>
```

Τα δεδομένα του NETCONF και αυτά που περιγράφει η YANG μπορεί να είναι εγγράψιμα ή και μη εγγράψιμα δεδομένα. Τα πρώτα αφορούν ρυθμίσεις που "τρέχει" ή θα "τρέξει" ο εξυπηρετητής και τα δεύτερα περιγράφουν την κατάσταση που βρίσκεται ο εξυπηρετητής. Η YANG κάνει αυτό το διαχωρισμό χρησιμοποιώντας τη δήλωση config που δέχεται μια λογική τιμή. Αν αυτή είναι true (προεπιλογή) τότε το δεδομένο είναι εγγράψιμο, αν είναι false τότε περιγράφεται δεδομένο κατάστασης. Η επέκταση των μονάδων της YANG γίνεται με τη χρήση της δήλωσης augment, που δέχεται σαν τιμή τη θέση ενός αρχείου που περιέχει μια μονάδα YANG, το οποίο και ενσωματώνεται. Για παράδειγμα augment /system/login/user.

Ορισμός RPC

Το NETCONF προκαλεί τις αλλαγές στον εξυπηρετητή με RPCs από τον πελάτη. Αυτές μπορεί να είναι οι βασικές κλήσεις του πρωτοκόλλου, που είναι υλοποιημένες εσωτερικά στον εξυπηρετητή και περιγράψαμε στο κομμάτι του κεφαλαίου για τον NETCONF. Παρόλα αυτά η YANG δίνει τη δυνατότητα να οριστούν οι κλήσεις επιπρόσθετα. Για να οριστεί μια κλήση πρέπει να περιγραφούν οι μεταβλητές ή τα δεδομένα που δέχεται, την επίδραση που θα έχεις στις datastore του εξυπηρετητή όπως επίσης και τι θα επιστρέψει τελικά στον πελάτη σαν απάντηση σε περίπτωση επιτυχία ή αποτυχίας. Οι κόμβοι input και output περιγράφουν την είσοδο και την έξοδο της RPC αντίστοιχα. Για παράδειγμα:

```
rpc activate-software-image {
       input {
          leaf image-name {
             type string;
          }
       }
       output {
          leaf status {
             type string;
          }
       }
    }
```

Με βάση αυτό μία κλήση NETCONF:

```
<rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <activate-software-image>
            <image-name>foo-bar.jpeg</image-name>
      </activate-software-image>
</rpc>
```

Θα λάβει απάντηση:

