ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

Διπλωματική Εργασία

Ανάπτυξη συστήματος

py-p2pinet

Συνεργατική Πρόσβαση στο Διαδίκτυο

Συγγραφέας

Αγγελίδης Νίκος

Επιβλέπων

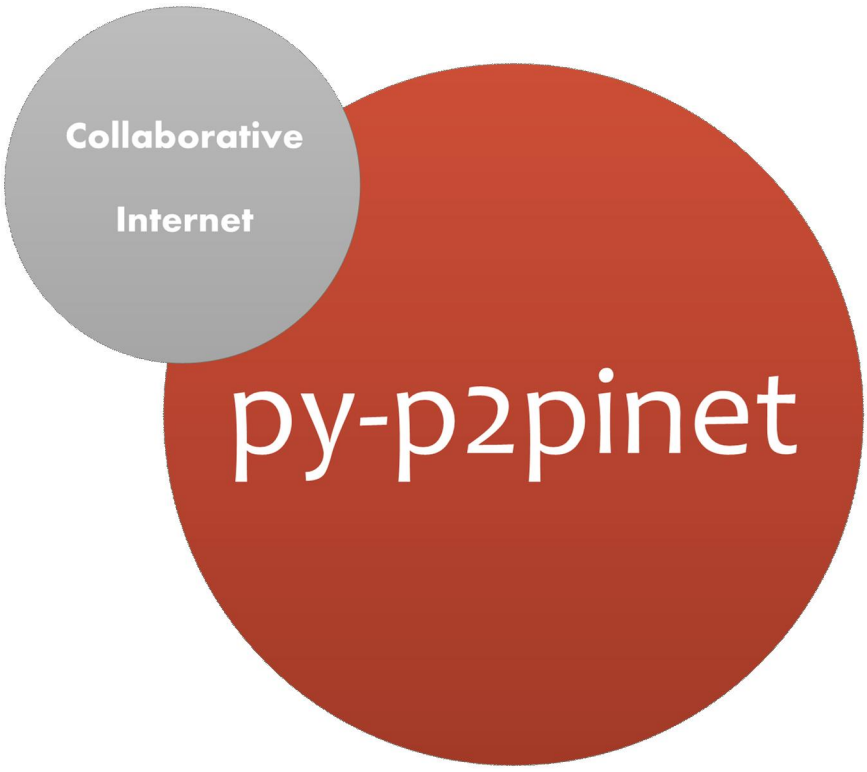
Καθ. Παύλος Σπυράκης

Συνεπιβλέπων

Δρ. Ιωάννης Χατζηγιαννάκης

Ευχαριστίες

Οι ευχαριστίες εδώ είναι δυναμικές. Απευθύνονται αρχικά σε όσους διαβάζουν αυτές τις γραμμές και ακόμα περισσότερο σε αυτούς που θα αφιερώσουν χρόνο να διαβάσουν μέχρι και τις τελευταίες. Επίσης θα ήθελα να ευχαριστήσω τους επιβλέποντες μου κύριο Παύλο Σπυράκη και κύριο Ιωάννης Χατζηγιαννάκη για το ενδιαφέρων θέμα και την καθοδήγησή τους, καθώς και τον ερευνητή κύριο Κονίνη Χρήστο για την υπομονή τους και τις πολύτιμες συμβουλές τους.



Πρόλογος

Το παρών κείμενο συγκεντρώνει αρκετή από την γνώση - ή τουλάχιστον όση δύναται να αποτυπωθεί στο χαρτί - που απέκτησα κατά το τελευταίο εξάμηνο των σπουδών μου στο Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του πανεπιστημίου Πατρών. Πραγματεύεται το συναρπαστικό θέμα του συνεργατικού Internet μέσω συνάθροισης συνδέσμων (link aggregation).

Στα πλαίσια της διπλωματικής αυτής αναπτύχθηκε το σύστημα **py-p2pinet**.

Πάτρα, Ιούνιος 2010

Περιεχόμενα

1 Εισαγωγή

1.1 Κίνητρο και σημασία του θέματος

Ένα δίκτυο υπολογιστών είναι η σύνδεση δύο ή περισσότερων υπολογιστών που τους επιτρέπει να μοιράζονται πόρους. Τα οφέλη της δικτύωσης είναι σημαντικά και είναι χωρίς υπερβολή είναι ένας από τους βασικούς λόγους που οι υπολογιστές έχουν εξαπλωθεί σχεδόν σε κάθε πτυχή τις καθημερινότητας μας.

Το αποτέλεσμα από τις διαφορετικές ανάγκες που καλούνται να καλύψουν τα δίκτυα υπολογιστών , είναι ότι έχει εμφανιστεί μια μεγάλη σειρά από ετερογενή μεταξύ τους δίκτυα, τα οποία διαφοροποιούνται τόσο στον τρόπο διασύνδεσης και στις οντότητες που διασυνδέονται όσο και στην τοπολογία, λειτουργία, χρησιμότητα, αξιοπιστία, και μια μεγάλη ακολουθία από άλλα χαρακτηριστικά.

Τις περισσότερες φορές για τον μέσο χρήστη, ή έννοια του δικτύου είναι συνυφασμένη με το Internet. Έτσι λοιπόν είναι λογικό και αναμενόμενο τα δίκτυα υπολογιστών και ακόμη πιο συγκεκριμένα ότι έχει σχέση με το Διαδίκτυο να αποτελούν βασικό τομέα έρευνας στην επιστήμη τον υπολογιστών.

Άλλο ένα χαρακτηριστικό τις εποχής μας είναι η εξάπλωση τον ασύρματων τηλεπικοινωνιών και τις ασύρματης δικτύωσης. Με τον όρο ασύρματη δικτύωση αναφερόμαστε στο WiFi[1] και στο πρότυπο που το ορίζει, το IEEE 802.11[2].

Είναι προφανές ότι η δυνατότητα διασύνδεσης χωρίς φυσικά μέσα είναι το χαρακτηριστικό που έχει κάνει τις ασύρματες τηλεπικοινωνίες τόσο ελκυστικές, καθώς οι περιορισμοί χρησιμοποίησης μειώνονται δραματικά, και οι δυνατότητες για τη χρήση και εφαρμογή των δικτυακών τεχνολογιών αυξάνονται.

[1] http://en.wikipedia.org/wiki/Wi-Fi
[2] http://en.wikipedia.org/wiki/IEEE_802.11

Μια ιδέα που αποτελεί αντικείμενο έρευνας αρκετό καιρό είναι η εκμετάλλευση της ασύρματης δικτύωσης και κάποιων τεχνικών συνάθροισης πολλαπλών συνδέσμων (multihoming) για την βελτίωση της Internet εμπειρίας (πιο συγκεκριμένα του last-mile connectivity[3]) ενός χρήστη.

[3] http://en.wikipedia.org/wiki/Last_mile

1.2 Τα Ασύρματα Δίκτυα WiFi

Ως ασύρματο δίκτυο χαρακτηρίζεται το τηλεπικοινωνιακό δίκτυο, συνήθως τηλεφωνικό ή δίκτυο υπολογιστών, το οποίο χρησιμοποιεί, ραδιοκύματα ως φορείς πληροφορίας. Τα δεδομένα μεταφέρονται μέσω ηλεκτρομαγνητικών κυμάτων, με συχνότητα φέροντος η οποία εξαρτάται κάθε φορά από τον ρυθμό μετάδοσης δεδομένων που απαιτείται να υποστηρίζει το δίκτυο. Η ασύρματη επικοινωνία, σε αντίθεση με την ενσύρματη, δεν χρησιμοποιεί ως μέσο μετάδοσης κάποιον τύπο καλωδίου. Στα ασύρματα δίκτυα εντάσσονται τα δίκτυα κινητής τηλεφωνίας, οι δορυφορικές επικοινωνίες, τα ασύρματα δίκτυα ευρείας περιοχής (WWAN), τα ασύρματα μητροπολιτικά δίκτυα [(WMAN),] τα ασύρματα τοπικά δίκτυα (WLAN) και τα ασύρματα προσωπικά δίκτυα (WPAN).

Η ονομασία WiFi περιγράφει την τεχνολογία των ασύρματων δικτύων τοπικής εμβέλειας (Wireless Local Area Networks - WLANs), τα οποία τυποποιούνται στο πρότυπο της IEEE 802.11.

Το IEEE 802.11 είναι μια οικογένεια προτύπων της IEEE για ασύρματα τοπικά δίκτυα (WLAN) που είχαν ως σκοπό να επεκτείνουν το 802.3 (Ethernet, το συνηθέστερο πρωτόκολλο ενσύρματης δικτύωσης υπολογιστών) στην ασύρματη περιοχή. Τα πρότυπα 802.11 είναι ευρύτερα γνωστά ως «WiFi» επειδή η WiFi Alliance, ένας οργανισμός ανεξάρτητος της IEEE, παρέχει την πιστοποίηση για τα προϊόντα που υπακούν στις προδιαγραφές του 802.11. Αυτή η οικογένεια πρωτοκόλλων αποτελεί το καθιερωμένο πρότυπο της βιομηχανίας στο χώρο των ασύρματων τοπικών δικτύων.

Ο όρος WiFi (Wireless Fidelity, κατά την ορολογία High Fidelity η οποία αφορά την εγγραφή ήχου) χρησιμοποιείται για να προσδιορίσει τις συσκευές που βασίζονται στην προδιαγραφή IEEE 802.11 b/g και εκπέμπουν σε συχνότητες 2.4GHz.

Ωστόσο το WiFi έχει επικρατήσει και ως όρος αναφερόμενος συνολικά στα ασύρματα τοπικά δίκτυα.

