ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)

στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λογισμικό ελεγκτή SDN

Μία βιβλιογραφική και συγκριτική μελέτη

Ηλίας Κουμουνδούρος
ΜΜ4140008

Επιβλέπων: Γιώργος Ξυλωμένος

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2016

Abstract

Software Defined Networking (SDN) has been a hot commercial and research topic in recent years. Many researchers claim that, by decoupling the control and data planes, it presents an antidote to the ossification of the Internet. The proposition for a centralized control plane does indeed promise to solve many of the problems posed by the traditional Internet architecture; however, the ICT community is challenged in terms of assessing the overhyped claims made by SDN technology vendors.

In this thesis, we attempt a comparison of two SDN controller software solutions, using metrics and techniques accepted by the software engineering community, along with conducting a survey on the subject. We start with an introduction to SDN, the controller concept and its significance. We discuss prior research in this area and existing controller software. We continue with a short introduction to software engineering metrics. We choose appropriate metrics, attempt to evaluate them and justify their selection in the context of this work. We then proceed with a comparative analysis of the controllers based on the selected metrics. Afterwards, the results are presented and discussed upon. We conclude with remarks and suggestions on future research.

Περίληψη

Τα τελευταία χρόνια, η Δικτύωση που Ορίζεται με Λογισμικό (Software Defined Networking, SDN) έχει γίνει θέμα αιχμής, και από εμπορική και από ερευνητική άποψη. Πολλοί ερευνητές θεωρούν ότι ο διαχωρισμός του επιπέδου ελέγχου από το επίπεδο δεδομένων που η SDN προτάσσει, αποτελεί το αντίδοτο στην απολίθωση του Διαδικτύου.

Μία πρόκληση που αντιμετωπίζει όμως η κοινότητα των Τεχνολογιών Πληροφορικής και Επικοινωνιών (ΤΠΕ) αφορά την αξιολόγηση των ενθουσιωδών ισχυρισμών των παρόχων τεχνολογίας SDN. Σε αυτή την εργασία, επιδιώκουμε την συγκριτική ανάλυση ελεγκτών SDN χρησιμοποιώντας μετρικές και τεχνικές αποδεκτές από την κοινότητα της τεχνολογίας λογισμικού.

Ξεκινούμε με μία εισαγωγή στην SDN, στην έννοια του ελεγκτή και στη σημασία αυτής. Επιθεωρούμε το ερευνητικό υπόβαθρο του αντικειμένου, καθώς και τις διαθέσιμες λύσεις ελεγκτή SDN. Συνεχίζουμε με μία σύντομη εισαγωγή στις μετρικές τεχνολογίας λογισμικού. Επιλέγουμε κατάλληλες μετρικές, επιχειρούμε την αξιολόγηση τους και στοιχειοθετούμε εμπεριστατωμένα τις επιλογές μας. Κατόπιν, προχωρούμε στη συγκριτική ανάλυση των ελεγκτών με βάση τις επιλεχθείσες μετρικές. Ακολουθεί παρουσίαση των αποτελεσμάτων και συζήτηση επ’αυτών. Ολοκληρώνουμε την εργασία με παρατηρήσεις και προτάσεις σε σχέση με το μέλλον της έρευνας στην SDN.

Περιεχόμενα

Εισαγωγή

Από τις απαρχές του έως σήμερα, το Internet έχει γνωρίσει αλματώδη ανάπτυξη, τόσο σε επίπεδο πλήθους διασυνδεδεμένων συσκευών όσο και απαιτήσεων και ποιοτικών μεταβολών. Μέσα σε 45 περίπου χρόνια έχει εξελιχθεί από αποκλειστικά ερευνητικό και στρατιωτικό δίκτυο σε επίκεντρο της επιχειρηματικής δραστηριότητας, τεχνολογικό καθρέπτη και καταλύτη αλλαγών στην κοινωνική ζωή και την καθημερινότητα. Οι μεταβολές αυτές δύσκολα θα μπορούσαν να είχαν προβλεφθεί από τους πατέρες της δικτύωσης. Η αρχιτεκτονική του είναι γέννημα των τότε αναγκών, προβλέψεων και οικονομικών συγκυριών. Τις πρώτες δεκαετίες από τον αρχικό σχεδιασμό του ARPANET ήταν συνδεδεμένα σε αυτό λίγα ερευνητικά κέντρα ανά τον κόσμο. Σήμερα χρησιμοποιείται από εκατομμύρια ανθρώπους σε σχεδόν κάθε πιθανή δραστηριότητα και διασυνδέει δισεκατομμύρια συσκευών που καλύπτουν μία τεράστια γκάμα υλικού, λογισμικού και τύπων δεδομένων.

Παρ'όλα αυτά, η αρχιτεκτονική του έχει μείνει στάσιμη, φτάνοντας στα όρια του μεγέθους δεδομένων που μπορεί να υποστηρίξει. Η σύζευξη του επιπέδου ελέγχου με το επίπεδο δεδομένων του Διαδικτύου είναι μείζον ανασταλτικό χαρακτηριστικό της κλασσικής σχεδίασης, το οποίο δυσχεραίνει την διεξαγωγή ελέγχων και πειραμάτων [SHE10], την πραγματοποίηση αλλαγών και γενικότερα τη διαχείριση του δικτύου [JSS14], ενώ εγκλωβίζει τον οργανισμό στη χρήση συγκεκριμένων τεχνολογιών δρομολόγησης και προώθησης πακέτων στο δίκτυο (το φαινόμενο γνωστό και ως vendor lock-in [CSD14]).

Η SDN αποτελεί μία νέα πρόταση για το σχεδιασμό συστημάτων δικτύων υπολογιστών και τηλεπικοινωνιών, η οποία προέκυψε σταδιακά, μέσα από προσπάθειες για την αντιμετώπιση των παραπάνω προβλημάτων. Λόγω των προοπτικών της, η SDN είναι ένα από τα κεντρικά θέματα στο πεδίο της δικτύωσης τα τελευταία χρόνια, προσελκύοντας μεγάλο επιχειρηματικό και ερευνητικό ενδιαφέρον. Όντας μία πολύ πρόσφατη εξέλιξη, χαρακτηρίζεται από πολλά προς το παρόν εκκρεμή ζητήματα τα οποία, δεδομένης της ιδιαίτερης σημασίας της SDN, επιδέχονται περισσότερης διερεύνησης.

Αρκετά από τα ανοικτά ερωτήματα αφορούν τον ελεγκτή (controller) ενός συστήματος SDN. Όπως θα δούμε στη συνέχεια, ο ελεγκτής είναι μία από τις κύριες συνιστώσες του συστήματος, όντας το κομμάτι λογισμικού το οποίο είναι υπεύθυνο για τη θεώρηση και διαχείριση του δικτύου.

Σε αυτή την εργασία, συγκρίνουμε δημοφιλείς λύσεις λογισμικού ελεγκτή, συμπληρώνοντας προηγούμενες ερευνητικές προσπάθειες συγκριτικής ανάλυσης με χρήση διαδεδομένων μετρικών από το πεδίο της τεχνολογίας λογισμικού (software engineering).

Αρχικά όμως, κάνουμε μία λεπτομερέστερη, αλλά συνοπτική και βεβαίως μη εξαντλητική, αναφορά στην SDN και τις τεχνολογίες που την υποστηρίζουν.

Σχετικά με την SDN

Διαφορές σε σχέση με την παραδοσιακή αρχιτεκτονική του Internet

Τί είναι, λοιπόν, η SDN; Παραθέτουμε τον ορισμό του οργανισμού Open Networking Foundation:

What is SDN? The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices. [OND16]

Με βάση τον παραπάνω ορισμό, η SDN βασίζεται στην ιδέα του διαχωρισμού του επιπέδου ελέγχου (control plane) από το επίπεδο δεδομένων (data plane) των συσκευών προώθησης πακέτων (switches) του δικτύου. Εδώ έχουμε μία σημαντική απόκλιση από την παραδοσιακή αρχιτεκτονική, στην οποία ένα switch λαμβάνει αποφάσεις τόσο για τη δρομολόγηση (routing) όσο και για την προώθηση (forwarding) των πακέτων μέσα στο δίκτυο.

Στα πλαίσια της αρχιτεκτονικής εκείνης, λόγω της σύζευξης των δύο επιπέδων, της τοποθέτησης και του σχεδιασμού της συσκευής προώθησης πακέτων, οι αποφάσεις για τη δρομολόγηση πρέπει να λαμβάνονται από κοινού με τα άλλα switches. Αυτή η διάταξη έχει τα εξής μειονεκτήματα:

- Επιβάρυνση του αποθηκευτικού χώρου της κάθε συσκευής με δεδομένα τοπολογίας του δικτύου, εκτός από τους πίνακες προώθησης των πακέτων, με αποτέλεσμα τη δυνάμει αύξηση του κόστους των συσκευών [JSS14]
- Επιβάρυνση της απόδοσης (performance) του δικτύου, καθώς κάθε συσκευή πρέπει να εκτελεί τον κατανεμημένο αλγόριθμο δρομολόγησης, εκτός από τις λειτουργίες αποθήκευσης και προώθησης πακέτων
- Λειτουργικά προβλήματα των πρωτοκόλλων δρομολόγησης που οφείλονται στην συμπεριφορά των κατανεμημένων αλγορίθμων δρομολόγησης (για τις τεχνικές λεπτομέρειες και τα προβλήματα των πρωτοκόλλων και αλγορίθμων δρομολόγησης βλ. [TAN03]), με ουσιώδεις επιπτώσεις στη διαθεσιμότητα (availability) του δικτύου

Στις αρχιτεκτονικές SDN ο έλεγχος της κίνησης γίνεται από λογισμικό ελεγκτή σε κεντρικό σημείο, ανεξάρτητο από τις συσκευές αποθήκευσης και προώθησης των πακέτων, εξαλείφοντας τα παραπάνω προβλήματα του κατανεμημένου ελέγχου.

Εφαρμογές και πλεονεκτήματα της αρχιτεκτονικής SDN

Η δρομολόγηση πακέτων είναι μόνο μία από τις εφαρμογές της SDN. Με βάση τον ορισμό που παρέχει το ONF, η θεώρηση του δικτύου από κεντρικό σημείο μέσω λογισμικού επιτρέπει τον αυτοματοποιημένο έλεγχο πολλών συσκευών του δικτύου. Ο διαχωρισμός όμως των επιπέδων ελέγχου και δεδομένων αναπόφευκτα προσθέτει την απαίτηση της επικοινωνίας μεταξύ των δύο

επιπέδων. Με άλλα λόγια, είναι απαραίτητη η ύπαρξη μίας διεπαφής μεταξύ του επιπέδου ελέγχου και του επιπέδου δεδομένων-η επονομαζόμενη και νότια διεπαφή (southbound interface-SBI) [JSS14].

Χωρίς μία πρότυπη (standard) διεπαφή, ο κάτοχος του δικτύου θα επωμιζόταν το βάρος της υλοποίησης μίας ιδιόκτητης διεπαφής, έχοντας να αντιμετωπίσει και τα προβλήματα πραγματοποίησης αλλαγών στο δίκτυο, με την αντικατάσταση π.χ. συσκευών με συσκευές από διαφορετικό πάροχο.

Για το λόγο αυτό, η νότια διεπαφή έχει προτυποποιηθεί από το ONF [ONF16] με αποτέλεσμα το ανοικτό πρότυπο Openflow. Η προτυποποίηση επιτρέπει αλλαγές στο υλικό και το λογισμικό του επιπέδου ελέγχου να πραγματοποιούνται χωρίς να απαιτούνται αντίστοιχες αλλαγές στο επίπεδο δεδομένων, καθ'όσον βέβαια και τα δύο επίπεδα υποστηρίζουν την πρότυπη διεπαφή [JSS14].

Ο διαχωρισμός των δύο επιπέδων, σε συνδυασμό με την προτυποποίηση της νότιας διεπαφής, εισαγάγει την έννοια των εφαρμογών δικτύωσης. Η εξέλιξη αυτή, σε συνδυασμό με την πρόοδο της τεχνολογίας εικονικοποίησης (virtualization [CAJ09, RAU11]) και την παράλληλη πρόοδο των τεχνολογιών υλικού, παρέχει σημαντικές διευκολύνσεις και νέες δυνατότητες αξιοποίησης του δικτύου. Ενδεικτικά αναφέρουμε τα παρακάτω:

- *Διευκολύνει τη μηχανική φορτίου (traffic engineering)* [JSS14]. Ο κεντρικός έλεγχος μειώνει το κόστος συγκέντρωσης στατιστικών δεδομένων για την κίνηση των πακέτων, εξασφαλίζοντας εκτός των άλλων και μια πιο αποδεκτή Ποιότητα Υπηρεσιών (Quality of Service-QoS), όπως αναφέρουμε αμέσως.

- *Διευκολύνει την παροχή Ποιότητας Υπηρεσιών*. Ο πάροχος μίας υπηρεσίας που βασίζεται σε δίκτυο αρχιτεκτονικής SDN μπορεί να εγγυηθεί ότι διάφορες παράμετροι ποιότητας της υπηρεσίας έχουν ένα συγκεκριμένο κάτω όριο, υψηλότερο σε σχέση με την παραδοσιακή αρχιτεκτονική. Για παράδειγμα, η SDN διευκολύνει την εξισορρόπηση φόρτου (load balancing), επιτρέποντας στον πάροχο να ανταποκριθεί δυναμικά στις μεταβολές του φορτίου και να εξασφαλίσει ένα υψηλότερο κάτω όριο απόδοσης ή διαθεσιμότητας του δικτύου, στοιχείο απαραίτητο σε απαιτητικές επιχειρηματικές εφαρμογές (business applications) όπως είναι η διεξαγωγή τηλεδιασκέψεων [KRN11].

- *Διευκολύνει τη διαχείριση του δικτύου.* Η διαχείριση του δικτύου της παλαιάς αρχιτεκτονικής είναι μία δραστηριότητα από δύσκολη έως οριακά ακατόρθωτη [JSS14]. Στη SDN, ο κεντρικός έλεγχος, σε συνδυασμό με το μηχανισμό αφαίρεσης (abstraction) που παρέχει μία πρότυπη νότια διεπαφή όπως το Openflow και τις τεχνικές ελέγχου φορτίου που αναφέραμε παραπάνω, καθιστά δυνατή τη συλλογή δεδομένων από οποιαδήποτε συσκευή, ανεξαρτήτως φορέα. Οι αρχικές ρυθμίσεις των συσκευών, καθώς και μεταβολές σε αυτές, μπορούν να λάβουν χώρα αυτόματα και μαζικά. Χάρη στην εικονικοποίηση και τους μηχανισμούς

αφαίρεσης, ο διαχειριστής μπορεί να κάνει αλλαγές στο δίκτυο, αντικαθιστώντας το υλικό μίας συσκευής, χωρίς να χρειαστούν αλλαγές σε άλλα στοιχεία του δικτύου.

- *Διευκολύνει την Εικονικοποίηση Λειτουργιών Δικτύου (Network Functions Virtualization-NFV)*. Για να επεξεργαστούν τα δεδομένα στο δίκτυο, οι οργανισμοί τηλεπικοινωνιών παραδοσιακά εγκαθιστούσαν εξειδικευμένο υλικό για την παροχή υπηρεσιών τείχους προστασίας (firewall), εξισορρόπησης φορτίου κοκ, τα επονομαζόμενα middleboxes. Η αντικατάσταση των middleboxes σε περίπτωση κατάρρευσης του υλικού καθώς και η πραγματοποίηση αλλαγών στην τοπολογία του δικτύου, αν για παράδειγμα χρειαζόταν να αλλάξει η σειρά των λειτουργιών, ήταν απαιτητικές διαδικασίες. Η εισαγωγή του λογισμικού στη δικτύωση επιτρέπει να πραγματοποιηθούν ενέργειες όπως οι παραπάνω με πολύ χαμηλότερο κόστος [CSD14].

- *Διευκολύνει την εφαρμογή πολιτικών στο φορτίο*. Ένας πάροχος μπορεί να διαχειριστεί ευκολότερα τη δρομολόγηση του φορτίου, ώστε να περνάει μόνο από τα δίκτυα που ορίζει. Μπορεί επίσης να εφαρμόσει πολιτικές ασφάλειας στο φορτίο, για παράδειγμα με τη λειτουργικότητα τείχους προστασίας, με καταγραφή (logging) της κίνησης για σκοπούς ελεγκτικής (auditing) κοκ.

- *Παρέχει προοπτικές καινοτομίας στο δίκτυο*. Η προγραμματισιμότητα του δικτύου αυξάνει τις επιλογές λειτουργικότητας-όχι μόνο στο επίπεδο ελέγχου, αλλά και στο επίπεδο δεδομένων. Θα μπορούσαμε, για παράδειγμα, να χρησιμοποιήσουμε τεχνικές Τεχνητής Νοημοσύνης (Artificial Intelligence-AI) ώστε ο ελεγκτής να προσαρμόζει τη συμπεριφορά του αυτόματα, μαθαίνοντας από τη συμπεριφορά του δικτύου.

Ανοικτά προβλήματα

Η SDN εγείρει και κάποιους νέους προβληματισμούς όσον αφορά το σχεδιασμό του ελεγκτή και γενικότερα του δικτύου. Αυτοί έχουν διερευνηθεί σε μεγάλη έκταση στη βιβλιογραφία της SDN γενικότερα, αλλά και στα πλαίσια των εργασιών σύγκρισης ελεγκτών που έχουν προηγηθεί ειδικότερα, στις οποίες θα αναφερθούμε στη συνέχεια. Για το λόγο αυτό, αξίζει η μνεία στα προβλήματα αυτά στην παρούσα ενότητα.

Η προτυποποίηση της βόρειας διεπαφής (northbound interface) αποτελεί ένα από τα μεγάλα ανοιχτά θέματα στον τομέα της SDN. Η βόρεια διεπαφή είναι η διεπαφή μεταξύ του ελεγκτή και των εφαρμογών SDN [JSS14]. Η υιοθέτηση ενός προτύπου θα επέτρεπε σε εφαρμογές SDN να επικοινωνήσουν με τον ελεγκτή ανεξαρτήτως της τεχνολογίας συγκεκριμένου παρόχου.