```
<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <status>
      The image foo-bar.jpeg is being installed.
   </status>
</rpc-reply>
```

Για την εύκολη μετάφρασή της σε XML η YANG προσφέρει τη YIN (YANG Independent Notation). Μια μορφή βασισμένη σε XML που προσφέρει το ακριβές ισοδύναμο ενός μοντέλου ή μια μονάδας YANG. Έτσι μπορούν να την επεξεργαστούν εργαλεία φτιαγμένα για την XML.

1.3 Quality of Service

Ο όρος Quality of Service στα δίκτυα υπολογιστών αναφέρεται σε ένα ευρύ σύνολο λειτουργιών που διενεργούν οι συσκευές δικτύου ώστε να παρέχουν ένα "καλύτερο" υφιστάμενο δικτυακό περιβάλλον για τις υπηρεσίες που το χρησιμοποιούν. Το βασικό πρόβλημα που έρχεται να λύσει η υλοποίηση QoS σε δικτυακές συσκευές είναι αυτό της συνύπαρξης εφαρμογών με διαφορετικές απαιτήσεις από το ίδιο το δίκτυο. Ειδικά αν ληφθεί υπόψιν πως το εύρος ζώνης είναι πεπερασμένο, το μέγεθος των ουρών, η επεξεργαστική ισχύς των συσκευών, σε συνδιασμό με έναν όγκο κίνησης που μπορεί να ξεπερνά τα όρια που θέτουν όλα αυτά. Πρέπει, επομένως, να βρεθεί τρόπος να διαφοροποιηθούν οι υπηρεσίες -ως εκ τούτου και η κίνηση τους- με βάση ένα χαρακτηριστικό στο οποίο είναι ευαίσθητες, ή αλλιώς με βάση τις απαιτήσεις τους. Για παράδειγμα η απευθείας μετάδοση εικόνας/βίντεο πλήττεται από το χαμηλό εύρος ζώνης κυρίαρχα και δευτερευόντως από την καθυστέρηση των

πακέτων (latency). Υπηρεσίες τηλεφωνίας με χρήση διαδικτύο (Voice Over IP) είναι ευαίσθητες στην καθυστέρηση και στην απώλεια πακέτων, λόγω της χρήσης UDP στο επίπεδο μεταφοράς και άρα της απώλειας ελέγχου ροής της κίνησης.

Με βάση αυτό το χαρακτηριστικό μπορεί να δοθεί στην κίνηση μία ετικέτα (label/marking) για να λάβει διαφορετική μεταχείριση από τις δικτυακές συσκεύες που θα κινηθεί ή ακόμα και να γίνει διαφορετική δρομολόγηση ανά κατηγορία κίνησης (Policy Based Routing) για να βελτιστοποιηθεί το μονοπάτι (λιγότεροι ενδιάμεσοι κόμβοι, μικρότερη γεωγραφική απόσταση).

Όπως είναι λογικό, η εφαρμογή μιας υπηρεσίας QoS σε ένα δίκτυο απαιτεί τη ρύθμιση και την διάδοση των ρυθμίσεων σε όλο το μήκος της τοπολογίας (point-to-point), ή έστω του μονοπατιού που θα μεταδοθεί η κίνηση. Ο λόγος είναι αυτονόητος, καθώς αν μια συσκευή παρέχει το απαιτούμενο από την υπηρεσία εύρος ζώνης ή υψηλή προτεραιότητα στην ουρά, αλλά ο επόμενος κόμβος διέλευσης της κίνησης δεν το κάνει, τότε δε θα υπάρχει καμιά βελτίωση στην συνολική απόδοση για την υπηρεσία για τον τελικό δέκτη/χρήστη.

Στην πραγματικότητα το QoS αναφέρεται στο σύνολο των τεχνικών που διενεργούνται από διαχειριστή δκτύου ώστε να γίνουν τα παραπάνω. Η τεχνική την οποία χρησιμοποιεί η υλοποίση και αναφέρεται και το παρόν ονομάζεται DSCP[3] (Differentiated Services Code Point) marking.

1.3.1 DSCP Marking

Η τεχνική αυτή χρησιμοποιεί το πεδίο Type of Service (ToS Byte) της επικεφαλίδας IP για να τοποθετήσει την ετικέτα που ανεφερθήκε παραπάνω. Η υλοποίηση μια υπηρεσίας DSCP ακολουθεί μια συγκεκριμένη διαδικασία ή οποία και φαίνεται στο σχήμα.

Ταξινόμηση/Κατηγοριοποίηση (Classification)

Η κίνηση που επιθυμεί ο διαχειριστής να λάβει ιδιαίτερης μεταχείρισης από την υποδομή του δικτύου για αρχή οφείλει να αναγνωριστεί και να διαχωριστεί με βάση κάποια κριτήρια. Τα κριτήρια αυτά μπορεί να αφορούν: α) το επίπεδο μεταφοράς, δηλαδή αν πρόκειται για κίνηση TCP ή UDP ή/και τη θύρα (TCP/UDP port) που χρησιμοποιεί, β) το επίπεδο δικτύου με τη διεύθυνση πηγής ή προορισμού, ή ακόμα και ολόκληρο υποδίκτυο πηγής ή προορισμού, γ) τη φυσική θύρα (interface) από το οποίο προέρχεται η κίνηση, ή δ) ένα συνδιασμός όλων των παραπάνω. Μετά από αυτό το φιλτράρισμα με βάση

τα κριτήρια τα πακέτα ταξινομούνται σε κάποια σχετική κατηγορία/κλάση. Ένα σύνολο κλάσεων μπορεί να συνυπάρχει σε μια συσκευή.

Μαρκάρισμα

Με βάση την παραπάνω διαλογή πακέτων και ταξινόμησή τους σε κλάσεις, τα πακέτα σημαδεύονται/μαρκάρονται έτσι ώστε να μπορούν πιο εύκολα να διαφοροποιηθούν κατά την υπόλοιπη διαδρομή του μέσα στο δίκτυο. Σε επίπεδο πακέτου αυτό γίνεται με την τοποθέτηση στο πεδίο ToS (Type of Service) της επικεφαλίδας IP ενός συγκριμένου byte, το οποίο θα είναι και το αναγνωριστικό για το είδος του πακέτου, και επομένως της αντιμετώπισης που πρέπει να λάβει. Κάθε δρομολογητής διατηρεί μια αντιστοίχιση ανάμεσα σε κάθε τιμή του byte και στη διαφορετική μεταχείριση της κίνησης.

"Αστυνόμευση" (Policing)

Η κάθε κατηγορία κίνησης ανατίθεται και σε μία ουρά σε κάθε συσκευή δικτύου. Η διαδικασία του Policing εφαρμόζεται σε κάθε ουρά και επομένως σε κάθε κατηγορία κίνησης. Ένας "αστυνόμος" (policer) μπορεί να ορίσει το εύρος ζώνης που θα δοθεί, την προτεραιότητα, το τι θα γίνει σε περίπτωση συμφώρησης (Congestion Control), τον αν σε περίπτωση πληρότητας της ουράς θα γίνεται τυχαία επιλογή στα πακέτα που θα πέφτουν, τι πιθανότητα (loss propability) θα έχει κάθε κίνηση να χάσει πακέτα σε αυτή την περίπτωση(Weighted Tail Drop) κ.ο.κ.

1.3.2 Τομέας DSCP

Το κομμάτι του δικτύου μέσα στο οποίο διενεργείται κατηγοριοποίηση/διαφοροποίηση της κίνησης με χρήση DSCP marking ονομάζεται DSCP Domain. Οι συσκευές που βρίσκονται στην είσοδο του χωρίου αυτού του δικτύου ονομάζονται ακραίες (edge) και είναι αυτά τα οποία συλλέγουν την κίνηση, την ταξινομούν και την μαρκάρουν αναλόγως. Οι εσωτερικές συσκευές ονομάζονται συσκευές "πυρήνα" και δέχονται προς τις δύο κατευθύνσεις κίνησης μαρκαρισμένη κίνηση, και απλά της παρέχουν την ιδιαίτερη μεταχείριση που απαιτεί η κάθε μία και επιθυμεί να δώσει ο διαχειριστής.

Κάθε πακέτο που καταφθάνει σε μια δικτυακή συσκευή περνάει σειριακά από την παραπάνω διαδικασία. Το μαρκάρισμα συμβαίνει στην θύρα/διεπαφή

εισόδου (ingress interface), το μαρκάρισμα γίνεται από την μηχανή δρομολόγησης και έπειτα τα μαρκαρισμένα πακέτα τοποθετούνται στην ουρά εξόδου όπου και διενεργείται το policing, που το αποτέλεσμά του είναι εμφανές στη θύρα/διεπαφή εξόδου (egress interface).

Κατά αντιστοιχία με τα παραπάνω οι συσκευές μέσα σε ένα DSCP Domain λαμβάνουν και ένα συγκεκριμένο ρόλο. Οι ακραίες συσκευές (edge) είναι εκείνες που έχουν τουλάχιστον ένα interface προς τα έξω του DSCP Domain, το οποίο παίζει ο ρόλο του edge interface, και ένα τουλάχιστον interface προς τα μέσα της τοπολογίας, το οποίο έχει ρόλο core interface. Οι συσκευές πυρήνα είναι εκείνες οι οποίες βρίσκονται στο εσωτερικό του DSCP domain και προφανώς έχουν interfaces που λειτουργούν ως core.

Σε κάθε συσκευή διενεργείται ταξινόμηση των πακέτων κατά την είσοδό τους σε αυτή. Στις συσκευές πυρήνα η κίνηση έρχεται από τις ακραίες συσκευές, όπου και αυτή έχει μαρκαριστή. Οι ακραίες συσκευές έχουν τουλάχιστον ένα interface εντός του DSCP domain και τουλάχιστον ένα εκτός του.

Συσκεύες Πυρήνα

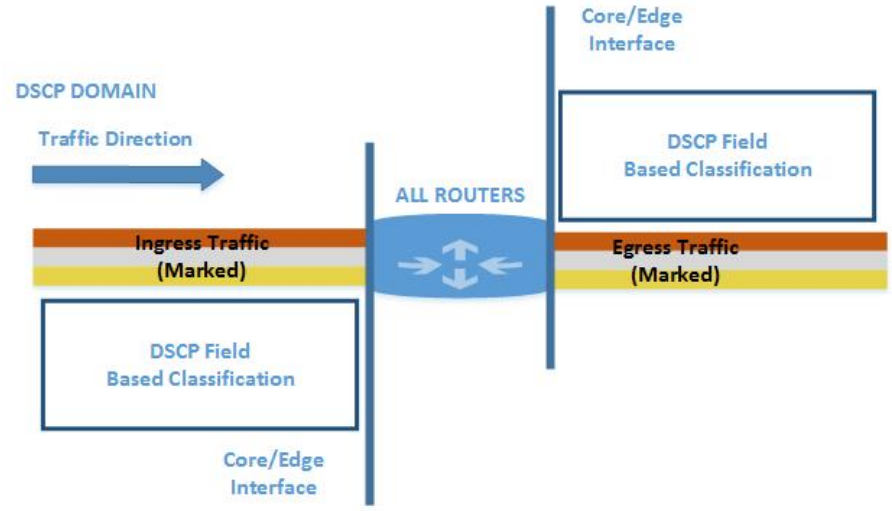
Συσκεύες πυρήνα είναι εκείνες που βρίσκονται από το 2ο άλμα (hop) και μετά, μέσα στο DSCP Domain. Αυτό σημαίνει πως σε αυτά καταφθάνει ήδη μαρκάρισμένη κίνηση, προς κάθε κατεύθυνση της. Σε όλα τα interfaces τους γίνεται διαλογή της κίνησης με βάση το πεδίο DSCP της IP επικεφαλίδας που έχει τεθεί από τις ακραίες συσκευές. Αυτά τα interfaces, καθώς δίνουν κίνηση στο εσωτερικό του DSCP domain ονομάζονται core interfaces.

Ακραίες Συσκευές

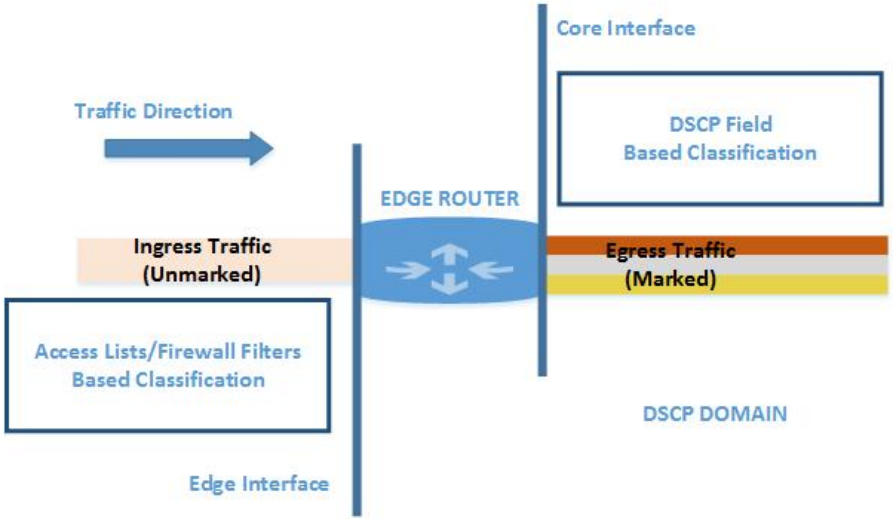
Οι ακραίες συσκευές καθώς βρίσκονται στο όριο του DSCP domain έχουν λίγο πιο περίπλοκο ρόλο. Τουλάχιστον ένα interface τους βρίσκεται εντός του DSCP domain (core interface) και τουλάχιστον ένα ”κοιτά” προς τα έξω και ονομάζεται edge interface. Αυτό σημαίνει πως η κίνηση που έρχεται από τα μέσα του DSCP domain πρέπει να διαλέγεται με βάση το πεδίο DSCP που έχει ήδη τεθεί, ενώ η κίνηση που έρχεται από τα έξω να διαλέγεται με βάση τα κριτήρια που έχουν τεθεί από το διαχειριστή. Αλλά παράλληλα στην έξοδο της κίνησης, σε κάθε κατεύθυνση, πρέπει να δίνεται η ειδική μεταχείριση σε κάθε κατηγορία κίνησης.

Core Interafeces

Έτσι ονομάζονται τα interfaces τα οποία προωθούν την κίνηση στο εσωτερικό του DSCP Domain. Στην εισερχόμενη (ingress traffic) σε αυτά κίνηση εφαρμόζεται η διαφοροποίηση/κατηγοριοποίηση της κίνησης βάσει του ήδη τοποθετημένου κωδικού DS που έχουν αναθέσει οι ακραίες συσκευές. Στην



Σχήμα 1.3: Διαχείριση εξερχόμενης κίνησης



Σχήμα 1.4: Διαχείριση εισερχόμενης κίνησης

εξερχόμενη από το interface κίνηση (egress) προσφέρεται η απαιτούμενη μεταχείριση ανάλογα την κλάση στην οποία έχει ενταχθεί.

Edge Interfaces

Τα interfaces αυτά είναι η είσοδος και έξοδος της κίνηση από το DSCP Domain. Αυτό σημαίνει πως πρέπει να δρουν στην εξερχόμενη κίνηση σαν ένα core facing interface, πράγμα που σημαίνει ότι οφείλει να "γνωρίζει" τι να κάνει στα μαρκαρισμένα πακέτα που έρχονται από το εσωτερικό του DSCP domain για να τα προωθήσει προς τα έξω.

Από την άλλη όμως δέχονται και "αχρωμάτιστη" κίνηση από το εξωτερικό του DSCP domain. Εκεί πρέπει να γίνεται η κατηγοριοποίηση/διαφοροποίηση όπως έχει αναφερθεί παραπάνω, και να μαρκαριστεί η κίνηση, για να περάσει στο εσωτερικό του DSCP domain πλέον με τον αντίστοιχο κωδικό στην IP επικεφαλίδα.

2

Αρχιτεκτονική

2.1 Αρχιτεκτονική

Η αρχιτεκτονική της εφαρμογής είναι βασισμένη σε αυτό που υλοποιούν οι περισσότερες εφαρμογές Προγραμματιζόμενων δικτύων με ελεγκτές[11]. Στο ρόλο του ελεγκτή επιλέχθηκε το Ansible[4], κυρίως λόγω της αξιόπιστης και πλήρως λειτουργικής υλοποίησης του πρωτοκόλλου NETCONF μέσω των modules του. Σαν διεπαφή με την τοπολογία (Southbound API) χρησιμοποιήθηκαν τα modules του Ansible που κάνουν χρήση NETCONF (junos_install_config[12] και cisco_client) για την προώθηση ρυθμίσεων στην τοπολογία και σαν διεπαφή προς τη μεριά του χρήστη (Northbound API) το Python API 2.0[13] του Ansible. Ως διεπαφή με τον διαχειριστή (User Interface) χρησιμοποιείται μια διαδικτυακή εφαρμογή, η οποία διατηρεί στη βάση δεδομένων της, τις πληροφορίες για τις υπηρεσίες που τρέχουν και ελέγχει την εκκίνηση/παύση/διαγραφή τους.

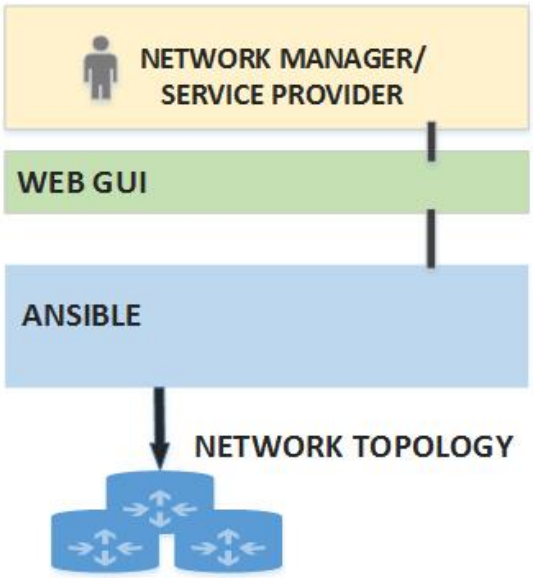
Ένα απλό πρόγραμμα μεταφράζει τα δεδομένα από τη βάση σε αρχεία YAML που είναι αυτά που χρησιμοποιεί το Ansible. Στο εσωτερικό του το Ansible κρατάει τις ρυθμίσεις που περιγράφηκαν στο προηγούμενο κεφάλαιο χωρισμένες "θεματικά", ώστε να μπορεί να τις προωθεί αποσπασματικά και κατά βούληση του διαχειριστή, αλλά και για να είναι επαναχρησιμοποιήσιμες από άλλες υλοποιήσεις.

Ως διεπαφή με την τοπολογία (Southbound Interface) χρησιμοποιήθηκαν δύο Ansible modules, το push_config για τις Cisco συσκευές, και το junos_install_config για τη Juniper.

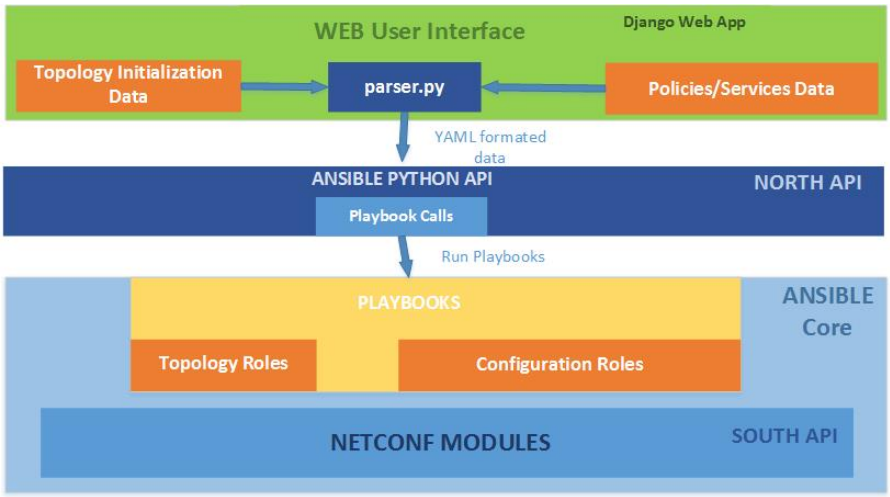
2.1.1 Το Ansible

Το Ansible είναι ένα εργαλείο διαχείρισης των ρυθμίσεων (configuration management), με βασικό σκόπό την αυτοματοποίηση των ενεργειών που απαιτούνται για τη ρύθμιση μιας πληροφοριακής υποδομής.

Κάθε ενέργεια που καλείται να υλοποιήσει το Ansible ονομάζεται task. Κάθε task καλεί μια ενέργεια, με τις απαιτούμενες εισόδους. Η κλήση του πυ-



Σχήμα 2.1: Επισκόπηση της εφαρμογής



Σχήμα 2.2: Η αρχιτεκτονική της εφαρμογής

ροδοτεί την διαδικασία σύνδεσης και υλοποιήσης της εκάστοτε ενέργειας. Στο τέλος το Ansible ενημερώνει για την εκπλήρωση (π.χ. ok ή error) αλλά και για τον αν στον κόμβο που στόχευε επήλθαν οι αλλαγές (change ή not changed).

Η σύνδεση με τους κόμβους που διαχειρίζεται το Ansible γίνεται με το πρωτόκολλο SSH και δεν χρησιμοποιεί κάποιου είδους "πράκτορα" (agent). Αυτό σημαίνει πως η βασική εγκατάσταση και χρήση του είναι απλή καθώς

δεν χρειάζεται κάτι παραπάνω από έναν SSH server σε κάθε κόμβο που θα διαχειριστεί το Ansible. Όπως επίσης δεν είναι αναγκαίο κάποιο επιπρόσθετο σύστημα πιστοποίησης πέρα από αυτό που χρησιμοποιεί το SSH.

2.1.2 Διεπαφή Χρήστη

Για επικοινωνία του χρήστη με την εφαρμογή αναπτύχθηκε μια διαδικτυακή εφαρμογή, με χρήση του Django Web Framework[5]. Μέσω αυτής εισάγονται όλα τα δεδομένα που αφορούν την υπηρεσία, αλλά και τα δεδομένα που αφορούν την τοπολογία. Εκτός από την εισαγωγή και αποθήκευση δεδομένων, ορίζονται κλήσεις για την εκκίνηση, παύση και διαγραφή υπηρεσιών.

2.1.3 Διεπαφή με την τοπολογία

Για την επικοινωνία του Ansible με την τοπολογία δικτύου χρησιμοποιήθηκαν δύο modules που καλούν υλοποιήσεις πελατων NETCONF. Δέχονται σαν ορίσματα την διεύθυνση IP της συσκευής, τα αναγνωριστικά του χρήστη (username//password), και ένα αρχείο με τις ρυθμίσεις που θα εγκατασταθούν σε μορφή κειμένου. Τα modules που χρησιμοποιούνται είναι τα push_config, βασισμένο στον cisco_client και το junos_install_config, ανεπτυγμένο από την Juniper και βασισμένο στον ncclient.

2.2 Δοκιμαστική Τοπολογία

Η εφαρμογή που υλοποιήθηκε για τους σκοπούς της παρούσας διπλωματικής εργασίας προσφέρει μια υπηρεσία Quality of Service για τη δικτυακή κίνηση δύο πελατών. Το απαιτούμενο από την υπηρεσία αποτέλεσμα είναι η παροχή ένός ζητούμενο εύρους ζώνης κατά τη ζήτηση (on demand) από τους δύο χρήστες.

2.2.1 Εξοπλισμός

Η τοπολογία στην οποία δοκιμάστηκε η εφαρμογή αποτελείται από τρεις κόμβους. Στα άκρα τις τοπολογίας βρίσκονται δύο φυσικοί δρομολογητές και στο εσωτερικό ένας εικονικός δρομολογητής. Ο ένας ακραίος δρομολογητής είναι της εταιρείας Cisco με αριθμό μοντέλου 2821 και το υλισμικό του είναι

IOS της έκδοσης 15.3(3)XB12. Ο άλλος είναι ένας μεταγωγέας με δυνατότητες επιπέδου 3 (Layer 3 switch), της εταιρείας Juniper, με κωδικό μοντέλου EX3300 και υλισμικό JUNOS 15.1R4.6. Ο εικονικός δρομολογητής είναι ένας Cisco CSR1000v με υλισμικό IOS-XE έκδοσης 03.13.01.S. Οι συσκευές Cisco δεν υποστηρίζουν τη χρήση της γλώσσας YANG, και επιτρέπουν χρήση του πρωτοκόλλου NETCONF με τη χρήση μόνο του μηχανισμού μετάδοσης [14], με περιεχόμενο τις εντολές που θα εγκαταστήσουν τις ρυθμίσεις. Για να ανταποκριθούμε σε αυτό αναπτύχθηκε το πρόγραμμα cisco_client που υλοποιεί το παραπάνω. Η συσκευή Juniper έχει εγκατεστημένο μοντέλο YANG, αλλά λόγω του όγκου της και της έλλειψης σπονδύλωσης της είναι εξαιρετικά δύσχρηστη. Για αυτό το λόγο και το module για το Ansible που έχει αναπτύξει η ίδια η εταιρεία δε βασίζεται σε αυτή. Στο ρόλο των χρηστών τοποθετήθηκαν δύο Raspberry Pi2 τοποθετημένα σε δύο διαφορετικά υποδίκτυα, το καθένα συνδεδεμένο και σε ένα ακραίο δρομολογητή.

2.2.2 Η υπηρεσία

Η διασύνδεση των πελατών

Κατά την αρχικοποίηση της υπηρεσίας και ένταξης των πελατών, ενημερώνεται η τοπολογία με την απαιραίτητη πληροφορία για τη διασύνδεση τους. Γίνεται χρήση στατικής δρομολόγησης ανάμεσα στους δρομολογητές. Για τις ζεύξεις ανάμεσα στους ακραίους δρομολογητές και τον εσωτερικό χρησιμοποιείται ενθυλάκωση σε GRE tunnels (Generic Routing Encapsulation). Η κίνηση, λοιπόν, των χρηστών μέσω των στατικών κανόνων εισάγεται μέσα στις διεπαφές των tunnel και με αυτό τον τρόπο αποφεύγεται η ”διαφήμιση” των δικτύων των χρηστών στον εσωτερικό δρομολογητή. Με αυτό τον τρόπο μειώνεται η πληροφορία που απαιτείται από τον δρομολογητή αυτό για τη διασύνδεση δύο χρηστών, και επομένως πιο γρήγορη οποιαδήποτε μετάβαση (πρόσθεση, αφαίρεση χρηστών).

To Quality of Service

Για την εφαρμογή του Quality of Service γίνεται διαφοροποίηση υπηρεσιών μέσω DSCP marking. Η κίνηση που αφορά τους δύο πελάτες θα επιλέγεται με χρήση φίλτρων (firewall filters για το Juniper, access-lists για τα Cisco). Σε αυτή θα εφαρμόζεται μια πολιτική QoS που θα έχει οριστεί. Καθώς η κίνηση ”μαρκάρεται”, ο χαρακτηρισμός συμβατικά είναι χρωματικός. Έχουμε τις πολι-

τικές Gold, Silver, Bronze, οι οποίες προσφέρουν και ένα ποσοστό του έυρους ζώνης έκαστη (π.χ. η Gold δίνει το 50% του διαθέσιμου). Αυτό βεβαια είναι απόφαση του διαχειριστή και μπορεί να δοθεί κατά το δοκούν. Έχει αφεθεί έλεύθερο κατά την εισαγωγή από το διαχειριστή των μεταβλητών της υπηρεσίας (μέσω της διαδικτυακής εφαρμογής) να μπορεί να γίνει οποιαδήποτε παραλλαγή.

3

Υλοποίηση

3.1 Δοκιμαστική Τοπολογία

Η εφαρμογή που υλοποιήθηκε για τους σκοπούς της παρούσας διπλωματικής εργασίας προσφέρει μια υπηρεσία Quality of Service για τη δικτυακή κίνηση δύο πελατών. Το απαιτούμενο από την υπηρεσία αποτέλεσμα είναι η παροχή ένός ζητούμενο εύρους ζώνης κατά τη ζήτηση (on demand) από τους δύο χρήστες.

3.1.1 Εξοπλισμός

Η τοπολογία στην οποία δοκιμάστηκε η εφαρμογή αποτελείται από τρεις κόμβους. Στα άκρα τις τοπολογίας βρίσκονται δύο φυσικοί δρομολογητές και στο εσωτερικό ένας εικονικός δρομολογητής. Ο ένας ακραίος δρομολογητής είναι της εταιρείας Cisco με αριθμό μοντέλου 2821 και το υλισμικό του είναι IOS της έκδοσης 15.3(3)XB12. Ο άλλος είναι ένας μεταγωγέας με δυνατότητες επιπέδου 3 (Layer 3 switch), της εταιρείας Juniper, με κωδικό μοντέλου EX3300 και υλισμικό JUNOS 15.1R4.6. Ο εικονικός δρομολογητής είναι ένας Cisco CSR1000v με υλισμικό IOS-XE έκδοσης 03.13.01.S. Οι συσκευές Cisco δεν υποστηρίζουν τη χρήση της γλώσσας YANG, και επιτρέπουν χρήση του πρωτοκόλλου NETCONF με τη χρήση μόνο του μηχανισμού μετάδοσης [14], με περιεχόμενο τις εντολές που θα εγκαταστήσουν τις ρυθμίσεις. Για να ανταποκριθούμε σε αυτό αναπτύχθηκε το πρόγραμμα cisco_client που υλοποιεί το παραπάνω. Η συσκευή Juniper έχει εγκατεστημένο μοντέλο YANG, αλλά λόγω του όγκου της και της έλλειψης σπονδύλωσης της είναι εξαιρετικά δύσχρηστη. Για αυτό το λόγο και το module για το Ansible που έχει αναπτύξει η ίδια η εταιρεία δε βασίζεται σε αυτή. Στο ρόλο των χρηστών τοποθετήθηκαν δύο Raspberry Pi2 τοποθετημένα σε δύο διαφορετικά υποδίκτυα, το καθένα συνδεδεμένο και σε ένα ακραίο δρομολογητή.

3.1.2 Η υπηρεσία

Η διασύνδεση των πελατών

Κατά την αρχικοποίηση της υπηρεσίας και ένταξης των πελατών, ενημερώνεται η τοπολογία με την απαιραίτητη πληροφορία για τη διασύνδεση τους. Γίνεται χρήση στατικής δρομολόγησης ανάμεσα στους δρομολογητές. Για τις ζεύξεις ανάμεσα στους ακραίους δρομολογητές και τον εσωτερικό χρησιμοποιείται ενθυλάκωση σε GRE tunnels (Generic Routing Encapsulation). Η κίνηση, λοιπόν, των χρηστών μέσω των στατικών κανόνων εισάγεται μέσα στις διεπαφές των tunnel και με αυτό τον τρόπο αποφεύγεται η "διαφήμιση" των δικτύων των χρηστών στον εσωτερικό δρομολογητή. Με αυτό τον τρόπο μειώνεται η πληροφορία που απαιτείται από τον δρομολογητή αυτό για τη διασύνδεση δύο χρηστών, και επομένως πιο γρήγορη οποιαδήποτε μετάβαση (πρόσθεση, αφαίρεση χρηστών).

To Quality of Service

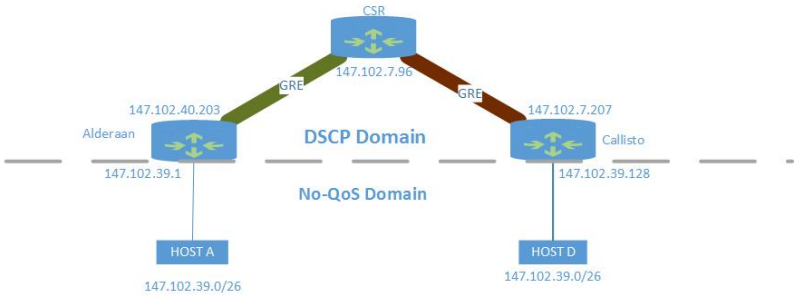
Για την εφαρμογή του Quality of Service γίνεται διαφοροποίηση υπηρεσιών μέσω DSCP marking. Η κίνηση που αφορά τους δύο πελάτες θα επιλέγεται με χρήση φίλτρων (firewall filters για το Juniper, access-lists για τα Cisco). Σε αυτή θα εφαρμόζεται μια πολιτική QoS που θα έχει οριστεί. Καθώς η κίνηση "μαρκάρεται", ο χαρακτηρισμός συμβατικά είναι χρωματικός. Έχουμε τις πολιτικές Gold, Silver, Bronze, οι οποίες προσφέρουν και ένα ποσοστό του έυρους ζώνης έκαστη (π.χ. η Gold δίνει το 50% του διαθέσιμου). Αυτό βεβαια είναι απόφαση του διαχειριστή και μπορεί να δοθεί κατά το δοκούν. Έχει αφεθεί ελεύθερο κατά την εισαγωγή από το διαχειριστή των μεταβλητών της υπηρεσίας (μέσω της διαδικτυακής εφαρμογής) να μπορεί να γίνει οποιαδήποτε παραλλαγή.

3.2 Οι ρυθμίσεις

Οι ρυθμίσεις που πρέπει να προωθηθούν στο δίκτυο με βάση τα παραπάνω χωρίστηκαν για πρακτικούς και εννοιολογικούς σκοπούς. Συγκεκριμένα χωρίστηκαν στις ρυθμίσεις: α) ορισμού των interfaces, β) των GRE tunnels, γ) δρομολόγηση, δ) του πυρήνα DSCP, ε) των ακραίων συσκευών του DSCP.

Έστω ότι έχουμε το παράδειγμα της παρακάτω τοπολογίας. Έστω επίσης

ότι θέλουμε να δώσουμε στους χρήστες το 50% του διαθέσιμου εύρους ζώνης των συσκευών και ότι θα μαρκάρουμε τα πακέτα της κίνησης με τον DSCP af11. Οι ρυθμίσεις είναι συνοπτικές και παραδειγματικές.



Σχήμα 3.1: Η τοπολογία

3.2.1 Interfaces

Για τις ακραίες συσκευές: Συσκευή CISCO (alderaan):

```
interface GigabitEthernet0/0
 description Core & Managment Interface
 ip address 147.102.40.203 255.255.255.0

interface GigabitEthernet0/1
 description Edge Interface
 ip address 147.102.39.1 255.255.255.192
```

Συσκευή Juniper (callisto):

```
#assign layer 2 interfaces to vlan
set interfaces ge-0/0/0 unit 0
      family ethernet-switching
            port-mode access
```

```
                vlan members v998
set interfaces ge-0/0/0 description EdgeInterface
#define layer 3 interface for vlan
set vlans v998 vlan-id 998 l3-interface vlan.998
set interfaces vlan unit 998 family inet address 147.102.39.129/30

#Core & Management interface
set interface me0 unit 0 family inet address 147.102.7.207/24
```

Για τις εσωτερικές συσκευές: Στην εικονική συσκευή χρησιμοποιείται ένα ”φυσικό” interface το οποίο είναι άκρο δύο διαφορετικών GRE Tunnel για σύνδεση με κάθε ακραία συσκευή, η ρύθμιση του οποίου θα αναπτυχθεί παρακάτω.