Στα ασύρματα δίκτυα WiFi έχουν τη δυνατότητα να συνδέονται συσκευές νέας τεχνολογίας, που είναι εξοπλισμένες με τους αντίστοιχους adapters και έτσι να μπορούν να συνδεθούν στο Διαδίκτυο. Τέτοιες συσκευές είναι οι προσωπικοί υπολογιστές, έξυπνα κινητά τηλέφωνα, PDAs και άλλες. Συνήθεις εφαρμογές του εκτός από την παροχή ασύρματων δυνατοτήτων πρόσβασης στο Internet, είναι η παροχή τηλεφωνίας μέσω διαδικτύου (VoIP) και διασύνδεσης μεταξύ ηλεκτρονικών συσκευών όπως τηλεοράσεις, ψηφιακές κάμερες, DVD Player και ηλεκτρονικοί υπολογιστές.

Μία ψηφιακή συσκευή με κάρτα ασύρματης δικτύωσης WiFi, π.χ. ένας ηλεκτρονικός υπολογιστής, μπορεί να συνδεθεί στο Διαδίκτυο όταν βρίσκεται σε ακτίνα κάλυψης ασύρματου δικτύου ήδη συνδεδεμένου στο Internet, το οποίο ονομάζεται σημείο πρόσβασης (Access Point). Μία περιοχή που καλύπτεται από ένα ή περισσότερα σημεία πρόσβασης συνδεδεμένα μεταξύ τους λέγεται Hotspot[4]. Ένα hotspot μπορεί να καλύπτει έναν χώρο έκτασης δωματίου ή και πολλών τετραγωνικών μέτρων, με εναλλασσόμενα σημεία πρόσβασης. Έτσι η τεχνολογία WiFi επιτρέπει τη σύνδεση μεταξύ δύο συσκευών μεταξύ τους, τη σύνδεση ενός προσωπικού υπολογιστή με ένα τοπικό δίκτυο και άλλους υπολογιστές και, στη συνέχεια, μέσω αυτών στο Internet. Η ψηφιακή συσκευή που υποστηρίζει το WiFi μπορεί να συνδεθεί οπουδήποτε υπάρχει σημείο πρόσβασης (π.χ. σε πάρκα ή πλατείες μεγάλων πόλεων, καφετέριες, βιβλιοθήκες κλπ).

Πλέον, πολλές εμπορικές επιχειρήσεις ή υπηρεσίες προσφέρουν συνδεσιμότητα σε WiFi δίκτυα για εξυπηρέτηση των πελατών τους. Ακόμα όλες οι ευρυζωνικές συνδέσεις συνοδεύονται από ένα router με δυνατότητα εκπομπής WiFi δικτύου, και όλοι οι φορητοί υπολογιστές είναι εφοδιασμένοι με το κατάλληλο υλικό για σύνδεση σε WiFi δίκτυα.

Πέρα από το σαφές πλεονέκτημα της ασύρματης συνδεσιμότητας που παρέχει το WiFi, το γεγονός της ευρείας κυκλοφορίας συσκευών που υποστηρίζουν σύνδεση σε ένα WiFi δίκτυο, οφείλεται στη διαρκή πτώση του κόστους του απαραίτητου υλικού. Εκτός αυτού, μεγάλο πλεονέκτημα αποτελεί η οικουμενικότητα που χαρακτηρίζει τις συσκευές αυτές και τα δίκτυα: μια συσκευή με δυνατότητα σύνδεσης σε WiFi, θα μπορεί να συνδεθεί σε οποιοδήποτε σημείο του κόσμου και να βρίσκεται το δίκτυο αυτό.

[4] http://en.wikipedia.org/wiki/Hotspot_(Wi-Fi)

1.3 Συνάθροιση Πολλαπλών Συνδέσμων (Multihoming)

Όπως έχει προαναφερθεί, στην εποχή μας ο τομέας των δικτύων υφίσταται ραγδαίες εξελίξεις μέρα με τη μέρα, και διαρκώς αναδύονται νέες τεχνολογίες και βελτιώσεις για τις ήδη υπάρχουσες πάνω στην αξιοπιστία και ποιότητα των υπηρεσιών. Οι ταχύτητες σύνδεσης στο Internet πλέον έχουν περάσει στο επίπεδο της Ευρείας Ζώνης Σύνδεσης, ή Broadband Connection. Υπάρχει μεγάλη ποικιλία παρόχων ψηφιακών συνδέσεων (Digital Subscriber Line --- DSL) που προσφέρουν τη δυνατότητα για διαρκή, αξιόπιστη και γρήγορη σύνδεση στο Internet.

Εκτός από οικιακή χρήση, πλέον πανεπιστήμια, σχολεία, νοσοκομεία, ερευνητικά κέντρα, αλλά και επιχειρήσεις και υπηρεσίες, χρησιμοποιούν τέτοιες συνδέσεις για τη βελτίωση της λειτουργίας τους. Είναι γεγονός πλέον ότι περιστοιχιζόμαστε από πολυάριθμα ασύρματα αλλά και ενσύρματα δίκτυα, τα οποία με συνδέσεις ευρείας ζώνης διασυνδέονται τελικά και με το Internet.

Πέρα από την εξέλιξή της και την υψηλή αποδοτικότητά της, η ευρυζωνική σύνδεση στο Internet έχει και μερικά ακόμη ενδιαφέροντα χαρακτηριστικά. Λόγω της ψηφιακής σύνδεσης, ανεξαρτητοποιημένης από τηλεφωνικές κλήσεις, μια ευρυζωνική σύνδεση είναι διαρκώς ενεργή, χωρίς να γίνονται αποσυνδέσεις, αφού η ψηφιακή γραμμή είναι αφιερωμένη σε αυτή τη σύνδεση.

Το γεγονός αυτό συνεπάγεται ότι κατά μέσο όρο, μια τέτοια σύνδεση θα είναι με μεγάλη πιθανότητα ανενεργή για μεγάλο χρονικό διάστημα. Πέρα από τη φύση της σύνδεσης, αν κανείς παρατηρήσει τη λειτουργία του λογισμικού που χρησιμοποιεί τη συνδεσιμότητα, θα δει πως κάθε σύνδεση προς το Internet δύσκολα μπορεί να υποστηρίξει πολλαπλές ροές προς το Internet.

Με την έννοια ροή περιγράφεται μια εγκαθίδρυση ζεύγους αποστολής-λήψης δεδομένων (ή και αντίστροφα) προς κάποιον απομακρυσμένο (δικτυακά) προορισμό. Πρακτικά αυτό σημαίνει ότι όταν ένας χρήστης εργάζεται στον Παγκόσμιο Ιστό, ή χρησιμοποιεί κάποιο σύστημα Ομότιμων, περιορίζονται οι δυνατότητες να εργαστεί και σε κάτι άλλο.

Η ύπαρξη αυτής της πληθωρικής ποικιλίας συνδέσεων προς το Internet, καθώς και των χαρακτηριστικών αυτών των συνδέσεων που περιγράφηκαν παραπάνω, γέννησε την ιδέα της Συνάθροισης Πολλαπλών Συνδέσμων, ή όπως αναφέρεται στη διεθνή βιβλιογραφία, του Multiple Link Aggregation ή Multihomed Collaborative Internet Access (εν συντομία, Multihoming).

Το Multihoming συνίσταται στο συνδυασμό και χρήση πολλαπλών συνδέσμων με το Internet με τρόπο παράλληλο, ώστε να αυξηθεί η ταχύτητα της σύνδεσης και η αξιοπιστία της. Η απόδοση αυτών των τεχνικών εξαρτάται από πολλούς παράγοντες, που φθάνουν μέχρι και το υλικό που χρησιμοποιείται για την υλοποίηση των συνδέσεων. Αξίζει να αναφερθεί ότι οι μέθοδοι υλοποίησης της τεχνικής του Multihoming δεν περιορίζονται μόνο στο λογικό επίπεδο (με χρήση προγραμματισμού στο λειτουργικό σύστημα). Έχουν γίνει προσπάθειες να υλοποιηθεί η τεχνική και με υλικά μέσα, με ειδικό δηλαδή εξοπλισμό. Το Multihoming υλοποιείται με διάφορες παραλλαγές, αναλόγως τη διαθεσιμότητα συνδέσμων με το Internet και του κατάλληλου υλικού. Πολλές φορές μάλιστα για την υλοποίηση συγκεκριμένων τεχνικών, απαιτείται και συνεργασία από τον πάροχο της υπηρεσίας Internet. Οι σημαντικότερες μορφές του Multihoming περιγράφονται παρακάτω :

- Μοναδικός Σύνδεσμος, Πολλαπλές IP διευθύνσεις

Η οντότητα στην οποία υλοποιείται η τεχνική αυτή, κατέχει ένα μοναδικό σύνδεσμο με το Internet, αλλά πολλαπλές IP διευθύνσεις με αποτέλεσμα να γίνεται η χρήση όλων των διευθύνσεων με λογισμικό. Όταν αποτύχει η μοναδική σύνδεση, χάνεται όλη η συνδεσιμότητα.

- Πολλαπλές Δικτυακές Διατάξεις (interfaces), Μοναδική IP ανά διάταξη

Λόγω της ύπαρξης μιας IP διεύθυνσης ανά διάταξη (interface), στην τεχνική αυτή μια αποτυχία δεν σημαίνει αποτυχία όλης της συνδεσιμότητας, καθώς οι υπόλοιπες διατάξεις εξακολουθούν να παρέχουν τη δική τους σύνδεση.

- Πολλαπλοί Σύνδεσμοι, Μοναδική IP διεύθυνση

Σε αυτή την τεχνική, χρησιμοποιείται κάποιο πρωτόκολλο δρομολόγησης ανάμεσα στους πολλαπλούς συνδέσμους, και ο στόχος εδώ είναι να απαλειφθεί η πιθανότητα όπου σφάλμα σε ένα σύνδεσμο σημαίνει και απώλεια της συνδεσιμότητας, καθώς οι υπόλοιποι σύνδεσμοι παραμένουν ενεργοί.