Ένα άλλο ζήτημα είναι αυτό της απόδοσης του δικτύου. Όπως είναι γνωστό, τα σημερινά μεγέθη δεδομένων απαιτούν πολύ υψηλή ρυθμαπόδοση (throughput) από τη μεριά της συσκευής επιπέδου δεδομένων. Στα πλαίσια της SDN, η συσκευή θα πρέπει να επικοινωνεί με τον ελεγκτή για κάθε

πακέτο που δε συμφωνεί με τα αποθηκευμένα πρότυπα, αυξάνοντας φυσικά κατά τον τρόπο αυτό το φόρτο του δικτύου. Ένας άλλος παράγοντας καθυστέρησης είναι βεβαίως και ο χρόνος επεξεργασίας των μηνυμάτων που δέχεται ο ελεγκτής από το επίπεδο δεδομένων [JSS14]. Η χωροχρονική πολυπλοκότητα των αλγορίθμων του ελεγκτή και οι αντίστοιχες απαιτήσεις της υλοποίησης μπορούν να καθορίσουν σε κάποιες περιπτώσεις την τάξη της καθυστέρησης στο δίκτυο.

Λειτουργικά, η αρχιτεκτονική μοναδικού στιγμιότυπου ελεγκτή έχει το χαρακτηριστικό του μοναδικού σημείο αποτυχίας (single point of failure) [JSS14], διακυβεύοντας τη διαθεσιμότητα του συστήματος [βλ. για παράδειγμα, μία περίπτωση επίθεσης άρνησης υπηρεσίας (Denial of service attack) στο [DOV15]] καθώς το επίπεδο δεδομένων είναι ανίκανο, σε περίπτωση πτώσης του ελεγκτή, να διαχειριστεί πακέτα που δεν αντιστοιχούν σε καμία αποθηκευμένη ροή. Εκτός των άλλων, η αρχιτεκτονική αυτή παρουσιάζει προβλήματα κλιμάκωσης.

Γι'αυτούς τους λόγους, σε μεγάλα δίκτυα στην πράξη δεν χρησιμοποιείται ένα στιγμιότυπο (instance) ελεγκτή, αλλά ένας αριθμός στιγμιότυπων σε ένα φυσικά κατανεμημένο σύστημα με λογικά κεντρικό επίπεδο ελέγχου. Μία κατανεμημένη (distributed) αρχιτεκτονική βελτιώνει την ανθεκτικότητα του συστήματος μέσω του πλεονασμού, καθώς και την κλιμάκωση του. Καθιστά όμως απαραίτητο τον συγχρονισμό της κατάστασης (state) των στιγμιότυπων, ώστε να έχουν κοινή οπτική του ελεγχόμενου δικτύου. Η επικοινωνία των στιγμιότυπων επιτυγχάνεται μέσω της διεπαφής ανατολής-δύσης (east-west interface) [JSS14]. Η απαίτηση της επικοινωνίας μεταξύ των στιγμιότυπων αυξάνει την πολυπλοκότητα του λογισμικού και αποτελεί άλλη μια πηγή καθυστέρησης στο δίκτυο. Και βέβαια δεν επιλύει δια παντός το πρόβλημα της διαθεσιμότητας, αφού μπορεί να προκύψει σημαντική καθυστέρηση μέχρι την εγκαθίδρυση ενός καναλιού επικοινωνίας μεταξύ στιγμιότυπων. Σημειώνεται ότι τα παραπάνω προβλήματα διαθεσιμότητας διογκώνονται στα πλαίσια των ad-hoc δικτύων [HYY16].

Παρ'όλα αυτά, τα κατανεμημένα συστήματα απολαμβάνουν ευρείας διάδοσης σε μεγάλους οργανισμούς. Είναι χαρακτηριστικό [και ενδεικτικό του υψηλού επιπέδου κλιμάκωσης των σύγχρονων κέντρων δεδομένων (datacenters)], το γεγονός ότι η κυκλοφορία ανατολής-δύσης παρουσιάζει μεγάλη αυξητική πορεία τα τελευταία χρόνια, φθάνοντας το ποσοστό του 70% της συνολικής κίνησης στο δίκτυο.

Από τα παραπάνω, προκύπτει το ζήτημα της τοποθέτησης του ελεγκτή στο δίκτυο. Η βέλτιστη τοποθέτηση θα μπορούσε να αντιμετωπίσει σε μεγάλο βαθμό τις δυσκολίες αυτές, όμως ακόμα αποτελεί ένα φλέγον ερευνητικό ερώτημα. Παρά το μεγάλο ενδιαφέρον του, δυστυχώς βρίσκεται έξω από τα όρια της συμπεριφοράς ενός ελεγκτή και συνεπώς αυτής της εργασίας. Για ένα παράδειγμα της έρευνας πάνω σε αυτό το πρόβλημα βλ. [YAO14].

Η εισαγωγή του λογισμικού στη δικτύωση αναμφίβολα επηρεάζει στην πράξη και την ασφάλεια του δικτύου [SCH13], καθώς αυξάνει την επιφάνεια επίθεσης (attack surface): το λογισμικό μπορεί να

περιλαμβάνει ευπάθειες, η εκμετάλλευση των οποίων θα μπορούσε να οδηγήσει στην απώλεια της εμπιστευτικότητας (confidentiality), ακεραιότητας (integrity) και διαθεσιμότητας (availability) των δεδομένων.

Ειδικά σε σχέση με τα συστήματα που χρησιμοποιούν το Openflow, το πρότυπο προβλέπει τη χρήση του πρωτοκόλλου ασφάλειας SSL/TLS για την επικοινωνία στο νότιο κανάλι. Όμως δεν επιβάλλει αυτή τη χρήση [SCH13], με αποτέλεσμα κάποιες υλοποιήσεις να είναι ευάλωτες σε επιθέσεις στην εμπιστευτικότητα.

Τέλος, ως προϊόν λογισμικού, ένας ελεγκτής σχετίζεται με όλα τα προβλήματα που αφορούν κάθε άλλο προϊόν ή υπηρεσία λογισμικού. Ιδανικά, θα πρέπει να είναι απρόσκοπτη η ανάπτυξη εφαρμογών που χρησιμοποιούν τις υπηρεσίες του ελεγκτή, όπως επίσης και η επέκταση ή τροποποίηση της συμπεριφοράς του ελεγκτή. Επίσης θα πρέπει το λογισμικό ελεγκτή να καθιστά ευχερή τον εντοπισμό σφαλμάτων, τον προσδιορισμό των αιτιών τους και την εξάλειψη τους. Αν το λογισμικό δεν καλύπτει αυτές τις προϋποθέσεις, αναπόφευκτα προκύπτουν προβλήματα λειτουργικότητας, απόδοσης και ασφάλειας, αύξηση του κόστους του συστήματος καθώς και οξύνονται πολλά από τα προβλήματα στα οποία αναφερθήκαμε προηγουμένως.

Όσον αφορά την ανάπτυξη εφαρμογών, μία εφαρμογή θα πρέπει, για παράδειγμα, να καθορίζει πώς το επίπεδο δεδομένων (και συνεπώς το επίπεδο ελέγχου) θα ανταποκρίνεται σε εισερχόμενα πακέτα, όπως θα δούμε προσεχώς. Όμως ο προσδιορισμός αυτών των κανόνων σε μία διαδεδομένη γλώσσα προγραμματισμού γενικού σκοπού, όπως η Java ή η Python, είναι μία δύσκολη διαδικασία [JSS14], καθώς υπάρχουν πολλές διαφορετικές περιπτώσεις εισερχόμενων πακέτων που θα πρέπει να προβλέψει ο σχεδιαστής της εφαρμογής. Αυτό το γεγονός υποχρεώνει την ομάδα ανάπτυξης είτε να δημιουργήσει πρόσθετους μηχανισμούς αφαίρεσης πάνω από το λογισμικό ελεγκτή, είτε να αντιμετωπίσει τα προβλήματα που προαναφέρθηκαν.

Η επίγνωση αυτών των ζητημάτων έχει κινητοποιήσει τη δημιουργία νέων γλωσσών προγραμματισμού υψηλότερου επιπέδου, εξειδικευμένων στην ανάπτυξη δικτυακών εφαρμογών, όπως οι γλώσσες που ανήκουν στην οικογένεια Frenetic [FHF11, FRE16].

Αξίζει να αναφέρουμε ότι το λογισμικό δεν αφορά μόνο το επίπεδο ελέγχου αλλά και το επίπεδο δεδομένων. Για ένα παράδειγμα προγραμματισιμότητας του επιπέδου δεδομένων βλ. τη γλώσσα προγραμματισμού P4 [BDG14, SKK15].

Γενικότερα, είναι ανοικτό το ζήτημα της αξιοποίησης των δυνατοτήτων, από πλευράς εργαλείων, που παρέχει η νέα αρχιτεκτονική-και θα παραμείνει ανοικτό για το μεγαλύτερο μέρος του κύκλου ζωής της αγοράς αυτής.

Συνεχίζουμε την εισαγωγή στις τεχνολογίες SDN με μία σύντομη αναφορά στο πρωτόκολλο Openflow, καθώς η δημιουργία του έπαιξε κομβικό ρόλο στην εξέλιξη της SDN. Η αναφορά αυτή είναι επίσης σημαντική για τις ανάγκες της εργασίας μας.

Το πρότυπο Openflow

Κεντρική έννοια στο πρότυπο Openflow είναι αυτή της ροής (δεδομένων) (flow). Μία ροή είναι ένα σύνολο μοτίβων με τα οποία συγκρίνονται τα πακέτα που εισέρχονται σε μία συσκευή επιπέδου δεδομένων, μαζί με ένα σύνολο οδηγιών οι οποίες εφαρμόζονται στα πακέτα που συμφωνούν (match) με το μοτίβο. Οι ροές είναι αποθηκευμένες στη συσκευή δεδομένων, σε σύνολα που στην ορολογία του Openflow ονομάζονται πίνακες ροών (flow tables).

Εάν δεν έχουν ακόμα εγκατασταθεί ροές στη συσκευή, ή εάν ένα εισερχόμενο πακέτο δε συμφωνεί με το μοτίβο καμίας από τις εγκατεστημένες ροές, η συσκευή επιχειρεί να επικοινωνήσει με τον ελεγκτή, ο οποίος και αποφαίνεται το πώς θα πρέπει να αντιμετωπιστεί το πακέτο που βρίσκεται στην ουρά της συσκευής. Ο έλεγχος πραγματοποιείται με τη μορφή ενός μηνύματος από τον ελεγκτή στη συσκευή και μπορεί να αφορά είτε μόνο το συγκεκριμένο πακέτο (μήνυμα PacketOut) είτε το συγκεκριμένο πακέτο καθώς και κάθε άλλο πακέτο το οποίο συμφωνεί με το μοτίβο που καθορίζει ο ελεγκτής, εγκαθιδρύοντας μία νέα ροή στον πίνακα ροών της συσκευής (μήνυμα FlowMod).

Εάν το πακέτο συμφωνεί με κάποιο από τα μοτίβα του πίνακα ροών, τότε εκτελούνται άμεσα οι ενέργειες της ροής, οι οποίες συνήθως (αλλά βεβαίως όχι αποκλειστικά) περιλαμβάνουν προώθηση (output) του πακέτου σε μία από τις θύρες της συσκευής. Εάν η λίστα των ενεργειών της ροής είναι κενή, τότε το πακέτο διαγράφεται από την ουρά της συσκευής (drop).

Είναι εμφανής η ομοιότητα της παραπάνω λειτουργικότητας με αυτή ενός τείχους προστασίας (firewall) και δεν είναι τυχαίο το γεγονός ότι μία από τις εφαρμογές του SDN είναι η υλοποίηση τείχους προστασίας [ONF16].

Ολοκληρώνοντας αυτή την σύντομη εισαγωγή στις τεχνολογίες που περιβάλλουν τον ελεγκτή SDN, θα στραφούμε στα ζητήματα των ελεγκτών καθ’εαυτών, ξεκινώντας από την επισκόπηση της παρελθούσας ερευνητικής δραστηριότητας ανάλυσης ελεγκτών.

Επισκόπηση προηγούμενης ερευνητικής δραστηριότητας

Στο παρελθόν έχουν λάβει χώρα διάφορες ερευνητικές εργασίες ανάλυσης ελεγκτών SDN.

Ο Erickson [ERI13] πραγματεύεται την υλοποίηση και την αρχιτεκτονική του Beacon. Η εργασία [KHA14] επικεντρώνεται στην αξιολόγηση της απόδοσης του Opendaylight.

Αξιοσημείωτη είναι η εργασία [ACI14], όπου αναλύεται η απόδοση της αρχιτεκτονικής SDN/Openflow σε ασύρματα δίκτυα με χρήση του περιβάλλοντος προσομοίωσης δικτύων OMNeT++ [OMN16] και μετρικές ποιότητας υπηρεσίας.

Στο πεδίο όμως της συγκριτικής ανάλυσης, σε σχέση με το συνολικό όγκο του ερευνητικού έργου στο αντικείμενο, ο οποίος παρουσιάζει διαρκώς αυξητική τάση λόγω της επικαιρότητας του, δεν έχουν γίνει πολλές ερευνητικές απόπειρες, όπως είναι για παράδειγμα η θέση του Romero, [ROM12], ή αυτή των [STA15].

Ο [TOO12] παρουσιάζει μία πολυνηματική (multithreaded) εκδοχή του NOX, τον NOX-MT, την οποία συγκρίνει με τους NOX, Beacon και Maestro όσον αφορά την απόδοση. Για τη διεξαγωγή των πειραμάτων χρησιμοποιεί το εργαλείο CBench, το οποίο εκτελείται σε υπολογιστικό σύστημα ανεξάρτητο από τους ελεγκτές.

Οι [SHA13] επιδιώκουν συγκριτική ανάλυση της αρχιτεκτονικής των ελεγκτών NOX, Beacon, Maestro και Floodlight, με κριτήριο εξίσου την απόδοση, χρησιμοποιώντας, μεταξύ άλλων, μετρικές κλιμάκωσης νήματος (thread scalability), κλιμάκωσης switch και καθυστέρησης (latency).

Οι [SHL13] συγκρίνουν δείκτες απόδοσης, κλιμάκωσης, αξιοπιστίας και ασφάλειας για επτά ελεγκτές (NOX, POX, Beacon, Floodlight, MuL, Maestro, Ryu) με χρήση του CBench και του hcprobe, ενός εργαλείου που αναπτύχθηκε στα πλαίσια του ερευνητικού έργου και, σύμφωνα με τους ερευνητές, παρέχει περισσότερες δυνατότητες υλοποίησης των πειραμάτων σε σχέση με το CBench.

Όπως και άλλες συγκριτικές εργασίες, συγκρίνουν τους ελεγκτές στα πλαίσια της εκτέλεσης μιας εφαρμογής switch εκμάθησης επιπέδου 2 (L2 learning switch), η οποία παρέχεται σχεδόν με κάθε πακέτο λογισμικού ελεγκτή και επιλέγεται αφ'ενός λόγω της απλότητας της, αφ'ετέρου επειδή συνιστά βασικό κομμάτι κάθε λύσης ελεγκτή.

Σε σχέση με την απόδοση και την κλιμάκωση, χρησιμοποιούν το CBench και καταγράφουν την καθυστέρηση του ελεγκτή με τη χρήση ενός switch το οποίο στέλνει ένα και μόνο μήνυμα στον ελεγκτή, πριν λάβει την απάντηση. Μετρούν, επίσης, τη ρυθμαπόδοση (throughput) του ελεγκτή σε σχέση με τον αριθμό των switches και τον αριθμό των πυρήνων επεξεργαστή, οι οποίοι εξυπηρετούν τα μηνύματα που λαμβάνουν από τα switches.

Καταλήγουν σε παρόμοια συμπεράσματα με τον [TOO12], καθώς η καθυστέρηση είναι παρόμοια για όλους τους ελεγκτές. Οι πολυνηματικοί ελεγκτές κλιμακώνονται εν γένει γραμμικά με τον αριθμό των switches όσο αυξάνουμε τους επεξεργαστικούς πυρήνες, αλλά η προσθήκη πυρήνων πέρα από τον αριθμό των switches δεν αυξάνει περαιτέρω την απόδοση του συστήματος. Με βάση τα ευρήματα και των δύο εργασιών, η χρήση ενός εξυπηρετητή (server) ελεγκτή με πάνω από 64 switches ελαττώνει σημαντικά την απόδοση του δικτύου, καθιστώντας απαραίτητη την υιοθέτηση μίας κατανεμημένης αρχιτεκτονικής για την κλιμακούμενη χρήση σε πολύ μεγάλα δίκτυα. Το αυτό ισχύει και για τους μονονηματικούς (single threaded) ελεγκτές, οι οποίοι δεν παρουσιάζουν δυνατότητες κλιμάκωσης.

Όσον αφορά την αξιοπιστία του συστήματος, εξομοιώνουν συνθήκες εργασίας ενός δικτύου, χρησιμοποιώντας το hcprobe για να δημιουργήσουν ένα προφίλ εργασίας (work profile) όπου ο ελεγκτής εκτελείται για 24 ώρες και το πλήθος των μηνυμάτων που λαμβάνει κατά τη διάρκεια της ημέρας παρουσιάζει καμπύλη με τη μορφή καμπάνας. Σε αυτό το χρονικό διάστημα μετράται το πλήθος των αποτυχιών (failures) του ελεγκτή (διακοπές εκτέλεσης, κλείσιμο συνδέσεων κλπ).

Το hcprobe χρησιμοποιείται και στην ανάλυση της ασφάλειας καθώς, σύμφωνα με τους ερευνητές, παρέχει τη δυνατότητα εύκολης δημιουργίας πακέτων με τροποποιημένες κεφαλίδες (headers). Ελέγχουν τη συμπεριφορά του ελεγκτή όταν λαμβάνει πακέτα τόσο με τροποποιημένες κεφαλίδες όσο και με τροποποιημένο μήνυμα Openflow.

Καταλήγουν στο συμπέρασμα ότι οι ελεγχθείσες εκδόσεις των ελεγκτών δεν είναι έτοιμες για χρήση σε περιβάλλοντα παραγωγής.

Οι [KHO14] υιοθετούν μία προσέγγιση από το πεδίο της επιστήμης του management. Συγκρίνουν τους ελεγκτές POX, Trema, Ryu, Opendaylight, Floodlight προκειμένου να επιλέξουν τον βέλτιστο με γνώμονα κριτήρια/απαιτήσεις που θα πρέπει αυτοί να ικανοποιούν κατά την κρίση των ερευνητών και σε συνάρτηση με δημόσιες πηγές πληροφόρησης. Αποδίδουν τιμές προτεραιότητας στα κριτήρια στην προκαθορισμένη κλίμακα 1-9 και τιμές προτεραιότητας στους ελεγκτές σύμφωνα με τα επιλεχθέντα κριτήρια. Συμπεραίνουν τον καλύτερο ελεγκτή με βάση αυτά τα κριτήρια, χρησιμοποιώντας μία τροποποιημένη εκδοχή της διαδικασίας λήψης αποφάσεων Analytic Hierarchy Process (AHP) [SAA08].