```
interface GigabitEthernet1
 description Core Interface
 ip address 147.102.7.96 255.255.255.240
```

3.2.2 GRE Tunnels

Σκοπός των tunnel γενικά είναι η προσφορά ασφαλών και ιδιωτικών μονοπατιών για την κίνηση των πακέτων μέσα από ένα δημόσιο δίκτυο. Το πρωτόκολλο GRE[15] το κάνει αυτό προσφέροντας μια εικονική ζεύξη από σημείο σε σημείο (point-to-point) ανάμεσα σε δύο δίκτυα. Το ότι είναι εικονική σημαίνει πως τα πακέτα διέρχονται μέσα από το υπόλοιπο δίκτυο. Το σημαντικό είναι πως το δημόσιο δίκτυο (public network) δεν αναμειγνύεται με το εσωτερικό του πακέτου, απλά το προωθεί από την αρχή του tunnel μέχρι το τέλος του.

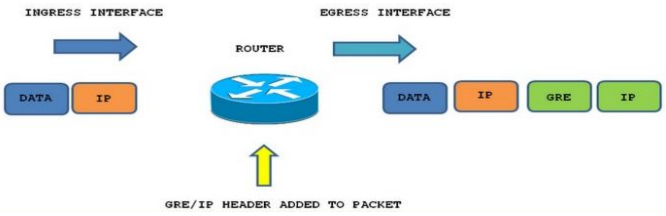
Αυτό επιτυγχάνεται μέσω της ενθυλάκωσης του IP πακέτου που προέρχεται από έναν χρήστη μέσα σε ένα δεύτερο IP πακέτο. Αυτό σημαίνει πως το δημόσιο δίκτυο κάνει τη δρομολόγηση με βάση την εξωτερική επικεφαλίδα IP καθώς αυτή είναι και η πρώτη που συναντά εξετάζοντας το πακέτο για να πάρει τις αποφάσεις δρομολόγησης.

Κάθε tunnel έχει μια πηγή και ένα προορισμό, που είναι δύο interfaces σε

δύο δρομολογητές, τα οποία είναι προσβάσιμα στο επίπεδο δικτύου (Layer 3) επομένως έχουν και την δική τους IP διεύθυνση. Κάθε δρομολογητής γνωρίζει μέσω του πίνακα δρομολόγησης του (routing table), για το ποια κίνηση θα προωθήσει μέσω ενός tunnel.Οι διευθύνσεις πηγής και προορισμού της εξωτερικής IP επικεφαλίδας είναι αυτές εκείνες της πηγής και τέλους του tunnel. Πλέον λοιπόν οι ενδιάμεσοι κόμβοι έχουν να δρομολογήσουν ένα πακέτο με βάση τον προορισμό του tunnel και όχι με βάση τον αρχικό προορισμό.

Το πρωτόκολλο GRE δεν είναι πρωτόκολλο δρομολόγησης (routing) αλλά δρομολογούμενο (routed). Πράγμα που σημαίνει πως για την εξωτερική IP επικεφαλίδα πρέπει να υπάρχει στο δημόσιο δίκτυο η πληροφορία που χρειάζεται για να δρομολογηθεί από το ένα άκρο του tunnel στο άλλο. Το πλεονέκτημα του πρωτοκόλλου είναι πως αυτή η πληροφορία είναι και η μόνη που χρειάζεται για να γίνει η δρομολόγηση. Πράγμα που σημαίνει πως το δημόσιο δίκτυο δεν είναι ανάγκη να μπορεί να δρομολογήσει την κίνηση ανάμεσα στα δύο δίκτυα που επιχειρούν να επικοινωνήσουν, αλλά αρκεί να μπορεί να το κάνει για τα δύο άκρα του tunnel. Με αυτό τον τρόπο απλοποιήθηκε η πληροφορία για τη δρομολόγηση που χρειάζεται.

Όταν το πακέτο φτάσει στην διεύθυνση προορισμού, όπως είναι γνωστό η επικεφαλίδα IP αφαιρείται και το πακέτο συνεχίζει να δρομολογείται με βάση την εσωτερική IP επικεφαλίδα.



Σχήμα 3.2: Η λειτουργία των GRE Tunnel

Τα GRE Tunnels στην υλοποίηση αυτή αποτελούν τις ζεύξεις ανάμεσα στις φυσικές ακραίες συσκευές και την εσωτερική εικονική. Ένα Tunnel για να οριστεί χρειάζεται ένα λογικό interface στα δύο άκρα, με μια ιδιωτική διεύθυνση IP, αλλά και τον ορισμό των άκρων του, που είναι φυσικά interfaces.

Ακολούθως και δεδομένων των ρυθμίσεων των φυσικών διεπαφών από παραπάνω, έχουμε:

```
interface Tunnel1
 ip address 10.10.10.2 255.255.255.252
 # This router's core interface
 tunnel source 147.102.40.203
 # Core router's core interface
 tunnel destination 147.102.7.96
# @Callisto
set interfaces gre unit 2 family inet address 10.10.10.6/30
set interfaces gre unit 2 family tunnel
      source 147.102.7.207
            destination 147.102.7.96
# @CSR
interface Tunnel1
  description To Alderaan Tunnel
  ip address 10.10.10.1 255.255.255.252
  tunnel source 147.102.7.96
  tunnel destination 147.102.40.203
interface Tunnel2
  description To Callisto Tunnel
  ip address 10.10.10.5 255.255.255.252
  tunnel source 147.102.7.96
  tunnel destination 147.102.7.207
```

3.2.3 Δρομολόγηση

Καθώς έχουμε πλέον τις λογικές point-to-point ζεύξεις των tunnel αρκεί να προωθήσουμε την επιθυμητή κίνηση μέσα από τα tunnel.

```
# @ Alderaan
ip route 147.102.39.128
      255.255.255.255.252 Tunnel1
# @CSR
ip route 147.102.39.128
      255.255.255.252 Tunnel2
ip route 147.102.39.62
      255.255.255.255 Tunnel1
# @ Callisto
edit routing-instance FOO
      set instance-type virtual-router
```

```
      set interface gre.2
      set interface vlan.998
      set routing-options static route
            147.102.38.62/32 next-hop 10.10.10.5
```

3.2.4 DSCP ρυθμίσεις

Οι ρυθμίσεις για την εφαρμογή του Quality of Service χωρίζονται σε δύο κομμάτια. Το DSCP Based classification που διενεργείται στα ήδη μαρκαρισμένα πακέτα, και το μαρκάρισμα που θα γίνει βάση των διεθύνσεων IP προορισμού και πηγής που διενεργείται στα εισερχόμενα αμαρκάριστα πακέτα.

DSCP Based Classification

```
# @Alderaan
# Catch traffic with af11 dscp
class-map match-all GOLD_CATCH
 match ip dscp af11
# Give this class 50\% bandwidth
policy-map GOLD
 class GOLD_CATCH
 bandwidth percent 50
# Assign action to interface
interface tunnel1
 service-policy output GOLD
# This is the core facing tunnel
# Exactly the same at the Core Router
# @Callisto
# Define a class
edit class-of-service
  set forwarding-classes queue 1 GOLD
# Catch traffic with af11 dscp
set classifiers dscp core-classifier
      forwarding-class GOLD loss-priority low code-points af11
# A default needed in JUNOS
set schedulers default priority low
set scheduler-maps LefterisDiploma
```

```
      forwarding-class best-effort scheduler default
# Give this class 50\% bandwidth
edit schedulers high_priority
      set transmit-rate percent 50
      set priority low
# Bind class and its scheduler
set scheduler-maps LefterisDiploma
      forwarding-class GOLD scheduler high_priority
#JUNOS define marking with the class
set rewrite-rules dscp dscp-mark-traffic
      forwarding-class GOLD loss-priority low code-point af11
set rewrite-rules dscp dscp-mark-traffic
      forwarding-class best-effort loss-priority low code-point be
# Assign to tunnel
set interfaces gre unit 2 scheduler-map LefterisDiploma
```

IP address classification

```
# @Alderaan
# ACL to catch traffic by Src-Dest
access-list 101 permit ip host 147.102.39.62 host 147.102.39.130
class-map match-all Diploma_MATCH
 match access-group 101
policy-map Diploma_MARKER
 class Diploma_MATCH
  set ip dscp af11
# Assign to Edge interface
interface gigabit0/1
 service-policy input Diploma_MARKER

# @Callisto
# JUNOS Firewall multifield classifier
edit firewall family inet filter LefterisDiploma
      term GOLD_traffic
  set from destination-address 147.102.39.62/32
  set from source-address 147.102.39.130/32
  set then forwarding-class GOLD loss-priority low
# Assign to Edge interface
set interfaces vlan.998 family inet filter input LefterisDiploma
```

Στην εσωτερική συσκευή η κίνηση φτάνει μαρκαρισμένη, επομένως δε χρειάζεται αυτή η διαδικασία. Μοναδική περίπτωση η απαίτηση από την εφαρμογή να γίνεται κάποιο διαφορετικό μαρκάρισμα ή ταξινόμηση.

3.3 Το Ansible

3.3.1 Ansible Modules

Η ενέργεια που εκτελείται από το Ansible ονομάζεται "μονάδα Ansible" (ansible module). Ένα Ansible module είναι ένα μικρό πρόγραμμα/σύνολο εντολών/script γραμμένο είτε σε Python (που είναι και η προεπιλογή λόγω συμβατότητας με το ίδιο το Ansible που είναι ανεπτυγμένο στην ίδια γλώσσα) είτε σε οποιαδήποτε άλλη επιλέξει ο διαχειριστής με κάποιες μικρές αλλαγές.

Το module μπορεί να εκτελεστεί με δύο τρόπους. Είτε να αποσταλλεί με μια μέθοδο SSH (scp για παράδειγμα) στον κόμβο, οποίος και θα το εκτελέσει (deploy), είτε να εκτελεστεί από τον ίδιο κόμβο που "τρέχει" και το ansible. Ο πρώτος τρόπος είναι η προεπιλογή και ο δεύτερος επιλέγεται με την παράμετρο connection και την τιμή local. Η πρώτη περίπτωση χρησιμοποιείται σε περιπτώσεις που ο κόμβος είναι σύστημα που έχει τη δυνατότητα να εκτελέσει προγράμματα/scripts Python ή οποιασδήποτε άλλης γλώσσας είναι γραμμένο το module. Η δεύτερη λοιπόν είναι όπως θα δείξουμε παρακάτω και η πρόσφορη λύση στην περίπτωση που ο κόμβος είναι μια δικτυακή συσκευή, η οποία δεν έχει αυτές τις δυνατότητες. Στην δεύτερη περίπτωση η σύνδεση περιγράφεται και εγκαθίσταται μέσα στο ίδιο το module και πρέπει να κάνει χρήση κάποιου API της δικτυακής συσκευής (π.χ το NETCONF). Υπάρχει ένα πολύ μεγάλο πλήθος modules τα οποία βρίσκουν εφαρμογή σε όλα τα πεδία ρυθμίσεων (π.χ. βάσεις δεδομένων, διαδικτυκοί εξυπηρετητές, εγκατάσταση και ενημέρωση λειτουργικών συστημάτων, δικτυακή τοπολογία, κ.α.).

Ορισμός Τοπολογίας (Inventory)

Το Ansible καταγράφει τους κόμβους τους οποίους διαχειρίζεται μέσα από ένα .ini αρχείο (η προεπιλογή το θέλει να ονομάζεται hosts) το οποίο περιέχει τις βασικές πληροφορίες για την διαχείριση του. Απαραίτητο είνα ένα domain-name ή μια IP διεύθυνση ώστε να μπορεί το Anisble να συνδεθεί σε αυτόν τον κόμβο. Πέρα από αυτό υπάρχουν προεπιλεγμένα

ονόματα μεταβλητών (πχ ansible_host που δίνει την IP διεύθυνση του κόμβου) αλλά μπορούν να οριστούν και οποιαδήποτε επιθυμεί ο διαχειριστής. Τέλος μπορούν οι κόμβοι να χωριστούν ανάλογα την χρήση (staging, production) ή το ρόλο (databases, web-servers) που εξυπηρετούν σε ομάδες (groups).

Tasks, Playbooks, Roles

Τα καθήκοντα (tasks) είναι μια απλή εκτέλεση ενός module όπως αυτή περιγράφηκε παραπάνω. Τα plays είναι ένα σύνολο από tasks και τα playbooks ένα σύνολο από plays. Κάθε play ή playbook πρέπει να ονοματίζει τους κόμβους στους οποίους θέλει να ρυθμίσει. Αυτό γίνεται με το πεδίο hosts που μπορεί να πάρει την τιμή all(αν θέλουμε να ρυθμίσουμε όλους του κόμβους), το όνομα μιας ομάδας ή ή/και ενός κόμβου μόνο. Το Ansible επαναλαμβάνει (loop) τα plays που περιγράφονται για κάθε ένα κόμβο ξεχωριστά.

Η χρηστικότητα όλων αυτών των αφαιρέσεων μπορεί να γίνει κατανόητή ύκολα με ένα παράδειγμα. Ας υποθέσουμε πως θέλουμε να εκκινήσουμε μια εφαρμογή σε ένα σύνολο κόμβων, οι οποίοι μπορεί να διατηρούν διαφορετικές εκδόσεις από τις απαιτήσεις της εφαρμογής (dependencies). Η διαδικασία που πρέπει να ακολουθηθεί είναι ο έλεγχος για τις τρέχουσες εκδόσεις (ή ακόμα και την ύπαρξη) των απαιτήσεων, η εγκατάσταση όσων λείπουν, και τέλος η εγκατάσταση και εκκίνηση της εφαρμογής. Τα playbooks περιγράφουν με απλό τρόπο (σε γλώσσα YAML) την ακολουθία των modules που θα τρέξουν, τον έλεγχο ροής που χρειάζεται, τους κόμβους που επιθυμεί ο διαχειριστής να στοχεύσει ανάλογα το group (πχ μια web εφαρμογή θα εγκατασταθεί στους web-servers). Ένα playbook μπορεί να κληθεί είτε μέσω γραμμής εντολών με την εντολή ansible-playbook είτε μέσω κάποιου script με τη χρήση του Python API του Ansible.

Μεταβλητές (Variables)

Οι μεταβλητές μπορεί να απαιτούνται είτε από το ansible για την εκτέλεση ενός task (πχ η IP διεύθυνση του κόμβου) είτε από το ίδιο το module που θα "τρέξει" (πχ η θύρα tcp μια εφαρμογής που θα εγκαταστήσει σε ένα κόμβο). Οι μεταβλητές μπορούν να δοθούν με μια σειρά τρόπων, στατικούς και δυναμικούς.

Το Ansible έχει ορίσει από προεπιλογή συγκεκριμένα αρχεία τα οποία φορτώνονται στην μνήμη με την κλήση οποιουδήποτε playbook.Αυτά είναι τα αρχεία μέσα στους φακέλους host_vars/ και group_vars/ τα οποία και φέρουν το καθένα το όνομα του κόμβου και της ομάδας που αναφέρονται αντίστοιχα.

Επομένως στο αρχείο host_vars/alderaan.yml περιέχει ορισμένες σε γλώσσα YAML τις μεταβλητές που αφορούν τον κόμβο με το όνομα alderaan. Το αρχείο group_vars/databases.yml περιέχει τις μεταβλητές που αφορούν όλους τους κόμβους που είναι ορισμένοι στο group databases στο αρχείο hosts όπως περιγράφηκε παραπάνω.

Μέσα σε ένα playbook μπορούν να εισαχθούν ad-hoc μέσα από άλλα αρχεία μεταβλητές. Η ισχύ τους διαρκεί μόνο μέχρι την ολοκλήρωση του playbook που της κάλεσε. Η εισαγωγή πρέπει να γίνει στην αρχή του playbook όπως φαίνεται παρακάτω.

Άλλες επιλογές για να εισαχθούν μεταβλητές είναι η προγραμματιστική, στην περίπτωση που η εκτέλεση ενός playbook καλείται με αυτόν τον τρόπο, ή μέσα από τους "ρόλους" (roles) που χρησιμοποιεί το Ansible και θα αναπτυχθούν παρακάτω.

Ένα Παράδειγμα

Η δομή των μεταβλητών,που γράφονται σε YAML, με τα interfaces του κόμβου alderaan φαίνεται παρακάτω:

```
---
#alderaan interfaces
core_interfaces:
  name: gigabit0/0
  ip: 147.102.40.203
  mask: 255.255.255.252

tunnel_ifce:
  -full_name: tunnel1
    ip: 10.10.10.2
    to: csr
    id: 1
    name: tunnel1
    dest_ip: 10.10.10.1

edge_interface:
  name: gigabit0/1
  ip: 147.102.39.1
  mask: 255.255.255.192
```

Εδώ έχουμε το παράδειγμα ενός απλού playbook, από αυτά που χρησιμοποιούνται στην εφαρμογή.