- Πολλαπλοί Σύνδεσμοι, Πολλαπλές IP Διευθύνσεις

Η προσέγγιση αυτή απαιτεί χρήση ειδικού υλικού, του λεγόμενου Link Load Balancer. Επιτρέπει ταυτόχρονη χρήση όλων τον συνδέσμων ώστε να αυξηθεί το συνολικό διαθέσιμο εύρος ζώνης, και λόγω του ειδικού υλικού παρέχει τη δυνατότητα εντοπισμού αποτυχιών στους συνδέσμους σε

πραγματικό χρόνο, καθώς και ανακατεύθυνση της μεταφοράς πληροφοριών επίσης σε πραγματικό χρόνο. Και εδώ χρησιμοποιούνται αλγόριθμοι για τη διαχείριση της μεταφοράς πληροφοριών.

1.4 Στόχοι της διπλωματικής

Οι τεχνολογίες αυτές δίνουν ξεκάθαρα τη δυνατότητα βελτίωσης των δικτυακών υπηρεσιών :

- § επιτυγχάνονται χαμηλότερες καθυστερήσεις στις συνδέσεις,
- § υψηλότερη ρυθμαπόδοση (throughput - μέσος ρυθμός επιτυχούς μετάδοσης δεδομένων)
- § υψηλότερη διαθεσιμότητα σύνδεσης.

Μια ακόμη πτυχή των βελτιώσεων είναι η προσαρμοστικότητα και η ανοχή σε σφάλματα. Πρακτικά αυτό σημαίνει ότι όταν έχει γίνει συνάθροιση των διαθέσιμων συνδέσμων, και κάποιος από αυτούς αποτύχει, αυτό δεν συνεπάγεται με πλήρη απώλεια της συνδεσιμότητας, αφού οι υπόλοιποι σύνδεσμοι εξακολουθούν να λειτουργούν κανονικά.

Έχοντας μια πλήρη εικόνα πάνω στα ασύρματα δίκτυα, σε τεχνολογίες όπως τα Συστήματα Ομότιμων (Peer 2 Peer systems), καθώς και στη σύγχρονη ιδέα του Multihoming, δίνεται η δυνατότητα συνδυασμού όλων αυτών και συνεπώς εκμετάλλευσης των πλεονεκτημάτων της κάθε μιας συνιστώσας, σε ένα σύστημα.

Αυτό που λείπει από τις υπάρχουσες εφαρμογές του Multihoming, είναι η δυνατότητα υποστήριξης περισσότερων από μια οντοτήτων, ίσως και ένα ολόκληρο τοπικό δίκτυο, με συναθροισμένη συνδεσιμότητα στο Internet

Τα 802.11 δίκτυα έχουν εξαπλωθεί αρκετά ανάμεσα στους χρήστες ηλεκτρονικών υπολογιστών, έτσι ώστε να είναι συνηθισμένο σε μια αστική περιοχή ένας χρήστης να δέχεται τα σήματα από τα ασύρματα δίκτυα διπλανών χρηστών.

Πιστεύουμε ότι ένας χρήστης είναι δυνατόν να εκμεταλλευθεί αυτήν την κατάσταση έτσι ώστε να βελτιώσει την συνδεσιμότητα του με το Internet, χωρίς επιπλέον κόστος, απλά συγκεντρώνοντας και δια-μοιράζοντας τις διάφορες ανεξάρτητες συνδέσεις με το internet ,την δικιά του και τον γειτόνων του.

Αυτό που θα προσπαθήσουμε να κάνουμε είναι να φέρουμε της βελτιώσεις του multihoming πιο κοντά στον τελικό χρήστη. Επειδή δεν είναι πρακτικό να περιμένουμε ότι ο τελικός χρήστης θα έχει τον ακριβό εξοπλισμό και ότι θα πληρώνει παραπάνω από 1 ISPs για να κάνει μόνος του multihoming , θα εκμεταλλευτούμε την διείσδυση τον ασύρματων δικτύων και θα κινηθούμε στην κατεύθυνση του Multihomed Collaborative Internet Access. Δηλαδή 2 οι παραπάνω χρήστες

θέλουμε να μπορούν να μοιραστούν την σύνδεση τους έτσι ώστε ο καθένας να έχει απόδοση παρεμφερή με το να πλήρωνε και να χρησιμοποιούσε παράλληλα παραπάνω από 1 σύνδεση.

Σε αυτήν τη διπλωματική παρουσιάζουμε τον σχεδιασμό , την υλοποίηση και την δοκιμή του **py-p2pinet** ενός συστήματος που προσπαθεί να βοηθήσει τους χρήστες να μοιραστούν την σύνδεση τους με γειτονικούς χρήστες που επιθυμούν και αυτοί τον διαμοιρασμό τις σύνδεσης τους.

1.5 Δομή Διπλωματικής

Το κείμενο είναι οργανωμένο ως εξής:

Κεφάλαιο 1 – Εισαγωγή

Γίνεται μια εισαγωγή στο θέμα της διπλωματικής

Κεφάλαιο 2 – Ανάλυση του συστήματος PERM

Πριν προχωρήσουμε στην περιγραφή του δικού μας συστήματος, παρουσιάζουμε το PERM την πιο γνωστή δουλεία, που έχει γίνει σε αυτόν το χώρο. Σε αυτό το κεφάλαιο θα δούμε τις επιλογές που κάνανε οι σχεδιαστές του PERM.

Κεφάλαιο 3 – py-p2pinet

Εδώ θα παρουσιάσουμε το δικό μας σύστημα

Κεφάλαιο 4 – Messaging

Σε αυτό το κεφάλαιο θα εξετάσουμε το πρωτόκολλο AMQP και γενικότερα έννοιες που άπτονται των Message-oriented middleware (MOM) συστημάτων.

Κεφάλαιο 5 – HTTP

Εδώ θα εξετάσουμε το HTTP πρωτόκολλο

Κεφάλαιο 6 – Proxy

Εδώ θα εξετάσουμε την υλοποίηση του Proxy.

Κεφάλαιο 7 – Βιβλιογραφία

2 PERM

Πριν προχωρήσουμε στην περιγραφή του δικού μας συστήματος, θα παρουσιάσουμε το PERM την πιο γνωστή δουλεία, που έχει γίνει σε αυτόν το χώρο και θα δούμε τις δικές τους σχεδιαστικές επιλογές.

Πριν τρία χρόνια, στο Πανεπιστήμιο του Illinois, δημιουργήθηκε το σύστημα PERM, ή Practical End-host Residential Multihoming. Η δουλεία αυτή περιγράφεται αναλυτικά σε 2 papers τα

- Flow Scheduling for End-host Multihoming
- PERM: A Collaborative System for Residential Internet Access

Πρόκειται για μια προσπάθεια εφαρμογής του Multihoming σε πραγματικά δεδομένα, σε επίπεδο οικιακής χρήσης, για αυτό άλλωστε και ο χαρακτηρισμός practical και residential του τίτλου. Συνοπτικά, πρόκειται για ένα σύστημα (οι δημιουργοί του το αναφέρουν σαν framework), το οποίο παρεμβάλλεται στο επίπεδο δικτύου ενός υπολογιστικού συστήματος, και με κατάλληλο λογισμικό, εφαρμόζει δρομολόγηση της μεταφοράς δεδομένων προς το Internet ανάμεσα στους διαθέσιμους συνδέσμους.

Το PERM επίσης εφαρμόζει ανάλυση σε πραγματικό χρόνο της διαδικτυακής συμπεριφοράς(networking behaviour) του χρήστη, ώστε με τεχνικές πρόβλεψης να βελτιώσει την απόδοση της δρομολόγησης (scheduling). Μια βασική προϋπόθεση για το PERM, την οποία οι δημιουργοί έλαβαν υπόψη τους εξ'αρχής και διατήρησαν μέχρι τέλους, ήταν η διατήρηση όλου του σχεδιασμού του συστήματος αποκλειστικά στην πλευρά του χρήστη, και όχι σε εξωτερικούς παράγοντες υποστήριξης, όπως υποστήριξη από τον παροχέα Υπηρεσίας Internet (εξ'ού και ο χαρακτηρισμός end-host Multihoming). Αυτό συνέβη γιατί θεωρήθηκε μη ρεαλιστική η ανάγκη εξωτερικής υποστήριξης, μιας και αυτό δεν είναι δυνατό να εξασφαλιστεί από όλους τους απομακρυσμένους προορισμούς στο Internet.

Κατά συνέπεια, το PERM έχει τα εξής χαρακτηριστικά. Αρχικά, η δρομολόγηση της κυκλοφορίας του Internet γίνεται στο επίπεδο ροών[5] του συστήματος. Επιπλέον, μια τέτοια ροή δεν έχει τη δυνατότητα να αναδρομολογηθεί, κάτι ούτως ή άλλως πρακτικά σχεδόν αδύνατο στις υπάρχουσες δικτυακές τεχνολογίες. Το PERM επιτυγχάνει αποδοτική δρομολόγηση διαφορετικών μεταξύ τους

[5] Ως ροή αναγωρίζεται το Σεύγος πηγή - απομακρυσμένος προορισμός, όπου η πηγή είναι φυσικά ο χρήστης του PERM.

ροών, όσον αφορά στον όγκο δεδομένων και προορισμών τους. Επίσης, είναι σημαντικό το γεγονός ότι η δρομολόγηση που εφαρμόζει το PERM, γίνεται λαμβάνοντας υπ'όψιν και εκτιμήσεις που βασίζονται στην ισχυρή χρονική συσχέτιση των ροών μεταξύ τους, καθώς και την μακροπρόθεσμη εξάρτηση του όγκου των δεδομένων της ροής, όταν αυτή γίνεται με έναν συγκεκριμένο προορισμό.