Η εργασία [THS14] επικεντρώνεται στις διαδικασίες εγκατάστασης ροής (flow setup) των ελεγκτών Trema, Floodlight και NOX, στα πλαίσια εικονικοποίησης δικτύου (network virtualization). Αξιολογούν την υλοποίηση σχετικών τεχνικών, όπως ο τεμαχισμός (slicing) του Trema και εικονικού switch των Floodlight (Virtual switch) και NOX (έμμεσα, μέσω της βιβλιοθήκης Libnetvirt [LNV16]). Στα πλαίσια της εκτέλεσης της εφαρμογής learning switch με κάθε ελεγκτή, καταγράφονται:

- Ο χρόνος μετ'επιστροφής (RTT) σε μηνύματα ICMP echo request και reply (ping)
- Ο χρόνος μεταφοράς TCP συνδέσεων
- Ο χρόνος απόκρισης με μηνύματα UDP

  Η απόδοση του Open VSwitch στις ίδιες μετρήσεις αποτελεί μέτρο σύγκρισης της απόδοσης των ελεγκτών.

  Οι μετρήσεις πραγματοποιούνται σε ένα εικονικό δίκτυο του Mininet [MIN16]. Στο υπολογιστικό σύστημα το οποίο εκτελεί την προσομοίωση συνδέονται δύο ακόμη υπολογιστικά συστήματα ελέγχου.

  Στο πείραμα ICMP, αποστέλλεται ένα μήνυμα ICMP Request το δευτερόλεπτο. Οι μετρήσεις ICMP επαναλαμβάνονται για το πρώτο μήνυμα με διαφορετική καθυστέρηση και για το δεύτερο μήνυμα χωρίς καθυστέρηση, με και χωρίς εγγραφές στους πίνακες ARP των switches καθώς και στα πλαίσια φόρτου δεδομένων TCP που παράγεται με τη χρήση του εργαλείου hping [HPI16]. Επίσης, οι μετρήσεις επαναλαμβάνονται για διαφορετικά μήκη διαδρομής (1, 2, 3 και 4 hops). Ενώ οι ελεγκτές παρουσιάζουν αρκετά διαφορετική μεταξύ τους συμπεριφορά από μέτρηση σε μέτρηση, εν γένει ο NOX φαίνεται να αποδίδει καλύτερα στο παραπάνω πείραμα σε σχέση με τους Trema και Floodlight, με το Open VSwitch να επιτυγχάνει τους χαμηλότερους χρόνους μετ'επιστροφής. Όλες πάντως οι λύσεις παρουσιάζουν σχεδόν τον ίδιο χρόνο μετ'επιστροφής στην περίπτωση του δεύτερου μηνύματος ICMP.

  Οι μετρήσεις TCP επαναλαμβάνονται και αυτές με και χωρίς καθυστέρηση (5ms), όπως επίσης χωρίς φόρτο και με επιπλέον φόρτο δικτύου. Στην απλούστερη περίπτωση της μεταφοράς δεδομένων χωρίς καθυστέρηση και χωρίς εισηγμένο φόρτο, οι ελεγκτές παρουσιάζουν παρόμοια συμπεριφορά, με αύξηση χρόνου μεταφοράς εκθετική σε σχέση με το μέγεθος των δεδομένων-όπως παρατηρούν δε οι ερευνητές, παρουσιάζει ενδιαφέρον το γεγονός ότι σε αυτή την περίπτωση το Open VSwitch (OVS) χαρακτηρίζεται από μεγαλύτερο χρόνο μεταφοράς σε σχέση με τους συγκρινόμενους ελεγκτές. Αποδίδουν το φαινόμενο αυτό στη λειτουργικότητα του OVS, με το οποίο εγκαθιδρύονται ροές σε κάθε βήμα, εν αντιθέσει με τις υπόλοιπες λύσεις που εγκαθιδρύουν ροές μία φορά για ολόκληρο το μονοπάτι. Στην περίπτωση καθυστέρησης χωρίς επιπλέον φόρτο, έχουμε και πάλι εκθετική καμπύλη, με το NOX να έχει την καλύτερη απόδοση, προσεγγίζοντας αυτή του OVS, ακολουθούμενο από το Floodlight και το Trema. Όταν όμως οι ερευνητές εισαγάγουν φόρτο στο υπόβαθρο, η σχέση χρόνου/όγκου δεδομένων αλλάζει, όπως και η απόδοση των ελεγκτών. Στην περίπτωση αυτή το NOX, που σύμφωνα με τους συγγραφείς δεν είναι βελτιστοποιημένο ώστε να εγκαθιδρύει πολλές ροές παράλληλα, αποδίδει κατώτερα σε σχέση με το Floodlight καθώς και το Trema, το οποίο δε φαίνεται να επηρεάζεται από την εισαγωγή του φόρτου στο δίκτυο.

  Οι ερευνητές ολοκληρώνουν την εργασία με πείραμα με UDP μηνύματα, παρατηρώντας ότι μία ροή UDP συμπεριφέρεται διαφορετικά από μία ροή TCP: αφ'ενός το UDP αποτελεί πρωτόκολλο βέλτιστης δυνατής προσπάθειας (best effort), κάτι που έχει ως αποτέλεσμα μεγαλύτερα ποσοστά

απώλειας πακέτων, αφ'ετέρου στο UDP δε νοείται φάση εγκαθίδρυσης σύνδεσης, η οποία δίνει στον ελεγκτή χρόνο να προετοιμάσει τις ροές, κάτι το οποίο αυξάνει την επικοινωνία μεταξύ του ελεγκτή και του switch στη φάση της εγκατάστασης των ροών. Για την προσμέτρηση της απώλειας UDP πακέτων με τρόπο ανεξάρτητο από το ρυθμό μετάδοσης πακέτων, οι ερευνητές εισαγάγουν την μετρική της Ισοδύναμης Απώλειας Πακέτων (Equivalent Packet Losses-EPL, η οποία υπολογίζεται ως εξής:

$$EPL=\frac{packetloss}{packetrate}$$

Έπειτα, συγκρίνουν τις τιμές EPL των ελεγκτών σε συνάρτηση με το ρυθμό μετάδοσης πακέτων. Το πείραμα διεξάγεται υπό συνθήκες παρεμφερείς του TCP, με πακέτα μεγέθους 64 bytes.

Γενικά, τα αποτελέσματα δείχνουν ότι το EPL ανεβαίνει όσο αυξάνεται ο ρυθμός μετάδοσης, για όλους τους ελεγκτές. Το NOX παρουσιάζει υψηλότερο δείκτη EPL σε σχέση με τους υπόλοιπους υπό σύγκριση ελεγκτές.

Οι παραπάνω εργασίες παρουσιάζουν ιδιαίτερο ενδιαφέρον καθώς αποτελούν τις πρώτες προσπάθειες συγκριτικής ανάλυσης ελεγκτών SDN. Είναι επόμενο λοιπόν να υπάρχουν περιθώρια βελτίωσης, τόσο στην ερευνητική περιοχή γενικότερα, όσο και στο δικό μας έργο, όπως θα διαπιστώσουμε στη συνέχεια.

Κατ'αρχάς, οι εργασίες δεν κάνουν αναφορά στις εκδόσεις του λογισμικού που χρησιμοποιείται, συνεπώς η αναπαραγωγή των πειραμάτων αποβαίνει σε κάποιο βαθμό δυσκολότερη. Ένα επιχείρημα που θα μπορούσε κάποιος να μας αντιπαραθέσει, είναι ότι στην συγκριτική ανάλυση θίγονται μεν κυρίως θέματα αρχιτεκτονικής του λογισμικού, η δε αρχιτεκτονική αλλάζει δύσκολα ανάμεσα σε εκδόσεις του λογισμικού [ΓΔΤ09]. Ενώ αυτή η παρατήρηση είναι σίγουρα εύστοχη, μπορούμε να υποστηρίξουμε ότι, αφ'ενός στα σημερινά πλαίσια ευέλικτων μεθοδολογιών ανάπτυξης λογισμικού (agile software development methodologies [AGM16, AGI16]) και αντικειμενοστραφούς προγραμματισμού (object oriented programming) δεν είναι απόλυτος κανόνας, αφ'ετέρου τόσο τα έργα λογισμικού ελεγκτή όσο και ολόκληρο το πεδίο της SDN βρίσκονται ακόμα σε πρώιμα στάδια εξέλιξης, στα οποία ακόμα και η αρχιτεκτονική χαρακτηρίζεται ως ρευστή.

Ούτως ή αλλιώς όμως δεν έχουμε μέχρι στιγμής λάβει γνώση κάποιας απόπειρας αναπαραγωγής των πειραμάτων των παρελθόντων εργασιών. Οι εργασίες που αναθεωρήσαμε είναι πρωτότυπες και διαφέρουν αρκετά ως προς το εύρος και τη μεθοδολογία: οι ερευνητές τείνουν να χρησιμοποιούν διαφορετικά εργαλεία για να αξιολογήσουν διαφορετικά χαρακτηριστικά. Θα μπορούσαμε όμως να πούμε ότι οι εργασίες αυτές λειτουργούν σε μεγάλο βαθμό συμπληρωματικά.

Ένα άλλο βιβλιογραφικό κενό αφορά την οπτική γωνία των ερευνητών. Με εξαίρεση την εργασία των [KHO14], η προσέγγιση που ακολουθείται είναι, όπως αναμένεται, κυρίως αυτή του ερευνητή της

δικτύωσης. Εμείς ακολουθούμε σε μεγαλύτερο, συγκριτικά, βαθμό την προσέγγιση της τεχνολογίας λογισμικού για την ανάλυση των ελεγκτών. Χωρίς φυσικά να αμφισβητούμε το γεγονός ότι ο ελεγκτής είναι πρωτίστως αντικείμενο μελέτης των περιοχών των δικτύων υπολογιστών και των τηλεπικοινωνιών, θεωρούμε ότι η τεχνολογία λογισμικού θα διαδραματίσει εφεξής σημαντικό ρόλο στην εξέλιξη των περιοχών αυτών και το αντίστροφο.

Συνεπώς, θα κινηθούμε επίσης συμπληρωματικά ως προς το ερευνητικό υπόβαθρο. Στις ενότητες που ακολουθούν περιγράφεται η ερευνητική μας δραστηριότητα, με αφετηρία τη μεθοδολογία που ακολουθήσαμε.

Μεθοδολογία-εργαλεία της έρευνας

Πρώτο μας βήμα είναι η επιλογή των λύσεων ελεγκτή που θα συγκριθούν. Παρουσιάζουμε λοιπόν τη συλλογιστική που οδήγησε στην επιλογή των συγκρινόμενων πακέτων λογισμικού, ξεκινώντας από μία ιστορική ανασκόπηση του επιπέδου ελέγχου, συνεχίζοντας με μια αναφορά στην κατάσταση της αγοράς σε σχέση με τους ελεγκτές που μας ενδιαφέρουν και καταλήγοντας στα κριτήρια επιλογής. Ελαχιστοποιούμε τα διαφοροποιητικά στοιχεία των ελεγκτών επιλέγοντας ελεγκτές γραμμένους στην ίδια γλώσσα προγραμματισμού.

Το επόμενο στάδιο είναι ο προσδιορισμός των κριτηρίων σύγκρισης των ελεγκτών, που εγείρει το ερώτημα "τι κάνει καλό έναν ελεγκτή;". Για να απαντήσουμε στο ερώτημα αυτό, εξετάζουμε το ζήτημα των απαιτήσεων ενός δικτύου επικοινωνιών γενικότερα και των ελεγκτών ειδικότερα.

Ύστερα στρέφουμε την προσοχή μας στα ποιοτικά χαρακτηριστικά που σχετίζονται με τις μη λειτουργικές απαιτήσεις των ελεγκτών. Επιλέγουμε την ποσοτική αξιολόγηση των χαρακτηριστικών με χρήση μετρικών από το πεδίο της τεχνολογίας λογισμικού. Περιγράφουμε και αξιολογούμε τις μετρικές που χρησιμοποιούμε.

Η σύγκριση γίνεται με στατική ανάλυση κώδικα των ελεγκτών POX 0.2.0 και Ryu 4.6. Το λογισμικό ελεγκτή είναι εγκατεστημένο στην εικονική μηχανή εκμάθησης του SDNHub [SDH16].

Η εικονική μηχανή εκτελείται στο λογισμικό εικονικοποίησης Virtualbox 5.0.22 r108108. (md5: 281cbbbae40c180318c59fa61abc24ac, ελεγμένο με το εργαλείο md5sum το οποίο βρίσκεται εγκατεστημένο στην ίδια την εικονική μηχανή) [VBO16, GNC16].

Δεν έχουν πραγματοποιηθεί μεταβολές στην κατάσταση (state) της εικονικής μηχανής, με εξαίρεση:

1. τη μετάβαση για το τοπικό αποθετήριο (local repository) του Git στον κυρίως κλάδο (branch) carp από τον κλάδο eel, με χρήση της εντολής git checkout carp από τον αρχικό φάκελο του POX,
2. την ανανέωση των τοπικών αποθετηρίων των δύο ελεγκτών με χρήση της εντολής git pull, ώστε να έχουμε τις τελευταίες εκδόσεις του λογισμικού,
3. την εγκατάσταση του λογισμικού ανάλυσης κώδικα SonarQube 5.6.1 (LTS) [SOQ16] καθώς και την εγκατάσταση λογισμικού που το SonarQube προαπαιτεί,
4. την εγκατάσταση του σαρωτή (scanner) κώδικα Sonar Scanner 2.7 [SSC16],
5. την εγκατάσταση του πρόσθετου (plugin) της γλώσσας προγραμματισμού Python για το SonarQube [SPP16] (εκδ. 1.6), με σκοπό τη σάρωση του κώδικα της Python,
6. την εκτέλεση του SonarQube με τα βήματα που περιγράφονται στην ενότητα Εκτέλεση – Αποτελέσματα,
7. Την εγκατάσταση και εκτέλεση του εργαλείου στατικής ανάλυσης radon 1.4.2 [RAD16].

   Ο host είναι ένας τυπικός προσωπικός υπολογιστής γραφείου με κεντρικό επεξεργαστή Athlon Phenom τριών πυρήνων και 8GB RAM, με δευτερεύοντα αποθηκευτικό χώρο μεγέθους 141GB. Εκτελεί πλατφόρμα Windows 7 Professional 64bit Service Pack 1.

Ιστορία και επισκόπηση των ελεγκτών SDN

Ο SDN είναι ένας από τους πιο πολυχρησιμοποιημένους όρους από το marketing στην εποχή μας. Παρ'όλα αυτά, καθώς δεν αποκτήσαμε πρόσβαση σε έγκυρες αναφορές ανάλυσης αγοράς, θα περιοριστούμε σε μία ιστορική επισκόπηση και απλή εμπειρική αναφορά στις κατά το παρόν διαθέσιμες λύσεις λογισμικού ελεγκτή. Για τη διευκόλυνση της διεξαγωγής της έρευνας περιορίζουμε επίσης την αναφορά μας σε ελεύθερα διαθέσιμες λύσεις ανοικτού κώδικα.

Μία από τις πρώτες πρωτοβουλίες για το διαχωρισμό του επιπέδου ελέγχου από το επίπεδο δεδομένων προέρχεται από το χώρο των τηλεπικοινωνιών, με την τεχνολογία NCP (network control point) της AT&T το 1981.

Παραδοσιακά, τα σήματα ελέγχου και δεδομένων του δικτύου διαδίδονταν στο ίδιο κανάλι, γεγονός που εισήγαγε διαχειριστικές δυσκολίες και ευπάθειες στο δίκτυο (καθώς σήματα ελέγχου του δικτύου μπορούσαν να σταλούν από οποιονδήποτε είχε πρόσβαση σε αυτό).

Με βάση το σχεδιασμό της AT&T, τα σήματα ελέγχου δρομολογούνταν στο NCP, εξαλείφοντας ουσιαστικά τις παραπάνω ευπάθειες. Το NCP επέτρεψε επίσης στον οργανισμό αφ'ενός να κατέχει πολύ καλύτερη οπτική του δικτύου, παρέχοντας υψηλότερη ποιότητα υπηρεσιών και υπηρεσίες κατ'απαίτηση (on demand), αφ'ετέρου να τμηματοποιήσει (modularize) την τεχνολογία του, δημιουργώντας στοιχειώδεις υπηρεσίες πάνω στις οποίες ήταν δυνατό να βασιστούν πιο περίπλοκες υπηρεσίες και εφαρμογές [KRE15, CSD15].

Το πρώτο έργο (project) λογισμικού ελεγκτή της SDN ήταν ο NOX [NOX08]. Αναπτύχθηκε από την Nicira Networks και έγινε έργο ανοικτού κώδικα (open source) το 2008. Ο NOX στην αρχική του μορφή (που σήμερα αναφέρεται ως NOX classic), ήταν γραμμένος στις γλώσσες προγραμματισμού C++ και Python. Σήμερα όμως αυτή η έκδοση του NOX δε χρησιμοποιείται και δεν αναπτύσσεται ενεργά. Ο κλασσικός NOX μετεξελίχθηκε στον σημερινό NOX, που είναι γραμμένος αποκλειστικά στη γλώσσα C++. Στη συνέχεια μεταφέρθηκε και εξ'ολοκλήρου στην Python με το όνομα POX. Ο σύγχρονος NOX και ο POX είναι ενεργά έργα, ο δε NOX εξασφαλίζει πολύ υψηλότερη απόδοση σε σχέση με τον κλασσικό NOX και τον POX [KRE15, CSD15, SDX16].

Σημαντικό στάδιο στην ιστορία των ελεγκτών είναι αυτό της κυκλοφορίας του Beacon. Πρόκειται για έναν ελεγκτή γραμμένο στη γλώσσα Java, με παράγωγα (forks) ιδιαίτερα δημοφιλείς ελεγκτές, όπως είναι ο Floodlight και ο Opendaylight [KRE15, CSD15]. Αυτοί οι ελεγκτές αποτελούν ουσιαστικά την κληρονομιά του Beacon, ο οποίος είναι επίσης ανενεργό project.

Τουλάχιστον στο πεδίο των λύσεων ανοικτού κώδικα, ο Opendaylight είναι ένας ιδιαίτερα διαδεδομένος ελεγκτής, απολαμβάνοντας μείζονος υποστήριξης εκ μέρους της βιομηχανίας.