```
---

-name: Intro play checking and configuring basics
 hosts: all
 connection: local

 vars_prompt:
  -name: ”pass”
   prompt: ”Enter password: ”
   private: yes

 pre_tasks:
  -name: Checking NETCONF connectivity on the devices
   wait_for: >
     host={{ansible_host}}
     port={{netconf_port}}
     timeout=5\n
  -name: Import Variables from cached files
   include_vars: ../cached_vars/policy.yml

 roles:
  -modules
  -dscp_core
 gather_facts: no
```

Στο παραπάνω playbook δηλώνεται με σειρά:

- Ότι θα εφαρμοστεί σε όλους τους κόμβους (hosts: all)
- Ότι η το module θα ”τρέξει” στον κόμβο που ”τρέχει” το Ansible (connection: local)
- Προτρέπεται ο χρήστης να δώσει του κωδικούς σύνδεσης (vars_prompt)
- Στα pre_tasks ελέγχεται η συνδεσιμότητα με τη θύρα NETCONF, και εισάγονται οι μεταβλητές
- εφαρμόνται οι ρόλοι (που όπως θα δούμε είναι ”πακέτα” από playbooks) που ονοματίζονται.

Ρόλοι (Roles)

Οι ρόλοι είναι ένα ακόμα επίπεδο αφαίρεσης στη λειτουργία του Ansible, το οποίο προσφέρει για αρχή ένα ολοκληρωμένο πακετάρισμα δράσεων προς εκτέλεση, και ως αποτέλεσμα αυτού και τη δυνατότητα ευκολότερου διαμοιρασμού και επανάχρησιμοποίησης.

Με την κλήση ενός ρόλου εκτελούνται αυτομάτως μια σειρά από playbooks. Οι μεταβλητές και οι απαιτήσεις των playbooks ενός ρόλου μπορούν να περιέχονται μέσα στο ρόλο ή να εισάγονται εξωτερικά. Τα βασικά απαιτούμενα στοιχεία ενός ρόλου είναι ο φάκελος tasks που περιέχει τα playbooks τα οποία θα εκτελεστούν. Από και πέρα η οργάνωση των φακέλων έχει να κάνει με τη φύση του ρόλου.

Εάν ο ρόλος περιέχει tasks που παράγουν αρχεία που θα μεταφορτωθούν σε κάποιον κόμβο, μπορεί να χρησιμοποιηθεί ένας φάκελος files, εάν τα αρχεία αυτά είναι προτυποποιημένα (templates) μπορεί να χρειαστεί ένας φάκελος templates. Τέλος μπορούν να οριστούν όπως και με τα απλά playbooks φάκελοι μεταβλητών όμοιοι με τους παραπάνω, ώστε να οριστούν παράμετροι που χρησιμοποιούνται αποκλειστικά για την εκτέλεση του ρόλου, καθώς δεν συγχωνεύονται με τις μόνιμες που περιγράφηκαν παραπάνω. Μια τυπική διάταξη φακέλων και αρχείων για ένα ρόλο θα μπορούσε να είναι, σύμφωνα με το επίσημο documentation[16] του Ansible:

```
roles/
   common/ # this hierarchy represents a "role"
      tasks/ #
         main.yml # <-- tasks file can include smaller files if
             warranted
      handlers/ #
         main.yml # <-- handlers file
      templates/ # <-- files for use with the template resource
         ntp.conf.j2 # <------- templates end in .j2
      files/ #
         bar.txt # <-- files for use with the copy resource
         foo.sh # <-- script files for use with the script resource
      vars/ #
         main.yml # <-- variables associated with this role
      defaults/ #
         main.yml # <-- default lower priority variables for this
             role
      meta/ #
```

```
   main.yml # <-- role dependencies
library/ # roles can also include custom modules
lookup_plugins/ # or other types of plugins, like lookup in
    this case
```

Διάταξη αρχείων του Ansible

Η διάταξη αρχείων και φακέλων που επιλέχθηκε για την εφαρμογή, όσων αφορά το κομμάτι του Ansible είναι η παρακάτω:

```
/sdn_manager
      /qos_provision
      |-- hosts
      |-- push_base.yml
      |-- push_policy.yml
      |-- push_service.yml
      |-- push_routing.yml
      |-- /host_vars
            |-- alderaan.yml
            |-- callisto.yml
            |-- csr.yml
      |-- /roles
            |-- /modules
                  |-- push_config.py
                  |-- junos_install_config.py
            |-- /base
            |-- /tunnels
            |-- /dscp_core
            |-- /marking
            |-- /add_router
            |-- /delete_dscp_core
            |-- /delete_marking
            |-- /peer_connectivity
```

3.3.2 Ορισμός Τοπολογίας

Στο αρχείο hosts τοποθετούνται οι κόμβοι που διαχειρίζεται το Ansible. Στην παρούσα υλοποίηση χωρίζονται σε δύο ομάδες, τους core και τους edge. Σαν μεταβλητές έχουν οριστεί οι IP διευθύνσεις, ο κατασκευαστής, και η θύρα που χρησιμοποιεί για το πρωτόκολλο NETCONF η εκάστοτε συσκευή. Ο κατασκευαστής χρησιμοποιείται για να διακρίνει το Ansible το module που θα χρησιμοποιήσει για την προώθηση των ρυθμίσεων.

```
[core]
csr ansible_host=147.102.7.96 vendor=cisco netconf_port=22

[edge]
callisto ansible_host=147.102.7.207 vendor=juniper netconf_port=830
alderaan ansible_host=147.102.40.203 vendor=cisco netconf_port=22
```

3.3.3 Playbooks

Τα playbooks είναι εκείνα που εισάγουν τις μεταβλητές και καλούν τους ρόλους που θα εφαρμοστούν. Ομαδοποιούν κατά κάποιο τρόπο τις ρυθμίσεις, και οργανώνουν τα διαδικαστικά για να εκτελεστεί η διάδοση των ρυθμίσεων με ομάλότητα. Όλα τα playbooks είναι πανομοιότυπα, με μόνη διαφορά τους ρόλους που καλεί το καθένα. Ένα παράδειγμα είναι το παρακάτω:

```
---

-name: Applying the QoS policies
 hosts: edge
 connection: local

 vars_prompt:
  -name: ”pass”
   prompt: ”Enter password”
   private: yes
```

```
pre_tasks:
  -name: Check connectivity
   wait_for: >
     host={{ansible_host}}
     port={{netconf_port}}
     timeout=5

  -include_vars: ../cached_vars/service.yml

roles:
  -modules
  -marking
gather_facts: no
```

3.3.4 Οι Ρόλοι

Οι ρόλοι που αναπτύχθηκαν μπορούν να χωριστούν σε δύο κατηγορίες. Η πρώτη αποτελείται από τους ρόλους που παράγουν και προωθούν ρυθμίσεις στο δίκτυο, και η δεύτερη από αυτούς που υλοποιούν λειτουργίες για το ίδιο το Ansible. Και οι δύο κατηγορίες έχουν την ίδια οργάνωση αρχείων, χρησιμοποιούν λίγο διαφορετικό μοτίβο στα tasks τους.

Διαχείριση Τοπολογίας

Ο ρόλος add_router διαχειρίζεται την τοπολογία και προσθέτει έναν δικτυακό κόμβο στην κατηγορία edge στο αρχείο hosts. Στο αρχείο /tasks/-main.yml έχουμε τα plays που θα εκτελεσθούν καλώντας το ρόλο.

```
---
#Create from template the host_vars.yml file
#for the router
-name: Create host_vars file and add it in host_vars
  template: >
    src=roles/add_router/templates/host_vars.j2
    dest=host_vars/{{router.name}}.yml

#Add router in the hosts file
```

```
-name: Update hosts file
 template: >
   src=roles/add_router/templates/hosts.j2
   dest=roles/add_router/files/hosts

-name: copy hosts file
 local_action: command rsync roles/add_router/files/hosts hosts
```

Ο ρόλος, λοιπόν, θα δημιουργήσει χρησιμοποιώντας το template module δύο αρχεία. Το ένα με το rsync module θα ανανεώσει το hosts αποθηκεύοντας πλέον μόνιμα το νέο κόμβο στην ομάδα edge, και το άλλο θα αποθηκευτεί στο φάκελο host_vars περιέχοντας τις μεταβλητές του νέου κόμβου.

Ρόλοι Ρυθμίσεων

Οι ρόλοι που αναλαμβάνουν ρυθμίσεις δικτύου έχουν πανομοιότυπη λειτουργία και οργάνωση αρχείων. Οι ρόλοι αυτοί είναι οι: base, tunnels, peer_connectivity, dcsp_core, marking, delete_dscp_core, delete_marking. Το μόνο που αλλάζει είναι τα αρχεία των προτύπων (templates) που παράγουν τις ρυθμίσεις. Για λόγους συνοπτικότητας θα παρουσιαστεί ένας ολόκληρος ρόλος. Η διάταξη αρχείων είναι η παρακάτω για όλους τους ρόλους.

```
|-- /roles
      |-- /marking
            |-- /tasks
                  |-- main.yml
            |-- /templates
                  |-- /cisco
                        |-- marking.set.j2
                  |-- /juniper
                        | --marking.set.j2
            |-- /files
                  |-- alderaan_marking.set
                  |-- callisto_marking.set
```

Στο αρχείο /tasks/main.yml περιγράφονται οι ενέργειες που εκτελούνται κατά την κλήση του ρόλου.

```
---

-name: DSCP marking configuration from templates
 template: >
   src=roles/marking/templates/{{vendor}}/marking.set.j2
   dest=roles/marking/files/{{inventory_hostname}}_marking.set

-name: Push configuration
 junos_install_config: >
   host={{ansible_host}}
   passwd="{{pass}}"
   file=roles/marking/files/{{inventory_hostname}}_marking.set
   replace=yes
   overwrite=false
 when: vendor == 'juniper'

-name: Push configuration
 push_config:
   host={{ansible_host}}
   passw="{{pass}}"
   file=roles/marking/files/{{inventory_hostname}}_marking.set
 when: vendor == 'cisco'
```

Η πρώτη ενέργεια είναι η κατασκευή του αρχείου ρυθμίσεων που θα τοποθετηθεί στον φάκελο files. Φαίνεται πως χρησιμοποιείται το module με το όνομα template το οποίο παίρνει σαν μεταβλητές το μονοπάτι του πρότυπου αρχείου (src) και τοποθετεί το παραγώμενο αρχείο στο μονοπάτι dest. Για το διαχωρισμό των αρχείων χρησιμοποιούνται οι ορισμένες στο αρχείο hosts μεταβλητές, vendor και inventory_hostname.

Με την χρήση του when τοποθετείται ένας έλεγχος - αντίστοιχος του if-then - για να προσδιοριστεί το module που αντιστοιχεί σε κάθε κατασκευαστή. Άρα θα εκτελεστεί η ενέργεια για την οποία αληθεύει η προϋπόθεση του when. Η αναλυτική λειτουργία των modules περιγράφεται.

Templates

Το module του Ansible με το όνομα template χρησιμοποιεί την μηχανή πρότυπων αρχείων (template engine) Jinja2[17] που έχει αναπτυχθεί σε Python. Η σύνταξη της είναι απλή, και μόνη του απαίτηση είναι ένα αρχείο (με κατάληξη .j2) που θα περιγράφει το πως θα κατασκευαστεί το τελικό αρχείο,

και ένα αρχείο με τις μεταβλητές και τα ονόματά τους (key-value format, εδώ σε JSON) ώστε να μπορούν να κληθούν μέσα στο πρώτο αρχείο.

Η λειτουργία αυτή φάνηκε ιδιαίτερα χρήσιμη καθώς οι ρυθμίσεις που χρειάζεται η εφαρμογή όπως παρουσιάζονται είναι οι ίδιες, με εξαίρεση τα ειδικά στοιχεία, οι διευθύνσεις των χρηστών, το εύρος ζώνης κ.ο.κ. Στα templates λοιπόν βρίσκεται ο πυρήνας της λογικής για την συναρμολόγηση των αρχείων με τις ρυθμίσεις για τις συσκευές δικτύου.

Η Jinja2 προσφέρει δυνατότητα επαναλήψεων, αν οι μεταβλητές περιέχουν λίστες, και έλεγχο ροής. Οι μεταβλητές που είναι διαθέσιμες είναι εκείνες που έχουν γίνει διαθέσιμες από το playbook που καλείται, με τον τρόπο που έχει περιγραφεί. Αυτές είναι οι μεταβλητές του αρχείου hosts, των host_vars, και των αρχείων που εισάγονται ανεξάρτητα από τον φάκελο cached_vars.

Με χρήση διπλών αγγειλών εισάγεται η τιμή της μεταβλητής που περιέχεται εντός τους, ενώ με % δήλώνεται μια ενέργεια (επανάληψη, έλεγχος κ.ο.κ.).

Για παράδειγμα έχουμε το template του ρόλου dscp_core είναι.

```
{# DSCP CORE CONFIGURATION #}
{# CORE CLASSIFIERS #}
{% for class in classes %}
class-map match-all {{class.name}}_CATCH
 match ip dscp {{class.code}}

policy-map {{class.name}}
 class {{class.name}}_CATCH
 bandwidth percent {{class.rate}}
{% endfor %}

{# START POLICY #}
{% for tun in tunnel_ifce %}
{% for class in classes %}
interface {{tun.full_name}}
 service-policy output {{class.name}}
{% endfor %}
{% endfor %}
```

Ενώ το πρόσθετο αρχείο μεταβλητών είναι το policy.yml που είναι:

```
---

classes:
  -name: GOLD
    slug: high_priority
    loss: low
    code: af11
    rate: 50
```

και ένα αρχείο host_vars :

```
---
#alderaan interfaces
core_interfaces:
  name: gigabit0/0
  ip: 147.102.40.203
  mask: 255.255.255.252

tunnel_ifce:
  -full_name: tunnel1
    ip: 10.10.10.2
    to: csr
    id: 1
    name: tunnel1
    dest_ip: 10.10.10.1

edge_interface:
  name: gigabit0/1
  ip: 147.102.39.1
  mask: 255.255.255.192
```

Το αρχείο template λοιπόν ακολουθεί κάνει μια πρώτη επαναληψη για κάθε στοιχείο της λίστας classes, και μια δεύτερη για κάθε στοιχείο του της λίστας tunnel_ifce. Οι τιμές των μεταβλητών αντικαθίστανται στις θέσεις των διπλών αγγειλών και εν τέλει παράγεται το παρακάτω αρχείο.

```
class-map match-all GOLD_CATCH
```

```
 match ip dscp af11

policy-map GOLD
 class GOLD_CATCH
 bandwidth percent 50

interface tunnel1
 service-policy output GOLD
```

Στη συνέχεια αυτό προωθείται όπως έχουμε περιγράψει στη συσκευή.

Με απλή ή πιο περίπλοκη χρήση και συνδιασμό αυτών των απλών μηχανισμών παράγονται από τους ρόλους το σύνολο των ρυθμίσεων που απαιτούνται.

3.3.5 To push_config module

Τα modules είναι τα κομμάτια εκείνα του κώδικα που περιγράφουν μια δράση που επιθυμούμε να εκτελεστεί. Μπορεί σε επίπεδο playbook απλά να τα ονοματίζουμε σε ένα αρχείο YAML και να τοποθετούμε ύστερα τις μεταβλητές, αλλά διαβάζοντας το, το Ansilbe βλέπει κώδικα που πρέπει είτε να στείλει για να εκτελέσει ένας κόμβος ή να εκτελέσει το ίδιο τοπικά (connection: local).

Το Ansible προσφέρει τις απαραίτητες βιβλιοθήκες για να μπορέσει να κάνει απλή την ανάπτυξη των modules σε Python, ειδικά με τη χρήση του αντικειμένου module που πολύ απλά δέχεται τις μεταβλητές εισόδου σαν μια απλή του ιδιότητα, και ομοίως τις εξόδους του.

```
module = AnsibleModule(
      argument_spec=dict(
         host=dict(required=True),
         user=dict(required=False, default=os.getenv('USER')),
         passw=dict(required=False, default=None),
         file=dict(required=True)
      )
   )

cur_config = dev.get_config()

   if cur_config != prev_config:
      results['changed'] = True
```

```
    logging.info("Completed")
    module.exit_json(**results)
```

Ένα απλό παράδειγμα είναι το push_config που αναπτύχθηκε για την παρούσα υλοποίηση. Στο παραπάνω κομμάτι φαίνεται ο ορισμός των εισόδων, αλλά και ο τρόπος που παράγεται η λογική τιμή changed που απαιτεί το Ansible, για να ενημερώσει αν άλλαξε κάτι στον κόμβο που εφαρμόστηκε το module. Για την ανάπτυξη του module αυτού χρησιμοποιήθηκε ο cisco_client οποίος αναλύεται παρακάτω.

3.3.6 Το Ansible API

Υπάρχουν δύο τρόποι για να κληθεί ένα playbook και επομένως και ο ρόλος που περιγράφει τις ρυθμίσεις που θέλουμε να εφαρμοστούν στην τοπολογια. Ο πιο εύκολος και απλός είναι από τη γραμμή εντολών χρησιμοποιώντας την εντολή ansible-playbook και το όνομά του μετά. Καθώς θέλαμε να δημιουργήσουμε και μια διεπαφή με το χρήστη/διαχειριστή του δικτύου και της εφαρμογής, αυτό έπρεπε να γίνεται με προγραμματιστικό και αυτόματο τρόπο, όταν εκείνος πατούσε ένα πεδίο στη διαδικτυακή εφαρμογή.

Η προγραμματιστική κλήση ενός playbook είναι δυνατή χρησιμοποιώντας το Python API του Ansible. Αυτό δίνει τη δυνατότητα να ορίσουμε ολόκληρο το περιβάλλον του Ansible με τεράστια λεπτομέρεια, να κάνουμε όσες εξατομισκεύσεις απαιτεί η εφαρμογή, ή ακόμα και να αφήσουμε τα πάντα στις προεπιλεγμένες λειτουργίες. Πέρα από την τοποθεσία των μεταβλητών, του inventory (αρχείο hosts), όλα τα υπόλοιπα ορίζονται εντός των playbooks (όπως για παράδειγμα οι επιπλέον εξωτερικές μεταβλητές που εισάγωνται από τον φάκελο cached_vars). Ένα playbook ορίζεται και εκτελείται με τον τρόπο που φαίνεται. Επιλέχθηκε η δυνατότητα επιλογής του playbook ανάλογα της δράσης που θέλει ο διαχειριστής να εφαρμόσει.

Η κλήση των playbooks

```
pb = PlaybookExecutor(playbooks=[playbook_path], inventory=inventory,
                                    variable_manager=variable_manager
                                        , loader=loader,
```

```
                                    options=options, passwords=
                                        passwords)
pb.run()

----

elif self.operation == 'service':
         playbook_path = '../qos_provision/push_service.yml'
```

Στα δύο κομμάτια του προγράμματος φαίνονται οι συνθήκες για επιλογή τοποθεσία playbook (playbook_path) και η εκτέλεση του με την εντολή run().

Η μετατροπή των δεδομένων

Το πρόγραμμα Ansible API δέχεται από το Django ένα Python Dictionary, δομή αντίστοιχη της JSON. Χρησιμοποιώντας την βιβλιοθήκη Jinja2 το αρχείο αυτό μετατρέπεται σε ένα αρχείο YAML, με τον ίδιο μηχανισμό με αυτό που χρησιμοποιήθηκε για να παραχθούν τα αρχεία ρυθμίσεων (config files). Παρακάτω φαίνεται το template για το αρχείο YAML που ορίζει μια υπηρεσία QoS.