Για την ανάλυση αυτών των δυο παρατηρήσεων, οι δημιουργοί του PERM εγκατέστησαν και χρησιμοποίησαν το σύστημα στις κατοικίες της πανεπιστημιούπολης του Dartmouth College, για τέσσερις μήνες σε διάρκεια. Το πείραμα συμπεριέλαβε συνολικά 1334 διαφορετικά υπολογιστικά συστήματα, και η διακίνηση δεδομένων έφτασε τα 8,6 gigabytes. Κατά την ανάλυση επαληθεύτηκε η θεωρία τους πάνω στην χρονική συσχέτιση και εξάρτηση όγκου των ροών. Πάνω σε αυτές τις παρατηρήσεις χτίστηκαν αλγόριθμοι πρόβλεψης οι οποίοι χρησιμοποιούνται από το PERM για εκτίμηση της καλύτερης δρομολόγησης.

2.1 Περιγραφή του Συστήματος

Η λειτουργικότητα του PERM διαχωρίζεται σε έξι λειτουργικές μονάδες, οι οποίες συνεργατικά στοχεύουν στην επιτυχή ανίχνευση άφιξης νέων ροών στο σύστημα, δρομολόγησής τους προς μια εκ των διαθέσιμων συνδέσεων, διατήρησης στατιστικών στοιχείων των συνδέσμων, απαραίτητων για τους αλγορίθμους πρόβλεψης και τελικά στην καταλληλότερη επιλογή συνδέσμου για χρήση. Οι λειτουργικές μονάδες του PERM είναι

- ένας Διαχειριστής Συνδέσεων
- η Μονάδα Παρακολούθησης Συστήματος
- η Μονάδα Πρόβλεψης
- ο Διαχειριστής Κινήτρου
- ο Διαχειριστής Ασφαλείας
- η Μονάδα Δρομολόγησης.

2.1.1 Διαχειριστής Συνδέσεων - Connection Manager

Οι νέες ροές που δημιουργούνται από το χρήστη του PERM στον υπολογιστή του, για παράδειγμα η αίτηση για άνοιγμα μιας ιστοσελίδας, η αποθήκευση ενός αρχείου από το Internet, η χρήση ενός P2P συστήματος, πρέπει να ανιχνεύονται έγκαιρα από το PERM και να γίνεται η δρομολόγησή τους προς έναν εκ των διαθέσιμων συνδέσμων. Ο ΔΣ παρεμβάλλεται ανάμεσα στο επίπεδο δικτύου και το Socket API του συστήματος, και ενεργοποιεί τη δρομολόγηση των ροών προς τους συνδέσμους.

Επίσης, ο ΔΣ δεν διαχειρίζεται εξωτερικά προερχόμενες ροές, μιας και η πλειοψηφία των ροών σε έναν οικιακό υπολογιστή ξεκινούν από το χρήστη (από το ίδιο το υπολογιστικό σύστημα).

2.1.2 Μονάδα Παρακολούθησης - Monitor Component

Για κάθε διαθέσιμο σύνδεσμο με το Internet το σύστημα διατηρεί ένα σύνολο από στοιχεία, τα οποία χρησιμοποιούνται για σωστή δρομολόγηση. Αυτά περιλαμβάνουν αποτυχίες και σφάλματα των συνδέσμων, εκτιμήσεις για τη χωρητικότητα των συνδέσμων, και χρονική καθυστέρηση προς τους απομακρυσμένους προορισμούς. Για τη μέτρηση χρονικής καθυστέρησης χρησιμοποιείται active probing (δηλαδή ενεργητική μέτρηση), ενώ για τη διατήρηση στοιχείων για την κατάσταση των συνδέσμων passive monitoring (ή παθητική παρακολούθηση).

2.1.3 Μονάδα Πρόβλεψης - Prediction Component

Η δρομολόγηση ροών προς συνδέσμους στο PERM δουλεύει με τρόπο που η γνώση εκ των προτέρων του χρόνου round trip[6] και του αναμενόμενου όγκου δεδομένων της ροής είναι στοιχεία απαραίτητα. Πρακτικά τέτοια στοιχεία δεν είναι διαθέσιμα πριν την εγκαθίδρυση μιας σύνδεσης με έναν απομακρυσμένο προορισμό, ή έστω γίνονται διαθέσιμα πολύ αργά πλέον για να λάβει χώρα η δρομολόγηση των ροών. Η ανάλυση που περιγράφηκε πριν και έγινε στο Dartmouth College, έδειξε ότι και οι απομακρυσμένοι προορισμοί αλλά και οι όγκοι δεδομένων των ροών είναι προβλέψιμοι. Η δρομολόγηση στο PERM αξιοποιεί τα μοντέλα πρόβλεψης που έχουν αναπτυχθεί και υλοποιούνται στη ΜΠ, ώστε να εκτιμηθούν οι όγκοι δεδομένων των ροών και η πρόβλεψη των πιθανών απομακρυσμένων προορισμών που θα ακολουθήσουν, ώστε να μετρηθεί εκ των προτέρων ο χρόνος round trip.

2.1.4 Διαχειριστής Κινήτρου - Incentive Manager

Για να υπάρχει αίσθημα δικαιοσύνης ανάμεσα στους χρήστες που κοινοποιούν τις συνδέσεις τους, μιας και το PERM στοχεύει σε περιπτώσεις που δυο χρήστες μοιράζονται ο κάθε ένας τη δική του σύνδεση με το Internet, εισάγεται η έννοια του κινήτρου (incentive). Η πολιτική κινήτρου και η εφαρμογή της είναι οι αρμοδιότητες του ΔΚ. Ο ΔΚ παρακολουθεί τη χρήση των διαθέσιμων συνδέσμων με το Internet, συνεργάζεται με τους ΔΚ των υπόλοιπων συμμετεχόντων συστημάτων και παρέχει πληροφορίες δρομολόγησης στη Μονάδα Δρομολόγησης.

[6] Ο συνολικός χρόνος που απαιτείται για ένα πακέτο δεδομένων να ταξιδέψει από μια πηγή, σε έναν προορισμό και πάλι πίσω.

2.1.5 Διαχειριστής Ασφαλείας - Security Manager

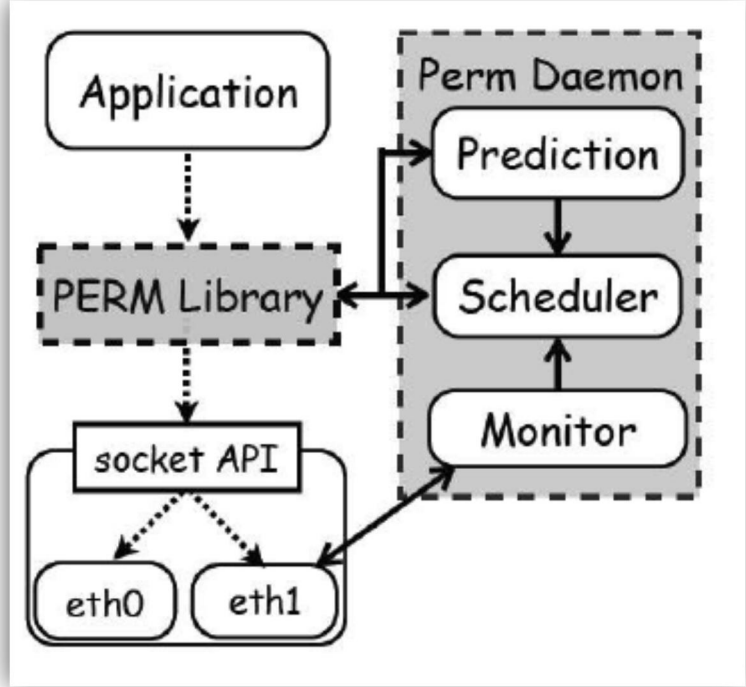
Οταν ένας χρήστης προσπελαύνει το Internet μέσω της σύνδεσης ενός γείτονα, εκθέτει τα ιδιωτικά του δεδομένα, φανερώνει τον τρόπο με τον οποίο χρησιμοποιεί το Internet και γενικώς αφήνει κενά ασφαλείας στο σύστημά του. Ο ΔΑ ευθύνεται για την ενίσχυση της ασφάλειας κατά τη χρήση του PERM. Στόχοι είναι η προστασία ευαίσθητων δεδομένων από τη διαρροή τους από το προστατευμένο δίκτυο του PERM, καθώς και ο αποκλεισμός από άλλους χρήστες στο να έχουν πρόσβαση σε σημαντικές ιστοσελίδες (όπως τραπεζικοί λογαριασμοί) μέσω της τοπικής σύνδεσης. Επίσης παρέχεται η δυνατότητα απόκρυψης της συμπεριφοράς στο Internet του χρήστη.