Ένα κύριο διαφοροποιητικό στοιχείο του Opendaylight σε σχέση με τους ανοικτού κώδικα πρόδρομους του είναι ο περισσότερο τμηματοποιημένος σχεδιασμός του, που επιτρέπει την ανεξάρτητη ανάπτυξη καθώς και τη δυναμική εκτέλεση τμημάτων (modules) λειτουργικότητας.

Επίσης, το Opendaylight παρέχει τη δυνατότητα διαχείρισης του δικτύου σε μαζική κλίμακα, καθώς και της εφαρμογής προσωρινών ρυθμίσεων και ελέγχων στο δίκτυο μέσω της δημιουργίας μοντέλων εκφρασμένων στη γλώσσα μοντελοποίησης YANG [ODL16].

Μία αξιοσημείωτη συνεισφορά, σε ερευνητικό επίπεδο, στο πεδίο των ελεγκτών είναι αυτή των [ZHA], οι οποίοι παρουσιάζουν μία υλοποίηση ελεγκτή προορισμένη για επεξεργαστές γραφικών (Graphics Processing Unit, GPU).

Με βάση την ευρύτερη βιβλιογραφία αλλά και τις εργασίες σύγκρισης λογισμικού ελεγκτή που έχουν προηγηθεί, οι ευρύτερα χρησιμοποιούμενοι, ελεύθερα διαθέσιμοι, ενεργοί ελεγκτές ανοικτού κώδικα που λάβαμε υπ'όψη προς σύγκριση ήταν οι NOX, POX, Ryu, Trema, Floodlight και OpenDaylight. Επιλέξαμε συνειδητά τη διεξαγωγή συγκριτικής ανάλυσης με έμφαση στο βάθος και όχι στο πλάτος. Ως εκ τούτου, περιορίζουμε την ανάλυση σε δύο μόνο ελεγκτές.

Αποφασίσαμε επίσης ότι οι ελεγκτές αυτοί θα πρέπει να είναι γραμμένοι την ίδια γλώσσα προγραμματισμού. Αφ'ενός, προς ελάττωση της ανομοιογένειας των συγκρινόμενων λύσεων, ώστε οι διαφορές ανάμεσα στις γλώσσες προγραμματισμού να μην επηρεάσουν τα αποτελέσματα της σύγκρισης. Αφ'ετέρου, για να αυξήσουμε το διαθέσιμο εύρος επιλογών εργαλείων σύγκρισης, καθώς πολλά από τα εργαλεία είναι προσανατολισμένα αποκλειστικά σε μία συγκεκριμένη γλώσσα προγραμματισμού.

Από τους προαναφερθέντες ελεγκτές:

- ο NOX είναι γραμμένος σε γλώσσα C,
- ο Trema σε Ruby,
- οι POX και Ryu σε Python και
- οι Floodlight και OpenDaylight σε Java.

Παρότι κατά την άποψή μας το Opendaylight είναι πιθανόν ο ελεγκτής με τη μεγαλύτερη διείσδυση στην αγορά αυτή τη στιγμή, η πολυπλοκότητα του ελεγκτή δεν μας επιτρέπει να προχωρήσουμε σε ανάλυση με το επιθυμητό βάθος. Οι παραπάνω περιορισμοί οδήγησαν λοιπόν στην επιλογή των ελεγκτών Ryu και POX προς σύγκριση.

Προτού υπεισέλθουμε στη σύγκριση των ελεγκτών, είναι απαραίτητη μία συζήτηση περί των απαιτήσεων (requirements) που θα πρέπει να ικανοποιεί ένας ελεγκτής γενικής χρήσης, καθώς όπως θα επισημάνουμε, θεωρούμε τις απαιτήσεις παράγοντα υψηλής προτεραιότητας στην εκτίμηση της ποιότητας μίας λύσης λογισμικού ελεγκτή.

Ο ελεγκτής και οι απαιτήσεις του

Σύμφωνα με τη δημοφιλή πλέον ρήση του Marc Andrseen, “Το λογισμικό τρώει τον κόσμο” [AND11]. Τα πεδία της δικτύωσης και της τεχνολογίας λογισμικού έρχονται πλέον πιο κοντά το ένα στο άλλο οπότε είναι αναπόφευκτη η μεταξύ τους αλληλεπίδραση. Μπορούμε λοιπόν να κάνουμε λόγο για τη μηχανική λογισμικού των ελεγκτών.

Όπως ισχύει για κάθε προϊόν λογισμικού [HUL11], η μηχανική των απαιτήσεων (requirements engineering) ενός ελεγκτή είναι ένα ιδιαίτερα περίπλοκο εγχείρημα, λόγω της θέσης του ελεγκτή στο δίκτυο. Άτυπα μπορούμε να πούμε πως το λογισμικό ελεγκτή πρέπει να μπορεί να διασφαλίσει ένα κάτω όριο ποιότητας για την απαιτητικότερη επιχειρηματική εφαρμογή που σχεδιάζεται να υποστηρίξει.

Για την ανάλυση λογισμικού (ή τη σύγκριση λύσεων λογισμικού, όπως θα δούμε παρακάτω), τόσο σε δημοσιογραφικό και ερασιτεχνικό, όσο και σε επιχειρηματικό ή ακαδημαϊκό επίπεδο, είθισται να μετρώνται ποιοτικά χαρακτηριστικά (quality attributes) όπως η απόδοση, σε επίπεδο κώδικα, είτε με εργαλεία στατικής ανάλυσης, είτε κατά την εκτέλεση του κώδικα με τεχνικές όπως benchmarking ή profiling.

Ενώ λάβαμε και εμείς την ίδια προσέγγιση, και ενώ φυσικά θεωρούμε και εμείς τις τεχνικές αυτές αναγκαίες για τη διαπίστωση της ποιότητας του λογισμικού, πρέπει να επισημάνουμε τη θέση μας ότι, το λογισμικό, όχι μόνο σε επίπεδο κώδικα αλλά και σε όλες τις φάσεις του κύκλου ζωής (life cycle), θα πρέπει να ελέγχεται σε σχέση με τις απαιτήσεις του λογισμικού.

Για να γίνει κατανοητή η πλήρης διατύπωση της θέσης μας πάνω στο ζήτημα αυτό, θα αναφερθούμε πολύ συνοπτικά σε αυτές τις πρώτες φάσεις της διαδικασίας ανάπτυξης λογισμικού.

Τυπικά, η ανάπτυξη ξεκινά με τη μηχανική απαιτήσεων, η οποία στο σύνολο της είναι η διαδικασία με την οποία οι εμπλεκόμενοι (stakeholders) στο έργο του λογισμικού (στους οποίους περιλαμβάνονται και μέλη της ομάδας ανάπτυξης) έρχονται σε συμφωνία ως προς το τι θα πρέπει να επιτυγχάνει το σύστημα λογισμικού [ΓΔΤ09].

Είναι μία δύσκολη (κατά τη γνώμη μας, ίσως η δυσκολότερη) φάση στη διαδικασία ανάπτυξης, καθώς:

- Οι εμπλεκόμενοι συχνά δε γνωρίζουν τι χρειάζονται από το σύστημα
- Είναι μία ανθρωποκεντρική δραστηριότητα η οποία επηρεάζεται από διαφορές απόψεων, επιπλοκές στις σχέσεις των μελών ανάπτυξης, γνωστικά σφάλματα ή μεροληψία [RAL11] και γενικότερα από ψυχολογικούς, αντιληπτικούς, πολιτικούς, οικονομικούς, οικολογικούς, εθνογραφικούς, κοινωνιολογικούς παράγοντες, παράγοντες συμμόρφωσης με το νόμο κοκ.
- Οι απαιτήσεις πρέπει να διατυπωθούν με τέτοιο τρόπο ώστε να είναι σαφείς στους υπεύθυνους ανάπτυξης.

  Για να καταστεί διαχειρίσιμη η διαδικασία μηχανικής των απαιτήσεων, το πρόβλημα χωρίζεται σε υποπροβλήματα, το καθένα εκ των οποίων αντιμετωπίζεται αυτόνομα [HULL11]. Οι απαιτήσεις χωρίζονται σε απαιτήσεις χρήστη (user requirements) και απαιτήσεις συστήματος (system requirements) [ΓΔΤ09].

Αρχικά εξετάζουμε μόνο το πρόβλημα που θέλουμε να λύσουμε με το σύστημα λογισμικού, χωρίς καμία αναφορά στο ίδιο το σύστημα. Μελετούμε δηλαδή το πεδίο (domain) [BJO98] του προβλήματος καθ'εαυτό, δημιουργώντας ένα μοντέλο του πεδίου προβλήματος (domain model). Από τη διαδικασία αυτή προκύπτουν οι απαιτήσεις χρήστη, που είναι ουσιαστικά οι απαιτήσεις των εμπλεκομένων χωρίς καμία αναφορά σε λεπτομέρειες του συστήματος.

Στις απαιτήσεις χρήστη βασίζεται η ανάλυση που γίνεται σε επόμενο επίπεδο και αφορά το τι θα πρέπει να κάνει το σύστημα λογισμικού, ώστε να ικανοποιήσει τις απαιτήσεις χρήστη. Από αυτό το στάδιο προκύπτουν οι απαιτήσεις συστήματος.

Σημειώνεται ότι και αυτό ακόμα το στάδιο ασχολείται αποκλειστικά με το ερώτημα του τι θα πρέπει να κάνει το σύστημα, όχι με το ερώτημα του πώς θα πρέπει να το κάνει. Το ερώτημα του πώς θα πρέπει να λειτουργεί το σύστημα απαντάται στο στάδιο που ακολουθεί τη μηχανική των απαιτήσεων και είναι ο σχεδιασμός του λογισμικού. Μέρος του σχεδιασμού είναι και η αρχιτεκτονική του λογισμικού.

Αναγνωρίζουμε, επίσης, ότι αναφερόμαστε σε αυτές τις δραστηριότητες σαν να επρόκειτο για μία απλή ακολουθία διακριτών σταδίων, στην οποία κάθε στάδιο ξεκινά μόνο αφού ολοκληρωθεί το προηγούμενο. Πρέπει να διευκρινίσουμε ότι στη σημερινή πρακτική της ανάπτυξης λογισμικού δεν ισχύει κάτι τέτοιο, αλλά για τους σκοπούς της επιχειρηματολογίας μας και για απλότητα θεωρούμε ότι ισχύει.

Οι δραστηριότητες που προαναφέρθηκαν, λαμβάνουν παραδοσιακά χώρα στα πλαίσια ενός συγκεκριμένου έργου (project) και επαναλαμβάνονται καθ'ολοκληρία σε επόμενα έργα του ίδιου οργανισμού. Έχει διατυπωθεί όμως και η ιδέα της επαναχρησιμοποίησης του μοντέλου πεδίου [FKR05] και των απαιτήσεων (για ένα παράδειγμα βλ.[BES13]), καθώς έχει παρατηρηθεί ότι αρκετές εφαρμογές σχεδιάζονται ώστε να λύνουν τα ίδια ή παρόμοια προβλήματα.

Με βάση αυτή την ιδέα, ένα μοντέλο πεδίου μπορεί να σχεδιαστεί για ένα έργο και να επαναχρησιμοποιηθεί σε άλλο, από τον ίδιο ή διαφορετικούς οργανισμούς, ή θα μπορούσε να σχεδιαστεί εξ'αρχής ώστε να χρησιμοποιηθεί σε μία πληθώρα έργων με κοινούς σκοπούς.

Μπορούμε να γενικεύσουμε την παραπάνω θεώρηση, προσθέτοντας ότι όχι μόνο τα συστήματα λογισμικού, αλλά κάθε τεχνητό σύστημα (για την έννοια του συστήματος βλ. παρακάτω), σχεδιάζεται για την επίλυση ενός συγκεκριμένου προβλήματος και πως πολλά συστήματα επιλύουν παρεμφερή προβλήματα, συνεπώς ορίζουμε πως ανήκουν στην ίδια οικογένεια.

Έτσι, όταν κάνουμε λόγο για το Πληροφοριακό Σύστημα μίας συγκεκριμένης τράπεζας ή ενός συγκεκριμένου νοσοκομείου, μπορούμε να μιλήσουμε και για τα πεδία της ευρύτερης τραπεζικής ή νοσοκομειακής πρακτικής που αυτά θεραπεύουν, τα οποία είναι κοινά, ή τουλάχιστον έχουν σημεία τομής, με Πληροφοριακά Συστήματα άλλων τραπεζών και νοσοκομείων. Ακόμα γενικότερα, μπορούμε να διακρίνουμε πεδία για την Οικονομική ή την Ιατρική πρακτική και επιστήμη, καθώς και για σχεδόν κάθε άλλο θέμα του επιστητού.

Ως εκ τούτου, θεωρούμε ότι είναι δυνατό να οριστούν μοντέλα πεδίου για κάθε τομέα των ΤΠΕ. Το να καταστήσουμε τα αντικείμενα των ΤΠΕ first class citizens στη διαδικασία ανάπτυξης ενός συστήματος θα έχει άμεσο αντίκτυπο στην επαναχρησιμοποιησιμότητα (reusability), με επακόλουθες επιδράσεις σε πολλά άλλα χαρακτηριστικά όπως είναι η συντηρησιμότητα (maintainability) και οι συνιστώσες της (αν)ασφάλειας του συστήματος.

Επιστρέφοντας στο χώρο της δικτύωσης, πιστεύουμε ότι ισχύουν κατ'αναλογία τα παραπάνω. Ένα σύστημα δικτύωσης σχεδιάζεται για τη μετάδοση πληροφοριών (με την έννοια της πληροφορίας του Shannon, [SHA48]) ανάμεσα σε μία ή περισσότερες οντότητες.

Φυσικά, κάθε σύστημα διαφέρει ανάλογα με τις περιπτώσεις χρήσης που καλείται να εξυπηρετήσει, αλλά οι ομοιότητες είναι αρκετές ώστε να μπορούμε να μιλήσουμε για το μοντέλο πεδίου των τηλεπικοινωνιών.

Μία επισκόπηση της βιβλιογραφίας φανερώνει πως όλες αυτές οι ιδέες δεν είναι κάτι νέο. Τα μοντέλα πεδίου μπορούν να αναπαρασταθούν με τη μορφή οντολογίας, ήτοι μίας τυπικής αναπαράστασης της γνώσης (του πεδίου-για έναν ορισμό βλ. το [GRU07]). Επιδιώξαμε την εκμετάλλευση αυτής της ιδέας, ώστε να διεκπεραιώσουμε την αξιολόγηση των ελεγκτών με ακόμα μεγαλύτερη πιστότητα από αυτή που παρέχουν οι συνήθως χρησιμοποιούμενες τεχνικές σε απομόνωση.

Αρχική μας πρόθεση ήταν να χρησιμοποιήσουμε ένα μοντέλο του πεδίου της δικτύωσης, ώστε να το χρησιμοποιήσουμε ως βάση σύγκρισης για τους ελεγκτές της επιλογής μας και συγκεκριμένα για τις απαιτήσεις των ελεγκτών ή τα παράγωγα (products) αυτών. Προς αυτή την κατεύθυνση μας κινητοποίησε μία εργασία που αφορούσε το πεδίο της επικοινωνίας ανθρώπου-υπολογιστή (Human-Computer Interaction-HCI) και πιο συγκεκριμένα των γραφικών διεπαφών χρήστη (GUIs) [SAM94].

Σε αυτή την εργασία επιδιώκεται σύγκριση λύσεων λογισμικού με βάση αρχιτεκτονικά μοντέλα και όχι με απευθείας μετρήσεις πάνω στον κώδικα.

Όμως συναντήσαμε αρκετά εμπόδια κατά την εφαρμογή αυτής της προσέγγισης. Κατ'αρχάς, όπως αναμέναμε με βάση την εμπειρία μας με τα έργα λογισμικού ανοικτού κώδικα και ελεύθερου λογισμικού, δε βρήκαμε δημόσια διαθέσιμα έγγραφα προσδιορισμού απαιτήσεων (requirements specification documents) ή άλλη τεκμηρίωση (documentation) των απαιτήσεων για οποιονδήποτε από τους δημοφιλείς ελεγκτές. Το σημαντικότερο είναι όμως ότι δεν ήμαστε σε θέση να ανακτήσουμε επαναχρησιμοποιήσιμες απαιτήσεις ή κάποια οντολογία για το πεδίο της δικτύωσης.

Αυτή η αποτυχία επαναχρησιμοποίησης της γνώσης πεδίου της δικτύωσης μας οδηγεί σε σκέψεις γύρω από το ποιες είναι τελικά οι απαιτήσεις ενός ελεγκτή, ανεξαρτήτως συγκεκριμένης εφαρμογής. Μία πρώτη σκέψη είναι ότι μία γενική λύση λογισμικού ελεγκτή, όπως αυτές που χρησιμοποιούνται ευρύτερα, αυτή τη στιγμή, σε κέντρα δεδομένων (data centers) και άλλα δίκτυα παραγωγής, θα πρέπει να μπορεί να υποστηρίξει ένα πλήθος εφαρμογών, από εφαρμογές Ιστού που εκτελούνται με υπολογιστική νέφους (cloud computing) όπως το Facebook ή η μηχανή αναζήτησης της Google μέχρι εφαρμογές διαμοιρασμού αρχείων (file sharing), εφαρμογές ζωντανής μετάδοσης βίντεο (live video streaming) και εφαρμογές τηλεδιάσκεψης [KRN01].

Αν όμως γυρίσουμε πίσω στις απαρχές της δικτύωσης, θα διαπιστώσουμε ότι η εξέλιξη της παρουσιάζει το ίδιο μοτίβο με αυτή των υπολογιστών καθ'εαυτών [GIH15].

Στη δεκαετία του 1940-1950 τα προγράμματα υπολογιστή εκτελούνταν χειροκίνητα από ανθρώπους, με απευθείας χειρισμό του υλικού, καθώς δεν υφίστατο ακόμα η έννοια του λειτουργικού συστήματος (operating system-OS).

Όπως η προσθήκη ενός επιπέδου λογισμικού στη δεκαετία του 1940-1950 ανάμεσα στα mainframes και στους χειριστές τους διευκόλυνε τη διαχείριση της μηχανής, έτσι και η προσθήκη επιπέδων λογισμικού πάνω από το υλικό του δικτύου έχει τη δυνατότητα να διευκολύνει τη διαχείριση του δικτύου.