```
---

service:
  name: {{service_name}}
  code: {{class_code}}
  id: {{service_id}}
  peers:{\%for peer in peers\%}
    -name: {{peer.name}}
      ip_add: {{peer.address}}
      router: {{peer.router}}{\% endfor \%}
```

Και το παραγώμενο αρχείο είναι:

```
---

service:
 name: Diploma
 code: af11
 id: 1
 peers:
  -name: HostAlderaan
   ip_add: 147.102.39.62
   router: alderaan

  -name: HostCallisto
   ip_add: 147.102.39.130
   router: callisto
```

Ενώ το κομμάτι κώδικα του API που καλεί την παραπάνω διαδικασία είναι:

```
----
def generate_vars(self):
      if self.operation == 'service':
         tmpl= env.get_template('service.yml.j2')
         with open('../cached_vars/service.yml', 'w') as var_file:
            var_file.write(tmpl.render(self.var_dict))
            var_file.close()
         self.var_file = 'cached_vars/service.yml'
```

Ανάλογα το operation που έχει δοθεί σαν όρισμα, ορίζεται και η τοποθεσία του template, και το όνομα του παραγόμενου αρχείου.

3.4 Ο cisco_client

Το πρόγραμμα του πελάτη αναπτύχθηκε για να καλύψει τις ιδιαίτερες απαιτήσεις των συσκευών του κατασκευαστή Cisco σε σχέση με το πρωτόκολλο NETCONF. Ακολουθήθηκε το πρότυπο της υλοποίησης NET-

CONF over SSH, όσον αφορά το επίπεδο σύνδεσης και μεταφοράς μηνυμάτων, και χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python και τη βιβλιοθήκη υλοποίησης συνδέσεων SSH με το όνομα paramiko.

Ο λόγος που έκανε απαραίτητη την ανάπτυξη ενός πελάτη για τις συσκευές αυτές, είναι η έλλειψη στην υλοποίηση του κατασκευαστή περιγραφής και μοντελοποίησης των δεδομένων που χειρίζεται το NETCONF, σε ορολογία πρωτοκόλλου δεν υπήρχε datastore και περιγραφή του σε μια όποιαδήποτε γλώσσα (ούτε καν την YANG που είναι η τυπική). Αυτό που υποστηρίζεται είναι το περιτύλιγμα των μηνυμάτων NETCONF (wrapping) με όλους τους βασικούς μηχανισμούς που έχουν περιγραφεί για τα μηνύματα σε XML, και ο μηχανισμός σύνδεσης με SSH. Στο επίπεδο του περιεχομένου έχει εισαχθεί από τον κατασκευαστή ένα στοιχείο στην XML, το <cmd>, στο οποίο εισάγωνται οι εντολές όπως θα δίνονταν από μια γραμμή εντολών, και μέσω NETCONF μεταφέρονται και εγκαθίστανται στη συσκευή.

Η έλλειψη datastore αφαιρεί τη δυνατότητα να ανασυρθούν και οι τρέχουσες ρυθμίσεις, επομένως αυτό γίνεται με μια απλή εντολή (show running-configuration) μέσω απλού SSH και όχι NETCONF, για να μπορέσει αν γίνει η σύγκριση και να ελεγχθεί η επιτυχία μιας εγκατάστασης νέων ρυθμίσεων.

3.4.1 Η συνδεση

Η αρχικοποίηση της σύνδεσης και κάποιες βασικές διαδικασίες ορίζονται στο αρχείο transport.py, από την κλάση Session. Με χρήση της βιβλιοθήκης paramiko[18] ανοίγει ένα socket, και επιστρέφεται για τους υπόλοιπους χειρισμούς από τις συναρτήσεις της κλάσης.

```
def set_channel(self):
      logger.info(”Connecting to the server”)
# Defining an SSH client to handle NETCONF connection
   client = paramiko.SSHClient()
   client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# Socket
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   s.connect((self.__ip, 22))
# Paramiko transport handler
   trans = paramiko.Transport(s)
   trans.connect(username=self.__username, password=self.__password)
   trans.set_keepalive(60)
# Gets a paramiko.Channel object
   c = trans.open_session(window_size=100000)
```

```
   self.channel = c
```

Σημαντική συνάρτηση είναι η hello_netconf που αποστέλλει το HELLO μήνυμα του NETCONF, και η send_rpc που αποστέλλει ένα μήνυμα NETCONF σε μορφή XML.

```
def send_rpc(self, rpc):
      logger.info("Preparing to send RPC")
   if self.channel.send_ready():
      logger.info(rpc)
      self.channel.send(rpc)
      reply = self.channel.recv(4096)
      reply = reply.decode('utf-8')
   if "ok" in reply:
      logger.info("RPC sent succesfully")
   elif "error" in reply:
      logger.info("Server error")
      logger.debug(reply)
   else:
      logger.info("Channel is down")
```

3.4.2 Κατασκευή Μηνυμάτων

Η κατασκευή μηνυμάτων κατά τον τρόπο που περιγράφηκε γίνεται από την κλάση EditConfigRPC που βρίσκεται στο αρχείο operation.py. Σαν όρισμα δέχεται ένα αρχείο με τις ρυθμίσεις σε μορφή εντολων, και τις εισάγει στο περιτύλιγμα του μηνύματος NETCONF σε μορφή XML. Διαβάζεται η κάθε σειρά του αρχείου και αφού της προστεθεί το στοιχείο XML, <cmd>, εισάγεται στο μήνυμα.

```
for line in a.readlines():
      line = line.replace('\n', '')
   command = wrapper \% line
```

```
   config_block += command
   rpc = NETCONF_WRAP \% config_block
```

Επομένως ένα παράδειγμα ρυθμίσεων και μηνύματος NETCONF θα μπορούσε να είναι το παρακάτω:

```
interface GigabitEthernet0/0
 description Uplink
 ip address 147.102.40.203 255.255.255.192
```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id ="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <edit-config>
    <target><running/></target>
    <config>
      <cli-config-data>
               <cmd>interface GigabitEthernet0/0</cmd>
               <cmd>description Uplink</cmd>
               <cmd>ip address 147.102.40.203 255.255.255.192</cmd>
         </cli-config-data>
    </config>
    </edit-config>
</rpc>]]>]]>
```

3.4.3 Higher Level Manager

Η κλάση manager προσφέρει μια αφαίρεση των παραπάνω κλάσεων για πιο εύκολη χρήση τους. Χρησιμοποιώντας τις προηγούμενες κλάσεις διατηρεί μια σύνδεση NETCONF, κατασκευάζει και αποστέλει τις RPC, και ανασύρρει τις τρέχουσες ρυθμίσεις.

Ένα παράδειγμα χρήσης του είναι το παρακάτω:

```
from cisco_client import manager
import logging


logging.basicConfig(level=logging.DEBUG)

# Define a device manager

alderaan = manager.Manager(ip="147.102.40.203", username="lefteris",
    password="******")
alderaan.connect()

if alderaan.is_connected():
   try:
      prev = alderaan.get_config()
      alderaan.edit_config('test_config.conf')
      cu = alderaan.get_config()
   except Exception as err:
      logging.error(err)
   if prev != cu:
      print("Changed")
   else:
      print("Nothing changed")
else:
   print("xaireta mou ton platano")
```

Εδώ γίνεται σύνδεση με τον alderaan.cn.ece.ntua.gr με διεύθυνση IP την 147.102.40.203. Γίνεται πιστοποίηση SSH, και προωθούνται οι εντολές που βρίσκονται στο αρχείο test_config.conf. Επίσης ανασύρρονται οι ρυθμίσεις πριν και μετά την αποστολή για να επιβεβαιωθεί αν κάτι άλλαξε.

3.5 Η διαδικτυακή εφαρμογή

Η διαδικτυακή εφαρμογή αναπτύχθηκε για να διευκολύνει τον διαχειριστή και τη λειτουργία της εφαρμογής σε μια σειρά πραγμάτων. Ένα βασικό είναι η προσφορα μιας εύκολης γραφικής αναπαράστασης

για τον διαχειριστή, ώστε να προσφέρει τις μεταβλητές που χρειάζεται η εφαρμογή. Ο διαχειριστής θα έπρεπε να γράφει σε ένα αρχείο YAML τις μεταβλητές που χρειάζονται, χρησιμοποιώντας και το σωστό σχήμα για να τις περιγράψει. Σε περίπτωση λάθους η κλήση των Ansible playbooks θα έσφαλε και μια μεταφορά ρυθμίσεων στην τοπολογία θα ακυρωνόταν. Το πρόγραμμα parser.py προσφέρει τη λογική για τη δημιουργία των αρχείων YAML, τα οποία και προσφέρει ύστερα στο Ansible για να εκτελέσει τα playbooks. Επομένως απαλοίφεται το πιθανό λάθος.

Προσφέρεται η αποθήκευση πολιτικών, χρηστών, ρυθμίσεων τοπολογίας, αλλά και να υπάρχει μια βασική παρακολούθηση για την κατάσταση υπηρεσιών, αλλά και η δυνατότητα εκκίνησης ή σταματήματός τους με ευκολότερο τρόπο. Επίσης με δεδομένα τα υφιστάμενα κομμάτια της υλοποίησης που έχουν περιγραφεί, είναι εύκολη η επέκταση της λογικής της εφαρμογής, και η πρόσθεση δυνατοτήτων παρακολούθησης της εφαρμογής (monitoring) κ.α.

3.5.1 Το framework Django

Το Django[5] είναι ένα πακέτο βιβλιοθηκών που προσφέρουν ένα σύνολο υψηλού επιπέδου λειτουργιών για την ανάπτυξη εφαρμογών διαδικτύου. Αυτό σημαίνει πως με αρκετά εύκολο και γρήγορο τρόπο, και με μια μικρή ποσότητα κώδικα μπορούν να παραχθουν αρκετά πλούσιες λειτουργίες.

Τα models

Με τον όρο models το Django περιγράφει τα δεδομένα που θα κρατήσει στη βάση του και τη σχέση που θα έχουν μεταξυ τους. Κάθε μοντέλο δέχεται μια σειρά μεταβλητών, ο τύπος των οποίων μπορεί να περιγραφεί από ένα μεγάλο σύνολο ήδη ορισμένων. Ένα παράδειγμα είναι το παρακάτω.

```
class Router(models.Model):

   name = models.CharField(max_length=20)
   ip = models.GenericIPAddressField(protocol='IPv4')
   vendor = models.CharField(max_length=10, choices=(('cisco', 'CISCO
       '), ('juniper', 'JUNIPER')))
```

```
    core = models.ForeignKey('Interface', related_name='core', default
        ='loop')
    edge = models.ForeignKey('Interface', related_name='edge', default
        ='loop')
    tunnel = models.ForeignKey('Tunnel', related_name='tunnel',
        default='0')

class Interface(models.Model):

    name = models.CharField(max_length=20, primary_key=True)
    ip = models.GenericIPAddressField(protocol='IPv4')
    mask = models.CharField(max_length=15)
```

Στο παράδειγμα αυτό φαίνεται ο ορισμός των δρομολογητών που αποθηκεύει το Django. Φαίνεται αυτός του ονόματός του, που είναι ένα πεδίο χαρακτήρων με μέγιστο πλήθος τους 20. Οι όροι ForeignKey αναφέρονται σε σχέση ανάμεσα στο μοντέλο Router και το Interface. Το Router είναι συσχετισμένο με το Interface. Αυτό σημαίνει πως αν ζητηθεί το core/edge ενός Router θα επιστραφεί ένα αντικείμενο σαν αυτό που περιγράφεται στο ορισμό του Interface. Ομοίως και με το tunnel.

Τα Views

Σαν views από το Django ορίζονται οι συναρτήσεις που απαντούν στα HTTP αιτήματα, και επιστρέφουν τα δεδομένα για να εμφανιστούν στο χρήστη. Εκτός από αυτό μπορεί να εκκινήσει και οποιαδήποτε άλλη διαδικασία έχει οριστεί. Για παράδειγμα με αυτό τον τρόπο αντιδρά η εφαρμογή στο πάτημα από το χρήστη στον κουμπί "Start" που εκκινεί μαι υπηρεσία Quality of Service.

```
def apply(request):

    service = Service.objects.get(name=request.POST['service_name'])
    peers = service.peers.all()
    var_dict = parser('service', service, peers=peers)
    push(var_dict, operation='service')
    service.is_pushed = True
    service.save()
```

```
   return HttpResponseRedirect('/services/')
```

Εδώ μόλις έρχεται στον εξυπηρετητή το αίτημα ανασύρρεται από τη βάση δεδομένων το αντικείμενο της εφαρμογής (Service) με βάση το όνομά του και ύστερα δίνεται στα δύο βασικά προγράμματα, το parser και το push.

3.5.2 Το πρόγραμμα parser

Ο σκοπός του προγράμματος είναι μετατροπή των δεδομένων που ανασύρρονται από τη βάση δεδομένων στη μορφή που απαιτούνται από το Ansible ώστε να μπορέσει να εκτελέσει τα playbooks. Σαν όρισμα δίνεται το είδος του μοντέλου που θα μετατραπεί. Για να προστεθεί ένας δρομολογητής, τα δεδομένα του θα επεξεργαστούν με τον τρόπο που φαίνεται:

```
if model == 'router':
      core = {}
      edge = {}
      tunnel = {}
      try:
         final_dict['router_name'] = obj.name
         final_dict['ip'] = obj.ip
         final_dict['vendor'] = obj.vendor
         if obj.vendor=='CISCO':
            final_dict['netconf_port'] = 22
         else:
            final_dict['netconf_port'] = 830

         tun = obj.tunnel
         core_ifce = obj.core
         edge_ifce = obj.edge

         core['name'] = core_ifce.name
         core['ip'] = core_ifce.ip
         edge['name'] = edge_ifce.name
         edge['ip'] = edge_ifce.ip
         edge['mask'] = edge_ifce.mask
```

```
        tunnel['name'] = tun.name
        tunnel['ip'] = tun.ip
        tunnel['to'] = tun.to
        tunnel['id'] = tun.tun_id
        tunnel['core_ip'] = tun.core_ip

        final_dict['core'] = core
        final_dict['edge'] = edge
        final_dict['tunnel'] = tunnel
```

Αντίστοιχη διαδικασία ακολουθείται και για κάθε δεδομένο που ορίζεται.

3.5.3 Το πρόγραμμα push

Η διαδικασία push είναι πάρα πολύ απλή. Το μόνο που κάνει είναι να προσφέρει στο Ansible API, ένα ζεύγος δεδομένων και διαδικασίας ώστε να εκτελεσθεί το αντίστοιχο playbook με τις σωστές μεταβλητές.

4

Αξιολόγηση λειτουργίας

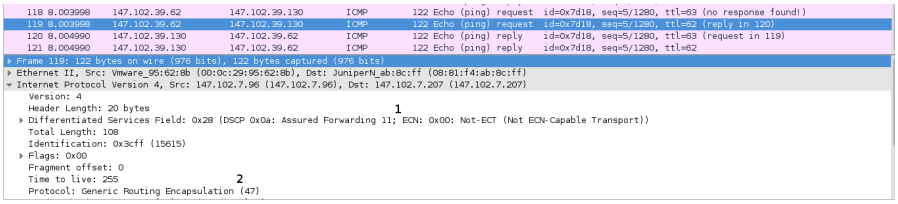
4.1 Μαρκάρισμα

Η αξιολόγηση λειτουργίας της εφαρμογής και η επιβεβαίωση ότι επιτυγχάνει τα επιθυμητά αποτελέσματα, μπορεί να γίνει εύκολα με μια καταγραφή πακέτων σε οποιοδήποτε σημείο της δοκιμαστικής τοπολογίας. Αυτό που θα έπρεπε να διαπιστωθεί είναι η τοποθέτηση του πεδίου DSCP της επικεφαλίδας IP στην επιθυμητή τιμή.

Μια δοκιμαστική καταγραφή πακέτων στον εσωτερικό δρομολογητή μπορεί να γίνει με τις εξής εντολές:

```
monitor capture lefteris interface GiganitEthernet 1 match any
monitor start
```

Η καταγραφή εξάγεται και εμφανίζεται για ευκολότερη ανάγνωση με το πρόγραμμα καταγραφή πακέτων (packet sniffer) Wireshark. Στη θέση 1 παρατηρούμε τον τοποθετημένο κωδικό στην επικεφαλίδα IP, και στη θέση 2 την ύπαρξη της ενθυλάκωσης των GRE tunnels.



Σχήμα 4.1: Καταγραφή πακέτων για την αξιολόγηση λειτουργίας

Με αντίστοιχο τρόπο, με τη χρήση του εργαλείου tcpdump μπορεί να διαπιστωθεί το ίδιο και σε οποιοδήποτε από τους πελάτες, στα άκρα της δοκιμαστικής τοπολογίας.

4.2 Μέτρηση εύρους ζώνης

Για την μέτρηση του εύρους ζώνης ανάμεσα στους δύο πελάτες χρησιμοποιήθηκε το εργαλείο εξέτασης δικτύου iperf[19]. Η λειτουργία του απαιτεί την εκκίνηση ενός πελάτη και ενός εξυπηρετητή στα δύο άκρα ανάμεσα στα οποία θέλουμε να μετρηθεί το εύρος ζώνης.

4.2.1 Βασικές παραδοχές

Για την μέτρηση του εύρους ζώνης χρησιμοποιήθηκε ως αναφορά ένας εξυπηρετητής εκτός των πελατών της υπηρεσίας. Οι πελάτες έχουν διεύθυνση IP 147.102.39.62 και 147.102.39.130 και η μεταξύ τους κίνηση μαρκάρεται από τους δρομολογητές. Ο εξυπηρετητής αναφοράς έχει διεύθυνση IP 147.102.7.38.

Λόγω έλλειψης της άδειας για πλήρη λειτουργία του δρομολογητή CSR1000v που είναι ο εσωτερικός δρομολογητής της δοκιμαστικής τοπολογίας, υπάρχει περιορισμός στο εύρος ζώνης στα 300kbps. Επομένως έγινε αξιολόγηση σε 2 βήματα. Αφενός στον έλεγχο με peak κίνησης σε ένα χαμηλό εύρος ζώνης για έλεγχο της λειτουργίας της αρχιτεκτονικής DSCP, δεδομένου ότι δεν μπορεί να ελεγχθεί μεγάλο εύρος ζώνης. Αφετέρου τον έλεγχο για παροχή ενός μεγάλου ποσοστού του εύρους ζώνης (πχ το 50% που αντιστοιχεί στο ποσοστό της GOLD υπηρεσίας).

Σαν αναφορά χρησιμοποιείται η μέτρηση ανάμεσα στο 147.102.39.62 και τον εξυπηρετητή αναφορά. Στον πίνακα αναφέρονται τα αποτελέσματα για τις δύο κινήσεις, και προς τις δύο κατευθύνσεις. Η κίνηση από το 147.102.39.130 είναι η μαρκαρισμένη και από το 147.102.39.62 είναι η αμαρκάριστη. Οι τιμές είναι όλες σε Kbit per seconend και ο χρόνος που διαρκεί το κάθε test είναι 100 δευτερόλεπτα.

1ο Σενάριο

Σε αυτό το σενάριο επιδιώκεται η επιβεβαίωσει πως τα μαρκαρισμένα πακέτα όντως λαμβάνουν μια ”ιδιαίτερη μεταχείριση”. Λόγω του χαμηλού παρεχόμενου εύρους ζώνης τοποθετείται άνω όριο στην κίνηση στους δρομολογητές. Αυτό τοποθετήθηκε στα 10Kbps σε κάθε φορά της κίνησης.

	Upload	Download	Total
Marked	16	14	30
Unmarked	179	68	247

2ο Σενάριο

Σε αυτό το σενάριο ελέγχεται η αντίδραση του δικτύου στην περίπτωση που δίνεται το 50% του ονομαστικού εύρους ζώνης των διεπαφών των δρομολογητών, το οποίο και είναι το 1Gbps. Το αναμενόμενο είναι να δοθεί ένα μεγάλο ποσοστό στην μαρκαρισμένη κίνηση σε βάρος της αμαρκάριστης.

	Upload	Download	Total
Marked	104	81	185
Unmarked	32	28	60

4.2.2 Σχόλια

Από τις τιμές που παρουσιάστηκαν φαίνεται πως η εφαρμογή QoS με τη χρήση μαρκαρίσματος DSCP αποδίδει, παρά τις ειδικές συνθήκες υπό τις οποίες και ελέγχθηκε. Οι αποκλίσεις στις αναμενόμενες τιμές οφείλονται στον χρονισμό με τον οποίο το πρόγραμμα iperf εκκινεί το test. Εάν ένας χρήστης καταλάβει πρώτος τον εξυπηρετητή του iperf, τότε αποκλείει τον άλλο. Αυτό σημαίνει πως στη διάρκεια των 100 δευτερολέπτων κάποιος πελάτης τρέχει test μόνος του. Δεδομένου λοιπόν, ότι το iperf υπολογίζει τον μέσο όρο των 100 δευτερολέπτων, εμφανίζεται αυτή η τόνωση του εύρους ζώνης, κυρίως της προς υποτίμηση κίνησης σε κάθε περίπτωση από τις δύο.