2.1.6 Μονάδα Δρομολόγησης Ροών - Scheduler

Στο PERM υλοποιείται ένας Υβριδικός Δρομολογητής, ο οποίος δρομολογεί με βάση τον αναμενόμενο, σύμφωνα με τη Μονάδα Πρόβλεψης, όγκο δεδομένων της ροής. Κατά τη δρομολόγηση μιας TCP ροής, οι σημαντικοί παράγοντες είναι ο χρόνος round trip, ο ανταγωνισμός μεταξύ των ροών όσον αφορά τον όγκο δεδομένων τους, σε σημεία συμφόρησης (bottleneck links) και το εύρος ζώνης στα σημεία συμφόρησης. Στην περίπτωση του PERM, τα σημεία συμφόρησης είναι οι διαθέσιμοι σύνδεσμοι με το Internet. Ακόμη, το γεγονός ότι ο εκάστοτε χρήστης από τον οποίο ξεκινούν οι ροές ελέγχει την δικτυακή κυκλοφορία στη δική του ευρυζωνική σύνδεση, σε συνδυασμό με τη μέτρηση του χρόνου round trip και την εκτίμηση των όγκων δεδομένων των ροών που εκτελεί η ΜΠ, δίνει τη δυνατότητα για μια αξιόπιστη πρόβλεψη του συνολικού χρόνου μετάδοσης μιας ροής. Σύμφωνα με την πρόβλεψη αυτή, το PERM προσαρμόζεται ανάμεσα σε δρομολογήσεις με βάση δυο διαφορετικά κριτήρια - απ' τη μία, δρομολόγηση που βασίζεται στην καθυστέρηση πάνω στους συνδέσμους, για την περίπτωση ροών με μικρό όγκο δεδομένων, και από την άλλη, δρομολόγηση με βάση τον εκτιμώμενο χρόνο μετάδοσης (για επίτευξη ισοκατανομής του φόρτου στους συνδέσμους) για ροές με μεγάλο όγκο δεδομένων[7].

Η λειτουργία του PERM ολοκληρώνεται με τη συνεργασία των λειτουργικών μονάδων του. Ο τρόπος αλληλεπίδρασης των μονάδων του φαίνεται και σχηματικά στην παρακάτω εικόνα.

[7] Το όριο ανάμεσα στον μικρό και το μεγάλο όγκο δεδομένων που θεωρείται στο PERM είναι τα 8 kilobytes.



2.2 Μοντελοποίηση Δικτυακής Κυκλοφορίας

Η αμερόληπτη, υψηλής απόδοσης δρομολόγηση που επιτυγχάνει ο δρομολογητής ροών του PERM, οφείλεται στην ανάλυση της δικτυακή κυκλοφορίας στην πλευρά του χρήστη και στην εξαγωγή πολύτιμων πληροφοριών για αυτή. Η επιδόσεις των ροών με μικρό όγκο δεδομένων κυριαρχούνται από το χρόνο round trip, και έτσι οι ροές αυτές δρομολογούνται στο σύνδεσμο με το μικρότερο χρόνο round trip κάθε φορά. Το PERM διατηρεί αυτό το χρόνο, παίρνοντας εκ των προτέρων μετρήσεις για το χρόνο αυτό, προς τους προορισμούς που προβλέπεται ότι θα ακολουθήσουν. Αυτή η εκ των προτέρων μέτρηση εξαλείφει την καθυστέρηση που θα δημιουργούνταν μετά την δημιουργία μιας ροής, αλλά πριν γίνει η δρομολόγησή της. Για τις ροές με μεγάλο όγκο δεδομένων, των οποίων οι επιδόσεις οριοθετούνται από το διαθέσιμο εύρος ζώνης, το PERM δρομολογεί τις ροές βάσει του εκτιμώμενου όγκου τους και τον τρέχον δικτυακό φόρτο σε κάθε σύνδεσμο.

Για να γίνει επιτυχώς αυτή η κλιμακούμενη a priori μέτρηση, και κατά συνέπεια και η υβριδική δρομολόγηση ροών, πρέπει να απαντηθούν δύο ερωτήματα :

1. Ποιό είναι το σύνολο των διευθύνσεων των απομακρυσμένων προορισμών με τους οποίους πρόκειται να επικοινωνήσει ο χρήστης μέσα στο επόμενο χρονικό διάστημα·

2. Ποιά είναι η κατανομή του όγκου δεδομένων της κυκλοφορίας που αντιστοιχεί σε έναν απομακρυσμένο προορισμό.

2.2.1 Διευθύνσεις IP των Απομακρυσμένων Προορισμών.

Τα αποτελέσματα που συντελούν στην απάντηση του ερωτήματος που τέθηκε, προήλθαν από τη μελέτη(μετρήσεις) στο Dartmouth College.

Στο σχήμα που ακολουθεί μπορούμε να δούμε το πλήθος των διαφορετικών διευθύνσεων τις οποίες επισκέπτεται ένας χρήστης, σε διάστημα μιας ημέρας.

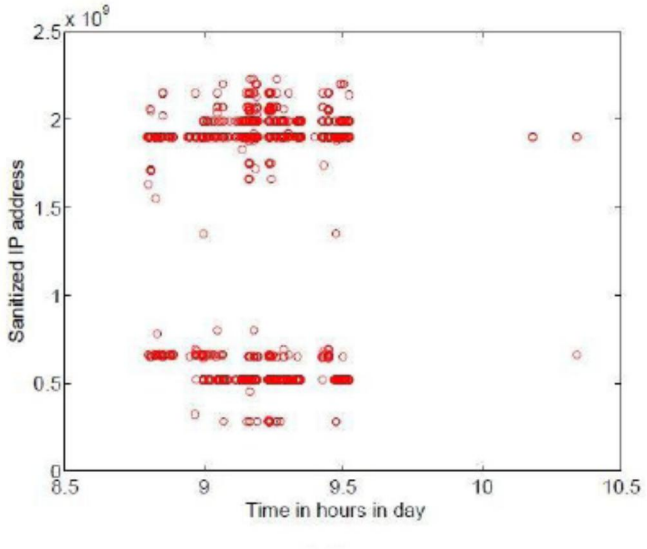


Figure 1 Διευθύνσεις IP μέσα σε μια μέρα

Μια παρατήρηση που κάνουμε, είναι ότι αυτές οι γραμμές είναι διασπασμένες σε δυο ομάδες, ένα πρότυπο το οποίο συχνά καλείται bi-modal, και συναντήθηκε σε πολλούς χρήστες κατά τη διάρκεια των πειραμάτων στο Dartmouth College.

Η εικόνα με τις διαφορετικές διευθύνσεις υποδεικνύει τη δυνατότητα πρόβλεψης σε πραγματικό χρόνο (online prediction) των IP διευθύνσεων τις οποίες ο χρήστης θα επισκευθεί με μεγαλύτερη πιθανότητα, με βάση την προσωρινή εμπειρία. Για την ακρίβεια, το σύστημα εκτελεί ένα συγκεκριμένο πλήθος από προβλέψεις, πράγμα που αυξάνει την πιθανότητα, η απομακρυσμένη IP διεύθυνση που θα ακολουθήσει να βρίσκεται μέσα στις προβλέψεις.

2.2.2 Όγκος δεδομένων των Ροών

Για την απάντηση του δεύτερου ερωτήματος που κάναμε πιο πάνω, δηλαδή της δυνατότητας πρόβλεψης της κατανομής του όγκου δεδομένων που αντιστοιχεί σε συγκεκριμένη απομακρυσμένη

IP διεύθυνση , οι δημιουργοί του PERM δείχνουν ότι με συγκεκριμένη τεχνική πρόβλεψης, το σφάλμα που προκύπτει είναι μέσα σε αποδεκτά όρια.

2.2.3 Υβριδική δρομολόγηση Ροών

Όλα τα προηγούμενα αποτελέσματα χρησιμοποιούνται στην πράξη από το PERM για την υλοποίηση της Υβριδικής δρομολόγησης Ροών (Hybrid Flow Scheduling), όπως το ονομάσανε οι δημιουργοί του PERM. Για το σχεδιασμό, λαμβάνονται υπ' όψιν μονάχα οι ροές που προκύπτουν από κυκλοφορία στον Παγκόσμιο Ιστό, μιας και έχει μετρηθεί ότι αποτελούν το 76% όλων των ροών στο διαδίκτυο.

2.3 Υλοποίηση του PERM

Το σύστημα PERM έχει σχεδιαστεί και υλοποιηθεί ως μια βιβλιοθήκη UNIX συστήματος, σε επίπεδο χρήστη. Η βιβλιοθήκη παρεμβάλλεται στις κλήσεις των εφαρμογών προς το Socket API[8] , και επικοινωνεί με την κύρια διεργασία του PERM, όπως φαίνεται και στο σχήμα στην ενότητα 2.1.6.

Το σύστημα διατηρεί πληροφορίες για τους απομακρυσμένους προορισμούς, όπως χρόνος round trip, συχνότητα επίσκεψης και όγκος δεδομένων, που χρειάζονται στον αλγόριθμο πρόβλεψης. Επίσης, διατηρούνται πληροφορίες για την κατάσταση των συνδέσμων, όπως η χωρητικότητα, η IP διεύθυνση και οι ροές που είναι ενεργές στο σύνδεσμο. Πέρα από τα πραγματικά δεδομένα, το PERM παράγει και προβλέψεις για τον αναμενόμενο όγκο των επερχόμενων ροών καθώς και για τους πιθανούς επόμενους προορισμούς. Η προμέτρηση του χρόνου round trip γίνεται περιοδικά, τόσο ώστε ούτε να επιβαρύνεται το σύστημα με το επιπλέον κόστος επικοινωνίας της μέτρησης (overhead) αλλά ούτε και οι τιμές που μετρώνται να είναι πεπαλαιωμένες.

Οι αποτυχίες των συνδέσμων εντοπίζονται παθητικά, ύστερα από τρεις συνεχόμενες και αποτυχημένες προσπάθειες σύνδεσης με απομακρυσμένους προορισμούς. Παρόλο που μπορεί να έχει εντοπιστεί αποτυχία σε ένα σύνδεσμο, δεν σταματούν οι προσπάθειες για προ- μέτρηση του χρόνου round trip σε αυτό το σύνδεσμο. Αν βρεθεί επιτυχής μέτρηση του χρόνου, ο σύνδεσμος θεωρείται ότι έχει επανέλθει.