Δεν πρέπει λοιπόν να είναι τυχαίο το γεγονός ότι, στη βιβλιογραφία της SDN, οι ελεγκτές συχνά αντιπαραβάλλονται με παραδοσιακά λειτουργικά συστήματα (operating systems). Το κίνημα SDN θεωρείται ως κίνημα προς ένα λειτουργικό σύστημα δικτύου (network operating system ή NOS) [NOX08, ERI13].

Παρόλα αυτά, ακόμα και με τη θεώρηση του ελεγκτή ως λειτουργικού συστήματος, τα προβλήματα παρέμειναν, καθώς δεν μπορέσαμε να βρούμε μοντέλο πεδίου για τα λειτουργικά συστήματα.

Ως εκ τούτου, βασιστήκαμε σε βιβλιογραφικές πηγές για τον προσδιορισμό των απαιτήσεων ενός ελεγκτή και καταφύγαμε σε κλασσικές τεχνικές μετρήσεων, έχοντας επίγνωση των δυσκολιών που αυτή η προσέγγιση ενέχει, στις οποίες θα αναφερθούμε προσεχώς.

Χαρακτηριστικά λογισμικού

Προϋποθέσεις για την πραγματοποίηση μετρήσεων είναι φυσικά η επιλογή των ποιοτικών χαρακτηριστικών (εφεξής χαρακτηριστικών) του λογισμικού τα οποία πρόκειται να μετρηθούν.

Στη βιβλιογραφία δεν συναντούμε ούτε κοινά αποδεκτό ορισμό του όρου χαρακτηριστικό, ούτε μία πρωτότυπη η ξεκάθαρη διάκριση μεταξύ των χαρακτηριστικών του λογισμικού. Κάθε οργανισμός έχει τη δική του ορολογία για τα χαρακτηριστικά ενός συστήματος, με αποτέλεσμα στην πρακτική της τεχνολογίας λογισμικού να χρησιμοποιείται ο ίδιος όρος με πολλές έννοιες ή να χρησιμοποιούνται πολλοί όροι για να δηλώσουν το ίδιο χαρακτηριστικό.

Κατά κοινή ομολογία όμως τα χαρακτηριστικά σχετίζονται με τις μη λειτουργικές απαιτήσεις του συστήματος. Πιο συγκεκριμένα, κάθε μη λειτουργική απαίτηση είναι εξ ορισμού απαίτηση για τη διατήρηση ενός χαρακτηριστικού κατά την ανάπτυξη, τον έλεγχο, τη συντήρηση ή/και την εκτέλεση του λογισμικού, μέσα σε κάποια όρια. Εν γένει, στη βιβλιογραφία των ΤΠΕ αναγνωρίζονται τα χαρακτηριστικά της απόδοσης, της συντηρησιμότητας, της ασφάλειας, της μεταφερσιμότητας, της διαλειτουργικότητας, της ευχρηστίας, τις ελεγξιμότητας και ούτω καθεξής.

Παρ'όλα αυτά, λόγω της αδυναμίας ορισμού τόσο της έννοιας του χαρακτηριστικού όσο και κάποιων από τα χαρακτηριστικά, και δεδομένου του πλήθους των διαφορετικών χαρακτηριστικών, δεν είναι δυνατή ούτε η εξαντλητική ανάλυση ενός πακέτου λογισμικού, ούτε η πλήρης ποσοτικοποίηση αυτής.

Γι'αυτό το λόγο, τα προς μέτρηση χαρακτηριστικά επιλέγονται με γνώμονα τους σκοπούς της ανάλυσης. Στην περίπτωσή μας θα επικεντρωθούμε σε χαρακτηριστικά τα οποία έχουν μεγαλύτερο αντίκτυπο στο πεδίο της δικτύωσης, με βάση την προβληματική του χώρου, την οποία περιγράψουμε συνοπτικά σε προηγούμενα εδάφια.

Όπως προαναφέρθηκε, μέχρι αυτή τη στιγμή-και με βάση όσα γνωρίζουμε-δεν υφίσταται τυπική διατύπωση των πεδίων της δικτύωσης, των τηλεπικοινωνιών ή των λειτουργικών συστημάτων, από την οποία θα μπορούσαμε να εξαγάγουμε τις απαιτήσεις των σχετικών προβλημάτων και των χαρακτηριστικών των αντίστοιχων λύσεων. Κατά συνέπεια προσπαθήσαμε να αντλήσουμε πληροφορίες από τη βιβλιογραφία και τη δική μας εμπειρία σχετικά με τις απαιτήσεις και τα ποιοτικά χαρακτηριστικά τα οποία σχετίζονται με τη δικτύωση και πιο συγκεκριμένα με τις αρχιτεκτονικές τύπου SDN.

Σημειώνεται ότι η συγκριτική ανάλυση λογισμικού με βάση τις απαιτήσεις μπορεί να λάβει χώρα στο καθεστώς ενός οργανισμού για τη σύγκριση έργων του ίδιου οργανισμού, με λιγότερους περιορισμούς. Με εξαίρεση τις απαιτήσεις που σχετίζονται με το πεδίο του προβλήματος, οι απαιτήσεις του λογισμικού καθορίζονται από τον οργανισμό, ενώ τα προβλήματα της ασάφειας και αμφισημίας των χαρακτηριστικών μπορούν να αντιμετωπιστούν με συναντήσεις των εμπλεκομένων

[ΓΔΤ09] και με τη δημιουργία λεξικών ή οντολογιών [PWC09] στα πλαίσια του οργανισμού. Τα προβλήματα τα οποία αντιμετωπίζουμε προκύπτουν στα πλαίσια της σύγκρισης λογισμικού προερχόμενου από τρίτο ή τρίτους οργανισμούς.

Για την ανάλυση μας, μία επίσημη πηγή πληροφόρησης σχετικά με τα χαρακτηριστικά σε αρχιτεκτονικές SDN είναι η [ONF15]. Ενώ η δημοσίευση αυτή αναφέρεται γενικά στην τεχνολογία SDN και όχι αποκλειστικά στον ελεγκτή, μας επιτρέπει να συμπεράνουμε τα χαρακτηριστικά του ελεγκτή τα οποία θα πρέπει να μετρήσουμε. Για παράδειγμα, προδιαγράφει ότι μία αρχιτεκτονική SDN θα πρέπει να υποστηρίζει όλους τους τύπους δικτύων, να υποστηρίζει τόσο τεχνολογίες προώθησης πακέτων όσο και μεταγωγής κυκλώματος και επιπρόσθετα να παραμένει συμβατή με παλαιές τεχνολογίες. Αυτό σημαίνει ότι ο ελεγκτής θα πρέπει να μπορεί να διαχειρίζεται ένα μεγάλο εύρος μηνυμάτων, πρωτοκόλλων και περιπτώσεων χρήσης, κάτι που επιβάλλει ένα άνω όριο στο χαρακτηριστικό της πολυπλοκότητας. Παράλληλα, αυτές οι απαιτήσεις αυξάνουν κατά πολύ την επισφάλεια (risk) που σχετίζεται με το σύστημα, καθώς αυξάνεται τόσο η συνολική επιφάνεια επίθεσης (attack surface) όσο και το πλήθος των απειλών (με τη μεγιστοποίηση του πλήθους των πιθανών χρηστών), επιβάλλοντας απαιτήσεις ασφάλειας, οι οποίες προλαμβάνονται ούτως ή άλλως ρητά από το ONF. Επιπλέον, θα πρέπει να υπάρχει πρόνοια για περισσότερα επίπεδα αφαίρεσης και γενικότερα για μελλοντικές αλλαγές, κάτι που κάνει απαραίτητη την επίτευξη ενός χαμηλού κάτω ορίου συντηρησιμότητας (χαρακτηριστικό απαραίτητο για κάθε προϊόν λογισμικού). Και φυσικά, ένας ελεγκτής γενικής χρήσης θα πρέπει να μπορεί να διαχειριστεί τόσο αποδοτικά τους διαθέσιμους πόρους [ΚΜΕ, αποθηκευτικός χώρος και ενέργεια, για τα κινητά και ενσωματωμένα (embedded) συστήματα], όσο χρειάζεται η απαιτητικότερη εφαρμογή που καλείται να υποστηρίξει.

Σχετικά με τη μέτρηση λογισμικού

Σημασία, επιπλοκές και ηθικές προεκτάσεις της μέτρησης λογισμικού

Στις προηγούμενες ενότητες κάναμε αναφορά στην έννοια της ποσοτικοποίησής χαρακτηριστικών του λογισμικού που είναι απαραίτητη για τις ανάγκες της σύγκρισης. Γι'αυτό τον σκοπό, χρειαζόμαστε θεωρία, τεχνικές και εργαλεία από το πεδίο της μέτρησης λογισμικού.

Το πεδίο της μέτρησης λογισμικού προκύπτει από την ανάγκη για τεχνολογία λογισμικού βασισμένη σε αποδείξεις. Αυτή είναι μάλιστα και μία από τις μεγάλες προκλήσεις στο πεδίο της τεχνολογίας λογισμικού [FIN11]. Συντασσόμαστε με τον Finkelstein, αναγνωρίζοντας αυτή την ανάγκη, καθώς η ποσοτικοποίηση ιδιοτήτων του λογισμικού μπορεί να οδηγήσει σε πιο πληροφορημένη λήψη αποφάσεων σχετικά με έργα και προϊόντα Πληροφορικής.

Αναγνωρίζουμε, επίσης, ότι οι τεχνικές μέτρησης λογισμικού, όπως οι περισσότερες τεχνικές της τεχνολογίας λογισμικού, έρχονται με κάποιο κόστος. Έτσι, ενώ η δυνατότητα για την πραγματοποίηση μετρήσεων είναι ευκταία για κάθε έργο Πληροφορικής, ίσως να μην αρμόζει σε μικρά έργα με στενά περιθώρια χρόνου και προϋπολογισμού, όταν το πεδίο του προβλήματος είναι γνωστό. Σε κάποιες δε κατηγορίες έργων Πληροφορικής, όπως για παράδειγμα αυτές των βιντεοπαιχνιδιών διασκέδασης (entertainment video games) ή ψυχαγωγίας, η τυχόν εξάρτηση από δεδομένα μέτρησης θα μπορούσε να αποβεί μέχρι και καταστροφική, εφ'όσον αποθαρρύνει την καλλιέργεια κλίματος αυθορμητισμού, έκφρασης της φαντασίας και παραγωγής ή εφαρμογής καινοτόμων ιδεών.

Σε κάθε περίπτωση όμως, πάντα είναι και θα πρέπει να είναι, στην ευχέρεια και την κρίση του διαχειριστή του έργου να επιλέξει μεν ποιες τεχνικές θα χρησιμοποιήσει σε διάφορες φάσεις του κύκλου ζωής του λογισμικού, η δε ερμηνεία των δεδομένων και τελική λήψη αποφάσεων απαιτεί την ανθρώπινη κρίση.

Παρ'όλα αυτά, θεωρούμε ότι σε άλλα, κρίσιμα έργα Πληροφορικής, οπού διακυβεύονται μεγάλα οικονομικά μεγέθη ή ακόμα και ανθρώπινες ζωές (βλ. πληροφοριακά συστήματα τραπεζών, νοσοκομείων, συστήματα πλοήγησης, συστήματα υποστήριξης ζωής, συστήματα τηλεϊατρικής, λοιπά συστήματα κρίσιμων υποδομών κ.α), η ανάγκη της ποσοτικοποίησης και στοιχειοθέτησης της ποιότητας του λογισμικού γίνεται επιτακτική.

Στο πεδίο της μέτρησης λογισμικού, κεντρικό ρόλο παίζουν, φυσικά, οι μετρικές λογισμικού. Μετρικές λογισμικού χρησιμοποιούνται εδώ και χρόνια για την αξιολόγηση της ποιότητας του λογισμικού. Αποτελούν εξίσου και μείζον ερευνητικό θέμα στο πεδίο της τεχνολογίας λογισμικού [SDS14]. Οι μετρικές ποσοτικοποιούν τα χαρακτηριστικά του λογισμικού, για παράδειγμα, όπως θα δούμε, το πλήθος των γραμμών κώδικα είναι μία μετρική με βάση την οποία μπορούμε να προσδιορίσουμε το χαρακτηριστικό του μεγέθους του λογισμικού.

Μειονεκτήματα της σημερινής πρακτικής της μέτρησης λογισμικού

Γνώμη μας είναι ότι, δυστυχώς, η σημερινή πρακτική της μέτρησης πάσχει σε αρκετά σημεία [FEN14], αλλά και ότι χρειαζόμαστε επίσης καλύτερα στοχευμένη έρευνα στον τομέα αυτό [CJH95].

Ένα πρόβλημα με τις επικρατούσες μετρικές που δεν αφορά άμεσα τη συγκριτική ανάλυση αλλά την αξιολόγηση των επιδόσεων των εργαζομένων είναι η ευαλωτότητα τους σε παραποίηση. Για παράδειγμα, εάν υπολογίζουμε την παραγωγικότητα ενός προγραμματιστή με βάση τις γραμμές κώδικα, ο προγραμματιστής θα μπορούσε να προσθέτει γραμμές που δεν έχουν αντίκτυπο στη λειτουργικότητα του συστήματος (τις λεγόμενες no-ops), πλην όμως μπορεί να επηρεάσουν αρνητικά την απόδοση ή τη συντηρησιμότητα του συστήματος.

Βεβαίως, οι μετρικές χαρακτηρίζονται και από προβλήματα τα οποία επηρεάζουν άμεσα και δυσχεραίνουν τη συγκριτική ανάλυση, όπως τα παρακάτω.

- *Προβλήματα σαφήνειας των μετρούμενων χαρακτηριστικών.* Για να είναι ελέγξιμη η ορθότητα μίας μετρικής, δεν πρέπει να υπάρχει ασάφεια ή αμφισημία σχετικά με τα χαρακτηριστικά του λογισμικού που επιδιώκουμε να μετρήσουμε, με άλλα λόγια πρέπει να γνωρίζουμε τι είναι αυτό που μετράμε. Για παράδειγμα, το μέτρο κυκλωματικής πολυπλοκότητας του McCabe [ΓΔΚ07, FEN97] αποτελεί μέτρο πολυπλοκότητας του λογισμικού-τί σημαίνει, όμως, πολυπλοκότητα; [CJH95, JOH09]. Με το ερώτημα αυτό θα καταπιαστούμε συνοπτικά στην ενότητα περί πολυπλοκότητας, όπου θα γίνει αντιληπτό το εν λόγω πρόβλημα.

- *Προβλήματα φάσης κύκλου ζωής λογισμικού.* Οι περισσότερες μετρικές ορίζονται σε επίπεδο σχεδιασμού και κώδικα. Ενώ είναι αδιαμφισβήτητη η αξία της μέτρησης και σε αυτά το επίπεδα, όπως συζητήθηκε παραπάνω θεωρούμε ότι βάση κάθε αξιολόγησης θα πρέπει να είναι οι απαιτήσεις λογισμικού. Ως εκ τούτου, αφ'ενός χρειαζόμαστε εντατικότερη έρευνα στο ζήτημα της αξιολόγησης των απαιτήσεων αυτών καθ'εαυτών, αφ'ετέρου θα πρέπει να οριστούν περισσότερες μετρικές ικανοποίησης των απαιτήσεων χρήστη και συστήματος.

- *Προβλήματα αξιολόγησης μετρικών.* Θα πρέπει για κάθε μετρική να υπάρχει ένας τρόπος εκτίμησης του κατά πόσο είναι κατάλληλη για ένα συγκεκριμένο χαρακτηριστικό. Για παράδειγμα, είναι το πλήθος των γραμμών κώδικα δηλωτικό του μεγέθους του λογισμικού;

  Οι [SDS14] διακρίνουν δύο μεθοδολογίες αξιολόγησης μετρικών, υποστηρίζοντας ότι πρέπει να χρησιμοποιούνται και οι δύο για την απόδειξη της ορθότητας μίας μετρικής: η εσωτερική αξιολόγηση (internal validation) είναι θεωρητική, ενώ η εξωτερική (external validation) στηρίζεται σε εμπειρικά δεδομένα. Η εσωτερική αξιολόγηση μπορεί περαιτέρω να είναι βασισμένη στη θεωρία αντιπροσωπευτικότητας της μέτρησης (representational theory of measurement) ή σε ιδιότητες που πρέπει να κατέχουν οι μετρικές ώστε να αποδεικνύεται η ορθότητα τους. Οι [FEN97] υποστηρίζουν τη χρήση του κριτηρίου της

αντιπροσωπευτικότητας, ενώ ερευνητές όπως η Weyuker [WEY88] ακολουθούν αξιωματικές προσεγγίσεις.

Δεν έχει επικρατήσει λοιπόν κάποια μεθοδολογία αξιολόγησης. Στα πλαίσια των στατιστικών μελετών, κάποιοι ερευνητές αξιολογούν μία νέα μετρική υπολογίζοντας τη συνάφεια (correlation) της με μετρικές που είναι γνωστό ότι είναι αντιπροσωπευτικές του ίδιου χαρακτηριστικού [SDS14]. Οι [FEN97] ενίστανται σε αυτή την πρακτική, καθώς μία σχέση συνάφειας μεταξύ των μετρικών δε δηλώνει κάτι για τη σχέση μετρικής-χαρακτηριστικού. Σε άλλες περιπτώσεις οι μετρικές αξιολογούνται με βάση τη διαίσθηση [SDS14].

- *Προβλήματα ολιστικής προσέγγισης*. Οι περισσότερες μετρικές αντιμετωπίζουν το λογισμικό ως αυτόνομη οντότητα, αποκομμένη από το περιβάλλον. Στην πράξη όμως, τα χαρακτηριστικά του συστήματος εξαρτώνται και από τον παρατηρητή του συστήματος, δηλ. από τον προγραμματιστή, τον σχεδιαστή, τον πελάτη, τον χρήστη κοκ. Ο [ASH56] υποστηρίζει ότι, χαρακτηριστικά ενός συστήματος όπως είναι η πολυπλοκότητα, ή η παρατήρηση του (γνωστού στους κύκλους των συστημικών επιστημών), φαινομένου της ανάδυσης (emergence) εξαρτώνται από την ικανότητα και τις γνώσεις του παρατηρητή. Πράγματι, με βάση την-κυρίως διαισθητική προς το παρόν-αντίληψη που έχουμε για την έννοια της πολυπλοκότητας του λογισμικού, μπορούμε εύλογα να συλλογιστούμε ότι το ίδιο τμήμα ενός συστήματος μπορεί να είναι πιο πολύπλοκο για έναν αρχάριο προγραμματιστή ή σχεδιαστή, σε σχέση με έναν βετεράνο του χώρου. Ο Norman [NOR11] διακρίνει ανάμεσα στους όρους complexity και complicated, χρησιμοποιώντας τον όρο complicated για να αναφερθεί στη νοητική κατάσταση του παρατηρητή. Οι [FEN97, FEN14] κάνουν τη διάκριση ανάμεσα στα εσωτερικά χαρακτηριστικά (internal attributes), τα οποία θεωρούνται εγγενή του συστήματος, όπως το μέγεθος και εξωτερικά χαρακτηριστικά (external attributes) που εξαρτώνται από το περιβάλλον, όπως η συντηρησιμότητα.