5

Μελλοντικές Επεκτάσεις

5.1 Άλλες εφαρμογές

Στην εργασία αυτή παρουσιάστηκε μια εφαρμογή για γρήγορή παραμετροποίηση δικτυακών συσκευών με σκοπό την παροχή μιας υπηρεσίας QoS. Η μεθοδολογία με την οποία σχεδιάστηκε και αναπτύχθηκε μπορεί να βρει εφαρμογή σε κάθε άλλη απαιτητική από άποψη χρόνου και πολυπλοκότητας παραμετροποίηση δικτύου. Το πρωτόκολλο NETCONF προσφέρει το μηχανισμό για την αξιόπιστη επικοινωνία και αποστολή ρυθμίσεων σε συσκευές δικτύου, ενώ το Ansible με τη χρήση των πρότυπων αρχείων (templates) είναι ένας συνδιασμός που μπορεί να λειτουργήσει σε κάθε περίπτωση πέρα από αυτή της εφαρμογής QoS με DSCP marking. Η περιγραφή των ρυθμίσεων μέσα στα πρότυπα αρχεία του Ansible, το "πακετάρισμα" τους μέσα σε ολοκληρωμένους, αυτοτελείς και επαναχρησιμοποιήσιμους ρόλους (Ansible Roles) προσφέρει τη δυνατότητα να παραχθεί μια πλήρης εργαλειοθήκη που θα παρέχει αυτοματοποιημένες μια σειρά από βασικές ρυθμίσεις δικτυακών συσκευών (αρχικοποίηση διεπαφών, ορισμός vlans κ.ο.κ).

5.2 Υιοθέτηση της YANG

Παρά τα θετικά που προσφέρει το πρωτόκολλο NETCONF σαν προγραμματιστική διεπαφή με τις δικτυακές συσκευές, η πλήρης λειτουργικότητά του δεν έχει ενσωματωθεί πλήρως από τις κατασκευάστριες εταιρείες δικτυακούς εξοπλισμού. Η εταιρεία Cisco την εισάγει αποκλειστικά στις συσκευές που "τρέχουν" λογισμικό IOS-XR (οι οποίες συνήθως απευθύνονται και μεγάλες υποδομές παρόχων υπηρεσιών), και η εταιρεία Juniper δεν ακολουθεί τον κανόνα για σπονδυλωτή YANG, κάνοντας το μοντέλο των 200.000 γραμμών δύσχρηστο από οποιοδήποτε controller (πχ. το Opendaylight), αλλά και εξαιρετικά χρονοβόρο -εώς αδύνατο- σε υπάρχοντες YANG parser να το επεξεργαστούν. Υιοθετείται παρόλα αυτά του OpenConfig από μια σειρά εταιρειών, αλλά και δίνεται η δυνατότητα από

την Juniper στην τελευταία έκδοση του λογισμικού Junos, για ενσωμάτωση αναπτυγμένων από το χρήστη μοντέλων YANG στις συσκευές της (με ένα απλό μεταφραστικό script για προσαρμογή στο ήδη υπάρχον μοντέλο).

5.3 Επιτάχυνση της αυτοματοποίησης

Το παραπάνω συνεπάγεται την πιο εύχρηστη περιγραφή του περιεχομένου των μηνυμάτων NETCONF. Κάτι που μπορεί να οδηγήσει και σε πιο ολοκληρωμένα εργαλεία για την διαχείριση και παραμετροποίηση των συκευών δικτύου. Ένα παράδειγμα στην εφαρμογή της συγκεκριμένης εργασίας θα μπορούσε να είναι ανάπτυξη ενός προγράμματος που θα λαμβάνει ένα μοντέλο YANG σαν όρισμα και από κει θα μπορεί να εξάγει το σχήμα των δεδομένων που χρησιμοποιεί το Ansible, τόσο για τις μεταβλητές στα αρχεία YAML, όσο και στα πρότυπα αρχεία που θα περιέχουν τις ρυθμίσεις πλέον όχι σε μορφή εντολών, αλλά σε μορφή δομημένων μηνυμάτων NETCONF σε XML. Αυτό μπορεί να παρέχει μεγάλη ευκολία καθώς η διαδίκασία παραγωγής αυτών των αρχείων θα γίνεται απευθείας από το μοντέλο YANG που ακολουθούν οι συσκευές. Και σαν συνέχεια αυτού θα ήταν και η αποτελεσματικότερη αποθήκευση των δεδομένων, σε δομημένα συστήματα και όχι απλά σε αρχεία YAML, για καλύτερη οργάνωση και πρόσβαση στα δεδομένα των ρυθμίσεων των συσκευών του δικτύου.

5.4 Ανάδραση

Όπως επιτάσσει η θεωρία του Αυτόματου Ελέγχου κάθε σύστημα χρειάζεται και μια ανάδραση. Στην παρούσα υλοποίηση αυτή δεν υλοποιείται προγραμματιστικά, και είναι ακόμα στο χέρι του ”ανθρώπου” παρακολουθώντας το δίκτυο να λάβει δράση ή να εκκινήσει οποιαδήποτε διαδικασία ρυθμισης. Η αυτοματοποίηση αυτής της διαδικασίας, όχι μόνο για την παρούσα εφαρμογή QoS, δηλαδή της επεξεργασίας των δεδομένων παρακολούθησης (monitoring) και η εισαγωγή ελέγχων ώστε να λαμβάνονται αυτόματα από το λογισμικό μέτρα για αποφυγή μιας δυσάρεστης κατάστασης ή ακόμα και χρονοπρογραμματισμού της παροχής μια υπηρεσίας θα ήταν κάτι που είναι αναφαίρετο κομμάτι του αυτοματισμού και ως εκ τούτου θα πρόσθετε σε οποιαδήποτε σχετική εφαρμογή.

6

Συμπεράσματα

Σκοπός της διπλωματικής εργασίας ήταν η μελέτη μέσω ενός πρακτικού παραδείγματος της χρηστική αξίας της αυτοματοποίησης της διαδικασίας παραμετροποίησης μιας δικτυακής υποδομής, ιδιαίτερα σε σχετικά σύνθετα σενάρια. Η παροχή μια υπηρεσίας QoS επιλέχθηκε γιατί απαιτεί μια χρονοβόρα διαδικασία σχεδιασμού και παραμετροποίησης για να γίνει διαθέσιμη.

Το πρωτόκολλο NETCONF αποτελεί έναν αξιόπιστο μηχανισμό αποστολής ρυθίσεων σε συσκευές δικτύου, παρότι η υιοθέτηση δεν είναι οριζόντια και αρκετά ομοιόμορφη, αναγκάζοντας στη χρήση διαφορετικών εργαλείων ανά συσκευή με βάση τον κατασκευάστή. Οι συνεδρίες ρύθμισης (configuration session) ασφαλίζονται και πιστοποιούνται επαρκώς με τη χρήση του πλήρως αξιόπιστου πρωτοκόλλου SSH χωρίς την ανάγκη σχεδίασης και ανάπτυξης άλλων μηχανισμών, αλλά και τη δομημένη περιγραφή των δεδομένων του με μια, πιο φιλική στον από την SMI του SNMP, γλώσσα προς τον άνθρωπο το κάνουν αυτή τη στιγμή αρκετά επαρκές. Μέσω της ρύθμισης της δοκιμαστική τοπολογίας για την παροχή της υπηρεσίας QoS για παροχή εύρους ζώνης σύμφωνα με τη ζήτηση, διαπιστώθηκε μια αρκετά μεγάλη διαφορά στον χρόνο ρύθμισης σε σχέση με την χειροκίνητη διαδικασία. Ειδικά με τη χρήση της μιας αφαιρετικής σχεδίασης, μόνο και μόνο λόγω της χρήσης λογισμικού, αλλά και μέσω σύγχρονων εργαλείων (Ansible, Opendaylight κ.α), οι εφαρμογές που αναπτυσσονται είναι επαναχρησιμοποιήσιμες, αποσβένοντας με τον καιρό τον κόστος ανάπτυξής τους με τη ταχύτατη παροχή υπηρεσίων.

Αλλά ακόμα περισσότερο η παραπάνω διαδικασία υποβάλλει τη διαδικασία ρύθμισης της δικτυακής υποδομή στις βέλτιστες πρακτικές που εφαρμόζονται ήδη για την υπόλοιπη πληροφοριακή υποδομή. Η καταγραφή της διαδικασίας ρύθμισης μέσω των εργαλείων διαχείρισης ρυθμίσεων (configuration management), η δυνατότητα επισκόπησης, ελέγχου, η καταγραφή "εκδόσεων" ρυθμίσεων (versioning) κάνει πολύ πιο μεθοδική τη εφαρμογή αλλαγών σε μια δικτυακή υποδομή, πιο εύκολη την ανάρρωση από κάποιο σφάλμα, την ελαχιστοποίηση της επίπτωσης του ανθρώπινου σφάλματος.

Παραρτήματα

Α: Πηγαίος Κώδικας

Ο cisco_client

transport.py

```
import socket
import paramiko
import logging


logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# A hello message for NETCONF session

HELLO = '''<?xml version="1.0" encoding=\"UTF-8\"?>
<hello><capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
     <capability>urn:ietf:params:netconf:capability:writeable-
         running:1.0</capability>
      <capability>urn:ietf:params:netconf:capability:rollback-on-
          error:1.0</capability>
       <capability>urn:ietf:params:netconf:capability:startup:1.0</
           capability>
        <capability>urn:ietf:params:netconf:capability:url:1.0</
            capability>
       <capability>urn:cisco:params:netconf:capability:pi-data-model:1
           .0</capability>
      <capability>urn:cisco:params:netconf:capability:notification:1
          .0</capability>
  </capabilities>
```

```
</hello>]]>]]>'''

# A message for closing NETCONF session

CLOSE = '''
<?xml version="1.0" encoding=\"UTF-8\"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1
    .0">
   <close-session/>
</rpc>]]>]]>'''


class Session:
   def __init__(self, ip, username, password):
      self.__ip = ip
      self.__username = username
      self.__password = password
      self.channel = None

   def set_channel(self):
      logger.info("Connecting to the server")
# Defining an SSH client to handle NETCONF connection
      client = paramiko.SSHClient()
      client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
# Socket
      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      s.connect((self.__ip, 22))
# Paramiko transport handler
      trans = paramiko.Transport(s)
      trans.connect(username=self.__username, password=self.
          __password)
      trans.set_keepalive(60)
# Gets a paramiko.Channel object
      c = trans.open_session(window_size=100000)
      self.channel = c

# Send the NETCONF HELLO message to the device
   def hello_netconf(self):
      if self.channel:
         self.channel.invoke_subsystem("netconf")
         self.channel.send(HELLO)
         rep = self.channel.recv(4096)
         logger.debug(rep)
      else:
         logger.info("Need to establish connection first")
```

```
# RPC Sender Function
   def send_rpc(self, rpc):
      logger.info(”Preparing to send RPC”)
      if self.channel.send_ready():
         logger.info(rpc)
         # Send the RPC
         self.channel.send(rpc)
         reply = self.channel.recv(4096)
         reply = reply.decode(’utf-8’)
         # Check Reply for <ok> element
         if ”ok” in reply:
            logger.info(”RPC sent succesfully”)
         elif ”error” in reply:
            logger.info(”Server error”)
            logger.debug(reply)
      else:
         logger.info(”Channel is down”)

# Send CLOSE message to end the session
   def close(self):
      logger.info(”Closing session”)
      try:
         self.send_rpc(CLOSE)
         self.channel.close()
      except Exception as err:
         logger.error(err)

# Send commands using plain SSH
   def command(self, com):
      if self.channel.send_ready():
         config = ’’
         try:
            # Paramiko’s exec_command
            self.channel.exec_command(com)
            while True:
               data = self.channel.recv(100000)
               data = data.decode(’utf-8’)
               config += data
               if data == ’’:
                  break
            return config
         except Exception as err:
            logger.error(err)
      else:
         logger.error(”Channel error”)
```

operations.py

```
import logging
from os.path import isfile


logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

NETCONF_WRAP = '''
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id ="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1
    .0">
   <edit-config>
   <target><running/></target>
   <config>
      <cli-config-data>
%s</cli-config-data>
   </config>
   </edit-config>
</rpc>]]>]]>'''

# An Edit Configuration RPC object
class EditConfigRPC:
   def __init__(self):
      # CLI command or block of commands
      self.text = None

# Config block must be of format:
# <cmd>mpla mpla</cmd>

   def build_rpc(self, file):
      try:
         # Open configuration file
         a = open(file, 'r+')
         logger.info("Constructing RPC")
         wrapper = '\t<cmd>%s</cmd>\n'
         config_block = ''
         # Include every line of config file
         for line in a.readlines():
            line = line.replace('\n', '')
            command = wrapper % line
            config_block += command
```

```
         # Wrap RPC
         rpc = NETCONF_WRAP % config_block
         logger.debug(rpc)
         logger.info("RPC ready to send")
                  self.text = rpc
      except isfile(file) == False:
         logger.info("%s not a valid filename" % file)
```

manager.py

```
import logging
import cisco_client.operations
import cisco_client.transport


logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Higher Level of NETCONF client

class Manager:
   def __init__(self, ip, username, password):
      self.__session = None
      self.ip = ip
      self.username = username
      self.password = password
   # Connect to a device given its IP and user/pass credentials
   def connect(self):
      s = cisco_client.transport.Session(ip=self.ip, username=self.
          username, password=self.password)
      s.set_channel()
      s.hello_netconf()
      self.__session = s
   # Send an Edit config RPC
   def edit_config(self, filename):
      # command in <cmd>mpla mpla</cmd> format
      rpc = cisco_client.operations.EditConfigRPC()
      rpc.build_rpc(filename)
```

```
      self.__session.send_rpc(rpc.text)
      logger.info("Configuration pushed")
      self.__session.close()
   def is_connected(self):
      if self.__session:
         return True
      else:
         return False
   # Get running config
   # Using plain SSH, not NETCONF
   def get_config(self):
      temp_chan = cisco_client.transport.Session(ip=self.ip, username
          =self.username, password=self.password)
      temp_chan.set_channel()
      config = temp_chan.command("sh run | begin version")
      temp_chan.close()
      return config
```

Ansible API

api.py

```
import os
import sys
from collections import namedtuple
import logging

from jinja2 import Environment, PackageLoader

from ansible.executor.playbook_executor import PlaybookExecutor
from ansible.parsing.dataloader import DataLoader
from ansible.vars import VariableManager
from ansible.inventory import Inventory

logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

# A simple API to remotely call Ansible Playbooks
# Using Ansible Python APIv2
```

```
class AnsibleHandler():
   def __init__(self, var_dict, operation):
      self.var_file = None
      self.operation = operation
      self.var_dict = var_dict

   def call_playbook(self):
      variable_manager = VariableManager()
      loader = DataLoader()
      self.generate_vars()
      # Host file location
      host_list = os.path.abspath('../qos_provision/hosts')
      inventory = Inventory(loader=loader,
                      variable_manager=variable_manager,
                      host_list=host_list)
      # Switch based on operation called
      if self.operation == 'router':
      # Give playbook location
         playbook_path = '../qos_provision/push_base.yml'
      elif self.operation == 'remove_policy':
         playbook_path = '../qos_provision/remove_policy.yml'
      elif self.operation == 'service':
         playbook_path = '../qos_provision/push_service.yml'
      elif self.operation == 'remove_service':
         playbook_path = '../qos_provision/remove_service.yml'
      elif self.operation == 'policy':
         playbook_path = '../qos_provision/push_policy.yml'
      elif self.operation == 'routing':
         playbook_path = '../qos_provision/push_routing.yml'
      else:
         logger.error("Invalid operation")
      if not os.path.exists(playbook_path):
         logger.error('Playbook does not exist')
      # Extra options
      Options = namedtuple('Options', ['listtags', 'listtasks', '
          listhosts',
                               'syntax', 'connection','module_path',
                               'forks', 'remote_user', '
                                   private_key_file',
                               'ssh_common_args', 'ssh_extra_args', '
                                   sftp_extra_args',
                               'scp_extra_args', 'become', '
                                   become_method',
                               'become_user', 'verbosity', 'check'])
```

```
    options = Options(listtags=False, listtasks=False, listhosts=
        False,
                  syntax=False, connection='local',module_path=None,

                  forks=100, remote_user='lefteris',
                      private_key_file=None,
                  ssh_common_args=None, ssh_extra_args=None,
                      sftp_extra_args=None,
                  scp_extra_args=None, become=False, become_method=
                      None,
                  become_user='root', verbosity=None, check=False)
    passwords = {}
    # Playbook execution
    pb = PlaybookExecutor(playbooks=[playbook_path], inventory=
        inventory,
                     variable_manager=variable_manager,
                     loader=loader, options=options, passwords=
                         passwords)
    logger.info(pb.run())
    # Generate variables for playbook
def generate_vars(self):
    # Set Jinja2 environment
    env = Environment(loader=PackageLoader('libansible', './vars'))
    # Switch based on operation called
    if self.operation == 'router':
       # Variables' Template location
       tmpl= env.get_template('base.yml.j2')
       # YAML (destination) file location
       with open('../cached_vars/base.yml', 'w') as var_file:
          var_file.write(tmpl.render(self.var_dict))
          var_file.close()
       self.var_file = 'cached_vars/base.yml'
    if self.operation == 'service':
       tmpl= env.get_template('service.yml.j2')
       with open('../cached_vars/service.yml', 'w') as var_file:
          var_file.write(tmpl.render(self.var_dict))
          var_file.close()
       self.var_file = 'cached_vars/service.yml'
    if self.operation == 'routing':
       tmpl= env.get_template('routing.yml.j2')
       with open('../cached_vars/routing.yml', 'w') as var_file:
          var_file.write(tmpl.render(self.var_dict))
          var_file.close()
       self.var_file = 'cached_vars/routing.yml'
    if self.operation == 'policy':
       tmpl = env.get_template('policy.yml.j2')
```

```
        with open('../cached_vars/policy.yml', 'w') as var_file:
           var_file.write(tmpl.render(self.var_dict))
           var_file.close()
        self.var_file = 'cached_vars/policy.yml'
     if self.operation == 'remove_service':
        tmpl = env.get_template('service.yml.j2')
        with open('../cached_vars/service.yml', 'w') as var_file:
           var_file.write(tmpl.render(self.var_dict))
           var_file.close()
        self.var_file = 'cached_vars/service.yml'
     if self.operation == 'remove_policy':
        tmpl = env.get_template('policy.yml.j2')
        with open('../cached_vars/policy.yml', 'w') as var_file:
           var_file.write(tmpl.render(self.var_dict))
           var_file.close()
        self.var_file = 'cached_vars/policy.yml'
```

vars

```
---

router:
 name: {{router_name}}
 core_interfaces:
  name: {{core.name}}
  ip: {{core.ip}}
 edge_interface:
  name: {{edge.name}}
  ip: {{edge.ip}}
  mask: {{edge.mask}}
 tunnels:
  name: {{tunnel.name}}
  ip: {{tunnel.ip}}
  to: {{tunnel.to}}
  id: {{tunnel.id}}
  core_ip: {{tunnel.core_ip}}
 ip: {{ip}}
 netconf_port: {{port}}
 vendor: {{vendor}}
```

```
---

classes:{% for class in policies %}
  -name: {{class.name}}
    slug: {{class.slug}}
    loss: {{class.loss}}
    code: {{class.code}}
    rate: {{class.rate}}
{% endfor %}

---

routing:
  core: {{core}}
  edge:{% for router in edge %}
    -{{router}}{% endfor %}
  peers:{% for peer in peers %}
    -name: {{peer.name}}
      ipadd: {{peer.ip}}
      router: {{peer.router}}{% endfor %}

---

service:
  name: {{service_name}}
  code: {{class_code}}
  id: {{service_id}}
  peers:{%for peer in peers%}
    -name: {{peer.name}}
      ip_add: {{peer.address}}
      router: {{peer.router}}{% endfor %}
```