Η λειτουργία του συστήματος ξεκινά με μια νέα αίτηση σύνδεσης από μια ροή. Τότε ο δρομολογητής πληροφορείται για τον εκτιμώμενο όγκο δεδομένων της ροής ανάλογα με τον προορισμό, και χαρακτηρίζει τη ροή ως μικρού ή μεγάλου όγκου. Εάν ένας προορισμός συναντάται πρώτη φορά και

[8] Application Programming Interface

δεν υπάρχει εκτίμηση για αυτόν, ακολουθείται στρατηγική χειρότερης περίπτωσης και επιλέγεται η ροή ως μεγάλου όγκου. Όσο δημιουργούνται ροές προς ένα προορισμό, η πρόβλεψη βελτιώνεται διαρκώς. Τελικά, όταν επιλεγεί ο σύνδεσμος στον οποίο η ροή θα δρομολογηθεί, γίνεται η σύνδεση των δυο, και ενώ η ροή δεδομένων λαμβάνει χώρα, διατηρούνται τα απαραίτητα στατιστικά στοιχεία. Κατά την ολοκλήρωση, γίνεται ενημέρωση των κατάλληλων μονάδων του συστήματος με τα στατιστικά και ανανεώνεται η εκτίμηση για τον προκείμενο απομακρυσμένο προορισμό.

2.4 Αξιολόγηση των επιδόσεων

Το σύστημα PERM αξιολογείται μέσα από τα δεδομένα που μετρώνται από τη λειτουργία του, σε αντιπαράθεση αφενός με την περίπτωση ύπαρξης μοναδικής ευρυζωνικής σύνδεσης, και αφετέρου με τρία διαφορετικά σχήματα δρομολόγησης που είναι δημοφιλή στο λεγόμενο enterprise Multihoming. Τα σχήματα αυτά είναι η Δρομολόγηση βάσει Κατακερματισμού (Hash based Scheduling), η Δρομολόγηση Εξισορρόπησης Δικτυακού Φόρτου (Load Balancing), και τέλος η μέθοδος Κυλιόμενου Παραθύρου (Sliding Window) με Active Probing. Οι ευρυζωνικές συνδέσεις που χρησιμοποιήθηκαν είναι μια DSL σύνδεση (κορύφωση στα 175KB ανά δευτερόλεπτο) και μια καλωδιακή σύνδεση (κορύφωση στα 500KB ανά δευτερόλεπτο. Κατά την εκτέλεση των πειραμάτων και των μετρήσεων, έγινε διαχωρισμός ανάμεσα στις δυο κατηγορίες ροών του PERM.

2.4.1 Ροές μικρού Όγκου δεδομένων

Όπως έχει ήδη αναφερθεί, η επίδοση των ροών μικρού όγκου εξαρτάται περισσότερο από το χρόνο round trip. Σε διάστημα μιας εβδομάδος έγιναν μετρήσεις του χρόνου round trip ανά πέντε λεπτά και για τις δυο συνδέσεις. Στο 44% των περιπτώσεων, η καλωδιακή σύνδεση είχε το στιγμιαίο χαμηλότερο χρόνο round trip, και η DSL για το υπόλοιπο 56%. Αναφορικά με τους προορισμούς, το 55% από αυτούς έδωσαν χαμηλότερο χρόνο round trip με την καλωδιακή, και το υπόλοιπο 45% με την DSL. Κατά μέσο όρο, για το 80% της διάρκειας των πειραμάτων, η σύνδεση που είχε την πλειοψηφία των χαμηλότερων στιγμιαίων μετρήσεων χρόνου round trip, ήταν και αυτή που είχε και συνολικά το χαμηλότερο χρόνο round trip. Αυτό οδηγεί στο συμπέρασμα ότι κατά ένα 20% η σύνδεση με την πλειοψηφία των χαμηλότερων στιγμιαίων μετρήσεων χρόνου round trip είναι η λάθος επιλογή.

Παρόλο που οι μετρήσεις έδειξαν ότι μακροπρόθεσμα ο χρόνος round trip είναι αρκετά σταθερός, η δυναμικότητα στις καθυστερήσεις είναι αρκετά υψηλή ώστε να απαιτείται διαρκής μέτρηση του χρόνου round trip. Για να φανεί το πλεονέκτημα του PERM, έγινε μια εξομοίωση πάνω στην δικτυακή

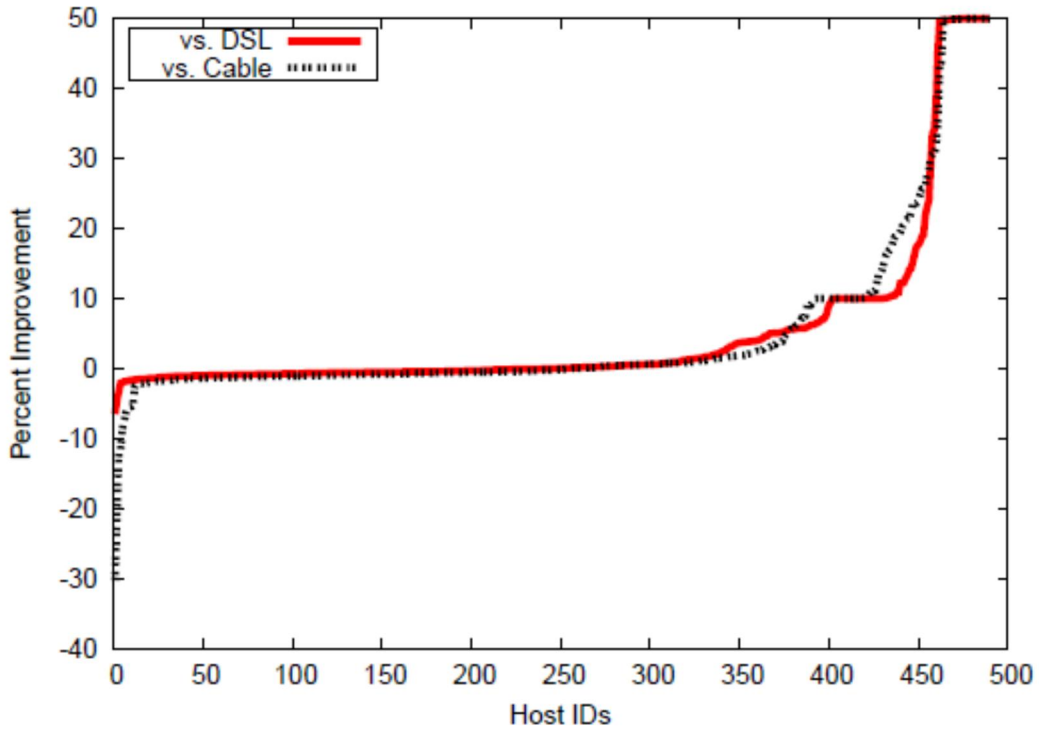
κυκλοφορία του πειράματος του Dartmouth της κλιμακούμενης προ- μέτρησης του χρόνου round trip του PERM και της τεχνικής Κυλιόμενου Παραθύρου με Active Probing. Για κάθε νέα εξομοιωμένη ροή, αν ο προορισμός ήταν στη λίστα με τις προβλέψεις, θεωρούνταν επιτυχία. Το PERM είχε ποσοστό επιτυχίας 53%, ενώ το Κυλιόμενο Παράθυρο μόλις 28%. Για καθαρά ροές HTTP, το PERM είχε ποσοστό επιτυχίας 92%, ενώ το Κυλιόμενο Παράθυρο 54%.

Αφού πλέον έχει δειχθεί η ακρίβεια της τεχνικής κλιμακούμενης προ-μέτρησης και πρόβλεψης του PERM, για να ποσοτικοποιηθεί και η ωφέλεια από την τεχνική αυτή, έγιναν μετρήσεις από μικρές μεταδόσεις δεδομένων από 150 δημοφιλείς ιστοσελίδες, με τρεις διαφορετικούς δρομολογητές - αυτόν του PERM, το Κυλιόμενο Παράθυρο, και το δρομολογητή Εξισορρόπησης Φόρτου. Τα αποτελέσματα φαίνονται στο παρακάτω σχήμα - το πλήθος απομακρυσμένων προορισμών για τους οποίους ο εκάστοτε δρομολογητής πέτυχε ελάχιστο χρόνο μετάδοσης, και το μέσο χρόνο μετάδοσης.

Scheduling scheme	# of hosts w/ min transmission time	Mean transmission time (sec)
PERM	147	2.790
Sliding Window	87	3.278
Load Balancing	80	3.052

Figure 2 Τα διαφορετικά σχήματα δρομολόγησης

Στο επόμενο σχήμα φαίνεται ο μέσος χρόνος μετάδοσης ανά προορισμό του δρομολογητή του PERM σε σχέση με την απλή DSL και καλωδιακή σύνδεση. Είναι χαρακτηριστικό ότι ο δρομολογητής του PERM έχει υψηλότερο μέσο χρόνο μετάδοσης μόνο για έναν έως δέκα προορισμούς. Επίσης, όταν οι προορισμοί ξεπερνούν τους 230 το PERM εμφανίζει όλο και μεγαλύτερη βελτίωση στο χρόνο.



2.4.2 Ροές μεγάλου Ογκου Δεδομένων.

Η κύρια επιρροή στις επιδόσεις ροών μεγάλου όγκου, είναι το διαθέσιμο εύρος ζώνης. Για να αξιολογηθεί η αποδοτικότητα του PERM για τις ροές με μεγάλο όγκο δεδομένων, εκτελέστηκαν ταυτόχρονες λήψεις αρχείων από απομακρυσμένους προορισμούς, των οποίων το μέγεθος κυμαίνεται από 2 έως 30 megabytes. Στην παρακάτω εικόνα βλέπουμε τους μέσους χρόνους μετάδοσης για κάθε σχήμα δρομολόγησης. Οι 2 πρώτοι τρόποι είναι υλοποιημένοι στο σύστημα PERM.