  Γνωρίζουμε όμως μέχρι στιγμής μία μόνο μετρική πολυπλοκότητας η οποία λαμβάνει υπ'όψην της και τον παρατηρητή του συστήματος, τη γνωστική πολυπλοκότητα (cognitive complexity) [BIE66, FEN97], η οποία επιδιώκει να αποτυπώσει την προσπάθεια που χρειάζεται ώστε να γίνει κατανοητό ένα σύστημα λογισμικού. Η γνωστική πολυπλοκότητα χρησιμοποιείται, μεταξύ άλλων, στον υπολογισμό της πολυπλοκότητας μίας διεπαφής χρήστη [TRA96].

  Μάλιστα, θα μπορούσαμε να προχωρήσουμε αυτή την ιδέα ένα βήμα παρακάτω, καθώς θεωρούμε ότι τα χαρακτηριστικά του λογισμικού δεν εξαρτώνται μόνο από τις ικανότητες και τις γνωστικές δεξιότητες του παρατηρητή. Εξαρτώνται, τουλάχιστον έμμεσα, και από τη συναισθηματική κατάσταση του παρατηρητή κατά τη διάρκεια της παρατήρησης, καθώς μπορεί να επηρεάσει τον τρόπο που ο παρατηρητής επεξεργάζεται τις πληροφορίες. Στις περιοχές των γνωστικών και ψυχολογικών επιστημών έχει προταθεί (τουλάχιστον) ένα

μοντέλο [FOR95] του τρόπου με τον οποίο η συναισθηματική κατάσταση ενός ατόμου επηρεάζει την επεξεργασία πληροφοριών και την λήψη αποφάσεων.

Αναγνωρίζουμε βεβαίως, ότι η παραπάνω περιγραφή αφ'εαυτή δε μπορεί να μας παράσχει μία μετρική κατάλληλη για την ανάλυση του λογισμικού. Ο επιτυχής ορισμός μίας τέτοιας μετρικής εξαρτάται από την πρόοδο στα πεδία της ψυχολογίας και της γνωστικής επιστήμης, τα οποία απέχουν ακόμα παρασάγγας από τον τελικό τους στόχο, παρ'ότι η πρόοδος τους αυτή είναι αξιοσημείωτη. Μέχρι τότε, ο ορισμός μετρικών που λαμβάνουν υπ'όψη και τον παρατηρητή θα παρουσιάζει σοβαρές δυσκολίες ακρίβειας και αντικειμενικότητας. Και ενώ το θέμα της ψυχολογίας του προγραμματισμού έχει διερευνηθεί και στο παρελθόν [HGS90], δε γνωρίζουμε κάποια προσπάθεια μοντελοποίησης στο πεδίο της τεχνολογίας λογισμικού η οποία να περιλαμβάνει τις ψυχολογικές πτυχές της μηχανικής λογισμικού.

Παρ'όλα αυτά επιχειρηματολογούμε υπέρ μίας ολιστικής προσέγγισης σε σχέση με τη μέτρηση του λογισμικού. Θεωρούμε ότι η διανοητική και ψυχολογική διάσταση της ομάδας ανάπτυξης του λογισμικού δε μπορούν να θεωρηθούν αμελητέες δυνάμεις στον κύκλο ζωής του συστήματος. Συνεπώς θα πρέπει να συμπεριληφθούν σε μελλοντικές μετρικές και μοντέλα τεχνολογίας λογισμικού.

- *Προβλήματα απουσίας θεωρίας μέτρησης λογισμικού.* Όπως προαναφέρθηκε, μία συστηματική μελέτη του πεδίου της μέτρησης λογισμικού μπορούμε να βρούμε στα [FEN97, FEN14]. Οι συγγραφείς χρησιμοποιούν τη θεωρία αντιπροσωπευτικότητας της μέτρησης για την αξιολόγηση των μετρικών. Παρ'όλα αυτά, το πεδίο έχει ακόμα ανάγκη από νόμους των χαρακτηριστικών και των μετρικών λογισμικού, στο πνεύμα της φυσικής λογισμικού του Halstead (Halstead's software physics) [HAL77]. Για παράδειγμα, πρέπει να υπάρχει κάποια σχέση ανάμεσα στα χαρακτηριστικά του μεγέθους, της πολυπλοκότητας του λογισμικού και στις αντίστοιχες μετρικές η ανακάλυψη της οποίας εκκρεμεί. Φυσικά, η διατύπωση σχέσεων τέτοιου είδους προϋποθέτει τον προσδιορισμό και εξακρίβωση των σχετικών εννοιών, όπως προαναφέραμε.

Μετρικές που χρησιμοποιήσαμε

Λαμβάνοντας γνώση των ανωτέρω προβλημάτων, σε αυτή την ενότητα παρουσιάζουμε συνήθεις μετρικές που χρησιμοποιήσαμε για την αξιολόγηση των ελεγκτών, ανά μετρήσιμο χαρακτηριστικό του λογισμικού και επιδιώκουμε την αξιολόγησή τους.

Σημειώνεται ότι οι μετρικές διακρίνονται σε μετρικές προϊόντος (product), διαδικασίας (process) και πόρων (resources) [KAN02]. Εμείς ασχολούμαστε αποκλειστικά με μετρικές προϊόντος, οι οποίες ποσοτικοποιούν το λογισμικό αυτό καθ'εαυτό. Επίσης, κάποιες από τις μετρικές που μας ενδιαφέρουν έχουν νόημα μόνο σε γλώσσες προγραμματισμού που ακολουθούν το παράδειγμα της αντικειμενοστρέφειας (object-orientation).

Μέγεθος

Ένα από τα σημαντικότερα προβλήματα στα πεδίο της μέτρησης λογισμικού και της διαχείρισης έργων λογισμικού είναι αυτό του προσδιορισμού του μεγέθους του λογισμικού. Το μέγεθος του λογισμικού θεωρείται συχνά ότι συσχετίζεται με την προσπάθεια ανάπτυξης και συντήρησης του καθώς και με την πολυπλοκότητα (complexity) του, γι'αυτό και ο σαφής προσδιορισμός του θεωρείται σημαντικός για την εκτίμηση του κόστους και του χρόνου που απαιτεί το έργο λογισμικού. Παραμένει, όμως, ένα ανοικτό πρόβλημα καθώς δεν είναι σαφές το τι εννοούμε με τον όρο "μέγεθος" λογισμικού, έτσι δεν υπάρχει συμφωνία ως προς το ποια είναι η πιο αντιπροσωπευτική μετρική για τη μέτρηση του μεγέθους.

Στο [FEN97] το χαρακτηριστικό του μεγέθους διακρίνεται σε τρία άλλα χαρακτηριστικά:

- Μήκος (Length)
- Λειτουργικότητα (Functionality)
- Πολυπλοκότητα (Complexity)

Στην παρούσα εργασία, θα χρησιμοποιήσουμε μετρικές μήκους και πολυπλοκότητας.

Μήκος

Μία οικογένεια μετρικών μήκους, την οποία λαμβάνουμε και εμείς υπ'όψην στη σύγκριση του μεγέθους των ελεγκτών, είναι αυτή του πλήθους των γραμμών κώδικα. Περιλαμβάνει διάφορες μετρικές, που απαριθμούν τις γραμμές κώδικα, οι οποίες διαφέρουν ως προς τον ορισμό της έννοιας της γραμμής κώδικα [FEN14].

Από αυτές τις μετρικές, χρησιμοποιήσαμε τις παρακάτω:

Γραμμές (lines). Το πλήθος των χαρακτήρων carriage return [SOQ16].

Γραμμές κώδικα άνευ σχολίων (Non-commented lines of code-NCLOC). Αυτή η μετρική δεν είναι κάτι άλλο παρά το πλήθος των γραμμών κώδικα, εξαιρουμένων των κενών γραμμών και των γραμμών σχολίων (comments) [GCS87]. Σε άλλες πραγματείες αναφέρεται απλά και ως "γραμμές κώδικα" LOC [ΓΔΤ09], συνήθως μετράται δε σε χιλιάδες γραμμών κώδικα (KLOC ή KNCLOC). Εμείς θα αναφερόμαστε σε αυτή με την συντομογραφία NCLOC, καθώς σε συμφωνία με τα [FEN97, FEN14] θα χρησιμοποιήσουμε το LOC όταν αναφερόμαστε στο συνολικό μέγεθος του λογισμικού (Total size), όπως θα δούμε προσεχώς.

Γραμμές σχολίων κώδικα (Comment Lines Of Code-CLOC). Όποια γραμμή αποτελεί σχόλιο κώδικα [FEN14]. Όπως και στην περίπτωση του NCLOC, δε λαμβάνονται υπ'όψην τυχόν κενές γραμμές. Χρησιμοποιείται και ως μετρική της τεκμηρίωσης (documentation) του λογισμικού, παρ'όλα αυτά μπορεί, όπως προαναφέρθηκε, να οδηγήσει σε λανθασμένα συμπεράσματα όταν πολλά από τα σχόλια είναι τετριμμένα ή στερούμενα νοήματος [WEL01].

Συνολικό μέγεθος. Στο [FEN14] ορίζεται ως το άθροισμα των γραμμών κώδικα και των γραμμών σχολίων κώδικα και αναφέρεται ως LOC, με λίγα λόγια

$$LOC=CLOC+NCLOC$$

Πυκνότητα σχολίων. Μία άλλη χρήσιμη μετρική η οποία αναφέρεται στο [FEN14] είναι αυτή της πυκνότητας σχολίων (comment density)-ας την ονομάσουμε CDEN-, η οποία ορίζεται ως εξής:

$$CDEN=\frac{CLOC}{LOC}$$

Η πυκνότητα σχολίων μπορεί επίσης να χρησιμοποιηθεί για την αξιολόγηση της τεκμηρίωσης του κώδικα, χαρακτηρίζεται όμως και από τα ίδια μειονεκτήματα με τη μετρική CLOC.

Πλήθος δηλώσεων. Μία μετρική που σχετίζεται με αυτές των γραμμών κώδικα είναι το πλήθος των δηλώσεων (statements), στο [FEN14] αναφέρεται με την συντομογραφία ES. Υπολογίζει ό,τι θεωρείται ως δήλωση από τη γλώσσα προγραμματισμού. Δύο ή περισσότερες δηλώσεις στην ίδια γραμμή κώδικα προσμετρώνται ξεχωριστά.

Άλλες μετρικές μήκους που χρησιμοποιούμε μετρούν το μέγεθος του λογισμικού βάσει άλλων μονάδων μέτρησης. Αυτές είναι οι εξής:

- Τάξεις ή κλάσεις (Classes). Το πλήθος των τάξεων του συστήματος λογισμικού.
- Αρχεία (Files). Το πλήθος των αρχείων του συστήματος λογισμικού.
- Φάκελοι (Directories). Το πλήθος των φακέλων του συστήματος λογισμικού.
- Μέθοδοι (Methods). Το πλήθος των μεθόδων του συστήματος λογισμικού.

Πολυπλοκότητα

Ένα άλλο χαρακτηριστικό που συνδέεται συχνά με το μέγεθος του λογισμικού είναι η πολυπλοκότητα. Θα μιλήσουμε συνοπτικά για τις διάφορες έννοιες του όρου πολυπλοκότητα, όπως αυτός χρησιμοποιείται στη βιβλιογραφία, προκειμένου να καταδείξουμε τα προβλήματα που δημιουργεί η σύγχυση περί αυτών αλλά και άλλων χαρακτηριστικών στην ανάλυση του λογισμικού.

Η πολυπλοκότητα συνδέεται συχνά με τον ορισμό του συστήματος [CJH95]. Ένα σύστημα ορίζεται γενικά ως μία ολότητα αλληλεπιδρώντων στοιχείων ή μερών και των μεταξύ τους σχέσεων, η οποία παρουσιάζει ιδιότητες που δεν κατέχουν τα μεμονωμένα μέρη ([ASH56, BER68, FCC99, KLIR01] κοκ. Οι [FCC99] και [KLIR01] αναφέρουν ότι η πολυπλοκότητα διακρίνεται σε οργανωμένη και ανοργάνωτη (organized/disorganized), με την ανοργάνωτη να αντιμετωπίζεται με στατιστικές μεθοδολογίες και την οργανωμένη με μεθοδολογίες από τις επιστήμες των συστημάτων και της

πολυπλοκότητας. Στο [FCC99] η πολυπλοκότητα ορίζεται ως το πλήθος των μερών και των αλληλεπιδράσεων.

Στο [FEN97] η πολυπλοκότητα διακρίνεται περαιτέρω στα εξής χαρακτηριστικά:

- Πολυπλοκότητα προβλήματος (problem complexity)
- Αλγοριθμική πολυπλοκότητα χώρου και χρόνου (space-time algorithmic complexity)
- Δομική πολυπλοκότητα (structural complexity)
- Γνωστική πολυπλοκότητα

Στο ίδιο έργο, η μεν έννοια της πολυπλοκότητας του προβλήματος ταυτίζεται με αυτή της υπολογιστικής πολυπλοκότητας (computational complexity) από το κλασσικό πεδίο της θεωρίας πολυπλοκότητας (complexity theory), η δε αλγοριθμική πολυπλοκότητα κατανοούμε ότι ταυτίζεται με την αντίστοιχη έννοια από το πεδίο της ανάλυσης αλγορίθμων και οι συγγραφείς την εναλλάσσουν με τον όρο απόδοση (efficiency). Η δομική πολυπλοκότητα δίνεται όχι με κάποιον ορισμό αλλά με παραδείγματα ελέγχου ροής, ιεραρχίας και σχέσεων μεταξύ τμημάτων (modules) του λογισμικού. Η γνωστική πολυπλοκότητα προσπερνάται από τους συγγραφείς με παραπομπή στο σώμα των κοινωνικών επιστημών.

Σε αυτό το σημείο διαπιστώνουμε δύο προβλήματα με την έννοια της πολυπλοκότητας, όπως αυτή ορίζεται στα προαναφερόμενα έργα. Το ένα πρόβλημα είναι ότι η πολυπλοκότητα φαίνεται να σχετίζεται εντέλει όχι μόνο με το μέγεθος του λογισμικού, αλλά επίσης και με την απόδοση. Κατ'αυτό τον τρόπο, δεν καθίσταται σαφές το ποιο ή ποια ποιοτικά χαρακτηριστικά μετρώνται από τις μετρικές πολυπλοκότητας. Το δεύτερο έχει να κάνει με τον ορισμό της πολυπλοκότητας αυτής καθ'εαυτής: όπως αναφέραμε και παραπάνω [JOH09], όχι μόνο δεν έχουμε σύμφωνο ορισμό της πολυπλοκότητας με τη γενική έννοια του όρου, αλλά φαίνεται ότι μας διαφεύγει η ίδια η έννοια της πολυπλοκότητας, την οποία προς το παρόν δεν μπορούμε να συλλάβουμε πάρα μόνο διαισθητικά.

Για απλοποίηση, θα επικεντρωθούμε αποκλειστικά στη δομική πολυπλοκότητα και θα χρησιμοποιήσουμε το μέτρο κυκλωματικής πολυπλοκότητας του McCabe, [WEY88, TRA96, FEN97, ΓΔΤ09], το οποίο συμβολίζεται ως $V(g)$ και ορίζεται ως εξής.

$$V(g)=d+1$$

όπου G είναι ο γράφος του διαγράμματος ελέγχου ροής που αντιστοιχεί στο τμήμα λογισμικού που μετράται και d είναι το πλήθος των σημείων απόφασης στο διάγραμμα αυτό.

Το μέτρο αυτό, παρ'όλη τη δημοτικότητα του, έχει δεχθεί μεταξύ πολλών άλλων κριτική (ή οποία μας βρίσκει σύμφωνους) ότι δεν αντιπροσωπεύει πάντα την πολυπλοκότητα του λογισμικού, καθώς εύκολα μπορεί κάποιος να φανταστεί απλό και συντηρήσιμο κώδικα με αρκετούς ελέγχους συνθήκης στην αρχή μίας συνάρτησης ή μεθόδου [FEN97]. Παρ'όλα αυτά, δεχόμαστε το ότι σε γενικές

γραμμές, η ύπαρξη πολλών διαφορετικών μονοπατιών κώδικα πλήττει την ελεγξιμότητα και συνεπώς την συντηρησιμότητα του συστήματος.

Όπως θα δούμε στην επόμενη ενότητα, για τη μελέτη της συντηρησιμότητας χρειαζόμαστε και μία τροποποιημένη εκδοχή της ανωτέρω μετρικής, την επεκτεταμένη κυκλωματική πολυπλοκότητα (extended cyclomatic complexity), $V(g')$ η οποία ισούται με τη $V(g)$ συν το άθροισμα των πράξεων AND και OR σε κάθε σημείο απόφασης.

Με λίγα λόγια, η $V(g')$ εκτός από το πλήθος των σημείων απόφασης, λαμβάνει υπ'όψην και την πολυπλοκότητα κάθε σημείου απόφασης.

Συντηρησιμότητα

Όπως και για τα περισσότερα ποιοτικά χαρακτηριστικά, δεν έχουμε σαφή ορισμό της έννοιας της συνηρησιμότητας. Διαισθητικά μπορούμε να πούμε ότι αντικατοπτρίζει την ευκολία πραγματοποίησης αλλαγών στο σύστημα λογισμικού. Επίσης διαισθητικά, εικάζουμε ότι η συντηρησιμότητα παρουσιάζει αρνητική συσχέτιση με την πολυπλοκότητα.

Για την μέτρηση της συντηρησιμότητας, θα χρησιμοποιήσουμε τη μετρική δείκτη συντηρησιμότητας (maintainability index-MI) [C4S97]. Πρώτα απ'όλα, όμως, πρέπει να εξετάσουμε πως υπολογίζεται η μετρική αυτή.