Ansible core

Playbooks

add_router.yml

```
---
```

```
-name: Adding new edge router
 hosts: all
 connection: local
# Include vars
 pre_tasks:
  -name: vars
   include_vars: ../cached_vars/base.yml
# Add new router in temp group
  -name: add host
   add_host: >
    hostname={{router.name}}
    groups=just_added
# Apply role to add the router
 roles:
  -add_router

-name: Configuring Tunnels
 hosts: all
 connection: local
# Demand password for user lefteris
 vars_prompt:
  -name: ”pass”
   prompt: ”Enter password: ”
   private: yes

 pre_tasks:
  -name: Checking NETCONF connectivity on the devices
   wait_for: >
    host={{ansible_host}}
    port={{netconf_port}}
    timeout=5
  -name: include vars
   include_vars: ../cached_vars/base.yml
# Apply roles
 roles:
  -modules
  -tunnels
  -base
 gather_facts: no
```

push_routing.yml

```
---

-name: Apply Routing
 hosts: all
 connection: local

 vars_prompt:
   -name: ”pass”
    prompt: ”Enter password”
    private: yes

 pre_tasks:
   -name: Check connectiity
    wait_for: >
      host={{ansible_host}}
      port={{netconf_port}}
      timeout=5

   -include_vars: ../cached_vars/routing.yml

 roles:
   -modules
   -peer_connectivity
 gather_facts: no
```

push_policy.yml

```
---

-name: Intro play checking and configuring basics
 hosts: all
 connection: local

 vars_prompt:
   -name: ”pass”
    prompt: ”Enter password: ”
    private: yes

 pre_tasks:
   -name: Checking NETCONF connectivity on the devices
    wait_for: >
```

```
      host={{ansible_host}}
      port={{netconf_port}}
      timeout=5
  -name: Import Variables from cached files
    include_vars: ../cached_vars/policy.yml

 roles:
  -modules
  -dscp_core
 gather_facts: no
```

push_service.yml

```
---

-name: Applying the QoS policies
 hosts: all
 connection: local

 vars_prompt:
  -name: ”pass”
   prompt: ”Enter password”
   private: yes

 pre_tasks:
  -name: Check connectivity
   wait_for: >
     host={{ansible_host}}
     port={{netconf_port}}
     timeout=5

  -include_vars: ../cached_vars/service.yml

 roles:
  -modules
  -marking
 gather_facts: no
```

Configuration Templates

Interfaces

```
{#Cisco interface configuration#}

{#Loop for every interface in host_vars#}
{% for ifce in interfaces %}
interface {{ifce.name}}
 description {{ifce.description}}
 ip address {{ifce.ipadd}} {{ifce.mask}}
 no shut
{% endfor %}

{# Juniper Interface Configuration#}
{#Routed Virtual Interface for a Layer 3 switch#}

{#Loop for every interface in host_vars#}

{% for ifce in interfaces %}
set interfaces vlan unit 998 description {{ifce.description}}
set interfaces vlan unit 998 family inet address {{ifce.ipadd}}/{{
    ifce.msk_prfx}}
set vlans v998 vlan-id 998
set vlans l3-interface vlan.998
set interfaces {{ifce.name}} description {{ifce.description}}
set interfaces {{ifce.name}} unit 0 family ethernet-swtiching port-
    mode access
set interfaces {{ifce.name}} unit 0 family ethernet-switching vlan
    members v998
{% endfor %}
```

Routing

```
{# Cisco static Routes configuration #}
{% if routing.core == inventory_hostname %}
{% for p in routing.peers %}
```

```
{% for tun in tunnel_ifce %}
{% if p.router==tun.to %}
ip route {{p.ipadd}} 255.255.255.255 {{tun.dest_ip}}
{% endif %}
{% endfor %}
{% endfor %}
{% endif %}
{% if inventory_hostname in routing.edge %}
{% for p in routing.peers %}
{% if p.router != inventory_hostname %}
{% for tun in tunnel_ifce %}
ip route {{p.ipadd}} 255.255.255.255 {{tun.dest_ip}}
{% endfor %}
{% endif %}
{% endfor %}
{% endif %}

{# Juniper Routing Instance with static routes #}
{% if routing.core == inventory_hostname %}
{% for p in routing.peers %}
{% for tun in tunnel_ifce %}
{% if p.router==tun.to %}
set routing-instances FOO interface {{tun.name}}
set routing-options static route {{p.ipadd}}/32 next-hop {{tun.
    dest_ip}}
{% endif %}
{% endfor %}
{% endfor %}
{% endif %}
set routing-instances FOO
set routing-instances FOO interface vlan.998
{% if inventory_hostname in routing.edge %}
{% for p in routing.peers %}
{% if p.router != inventory_hostname %}
{% for tun in tunnel_ifce %}
set routing-options static route {{p.ipadd}}/32 next-hop {{tun.
    dest_ip}}
{% endfor %}
{% endif %}
{% endfor %}
{% endif %}
```

DSCP based classification

```
{# DSCP CORE CONFIGURATION #}
{# CORE CLASSIFIERS #}
{% for class in classes %}
class-map match-all {{class.name}}_CATCH
 match ip dscp {{class.code}}

policy-map {{class.name}}
 class {{class.name}}_CATCH
 bandwidth percent {{class.rate}}
{% endfor %}

{# Apply policy to interfaces #}
{% for tun in tunnel_ifce %}
{% for class in classes %}
interface {{tun.full_name}}
 service-policy output {{class.name}}
{% endfor %}
{% endfor %}

{# Juniper DSCP Configuration #}
{# Classification based on marked packets with DSCP #}
{# Set classes #}

{# Loop for every class #}

{%for class in classes%}
set class-of-service forwarding-classes queue 1 {{class.name}}
set class-of-service classifiers dscp core-classifier forwarding-
    class {{class.name}} loss-priority {{class.loss}} code-points {{
    class.code_point}}

{# Queue schedulers #}
set class-of-service schedulers {{class.slug}} transmit-rate percent
    {{class.rate_percent}}
set class-of-service schedulers {{class.slug}} priority low
set class-of-service scheduler-maps LefterisDiploma forwarding-class
    {{class.name}} scheduler {{class.slug}}
set class-of-service schedulers default priority low
set class-of-service scheduler-maps LefterisDiploma forwarding-class
    best-effort scheduler default

{# Mark packets #}
set class-of-service rewrite-rules dscp dscp-mark-traffic forwarding-
    class {{class.name}} loss-priority {{class.loss}} code-point {{
```

```
    class.code_point}}
set class-of-service rewrite-rules dscp dscp-mark-traffic forwarding-
    class best-effort loss-priority low code-point be

{# Apply to interfaces #}

{%for tun in tunnel_ifce%}
set class-of-service interfaces {{tun.name}} unit {{tun.id}}
    scheduler-map LefterisDiploma
{%endfor%}
{%endfor%}
```

DSCP Marking

```
{# Cisco DSCP marking Configuration #}

{# For edge routers only #}

{# Generate access-list for to-mark traffic #}
{# A (not very) simple set of if conditions and for loops#}

{% if inventory_hostname not in groups['core'] %}
access-list 10{{service.id}} permit ip {%for peer in service.peers
    %}{%if peer.router==inventory_hostname%}host {{peer.ip_add}} {%
    endif%}{%endfor%}{%for peer in service.peers%}{%if peer.router!=
    inventory_hostname%}host {{peer.ip_add}} {%endif%}{%endfor%}

{# Access-list based classifier #}

{%for peer in service.peers%}
{%if peer.router==inventory_hostname%}
class-map match-all {{service.name}}_MATCH
 match access-group 10{{service.id}}
{%endif%}
{%endfor%}
policy-map {{service.name}}_MARKER
{%for peer in service.peers%}
{%if peer.router==inventory_hostname%}
 class {{service.name}}_MATCH
  set ip dscp {{service.code}}
{%endif%}
```

```
{%endfor%}

{# Apply classifiers ton interfaces #}

{% for peer in service.peers %}
{% if peer.router == inventory_hostname %}
interface {{edge_interface.name}}
 service-policy input {{service.name}}_MARKER
{% endif %}
{% endfor %}
{% endif %}

{# Minor tweak for core router marking #}
{# Not mandatory #}
{% if inventory_hostname in groups['core'] %}
{% for peer in service.peers %}
access-list {{service.id}} permit {{peer.ip_add}}
{% endfor %}
class-map match-all {{service.name}}_MATCH
 match access-group {{service.id}}
policy-map {{service.name}}_MARKER
 class {{service.name}}_MATCH
 set ip dscp {{service.code}}
{% for tunnel in tunnel_ifce %}
interface {{tunnel.name}}
 service-policy input {{service.name}}_MARKER
{% endfor %}
{% endif %}

{# Juniper configuration for DSCP marking #}
{# Generate a firewall filter to catch to-mark traffic #}

{% for service in services %}
{% for peer in service.peers %}
{% if peer.router==inventory_hostname %}
set firewall family inet filter LefterisDiploma term {{service.class
    }}_traffic from source-address {{peer.ipadd}}/32
{% endif %}
{% if peer.router!=inventory_hostname %}
set firewall family inet filter LefterisDiploma term {{service.class
    }}_traffic from destination-address {{peer.ipadd}}/32
{% endif %}
{% endfor %}
set firewall family inet filter LefterisDiploma term {{service.class
    }}_traffic then forwarding-class {{service.class}} loss-priority
    {{service.policy_loss}}
```

```
set firewall family inet filter LefterisDiploma term default then
    forwarding-class best-effort loss-priority low
{% endfor %}
set interfaces {{edge_interface.name}} family inet filter input
    LefterisDiploma
```

Django Application

models.py

```
from __future__ import unicode_literals

from django.db import models

import os
from collections import namedtuple

from ansible.parsing.dataloader import DataLoader
from ansible.vars import VariableManager
from ansible.inventory import Inventory
from ansible.executor.playbook_executor import PlaybookExecutor

import sys

imp_path = os.path.abspath('../lib')
sys.path.insert(0, imp_path)

from libansible.api import AnsibleHandler
from parsing import *

# Definition of data models

class Router(models.Model):

   name = models.CharField(max_length=20)
   ip = models.GenericIPAddressField(protocol='IPv4')
   vendor = models.CharField(max_length=10, choices=(('cisco', 'CISCO
       '), ('juniper', 'JUNIPER')))
```

```
    core = models.ForeignKey('Interface', related_name='core', default
        ='loop')
    edge = models.ForeignKey('Interface', related_name='edge', default
        ='loop')
    tunnel = models.ForeignKey('Tunnel', related_name='tunnel',
        default='0')

    def __str__(self):
       return self.name

class Interface(models.Model):

    name = models.CharField(max_length=20, primary_key=True)
    ip = models.GenericIPAddressField(protocol='IPv4')
    mask = models.CharField(max_length=15)

    def __str__(self):
       return self.name
class Tunnel(models.Model):

    name = models.CharField(max_length=10, primary_key=True)
    tun_id = models.SmallIntegerField(default=0)
    ip = models.GenericIPAddressField(protocol='IPv4', default='
        10.10.10.0')
    to = models.CharField(max_length=20, default='csr')
    core_ip = models.GenericIPAddressField(protocol='IPv4', default='
        10.10.10.0')

    def __str__(self):
       return self.name

class Policy(models.Model):

    name = models.CharField(max_length=20, primary_key=True)
    slug = models.SlugField()
    loss = models.CharField(max_length=15)
    code_point = models.CharField(max_length=4)
    rate_percent = models.SmallIntegerField(default=0)
    is_pushed = models.BooleanField(default=False)

    def __str__(self):
       return self.name

class Service(models.Model):
    name = models.CharField(max_length=20)
    id = models.SmallIntegerField(primary_key=True)
```

```
   peers = models.ManyToManyField('Peer')
   traffic_class = models.ForeignKey('Policy')
   is_pushed = models.BooleanField(default=False)
   running = models.BooleanField(default=False)

   def __str__(self):
      return self.name
class Peer(models.Model):
   name = models.CharField(max_length=20, primary_key=True)
   ip = models.GenericIPAddressField(protocol='IPv4')
   router = models.ForeignKey('Router', on_delete=models.PROTECT)

   def __str__(self):
      return self.name

# Global method to call Ansible API
# Give operation name and variables location

def push(var_file, operation):

   pusher = AnsibleHandler(var_file, operation)
   pusher.call_playbook()
```

views.py

```
from django.shortcuts import render
from django.http import HttpResponse, HttpResponseRedirect
from django.template import loader
from django.views import View
from django.views.generic import TemplateView
from django.views.generic.edit import FormMixin
from django.views.generic.list import ListView
from django.views.generic.edit import FormView
from .models import *
from .forms import *
from django.urls import reverse
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
from django.contrib.auth.mixins import LoginRequiredMixin
```

```
import sys
from parsing import *

# HTTP Request Handlers

class TopologyView(LoginRequiredMixin, TemplateView):

   template_name = 'basic_app/topology.html'

   def get(self, request, *args, **kwargs):
      form = RouterForm
      ifce_form = InterfaceForm
      tunnel_form = TunnelForm

      object_list = Router.objects.all()

      context = self.get_context_data(**kwargs)
      context['form'] = form
      context['object_list'] = object_list
      context['ifce_form'] = ifce_form
      context['tunnel_form'] = tunnel_form

      return self.render_to_response(context)
class RouterView(LoginRequiredMixin, FormMixin, View):

   model = Router
   success_url = '/policies/'
   login_url = '/login/'
   redirect_field_name = ''
   form_class = RouterForm

   # Get variables from forms
   # If form is valid add an object

   def form_valid(self, form):
      name = form.cleaned_data['name']
      core = form.cleaned_data['core']
      edge = form.cleaned_data['edge']

      tunnel = form.cleaned_data['tunnel']
      ip = form.cleaned_data['ip']
      vendor = form.cleaned_data['vendor']

      new_router = Router(name=name, ip=ip, vendor=vendor)
      router_tunnel = Tunnel.objects.get(name=tunnel)
```

```
      core_ifce = Interface.objects.get(name=core)
      edge_ifce = Interface.objects.get(name=edge)
      new_router.save()
      new_router.core = core_ifce
      new_router.edge = edge_ifce
      new_router.tunnel = router_tunnel
      new_router.save()
      var_dict = parser('router', new_router)
      push(var_dict, operation='router')

      return super(RouterView, self).form_valid(form)

   def post(self, request, *args, **kwargs):
      form = self.get_form()
      if form.is_valid():
         return self.form_valid(form)

class InterfaceView(LoginRequiredMixin, FormMixin, View):

   model = Interface
   success_url = '/topology/'
   login_url = '/login/'
   related_field_name = ''
   form_class = InterfaceForm

   def form_valid(self, form):
      name = form.cleaned_data['name']
      ip = form.cleaned_data['ip']
      mask = form.cleaned_data['mask']
      new_interface = Interface(name=name, ip=ip, mask=mask)
      new_interface.save()
      return super(InterfaceView, self).form_valid(form)

   def post(self, request, *args, **kwargs):
      form = self.get_form()
      if form.is_valid():
         return self.form_valid(form)
      else:
         self.form_invalid(form)

class TunnelView(LoginRequiredMixin, FormMixin, View):

   model = Tunnel
   form_class = TunnelForm
   success_url = '/topology/'
   login_url = '/login/'
```

```
    redirect_field_name = ''

    def form_valid(self, form):
        name = form.cleaned_data['name']
        id = form.cleaned_data['id']
        ip = form.cleaned_data['ip']
        to = form.cleaned_data['to']
        core_ip = form.cleaned_data['core_ip']

        new_tunnel = Tunnel(name=name, tun_id=id,
                                    ip=ip, to=to, core_ip=core_ip)
        new_tunnel.save()

        return super(TunnelView, self).form_valid(form)

    def post(self, request, *args, **kwargs):
        form = self.get_form()
        if form.is_valid():
            return self.form_valid(form)

class PolicyListView(LoginRequiredMixin, ListView, FormMixin):
# Add policy and view policies view
    model = Policy
    success_url = '/services/'
    login_url = '/login/'
    redirect_field_name = ''
    form_class = PolicyForm

    def get_context_data(self, **kwargs):
        context = super(PolicyListView, self).get_context_data(**kwargs
            )
        context['form'] = PolicyForm()
        return context

    def form_valid(self, form):
        name = form.cleaned_data['name']
        slug = form.cleaned_data['slug']
        code_point = form.cleaned_data['code_point']
        loss = form.cleaned_data['loss']
        rate_percent = form.cleaned_data['rate_percent']
        # Add policy to database
        new_policy = Policy(name=name, slug=slug, code_point=code_point
            ,
                                    loss=loss, rate_percent=
                                        rate_percent)
        new_policy.save()
```

```
      return super(PolicyListView, self).form_valid(form)

   def post(self, request, *args, **kwargs):
      form = self.get_form()
      if form.is_valid():
         return self.form_valid(form)
      else:
         return self.form_invalid(form)

class PeerListView(LoginRequiredMixin, ListView, FormMixin):
# add peer and view peers list
   redirect_field_name = ''
   login_url = '/login/'
   model = Peer
   form_class = PeerForm
   success_url = '/policies/'

   def get_context_data(self, **kwargs):
      context = super(PeerListView, self).get_context_data(**kwargs)
      context['form'] = PeerForm()
      return context

   def form_valid(self, form):
      name = form.cleaned_data['name']
      ip = form.cleaned_data['ip']
      router = form.cleaned_data['router']
      # Add the peer to database
      new_peer = Peer(name=name, ip=ip, router=router)
      new_peer.save()
      return super(PeerListView, self).form_valid(form)

   def post(self, request, *args, **kwargs):
      form = self.get_form()
      if form.is_valid():
         return self.form_valid(form)
      else:
         return self.form_invalid(form)

class ServiceListView(LoginRequiredMixin, ListView, FormMixin):
# add service, services, apply service
   login_url = '/login/'
   redirect_field_name = ''
   model = Service
   form_class = ServiceForm
   success_url = '/services/'
```

```
    def get_context_data(self, **kwargs):
        context = super(ServiceListView, self).get_context_data(**
            kwargs)
        context['form'] = ServiceForm()
        return context

    def post(self, request, *args, **kwargs):

        form = self.get_form()
        if form.is_valid():
            return self.form_valid(form)

    def form_valid(self, form):
        name = form.cleaned_data['name']
        id = form.cleaned_data['id']
        peers = form.cleaned_data['peers']
        traffic_class = form.cleaned_data['traffic_class']
        # Add service to database
        new_service = Service(name=name, id=id)
        # Get ForeignKey objects
        tr_class = Policy.objects.get(name=traffic_class)
        new_service.traffic_class = tr_class
        new_service.peers.add(peers)
        new_service.save()
        var_dict = parser('peer', peers)
        push(var_dict, operation="routing")

        return super(ServiceListView, self).form_valid(form)

def index(request):
    if request.user.is_authenticated:
        tmpl = loader.get_template('basic_app/index.html')
        c = {}
        return HttpResponse(tmpl.render(c, request))
    else:
        return HttpResponseRedirect('/login/')

# Function to activate a service

def apply(request):

    service = Service.objects.get(name=request.POST['service_name'])
    peers = service.peers.all()
    # Call parser to give the desired format to the varibles
    var_dict = parser('service', service, peers=peers)
    push(var_dict, operation='service')
```

```
   service.is_pushed = True
   service.save()

   return HttpResponseRedirect('/services/')

# Function to add a policy

def commit(request):

   policy = Policy.objects.get(name=request.POST['policy_name'])
   var_dict = parser('policy', policy)
   push(var_dict, operation='policy')
   policy.is_pushed = True
   policy.save()

   return HttpResponseRedirect('/services/')

# Function to stop a service

def stop(request):

   service = Service.objects.get(name=request.POST['service_name'])
   peers = service.peers.all()
   var_dict = parser('service', service, peers=peers)
   push(var_dict, operation='stop_service')
   service.running = False
   service.save()

   return HttpResponseRedirect('/services/')

# Function to remove policy

def remove_policy(request):

   policy = Policy.objects.get(name=request.POST['policy_name'])
   var_dict = parser('policy', policy)
   push(var_dict, operation='remove_policy')
   policy.is_pushed = False
   policy.save()

   return HttpResponseRedirect('/policies/')

# Delete service

def remove_service(request):
```

```
    service = Service.objects.get(name=request.POST['service_name'])
    peers = service.peers.all()
    var_dict = parser('service', service, peers=peers)
    push(var_dict, operation='remove_service')
    service.is_pushed = False
    service.save()

    return HttpResponseRedirect('/services/')

def login_view(request):
    tmpl = loader.get_template('basic_app/login.html')
    c = {}
    username = request.POST.get('username')
    password = request.POST.get('password')
    user = authenticate(username=username, password=password)
    if user is not None:
        login(request, user)
        return HttpResponseRedirect('/')
    return HttpResponse(tmpl.render(c, request))

def logout_view(request):
    logout(request)
    return HttpResponseRedirect('/login/')
```