Scheduler	Mean Completion Time (sec)
Min-Total	186.21
Min-Max	199.14
Sliding Window	201.23
Load Balancing	211.37
Cable	257.21
Hash-Based	257.57
DSL	500.54

Και εδώ είναι εμφανής η υπεροχή των σχημάτων δρομολόγησης του PERM. Ο "min-total" δρομολογητής υπερέχει της δρομολόγησης Εξισορρόπησης Φόρτου κατά 11.9%, της δρομολόγησης

βάσει Κατακερματισμού των διευθύνσεων κατά 27.7% και της δρομολόγησης Κυλιόμενου Παραθύρου κατά 7.4%. Η χρήση μόνο της καλωδιακής και DSL σύνδεσης βελτιώθηκε κατά 28% και 62% αντίστοιχα.

Το επόμενο σχήμα δίνει μια καλύτερη άποψη στη σχέση ανάμεσα στην απόδοση όλων των σχημάτων δρομολόγησης, δείχνοντας τη χρησιμοποίηση (utilization)[9] της κάθε σύνδεσης κατά την εφαρμογή των διαφόρων σχημάτων.

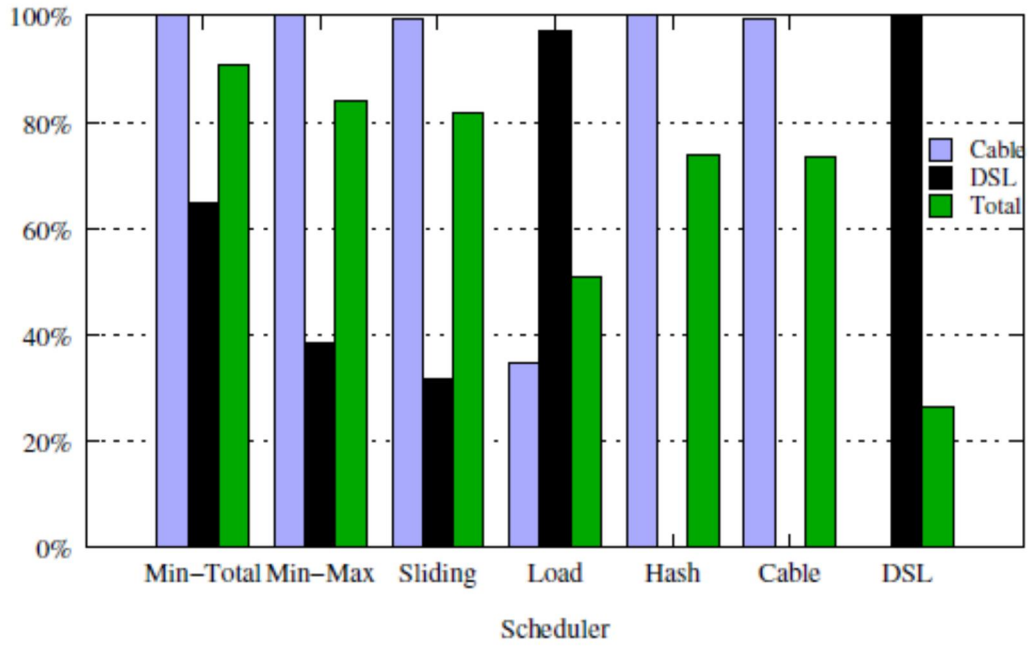


Figure 3 Χρησιμοποίηση της Καλωδιακής, DSL και συνολικής (Καλωδιακής+DSL) σύνδεσης με κάθε δρομολογητή.

Και εδώ, τα δυο σχήματα δρομολόγησης που επιστρατεύει το PERM πέτυχαν το μεγαλύτερο ποσοστό χρησιμοποίησης και στους δυο συνδέσμους ταυτόχρονα. Λόγω μεγαλύτερης χωρητικότητας σύνδεσης όμως, η καλωδιακή σύνδεση ωφελείται περισσότερο από το PERM. Στο σχήμα επίσης φαίνεται ότι η δρομολόγηση βάσει Κατακερματισμού αξιοποιεί αποκλειστικά την καλωδιακή σύνδεση, πράγμα που σημαίνει ότι δεν επιτυγχάνεται καλή εξισορρόπηση φόρτου όταν το πλήθος των συνδέσεων είναι μικρό και η δικτυακή κυκλοφορία αραιή. Ενδεχομένως κατά την επιλογή διαφορετικής συλλογής από προορισμούς, η δρομολόγηση βάσει Κατακερματισμού να επέλεγε την DSL σύνδεση μόνο, κάτι που θα κατέληγε σε ακόμη χειρότερη εξισορρόπηση του δικτυακού φόρτου.

[9] Το ποσοστό της χωρητικότητας του συνδέσμου που χρησιμοποιείται.

2.4.3 Ανθεκτικότητα.

Έχει σημασία να φανεί πως αντιδρά το σύστημα σε περιπτώσεις αποτυχίας ενός από τους διαθέσιμους συνδέσμους, μιας και η ανοχή σε σφάλματα είναι βασικός στόχος στο Multihoming. Χρησιμοποιώντας τις ίδιες προ-μετρήσεις (probes) με την ενότητα (Α. Ροές μικρού Όγκου δεδομένων), ο συνολικός ρυθμός αποτυχιών κατά την εκτέλεση των μετρήσεων ήταν 20.3% στην Καλωδιακή και 12.8% στην DSL. Από αυτές τις αποτυχίες, μόνον το 4% αυτών της Καλωδιακής και το 6% αυτών της DSL συνέβησαν ταυτόχρονα, οπότε και η συνδεσιμότητα του συστήματος κατέρρευσε εντελώς. Συνολικά, κατά το 94% των αποτυχιών αποφεύχθηκε χρησιμοποιώντας το σχήμα δρομολόγησης του PERM.

2.4.4 Επιβαρύνσεις

Το υβριδικό σχήμα δρομολόγησης του PERM επιφέρει επιπλέον καθυστερήσεις στη δημιουργία και τερματισμό μιας σύνδεσης. Πιο συγκεκριμένα, οι πιο αξιοσημείωτες καθυστερήσεις που παρατηρήθηκαν περιλαμβάνουν μια καθυστέρηση κατά 33% κατά την λήψη δεδομένων από μια σύνδεση και 16% κατά την αποστολή. Ο λόγος είναι η παρεμβολή της βιβλιοθήκης του PERM στο Socket API του συστήματος. Η καθυστέρηση που προστέθηκε στην ειδική κλήση συστήματος connect ήταν 1.40ms (ή 159%) ενώ στην κλήση συστήματος socket ήταν 0.050ms (ή 931%). Παρόλο που τα ποσοστά της καθυστέρησης που προστίθεται από το PERM είναι φαινομενικά μεγάλα, η συνολική χρονική διάρκεια είναι πολύ μικρή για να γίνει αντιληπτή από το χρήστη.

2.5 Συμπεράσματα

Συμπερασματικά, το PERM είναι ένα σύστημα που επιτυγχάνει αυτό για το οποίο έχει σχεδιαστεί - να υλοποιήσει δηλαδή στην πράξη μια αποδοτική λύση για το Multihoming, από την οπτική της δρομολόγησης των ροών στους διαθέσιμους συνδέσμους. Η ενίσχυση του δρομολογητή του PERM με τους μηχανισμούς πρόβλεψης που αναλύθηκαν, δίνει έναν ισχυρό συνδυασμό που βελτιώνει ακόμη περισσότερο την απόδοση της δρομολόγησης. Έτσι το PERM επιτυγχάνει συνολικά σχεδόν 15% μείωση του μέσου χρόνου μετάδοσης των μικρού όγκου ροών και μέχρι 27% για τις μεγάλου όγκου ροές, σε σύγκριση με σχήματα δρομολόγησης που χρησιμοποιούνται σε άλλες παρεμφερείς υλοποιήσεις του Multihoming. Ιδιαίτερα σε σχέση με την απλή καλωδιακή και DSL σύνδεση, για τις ροές μικρού όγκου η καθυστέρηση μετάδοσης μειώνεται μέχρι και 50%, ενώ ο μέσος χρόνος μετάδοσης για τις ροές μεγάλου όγκου μειώνεται κατά 28% και 62% αντίστοιχα για την κάθε σύνδεση.

Ενώ οι επιδόσεις της δρομολόγησης του PERM είναι εμφανείς, υπάρχουν και κάποια θέματα που δεν έχουν ακόμη υλοποιηθεί από τους δημιουργούς, και βρίσκονται υπό συζήτηση. Η χρήση του PERM, προϋποθέτει τη θέληση για διαμοιρασμό της σύνδεσης ενός χρήστη, διότι κανείς χρήστης δεν έχει στην ιδιοκτησία του δυο διαφορετικές συνδέσεις, και βασίζεται στο διαμοιρασμό με κάποιο γείτονα. Για να υπάρξει όμως αυτή η συνεργασία, θα πρέπει να υπάρχει και κάποια μορφή κινήτρου. Αυτό το ρόλο ακριβώς παίζει ο διαχειριστής κινήτρου που έχει περιγραφεί. Ένα τέτοιο σύστημα βρίσκεται υπό υλοποίηση από τους δημιουργούς του PERM, μιας και η χρήση του κρίνεται αναγκαία σε αυτή την περίπτωση.