Η μετρική δείκτη συντηρησιμότητας είναι μία σύνθετη (compound) μετρική, δηλαδή ο ορισμός της εξαρτάται από άλλες μετρικές. Οι μετρικές αυτές είναι:

- ο μέσος όρος πλήθους των γραμμών κώδικα ανά μονάδα λογισμικού $aveLOC$,
- ο μέσος όρος επεκτεταμένης κυκλωματικής πολυπλοκότητας του McCabe ανά μονάδα λογισμικού, $V(g')$ και
- ο μέσος όρος όγκου του Halstead, V.

Ο MI ορίζεται ως εξής:

$$171-5{,}2\,ln(aveV)-0{,}23\,aveV(g')-16{,}2\,ln(aveLOC)$$

Υπάρχει όμως και μία δεύτερη εκδοχή της εξίσωσης του MI, η οποία συμπεριλαμβάνει τη μετρική πλήθους σχολίων, αντικατοπτρίζοντας τη θέση ότι ο σχολιασμός του κώδικα διευκολύνει τη συντήρηση του, η οποία ορίζεται ως εξής:

$$171-5{,}2\,ln(aveV)-0{,}23\,aveV(g')-16{,}2\,ln(aveLOC)+50{,}0\eta\mu\sqrt{2{,}46\,perCM}$$

όπου το $perCM$ αποτελεί το μέσο όρο του ποσοστού των γραμμών σχολίων ανά μονάδα λογισμικού.

Αυτό το μοντέλο υποθέτει ότι τα σχόλια κώδικα έχουν πάντα θετική επίδραση στην συντηρησιμότητα του συστήματος. Όπως όμως προαναφέραμε, τα σχόλια κώδικα μπορεί σε κάποιες περιπτώσεις αφ'ενός να αλλοιώνουν τα αποτελέσματα άλλων μετρικών όπως είναι το LOC, αφ'ετέρου να παραπλανούν τους υπεύθυνους ανάπτυξης, ουσιαστικά δυσχεραίνοντας το έργο της συντήρησης.

Γι'αυτούς τους λόγους, ο Welker [WEL01] συνιστά την επιλογή μίας από τις δύο εξισώσεις, αναλόγως της βαρύτητας που έχουν τα σχόλια στο προς ανάλυση project. Βεβαίως όμως, καθώς η ερμηνεία και αξιολόγηση των σχολίων κώδικα είναι μέχρι στιγμής κυρίως ανθρώπινη δυνατότητα, ο Welker τονίζει την ανάγκη ανθρώπινης παρέμβασης σε αυτή τη δραστηριότητα.

Πρέπει να σημειωθεί πως οι συντελεστές του δείκτη συντηρησιμότητας που παραθέτουμε έχουν επιλεγεί εδώ και αρκετά χρόνια, με βάση δεδομένα που έχουν συλλεχθεί από πολλά έργα λογισμικού σε βάθος χρόνου [C4S97]. Όμως δε γνωρίζουμε κατά πόσο εκείνα τα ευρήματα παραμένουν σε ισχύ σήμερα.

Μέχρι στιγμής δεν αναφερθήκαμε στη μετρική όγκου του Halstead, από την οποία εξαρτάται ο MI, κάτι που κάνουμε στην επόμενη ενότητα.

Επιστήμη λογισμικού του Halstead

Ο Maurice Halstead φιλοδοξούσε να θεμελιώσει την επιστήμη της μέτρησης λογισμικού πάνω σε μετρικές που θυμίζουν τους φυσικούς νόμους [HALL77]. Ξεκινά με απλούς δείκτες, με βάση τους οποίους ορίζει απλές και στη συνέχεια σύνθετες μετρικές.

Η μετρική όγκου που μας ενδιαφέρει, ορίζεται ως εξής:

$$V = N log_2 \mu$$

Ο όρος N είναι το μήκος του προγράμματος, για το οποίο ισχύει

$$N = N_1 + N_2$$

όπου N_1 είναι ο συνολικός αριθμός των τελεστέων (operands) του προγράμματος και N_2 ο συνολικός αριθμός των τελεστών (operators).

Με παρόμοιο τρόπο ορίζεται και η ποσότητα μ του συνόλου των διακριτών τελεστέων και τελεστών

$$\mu = \mu_1 + \mu_2$$

Εκτέλεση - Αποτελέσματα

Εκτέλεση

Σε αυτή την ενότητα, υποθέτουμε ότι το λογισμικό σάρωσης του κώδικα Sonar Scanner είναι εγκατεστημένο στην εικονική μηχανή, στον φάκελο /home/ubuntu/sonar-scanner-2.7/. Επίσης έχουμε ήδη εγκαταστήσει το SonarQube και το πρόσθετο του SonarQube για τη γλώσσα Python.

Από τερματικό (terminal) στην εικονική μηχανή, αλλάξαμε το branch του τοπικού αποθετηρίου του POX, πηγαίνοντας στον φάκελο pox και πληκτρολογώντας:

git checkout carp

Ύστερα, βεβαιώσαμε ότι το τοπικό αποθετήριο git για το POX είναι ανανεωμένο, με χρήση της εντολής

git pull

Επαναλάβαμε την παραπάνω εντολή για το τοπικό αποθετήριο του Ryu.

Στη συνέχεια, εκτελέσαμε το σαρωτή από τον αρχικό φάκελο του POX με την παρακάτω εντολή:

/home/ubuntu/sonar-scanner-2.7/bin/sonar-scanner -Dsonar.projectKey=pox
-Dsonar.projectVersion=0.2.0 -Dsonar.projectName=POX -Dsonar.sources=. -Dsonar.exclusions=tests

Εκτελέσαμε το σαρωτή για τον κώδικα του Ryu από τον υποφάκελο ryu/ryu:

/home/ubuntu/sonar-scanner-2.7/bin/sonar-scanner -Dsonar.projectKey=ryu
-Dsonar.projectVersion=4.6 -Dsonar.projectName=Ryu -Dsonar.sources=.
-Dsonar.exclusions=tests,sdnhub_apps

Μπορούμε να δούμε τα αποτελέσματα μέσω της επιλογής Compare Projects της διεπαφής Ιστού του SonarQube (από προεπιλογή, ο εξυπηρετητής του SonarQube ακούει στη διεύθυνση http://localhost:9000 .)

	POX 0.2.0 SEP 26 2016	RYU 4.6 07:10
Lines of Code	32,365	152,418
Complexity	10,593	34,430
Comments (%)	12.0%	14.0%
Duplicated Lines (%)	1.9%	38.6%
Issues	2,563	5,360
Statements	24,850	96,073
Complexity / File	68.3	77.5
Blocker Issues	0	0
Critical Issues	11	48
Major Issues	1,062	4,037
Open Issues	2,563	5,360
Reopened Issues	0	0
Bugs	39	604
Vulnerabilities	198	923

Για την ανάλυση συντηρησιμότητας χρησιμοποιήσαμε το εργαλείο radon [RAD16]. Το radon παρέχει τη δυνατότητα υπολογισμού του δείκτη συντηρησιμότητας που περιγράψαμε προηγουμένως. Όμως υπολογίζει τη συντηρησιμότητα ανά αρχείο κώδικα. Εμείς χρειαζόμαστε, και υπολογίσαμε, μία τιμή συντηρησιμότητας ανά συγκρινόμενο project, ως εξής.

Αρχικά χρησιμοποιήσαμε την επιλογή του radon για καταγραφή των αποτελεσμάτων σε μορφή JSON, δίνοντας την εντολή, από το φάκελο ryu/ryu:

```
radon mi -j -s -e "tests/*" . > ../mi.json
```

Αμέσως μετά υπολογίσαμε το μέσο όρο του MI ανά αρχείο κώδικα με το παρακάτω Python script:

```
import json

json_file = open("mi.json")

json_str = json_file.read()

json_dict = json.loads(json_str)

json_values = json_dict.values()

sum = 0

n = len(json_values)

for value in json_values:
```

```
        sum += value["mi"]

avg_mi = sum / n
```

όπου avg_mi είναι ο μέσος όρος του δείκτη συντηρησιμότητας.

Επαναλάβαμε αναλόγως για τον ελεγκτή POX από το φάκελο pox.

Αποτελέσματα

Μία πρώτη ματιά στα αποτελέσματα κάνει αντιληπτό το εξής πρόβλημα. Ο POX βρίσκεται σε πολύ πρωιμότερη έκδοση σε σχέση με το Ryu, κάτι που καθιστά αυτονόητο το μικρό του μέγεθος (είτε μετράμε το μέγεθος με γραμμές κώδικα, είτε με πλήθος τάξεων ή αρχείων). Καθώς ο POX είναι έως 4 φορές μικρότερος του Ryu, είναι αναμενόμενο το γεγονός ότι ο POX περιλαμβάνει και μικρότερο αριθμό προβλημάτων (issues), σφαλμάτων (bugs), ευπαθειών (vulnerabilities) και ούτω καθεξής. Γι'αυτό το λόγο, μία απόπειρα άμεσης σύγκρισης μπορεί να οδηγήσει σε λανθασμένα συμπεράσματα.

Συνεπώς, χρειαζόμαστε έναν τρόπο κανονικοποίησης των δεδομένων. Μία απλή λύση σε αυτό το πρόβλημα θα ήταν να συγκρίνουμε τη σύγχρονη έκδοση του POX με μία παλαιότερη έκδοση του Ryu. Όμως δεν γνωρίζουμε ποια πρακτική αξία θα είχε μία τέτοια συγκριτική ανάλυση.

Επίσης θα μπορούσαμε να ακολουθήσουμε την αντίστροφη πορεία, ήτοι να κάνουμε προβολή σε μία υποθετική μεγαλύτερη μελλοντική έκδοση του POX, ίσως χρησιμοποιώντας την τεχνική της παρεκβολής (extrapolation) με το ιστορικό της βάσης κώδικα (codebase) του.

Όπως προαναφέρθηκε όμως, δυστυχώς γνωρίζουμε ελάχιστα όσον αφορά τις σχέσεις μεταξύ των μετρικών (και ιδιαίτερα τη σχέση μεταξύ του πλήθους γραμμών κώδικα και των άλλων μετρικών) ώστε να οδηγηθούμε σε ασφαλή συμπεράσματα.

Θα λάβουμε την ενδιάμεση οδό και θα χρησιμοποιήσουμε το πλήθος των δηλώσεων κώδικα ως βάση για την σύγκριση των άλλων τιμών που μας ενδιαφέρουν. Επιλέγουμε αυτή τη μετρική, καθώς θεωρούμε ότι παρέχει μεγαλύτερη ακρίβεια σε σχέση με τις απλές μετρικές τύπου LOC. Διαιρώντας ένα αποτέλεσμα με το πλήθος των δηλώσεων κώδικα, λαμβάνουμε την αντίστοιχη μέτρηση σε σχέση με το πλήθος των δηλώσεων κώδικα.

Έτσι, όσον αφορά θέματα δυσλειτουργιών και ευπαθειών του λογισμικού, μπορούμε να θεωρήσουμε, απλοποιητικά, ότι ο POX περιλαμβάνει, κατά μέσο όρο, 24850 / 39 = μία δυσλειτουργία ανά 637 δηλώσεις κώδικα, ενώ για το Ryu ισχύει 96,073 / 604 = μία δυσλειτουργία ανά 159 δηλώσεις κώδικα. Παρομοίως βγάζουμε συμπεράσματα και για τις άλλες κατηγορίες δυσλειτουργιών που υποστηρίζει το SonarQube. Οι δυσλειτουργίες όμως αυτές αφορούν τη σημασιολογία της γλώσσας προγραμματισμού και δεν αποτελούν λογικά σφάλματα, τα οποία είναι τα πλέον επικίνδυνα και συνήθη σφάλματα.

Όσον αφορά τις ευπάθειες, η μόνη σχετική μετρική του SonarQube για τη γλώσσα Python είναι αυτή του πλήθους των hardcoded διευθύνσεων IP. Ο POX περιλαμβάνει μία hardcoded διεύθυνση IP ανά 125 δηλώσεις, ενώ ο Ryu μία ανά 104. Δυστυχώς το SonarQube στην παρούσα διευθέτηση δε μπορεί να μας παρέχει πλήρη στατική εικόνα της ασφάλειας του ελεγκτή.

Συνολικά, καμία από τις ελεγχθείσες εκδόσεις δεν περιλαμβάνει blocking issues, δηλ. προβλήματα τα οποία παρεμποδίζουν την ομαλή ανάπτυξη του project. Όσον αφορά κρίσιμα ζητήματα, τα οποία μπορεί να αποτελέσουν πρόβλημα για τη λειτουργικότητα ή τη διαχείριση πόρων σε ένα μέρος του συστήματος, ο POX παρουσιάζει ένα κρίσιμο ζήτημα ανά 2259 δηλώσεις κώδικα, ενώ ο Ryu ένα κρίσιμο ζήτημα ανά 2001 δηλώσεις κώδικα.

Σημαντικό είναι επίσης και το γεγονός ότι κανένα από τα έργα δεν παρουσιάζει παλινδρόμηση (regression). Η παλινδρόμηση αφορά το φαινόμενο της επανεμφάνισης προβλημάτων που θεωρείτο ότι είχαν επιλυθεί στο παρελθόν. Η απουσία παλινδρόμησης συνιστά ένδειξη υψηλής συντηρησιμότητας, αν και επαναλαμβάνουμε ότι στις μετρήσεις δεν περιλαμβάνεται η σημαντική κατηγορία των λογικών σφαλμάτων.

Έχουμε όμως στη διάθεση μας και ποσοστιαία αποτελέσματα που είναι απευθείας συγκρίσιμα. Ένα από αυτά αποτελεί το ποσοστό των γραμμών σχολίων κώδικα. Η τεκμηρίωση του κώδικα και στους δύο ελεγκτές είναι της τάξης του 12%, για τον POX και του 14% για τον Ryu, αντικατοπτρίζοντας, δυστυχώς, την κατάσταση που επικρατεί σε πολλά έργα λογισμικού ελεύθερου/ανοικτού κώδικα, αλλά και τις δυσκολίες της διαδικασίας. Στην περίπτωση του POX κάτι τέτοιο ίσως είναι εύλογο καθώς το project δε φαίνεται να έχει ακόμα βγει από το πειραματικό στάδιο. Το ποσοστό δεν το κρίνουμε αρκετά μεγάλο, ώστε να έχει αξία η ποιοτική ανάλυση των σχολίων.

Ένα άλλο αξιοσημείωτο ποσοστό είναι αυτό των διπλοτύπων γραμμών κώδικα. Ο Ryu φαίνεται να έχει ένα πολύ σημαντικό ποσοστό διπλοτύπων (38.6%), που θα έπρεπε να έχει σοβαρό αντίκτυπο στην συντηρησιμότητα του συστήματος. Θα μπορούσε όμως αυτό το ποσοστό να οφείλεται σε αυτόματα παραγόμενο κώδικα και όχι σε χειροκίνητη εισαγωγή των διπλοτύπων. Δυστυχώς όμως, δεν είμαστε σε θέση να ελέγξουμε αυτή την υπόθεση. Απεναντίας, ο POX παρουσιάζει πολύ μικρό ποσοστό, ένδειξη προσεκτικού σχεδιασμού (design) από τα πρώτα κιόλας στάδια του έργου.

Όσον αφορά τη δομή του κώδικα, μας ενδιαφέρει το μέτρο της πολυπλοκότητας-το οποίο στο SonarQube δεν είναι άλλο από την κυκλωματική πολυπλοκότητα του McCabe-και πιο συγκεκριμένα, το μέτρο Complexity / File, ο μέσος όρος πολυπλοκότητας ανά αρχείο του λογισμικού. Ο POX παρουσιάζει χαμηλότερη τιμή (68.3 σημεία απόφασης έναντι 77.5 για το Ryu).

Οι αντίστοιχες τιμές του δείκτη συντηρησιμότητας, όπως το μετρήσαμε με τη χρήση του radon που περιγράψαμε, είναι περ. 64.25 για τον POX και 71.26 για το Ryu. Το αποτέλεσμα αυτό αποτελεί μία ένδειξη ότι υψηλότερη κυκλωματική πολυπλοκότητα δε συνεπάγεται απαραιτήτως χαμηλότερη συντηρησιμότητα, όπως προειπώθηκε.

Σημειώνουμε ότι το radon λαμβάνει υπ'όψη και τα σχόλια του κώδικα στη μετρική συντηρησιμότητας. Συνεπώς, η απόδοση χαμηλότερου δείκτη συντηρησιμότητας στον POX ίσως είναι απόρροια του βάρους που η μετρική αυτή αποδίδει στα σχόλια, καθώς ο Ryu παρουσιάζει όπως είδαμε λίγο μεγαλύτερη πυκνότητα σχολίων.

Συνολικά, υπενθυμίζουμε ότι οι δύο ελεγκτές είναι δύσκολα συγκρίσιμοι λόγω απόκλισης μεγέθους, παρ'όλο που με την πρώτη ματιά ο POX μοιάζει να είναι ένας Ryu σε μικρότερη κλίμακα. Παρ'ότι προσπαθήσαμε να κανονικοποιήσουμε τα αποτελέσματα, ο POX παρουσιάζει και πάλι γενικά χαμηλότερες τιμές σε σχέση με το Ryu, γεγονός που ίσως σημαίνει ότι η παραπάνω κανονικοποίηση δεν επιτυγχάνει να απαλείψει πλήρως την επιρροή της διαφοράς μεγέθους.

Επιπλέον, τα δεδομένα μας δεν παρέχουν επαρκείς πληροφορίες σχετικά με την κατάσταση των ελεγκτών από πλευράς ορθότητας και ασφάλειας, που είναι δύο σημαντικότατα κριτήρια επιλογής ελεγκτή. Τέλος, δε διαθέτουμε δεδομένα κάλυψης κώδικα (code coverage) από αυτοματοποιημένες διαδικασίες ελέγχου (testing), καθώς δεν εντοπίσαμε εργαλεία στατικής ανάλυσης κάλυψης κώδικα για τη γλώσσα Python. Ο αυτοματοποιημένος έλεγχος είναι κρίσιμος παράγοντας συντηρησιμότητας σε σύγχρονα έργα λογισμικού. Όλες αυτές οι παραλείψεις θα μπορούσαν να καλυφθούν σε μελλοντική έρευνα.

Δοθέντων των ανωτέρω, η κρίση μας μας λέει ότι από τους δύο ελεγκτές στις εκδόσεις που συγκρίθηκαν, ο Ryu είναι εν γένει καταλληλότερος για περιβάλλοντα παραγωγής, εξ'αιτίας της διαφοράς αυτής καθώς και του γεγονότος ότι ο POX βρίσκεται ακόμα σε pre-release επίπεδο.