forms.py

```
from django import forms
from .models import Peer, Policy, Tunnel, Router, Interface, Service

class PolicyForm(forms.Form):
    name = forms.CharField(max_length=20)
    slug = forms.SlugField()
    code_point = forms.CharField(max_length=4)
    loss = forms.CharField(max_length=15)
    rate_percent = forms.IntegerField()

class PeerForm(forms.Form):

    routers = Router.objects.all()
```

```
   name = forms.CharField(max_length=20)
   ip = forms.GenericIPAddressField(protocol='IPv4')
   router = forms.ModelChoiceField(queryset=routers)

class ServiceForm(forms.Form):
   # Build Choices
   peer_choices = Peer.objects.all()
   policy_choices = Policy.objects.all()
   # Form
   name = forms.CharField(max_length=20)
   id = forms.IntegerField()
   traffic_class = forms.ModelChoiceField(queryset=policy_choices,
       widget=forms.RadioSelect)
   peers = forms.ModelMultipleChoiceField(queryset=peer_choices,
       widget=forms.CheckboxSelectMultiple)

class InterfaceForm(forms.Form):

   name = forms.CharField(max_length=20)
   ip = forms.GenericIPAddressField(protocol='IPv4')
   mask = forms.CharField(max_length=15)

class TunnelForm(forms.Form):

   name = forms.CharField(max_length=30)
   id = forms.IntegerField()
   ip = forms.GenericIPAddressField(protocol='IPv4')
   to = forms.CharField(max_length=20)
   core_ip = forms.GenericIPAddressField(protocol='IPv4')

class RouterForm(forms.Form):

   interfaces = Interface.objects.all()
   tunnels = Tunnel.objects.all()

   name = forms.CharField(max_length=20)
   ip = forms.GenericIPAddressField(protocol='IPv4')
   vendor = forms.CharField()
   core = forms.ModelChoiceField(queryset=interfaces)
   edge = forms.ModelChoiceField(queryset=interfaces)
   tunnel = forms.ModelChoiceField(queryset=tunnels)
```

HTML templates

base.html

```
<html>
<head>
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.7/css/bootstrap.min.css">

<!-- jQuery library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/
    jquery.min.js"></script>

<!-- Latest compiled JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/
    bootstrap.min.js"></script>
</head>
<title>QoS on demand</title>
<body>
<nav class="navbar navbar-default navbar-static-top">
  <div class="container">
    <div class="navbar-header">
     <a class="navbar-brand" href="{% url 'basic_app:index'%}">QoS on
          Demand</a>
     </div>
     <div id="navbar" class="navbar-collapse collapse">
     <form class="navbar-form navbar-right" action="{% url '
         basic_app:logout' %}" method="post">
     {% csrf_token %}
       <div class="control-group">
         <div class="controls">
          <button type="submit" class="btn">Logout</button>
         </div>
       </div>
     </div>
     </form>
</nav>
{%block content%}
{%endblock%}
</body>
</html>
```

login.html

```
<html>
<head>
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-
    BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
     crossorigin="anonymous">

<!-- Optional theme -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
    bootstrap/3.3.7/css/bootstrap-theme.min.css" integrity="sha384-
    rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
     crossorigin="anonymous">

<!-- Latest compiled and minified JavaScript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/
    bootstrap.min.js" integrity="sha384-
    Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
     crossorigin="anonymous"></script>
</head>
<body>

<form class="form-signin" action={% url 'basic_app:login' %} method="
    post" id="login_form">
   {% csrf_token %}
   <h3 class="form-signin-heading">Please sign in</h3>
   <div class="control-group">
     <label class="control-label" for="username">Username</label>
       <div class="controls">
         <input type="text" id="username" name="username" placeholder
             ="Username">
       </div>
   </div>
   <div class="control-group">
     <label class="control-label" for="password">Password</label>
       <div class="controls">
         <input type="password" name="password" id="password"
             placeholder="Password">
       </div>
   </div>
   <div class="control-group">
     <div class="controls">
       <button type="submit" class="btn">Login</button>
     </div>
```

```
   </div>
</form>
</body>
</html>
```

index.html

```
{% extends 'basic_app/base.html' %}
{% block content %}
<div class="jumbotron">
  <div class="container">
    <h2>QoS on Demand</h2>
    <p>
      This is a QoS on demand application frontend.
      Adding details for a DSCP marking configuration, over GRE
          Tunnels
      you can specify a QoS service.
      The settings and variables provided from this web application
      are passed to the Ansible backend, which using NETCONF
          management protocol
      over SSH configures the topology.
    </p>
    <p>
    <a class="btn btn-primary btn-lg" href="/about/" role="button">
        Learn More</a>
    </p>
  </div>
</div>
<div class="container">
  <div class="row">
    <div class="col-md-4">
      <h4> Start Here! </h4>
      <p>
        If you want to set a new service from the start go here
      </p>
      <a class="btn btn-primary btn-lg" href="/topology/" role="button"
          >Let's start</a>
    </div>
  </div>
  <div class="row">
    <div class="col-md-4">
      <h4> Add a peer </h4>
      <p>
```

```
      Add a host to the application, to be used in a service
    </p>
    <a class="btn btn-primary btn-lg" href="/peers/" role="button">
        Add a peer</a>
  </div>
  <div class="col-md-4">
    <h4> Commit a Policy </h4>
    <p>
      Pre-commit a policy, and save it until apply it
    </p>
    <a class="btn btn-primary btn-lg" href="/policies/" role="button
        ">Commit a policy</a>
  </div>
  <div class="col-md-4">
    <h4> Apply a service </h4>
    <p>
      Apply services
    </p>
    <a class="btn btn-primary btn-lg" href="/services/" role="button
        ">Services</a>
  </div>
 </div>
</div>
{% endblock %}
```

service_list.html

```
{% extends 'basic_app/base.html' %}
{% load widget_tweaks %}
{% block content %}
<div class="container">
 <div class="row">
  <div class="col-md-6">
    {% for service in object_list %}
    {% if not service.is_pushed %}
    <div class="panel panel-danger">
     <div class="panel-heading">
       <h4 class="panel-tittle">{{service.name}}</h4>
       <form action="{% url 'basic_app:apply' %}" method="post" id="
           apply_form">
       {% csrf_token %}
       <button class="btn btn-primary btn-sm" type="submit" name="
           service_name" value={{service.name}} form="apply_form"/>
```

```
            Push Service</button>
        </form>
        {% else %}
    <div class="panel panel-success">
      <div class="panel-heading">
        <h4 class="panel-tittle">{{service.name}}</h4>
        <form action="{% url 'basic_app:remove_service' %}" method="
            post" id="remove_form">
        {% csrf_token %}
        <button class="btn btn-primary btn-sm" type="submit" name="
            service_name" value={{service.name}} form="remove_form"/>
            Remove Service </button>
        {% endif %}
      </div>
      <div class="panel-body">
        <div class="row">
          <div class="col-md-6">
            <ul>
              <li> Policy: {{service.traffic_class.name}} </li>
              <li> Guaranted rate: {{service.traffic_class.rate_percent
                  }}%</li>
              {%for peer in service.peers.all%}
              <li>{{peer.name}}@{{peer.ip}}</li>
              {%endfor%}
            </ul>
          </div>
        </div>
      </div>
    </div>
    <br/>
    {% endfor %}
    </form>
  </div>
  <div class="col-md-6">
    <div class="panel panel-info">
      <div class="panel-heading">
        <h4 class="panel-tittle">New Service</h4>
      </div>
      <div class="panel-body">
        <form class="form-horizontal" action="{% url '
            basic_app:services' %}" method="post">
          {% csrf_token %}
          {% for field in form %}
          <div class="form-group">
            <div class="row">
              <div class="col-sm-5">
```

```
            <label for={{field.name.id_for_label}} class="control-
                label">Service {{field.label}}</label>
            </div>
            <div class="col-sm-7 col-sm-pull-1">
            {% render_field field|add_class:"form-control" %}
            </div>
          </div>
        </div>
       {% endfor %}
       <button class="btn btn-primary btn-md" type="submit">Add
           Servcice</button>
       </form>
     </div>
   </div>
  </div>
</div>
{% endblock %}
```

policy_list.html

```
{% extends 'basic_app/base.html' %}
{% load widget_tweaks %}
{% block content %}
<div class="container">
  <div class="row">
    <div class="col-md-6">
     {% for policy in object_list %}
     {% if not policy.is_pushed %}
     <div class="panel panel-danger">
       <div class="panel-heading">
         <h4 class="panel-tittle">{{policy.name}}</h4>
          <form action="{% url 'basic_app:commit' %}" method="post" id
              ="commit_form">
           {% csrf_token %}
           <button class="btn btn-info btn-sm" type="submit" name="
               policy_name" value={{policy.name}} form="commit_form">
               Push Policy</button>
          </form>
       </div>
       {% else %}
       <div class="panel panel-success">
       <div class="panel-heading">
         <h4 class="panel-tittle">{{policy.name}}</h4>
```

```
        <form action="{% url 'basic_app:remove_policy' %}" method="
            post" id="remove_form">
        {% csrf_token %}
          <button class="btn btn-info btn-sm" type="submit" name="
              policy_name" value={{policy.name}} form="remove_form">
              Remove Policy</button>
        </form>
      </div>
      {% endif %}
      <div class="panel-body">
        <p>{{policy.rate_percent}}% Guaranted Bandwidth</p>
        <p>Marking: {{policy.code_point}} DSCP</p>
        <p>Loss Probability {{police.loss}}</p>
      </div>
    </div>
    {% endfor %}
    </form>
  </div>
  <div class="col-md-6">
    <div class="panel panel-info">
      <div class="panel-heading">
        <h4 class="panel-tittle">New Policy</h4>
      </div>
      <div class="panel-body">
        <form class="form-horizontal" action="{% url '
            basic_app:policies' %}" method="post">
          {% csrf_token %}
          {% for field in form %}
          <div class="form-group">
            <div class="row">
              <div class="col-sm-4">
                <label for={{field.name.id_for_label}} class="control-
                    label">Policy {{field.label}}</label>
              </div>
              <div class="col-sm-8 col-sm-pull-1">
              {% render_field field|add_class:"form-control" %}
              </div>
            </div>
          </div>
        {% endfor %}
        <button class="btn btn-primary btn-md" type="submit">Add
            Policy</button>
      </form>
    </div>
  </div>
```

```
    <a class="btn btn-primary btn-md" href="/services" role="button"
        >Continue</a>
 </div>
</div>
{%endblock%}
```

topology.html

```
{% extends 'basic_app/base.html' %}
{% load widget_tweaks %}
{% block content %}
<div class="container">
<div class="row">
 <div class="col-md-6">
  {% for router in object_list %}
  <div class="panel panel-info">
   <div class="panel-heading">
         <h4 class="panel-tittle">{{router.name}}</h4>
   </div>
   <div class="panel-body">
    <p>Management IP: {{router.ip}}</p>
    <h5><b>Interfaces</b></h5>
    <p>Edge Facing Interface: {{router.edge.name}} {{router.edge.ip
        }}</p>
    <p>Core Facing Interface: {{router.core.name}} {{router.core.ip
        }}</p>
    <h5><b>Tunnel to core router</b></h5>
    <p>Tunnel interface: {{router.tunnel.name}}</p>
    <p>Local ip: {{router.tunnel}}</p>
   </div>
  </div>
  {% endfor %}
 </div>
 <div class="col-md-6">
  <div class="row">
  <div class="panel panel-success">
   <div class="panel-heading">
    <h4 class="panel-tittle">New Router Interfaces</h4>
   </div>
   <div class="panel-body">
    <form class="form-horizontal" action="{% url '
        basic_app:interfaces' %}" method="post">
     {% csrf_token %}
```

```
      {% for field in ifce_form %}
       <div class="form-group">
        <div class="row">
         <div class="col-sm-4">
          <label for={{field.name.id_for_label}} class="control-
              label"> {{field.label}}</label>
         </div>
         <div class="col-sm-8 col-sm-pull-1">
         {% render_field field|add_class:"form-control" %}
         </div>
        </div>
       </div>
      {% endfor %}
      <button class="btn btn-primary btn-md" type="submit">Add
          Interface</button>
    </form>
    </div>
</div>
</div>
<div class="row">
<div class="panel panel-success">
  <div class="panel-heading">
    <h4 class="panel-tittle">Tunnel to core router</h4>
  </div>
  <div class="panel-body">
    <form class="form-horizontal" action="{% url 'basic_app:tunnels
        ' %}" method="post">
      {% csrf_token %}
      {% for field in tunnel_form %}
       <div class="form-group">
        <div class="row">
         <div class="col-sm-4">
          <label for={{field.name.id_for_label}} class="control-
              label"> {{field.label}}</label>
         </div>
         <div class="col-sm-8 col-sm-pull-1">
         {% render_field field|add_class:"form-control" %}
         </div>
        </div>
       </div>
      {% endfor %}
      <button class="btn btn-primary btn-md" type="submit">Add
          Tunnel</button>
    </form>
  </div>
</div>
```

```
  </div>
  <div class="row">
  <div class="panel panel-success">
    <div class="panel-heading">
      <h4 class="panel-tittle">Router Configuration</h4>
    </div>
    <div class="panel-body">
      <form class="form-horizontal" action="{% url 'basic_app:routers
          ' %}" method="post">
        {% csrf_token %}
        {% for field in form %}
          <div class="form-group">
           <div class="row">
            <div class="col-sm-4">
             <label for={{field.name.id_for_label}} class="control-
                 label"> {{field.label}}</label>
            </div>
            <div class="col-sm-8 col-sm-pull-1">
            {% render_field field|add_class:"form-control" %}
            </div>
                  </div>
          </div>
        {% endfor %}
        <button class="btn btn-primary btn-md" type="submit">Add
            Router</button>
      </form>
    </div>
  </div>
  </div>
 </div>
   <a class="btn btn-primary btn-md" href="/peers/" role="button">
       Continue</a>
</div>
</div>
{% endblock %}
```

Βιβλιογραφία

[1] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho. Network Function Virtualization: State-of-the-Art and Research Challenges. 2015.

[2] IETF. NETCONF, RFC6241, 2011.

[3] Network Working Group. DSCP, RFC 2474, 1998.

[4] Ansible Community. Ansible, 2012. URL http://www.ansible.com.

[5] Django Community. Django Web Framework, 2005. URL http://www.docs.djangoproject.com.

[6] Internet Engineering Task Force. URL http://www.ietf.org.

[7] eXtensive Markup Language, 1996. URL https://en.wikipedia.org/wiki/XML.

[8] Internet Engineering Task Force. YANG, RFC 6020, 2010.

[9] Rob Shakir, Anees Shaikh. OpenConfig, 2015. URL https://github.com/openconfig.

[10] Linux Foundation. Opendaylight, 2013. URL https://www.opendaylight.org/.

[11] Open Network Foundation. SDN Architerecture. 2016.

[12] Juniper Networks. Junos Ansible module, 2014. URL https://github.com/Juniper/ansible-junos-stdlib.

[13] Red Hat, Inc. Ansible Python API 2.0, 2012. URL http://docs.ansible.com/ansible/dev_guide/developing_api.html#python-api-2-0.

[14] Cisco. Cisco IOS NETCONF implementation, 2016. URL http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cns/configuration/xe-3s/cns-xe-3s-book/cns-netconf.html.

[15] Network Working Group. Generic Routing Encapsulation, RFC 2784, 2000.

[16] Ansible roles layout.

[17] Jinja2 Community. Jinja2 Template Engine, 2008. URL http://jinja.pocoo.org/.

[18] Jeff Forcier. Paramiko SSHv2 Library, 2008. URL https://github.com/paramiko/paramiko/.

[19] iperf Team. iperf2, Network Testing Tool, 2003. URL http://sourceforge.net/projects/iperf2.