Παρ'όλα αυτά, με βάση τα δεδομένα μας ο POX παρουσιάζει υψηλά επίπεδα συντηρησιμότητας, μάλιστα δεν χαρακτηρίζεται από τις μικρές αδυναμίες του Ryu. Λόγω δε του μικρού μεγέθους, έχει μεγαλύτερη ευελιξία και περιθώρια πραγματοποίησης αλλαγών, βελτιώσεων και προσθήκης νέων δυνατοτήτων. Για παράδειγμα, ο POX θα μπορούσε λόγω μεγέθους να τροποποιηθεί εύκολα σε αρχιτεκτονικό επίπεδο, ώστε να υποστηρίξει πολυνηματική λειτουργικότητα. Επιπλέον, μπορεί με σχετική ευκολία να προσαρτήσει νέα επίπεδα αφαίρεσης, για την υποστήριξη π.χ. μίας μελλοντικής πρότυπης διεπαφής βορρά.

Συνεπώς, εάν δεχθούμε ότι οι δύο ελεγκτές έχουν όμοιες απαιτήσεις, θα μπορούσε να φτάσει με σχετική ευκολία στα επίπεδα μεγέθους του Ryu με πιθανά μικρότερο κόστος, καθώς το πεδίο του Software Defined Networking θα ωριμάζει.

Προς το παρόν όμως, δεδομένου του νεαρού της ηλικίας του SDN και της έντονης δραστηριότητας γύρω από αυτό, είμαστε βέβαιοι ότι επίκεινται μεγάλες αλλαγές στην τεχνολογία των ελεγκτών και στο περιβάλλον τους, συνεπώς η υψηλή συντηρησιμότητα και των δύο projects είναι σημαντικός παράγοντας για τη μελλοντική τους επιβίωση και επιτυχία.

Υπενθυμίζουμε, πάντως, ότι σε τελική ανάλυση, η μελλοντική πορεία και η καταλληλότητα ενός ελεγκτή δε δύναται να διαπιστωθούν σε απομόνωση, ανεξάρτητα από τις απαιτήσεις του πεδίου προβλήματος ή των εφαρμογών που καλείται να εξυπηρετήσει. Οι παράγοντες αυτοί θα πρέπει να λαμβάνονται υπ'όψην σε μελλοντικές εργασίες πάνω στο ζήτημα αυτό.

Επίλογος

Άμεσος στόχος της εργασίας αυτής ήταν η σύγκριση λογισμικού ελεγκτή SDN. Επιθεωρήσαμε την προηγούμενη ερευνητική δραστηριότητα πάνω στο θέμα αυτό. Καθώς δεν είχαμε τη δυνατότητα για έλεγχο των αποτελεσμάτων των ερευνητών, επιδιώξαμε να λειτουργήσουμε συμπληρωματικά προς αυτές τις εργασίες, ακολουθώντας την οδό της στατικής ανάλυσης λογισμικού με χρήση τεχνικών τεχνολογίας λογισμικού. Στην προσπάθεια μας αυτή, καταδείξαμε αρκετά από τα προβλήματα που εντοπίζονται τόσο στο σημείο τομής των περιοχών της δικτύωσης με την τεχνολογία λογισμικού, όσο και στις δύο αυτές περιοχές ξεχωριστά, αλλά και στο πεδίο της μέτρησης λογισμικού ειδικότερα.

Τα βασικότερα από αυτά τα προβλήματα είναι, κατά τη γνώμη μας, τα διαδεδομένα ανοικτά ζητήματα των περιοχών αυτών (βλ. βόρεια διεπαφή, τοποθέτηση ελεγκτή, τεχνολογία λογισμικού βασισμένη σε αποδείξεις, εργαλεία ανάπτυξης εφαρμογών SDN κλπ), όπως επίσης και η έλλειψη συμφωνίας σχετικά με τις απαιτήσεις που θα πρέπει να ικανοποιεί το λογισμικό ελεγκτή, η έλλειψη συμφωνίας σχετικά με τα ποιοτικά χαρακτηριστικά του λογισμικού, η ασάφεια στη σχέση μεταξύ ποιοτικών χαρακτηριστικών και απαιτήσεων, οι αδυναμίες αξιολόγησης των μετρικών λογισμικού, οι ελλείψεις σε εργαλεία μέτρησης λογισμικού.

Γι’αυτούς τους λόγους, η σύγκριση λογισμικού ελεγκτή SDN κρίνεται ένα ιδιαίτερα δύσκολο εγχείρημα. Ενώ προχωρήσαμε σε μία συγκριτική σκιαγράφηση των χαρακτηριστικών των POX και Ryu, δεν κατορθώσαμε να αποκριθούμε στο ερώτημα της επιλογής ελεγκτή στη γενική περίπτωση.

Παρ’όλα αυτά, ελπίζουμε ότι επιτύχαμε τον έμμεσο στόχο της εργασίας αυτής: την ανάδειξη της ανάγκης για στενότερη συνεργασία μεταξύ των κλάδων της δικτύωσης και της τεχνολογίας λογισμικού. Στην εποχή του λογισμικού στη δικτύωση, πιστεύουμε ότι αυτή η συνέργεια θα είναι απαραίτητη.

Βιβλιογραφικές αναφορές

[ACI14] Araniti, G., J. Cosmas, A. Iera, A. Molinaro, R. Morabito, και A. Orsino. ‘OpenFlow over wireless networks: Performance analysis’, 1–5. IEEE, 2014. doi:10.1109/BMSB.2014.6873559.

[AGI16] ‘Agile software development’, *Wikipedia, the free encyclopedia*. 26-Σεπτεμβρίου-2016.

[AGM16] ‘Manifesto for Agile Software Development’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://agilemanifesto.org/. [Ημερομηνία πρόσβασης: 02-Οκτωβρίου-2016].

[AND11] M. Andreessen, ‘Why Software Is Eating The World’, *Wall Street Journal*, 20-Αυγούστου-2011.

[ASH56] W. R. Ashby, ‘An introduction to cybernetics’, 1957.

[BDG14] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, και others, ‘P4: Programming protocol-independent packet processors’, *ACM SIGCOMM Computer Communication Review*, τ. 44, τχ. 3, σσ 87–95, 2014.

[BIE66] J. Bieri, ‘Cognitive Complexity and Personality Development’, στο *Experience Structure & Adaptability*, O. J. Harvey, Επιμ. Springer Berlin Heidelberg, 1966, σσ 13–37.

[BJO98] D. Bjørner, ‘Domains as a prerequisite for requirements and software domain perspectives and facets, requirements aspects and software views’, στο *Requirements Targeting Software and Systems Engineering*, Springer, 1998, σσ 1–41.

[C4S97] M. Bray, K. Brune, D. A. Fisher, J. Foreman, και M. Gerken, ‘C4 Software Technology Reference Guide - A Prototype.’, Ιανουαρίου 1997.

[CAJ] J. Carapinha και J. Jiménez, ‘Network virtualization: a view from the bottom’, στο *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, 2009, σσ 73–80.

[CJH95] S. Cant, D. Jeffery, και B. Henderson-Sellers, ‘A conceptual model of cognitive complexity of elements of the programming process’, *Information and Software Technology*, τ. 37, τχ. 7, σσ 351–362, 1995.

[DOV13] DOVER, Jeremy M. A denial of service attack against the Open Floodlight SDN controller. 2013.

[ERI13] D. Erickson, ‘The beacon openflow controller’, στο *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, σσ 13–18.

[FCC93] R. L. Flood και E. R. Carson, *Dealing with Complexity*. Boston, MA: Springer US, 1993.

[FEN97] N. E. Fenton, *Software metrics*, 2. ed., 1. print. London [u.a.]: PWS Publ. [u.a.], 1997.

[FEN14] Norman Fenton και James Bieman, *Software Metrics*. CRC Press, 2014.

[FHF14] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, και D. Walker, 'Frenetic: A network programming language', στο *ACM Sigplan Notices*, 2011, τ. 46, σσ 279–291.

[FIN11] A. Finkelstein, 'ENGINEERING CHALLENGES OF NEW BUSINESS MODELS IN SOFTWARE'.

[FKR05] W. B. Frakes και K. Kang, 'Software reuse research: status and future', *IEEE Transactions on Software Engineering*, τ. 31, τχ. 7, σσ 529–536, Ιουλίου 2005.

[FOR95] J. P. Forgas, 'Mood and judgment: the affect infusion model (AIM)', *Psychol Bull*, τ. 117, τχ. 1, σσ 39–66, Ιανουαρίου 1995.

[FRE16] 'The Frenetic Project'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://www.frenetic-lang.org/. [Ημερομηνία πρόσβασης: 02-Οκτωβρίου-2016].

[GCS87] R. B. Grady, *Software Metrics: Establishing a company-wide program*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[GIH15] '1.3 Introduction to Clouds History 07:40 - YouTube'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.youtube.com/watch?v=waqgUgHbhZY&index=5&list=PLFd87qVsaLhOkTLvfp6MC94iFa_1c9wrU. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[GIN15] '1.4 Introduction to Clouds What s New in Today's Clouds 07:34 - YouTube'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.youtube.com/watch?v=KLb4DRfh2Vk&index=6&list=PLFd87qVsaLhOkTLvfp6MC94iFa_1c9wrU. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[GNC16] 'gnu.org'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.gnu.org/software/coreutils/coreutils.html. [Ημερομηνία πρόσβασης: 18-Σεπτεμβρίου-2016].

[GRU09] T. Gruber, 'Ontology', στο *Encyclopedia of Database Systems*, L. LIU και M. T. ÖZSU, Επιμ. Springer US, 2009, σσ 1963–1965.

[HAL77] M. H. Halstead, *Elements of software science*. New York [u.a.]: Elsevier, 1977.

[HGS90] J. M. Hoc, T. Green, R. Samurcay, και D. Gilmore, *Psychology of Programming*. Academic Press Inc., 1990.

[HPI16] ‘Hping - Active Network Security Tool’. [Έκδοση σε ψηφιακή μορφή] . Διαθέσιμο στο: http://www.hping.org/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[HUL11] E. Hull, K. Jackson, και J. Dick, *Requirements Engineering*. London: Springer London, 2011.

[HYY16] T. Huang, S. Yan, F. Yang, T. Pan, και J. Liu, ‘Building SDN-Based Agricultural Vehicular Sensor Networks Based on Extended Open vSwitch’, *Sensors*, τ. 16, τχ. 1, σ 108, Ιανουαρίου 2016.

[JOH09] N.F. Johnson, ‘Two's company, three is complexity: a simple guide to the science of all sciences’, Oxford:Oneworld, 2007.

[JSS14] M. Jammal, T. Singh, A. Shami, R. Asal, και Y. Li, ‘Software defined networking: State of the art and research challenges’, *Computer Networks*, τ. 72, σσ 74–98, Οκτωβρίου 2014.

[JSW13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, και others, ‘B4: Experience with a globally-deployed software defined WAN’, *ACM SIGCOMM Computer Communication Review*, τ. 43, τχ. 4, σσ 3–14, 2013.

[KAN02] S. H. Kan, ‘Software quality metrics overview’, *Metrics and Models in Software Quality Engineering*, σσ 85–120, 2002.

[KHA14] Z. K. Khattak, M. Awais, και A. Iqbal, ‘Performance evaluation of OpenDaylight SDN controller’, στο *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2014, σσ 671–676.

[KHO14] R. Khondoker, A. Zaalouk, R. Marx, και K. Bayarou, ‘Feature-based comparison and selection of Software Defined Networking (SDN) controllers’, στο *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, 2014, σσ 1–7.

[KLIR01] G. J. Klir, *Facets of Systems Science*. Boston, MA: Springer US, 2001.

[KRE15] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, και S. Uhlig, ‘Software-Defined Networking: A Comprehensive Survey’, *arXiv:1406.0440 [cs]*, Ιουνίου 2014.

[KRN01] Kurose, J., and K. Ross. "Computer networks: a top-down approach." (2001).

[LNV16] ‘danieltt/libnetvirt’, *GitHub*. [Έκδοση σε ψηφιακή μορφή] . Διαθέσιμο στο: https://github.com/danieltt/libnetvirt. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[MIN16] ‘Mininet: An Instant Virtual Network on your Laptop (or other PC) - Mininet’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://mininet.org/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[NOX08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, και S. Shenker, 'NOX: towards an operating system for networks', *ACM SIGCOMM Computer Communication Review*, τ. 38, τχ. 3, σσ 105–110, 2008.

[ODL16] 'The OpenDaylight Platform | OpenDaylight'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.opendaylight.org/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[OMN16] 'OMNeT++ Discrete Event Simulator - Home'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://omnetpp.org/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[OND16] 'Software-Defined Networking (SDN) Definition - Open Networking Foundation'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.opennetworking.org/sdn-resources/sdn-definition. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[ONF15] 'Framework for SDN: Scope and Requirements – Open Networking Foundation', TR-516, June 2015. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.opennetworking.org/sdn-resources/technical-library. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[ONF16] 'Home - Open Networking Foundation'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.opennetworking.org/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[PWC09] PricewaterhouseCoopers, 'Technology Forecast: Spring 09', *PwC*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://www.pwc.com/us/en/technology-forecast/spring2009.html. [Ημερομηνία πρόσβασης: 03-Οκτωβρίου-2016].

[RAD16] 'radon 1.4.2 : Python Package Index'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://pypi.python.org/pypi/radon. [Ημερομηνία πρόσβασης: 27-Σεπτεμβρίου-2016].

[RAL11] P. Ralph, 'Toward a theory of debiasing software development', στο *EuroSymposium on Systems Analysis and Design*, 2011, σσ 92–105.

[RAU11] K. Rausch, 'Network Virtualization – An Overview', 2011.

[ROM12] ROMERO DE TEJADA MUNTANER, Guillermo. Evaluation of OpenFlow Controllers. 2012.

[SAA08] T. L. Saaty, 'Decision making with the analytic hierarchy process', *International journal of services sciences*, τ. 1, τχ. 1, σσ 83–98, 2008.

[SCH13] S. Scott-Hayward, G. O'Callaghan, και S. Sezer, 'Sdn Security: A Survey', 2013, σσ 1–7.

[SDH16] 'All-in-one SDN App Development Starter VM | SDN Hub'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://sdnhub.org/tutorials/sdn-tutorial-vm/?ModPagespeed=noscript. [Ημερομηνία πρόσβασης: 18-Σεπτεμβρίου-2016].

[SDS14] K. P. Srinivasan και T. Devi, ‘Software Metrics Validation Methodologies in Software Engineering’, *International Journal of Software Engineering & Applications*, τ. 5, τχ. 6, σσ 87–102, Νοεμβρίου 2014.

[SDX16] ‘What are SDN Controllers (or SDN Controller Platforms)?’, *SDxCentral*, 14-Ιανουαρίου-2014. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.sdxcentral.com/sdn/definitions/sdn-controllers/. [Ημερομηνία πρόσβασης: 28-Αυγούστου-2016].

[SHA48] C. E. Shannon, ‘A mathematical theory of communication’, *ACM SIGMOBILE Mobile Computing and Communications Review*, τ. 5, τχ. 1, σσ 3–55, 2001.

[SHA13] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, και S. A. Mehdi, ‘An architectural evaluation of SDN controllers’, 2013, σσ 3504–3508.

[SHE10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, και G. M. Parulkar, ‘Can the production network be the testbed?’, στο *OSDI*, 2010, τ. 10, σσ 1–6.

[SHL13] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, και R. Smeliansky, ‘Advanced study of SDN/OpenFlow controllers’, 2013, σσ 1–6.

[SKK15] A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, και M. Budiu, ‘DC.p4: programming the forwarding plane of a data-center switch’, 2015, σσ 1–8.

[SOQ16] ‘Metric Definitions - SonarQube Documentation - SonarQube’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://docs.sonarqube.org/display/SONAR/Metric+Definitions. [Ημερομηνία πρόσβασης: 11-Σεπτεμβρίου-2016].

[SPP16] ‘Python Plugin - Plugins - SonarQube’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://docs.sonarqube.org/display/PLUG/Python+Plugin. [Ημερομηνία πρόσβασης: 18-Σεπτεμβρίου-2016].

[SSC16] ‘Analyzing with SonarQube Scanner - Scanners - SonarQube’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner. [Ημερομηνία πρόσβασης: 18-Σεπτεμβρίου-2016].

[STA15] STANCU, Alexandru L., et al. A comparison between several Software Defined Networking controllers. In: *Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS), 2015 12th International Conference on*. IEEE, 2015. p. 223-226.

[TAN03] Tanenbaum, Andrew S. "Computer networks, 4th edition." *ed: Prentice Hall* (2003).

[THS14] D. Turull, M. Hidell, και P. Sjödin, ‘Performance evaluation of openflow controllers for network virtualization’, στο *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*, 2014, σσ 50–56.

[TOO12] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, και R. Sherwood, 'On Controller Performance in Software-Defined Networks', στο *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.

[TRA96] R. Trappl, Österreichische Studiengesellschaft für Kybernetik, και European Meeting on Cybernetics and Systems Research, Επιμ., *Cybernetics and systems '96: proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research, held at the University of Vienna, Austria, 9 - 12 April 1996*. Vienna: Austrian Society for Cybernetic Studies, 1996.

[VBO16] 'Oracle VM VirtualBox'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.virtualbox.org/. [Ημερομηνία πρόσβασης: 18-Σεπτεμβρίου-2016].

[WCT92] [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://c2.com/doc/oopsla92.html. [Ημερομηνία πρόσβασης: 12-Σεπτεμβρίου-2016].

[WEL01] K. D. Welker, 'The software maintainability index revisited', *CrossTalk*, τ. 14, σσ 18–21, 2001.

[WEY88] E. J. Weyuker, 'Evaluating software complexity measures', *IEEE Transactions on Software Engineering*, τ. 14, τχ. 9, σσ 1357–1365, Σεπτεμβρίου 1988.

[YAO14] G. Yao, J. Bi, Y. Li, και L. Guo, 'On the Capacitated Controller Placement Problem in Software Defined Networks', *IEEE Communications Letters*, τ. 18, τχ. 8, σσ 1339–1342, Αυγούστου 2014.

[ZHA] E. G. Renart, E. Z. Zhang, και B. Nath, 'Towards a GPU SDN controller', στο *Networked Systems (NetSys), 2015 International Conference and Workshops on*, 2015, σσ 1–5.

[ΓΔΤ09] Ε. Γιακουμάκης και Ν. Διαμαντίδης, 'Τεχνολογία Λογισμικού', Αθήνα: Σταμούλης, 2009.