Ένα ακόμη θέμα υπό έρευνα είναι το θέμα της ασφάλειας. Πάνω σε αυτό, ερευνώνται μέθοδοι αύξησης της ασφάλειας σύνδεσης χρησιμοποιώντας φιλτράρισμα των διευθύνσεων MAC[10] και έλεγχος στη σύνδεση σε τοπικό ασύρματο δίκτυο με χρήση 802.11i (WPA) κλειδιών. Για την απόκρυψη πληροφοριών της δικτυακής κυκλοφορίας του κάθε χρήστη, μιας και εδώ τίθεται θέμα ιδιωτικού απορρήτου, συζητείται η υλοποίηση μιας μεθόδου δημιουργίας εικονικής κυκλοφορίας, η οποία ουσιαστικά θα αποκρύπτει από τον έξω κόσμο την καθαρή, πραγματική κυκλοφορία του χρήστη.

Η ανάλυση που έγινε για τη δυνατότητα πρόβλεψης στη δρομολόγηση του PERM έδειξε ότι η κυκλοφορία στον παγκόσμιο ιστό και στο πρωτόκολλο του e-mail είναι προβλέψιμη. Άλλα είδη κυκλοφορίας στο διαδίκτυο δεν είναι προβλέψιμα, αλλά μάλλον τυχαία, όπως κυκλοφορία σε P2P και Voice Over IP (VoIP) συστήματα. Αυτά τα είδη δικτυακής κυκλοφορίας βρίσκονται υπό μελέτη ώστε να βρεθούν τρόποι πρόβλεψης αλλά και δρομολόγησης που να είναι αποδοτικοί.

Το Σύστημα PERM αποτελεί μια υψηλού επιπέδου προσπάθεια στα πλαίσια του Multihoming, η οποία εφαρμόστηκε στην πράξη και τα αποτελέσματα δείχνουν πολλά πράγματα. Το γεγονός ότι κάποιες πτυχές δεν έχουν ερευνηθεί εις βάθος ακόμη δεν μπορεί να χαρακτηρίσει το PERM ως ελλιπές, μιας και η ουσιαστική αξία του Multihoming αναδείχθηκε επιτυχώς και δημιουργήθηκε ένα σύστημα που στην πράξη και στην καθημερινότητα μπορεί να εφαρμοστεί εύκολα. Καθώς η τεχνολογία των δικτύων και τον υπολογιστικών συστημάτων προχωρά, ενδεχομένως το Multihoming να γίνει κατεστημένο στη δικτυακή καθημερινότητα, και σίγουρα το PERM είναι ένα βήμα προς αυτή την κατεύθυνση.

[10] Medium Access Control

3 py-p2pinet

Εδώ θα παρουσιάσουμε το py-p2pinet, το σύστημα-framework που υλοποιήθηκε στα πλαίσια αυτής της διπλωματικής εργασίας.

Γίνεται αναφορά στα κίνητρα που ώθησαν στην υλοποίηση ενός τέτοιου συστήματος, καθώς και στους στόχους από την υλοποίηση. Ακολουθεί μια πλήρης περιγραφή του συστήματος και της λειτουργίας του, και τέλος μια αναλυτική παρουσίαση των πειραμάτων που έγιναν για την διερεύνηση της λειτουργίας του από την άποψη της ορθότητας, της ευελιξίας και της απόδοσης. Τέλος, γίνεται μια συζήτηση για τα συμπεράσματα, τις προοπτικές για εξέλιξη του συγκεκριμένου έργου, καθώς και τις δυνατότητες για περαιτέρω μελέτη και έρευνα πάνω στο θέμα.

Το project είναι μια καινούρια και διαφορετική υλοποίηση, ενός συστήματος με την ονομασία p2pinet, το οποίο είχε υλοποιηθεί στα πλαίσια παλαιότερης διπλωματικής. Έτσι κάποιες στιγμές θα τονίσουμε τις διαφορετικές προσεγγίσεις που ακολουθήθηκαν.

3.1 Εισαγωγή

Αυτό που λείπει από τις υπάρχουσες λύσεις και εφαρμογές του Multihoming είναι η δυνατότητα υποστήριξης περισσότερων από ενός οντοτήτων, έως ίσως και ένα ολόκληρο τοπικό δίκτυο, με συναθροισμένους συνδέσμους προς το Internet, όπως για παράδειγμα στην υποθετική περίπτωση μιας μικρής κοινωνίας διασκορπισμένων χρηστών, όπου μόνον μερικοί από αυτούς έχουν διαθέσιμη ευρυζωνική πρόσβαση.

Ακόμη και σε μια τέτοια περίπτωση, το Multihoming μπορεί να εφαρμοστεί με τρόπο τέτοιο ώστε η αποδοτικότητα να επαυξηθεί. Στην παρούσα εργασία γίνεται μια προσπάθεια εφαρμογής αυτής της ιδέας με την ανάπτυξη ενός συστήματος που προσφέρει διαμοιρασμό των πολλαπλών διαθέσιμων συνδέσεων προς το Internet[11] σε ένα οργανωμένο δίκτυο οντοτήτων που θα τις αξιοποιούν. Στόχος μας είναι η βελτίωση της διαδικτυακής πρόσβασης και απόδοσης των οντοτήτων που συνδέονται στο σύστημα.

Η **απόδοση** επιδιώκεται από την οπτική γωνία του ρυθμού μεταφοράς δεδομένων (όπως το εύρος ζώνης), της συνδεσιμότητας και της αξιοπιστίας της, στην περίπτωση όπου ένας διαθέσιμος σύνδεσμος αποτυγχάνει, οπότε και γίνεται αδιαφανής μετάβαση σε κάποιον άλλο ενεργό.

[11] Η και γενικότερα προς μια πύλη (gateway) ενδιαφέροντος

Επιπλέον, ενδιαφέρον υπάρχει προς την κατεύθυνση της οργάνωσης των διαθέσιμων συνδέσμων με χρήση σχημάτων δρομολόγησης των μεταβιβάσεων δεδομένων, καθώς και ο αντίκτυπος της επιλογής του εκάστοτε σχήματος.

Τέλος, στόχος είναι η χρήση του συστήματος τόσο σε πλασματικές συνθήκες για επιβεβαίωση της λειτουργίας, όσο και σε πραγματικές συνθήκες για ποσοτικοποίηση του κέρδους μιας τέτοιας προσέγγισης.

Για την υλοποίηση του συστήματος χρησιμοποιήθηκε ο RabbitMQ broker και η γλώσσα Python. Το σύστημα ονομάστηκε py-p2pinet, από το συνδυασμό των P2P δικτύων με το Internet και το py για την επιλογή της γλώσσας υλοποίησης.

3.2 Ορολογία

3.2.1 Οντότητα

Σαν οντότητα περιγράφεται οποιοδήποτε υπολογιστικό σύστημα που έχει τη δυνατότητα διασύνδεσης μέσω ενός ενσύρματου ή ασύρματου interface, όπως Υπολογιστές, Κινητά Τηλέφωνα, PDAs, αισθητήρες, κ.ά.

3.2.2 Ροή

Για την περιγραφή της επικοινωνίας μεταξύ των κόμβων, αλλά και ενός κόμβου με το Internet χρησιμοποιείται η έννοια της ροή. Η ροή αποτελεί ουσιαστικά μια ανταλλαγή δεδομένων και θα έχει πάντοτε έναν κόμβο προέλευσης και έναν προορισμό. Δηλαδή ροή θα είναι η διαδικασία κατά την οποία ένας κόμβος κάνει μια αίτηση για δεδομένα προς έναν προορισμό, και ο προορισμός απαντά με τα δεδομένα.

Ακόμα ορίζουμε σαν **εξωτερική ροή**, μια ροή της οποίας η προέλευση είναι ένας κόμβος του P2P δικτύου, και προορισμός είναι κάποιος απομακρυσμένος server του Διαδικτύου.

Έτσι, αυτές οι ροές θα εκτελούνται μόνον από κόμβους που διαθέτουν σύνδεση με το Internet. **Εσωτερικές ροές** θα καλούνται οι ροές των οποίων και η προέλευση και ο προορισμός είναι κάποιοι δυο κόμβοι του P2P δικτύου.

Όταν ένας κόμβος επιθυμεί την ανταλλαγή δεδομένων με έναν απομακρυσμένο προορισμό του διαδικτύου, όχι μέσω της δικιά του γραμμής, αλλά μέσω της γραμμής ενός άλλου κόμβου, θα

αναγκάζεται να εγκαθιστά μια εσωτερική ροή δεδομένων προς τον άλλο κόμβο, ο οποίος με τη σειρά του θα εγκαθιστά μια εξωτερική ροή για να εξυπηρετήσει την αίτηση του πρώτου.

Ουσιαστικά λοιπόν μια **εξερχόμενη ροή** ίσως χρειαστεί να διασπαστεί σε μια εσωτερική και μια εξωτερική ροή, ίσως όμως να ταυτίζεται μερικές φορές με την εξωτερική ροή.

Στη συνέχεια θα γίνει πιο σαφής ο τρόπος με τον οποίο οι κόμβοι θα διαμοιράζονται τη συνδεσιμότητά τους και το πώς θα γίνεται η επικοινωνία ανάμεσά τους.

3.3 Σενάρια Χρήσης (use cases)

Πριν προχωρήσουμε, θα παρουσιάσουμε μερικά σενάρια χρήσης, που χρησιμοποιήσαμε για να συγκεντρώσουμε τα επιθυμητά specifications του συστήματός μας.

Όπως αναφέρουμε και στην εισαγωγή βασικός στόχος είναι ο διαμοιρασμός των πολλαπλών διαθέσιμων συνδέσεων προς το Internet[12] σε ένα οργανωμένο δίκτυο οντοτήτων που θα τις αξιοποιούν.

Το πρώτο σενάριο είναι αυτό όπου 2 ή περισσότεροι γείτονες θέλουν να μοιραστούν τις συνδέσεις τους στο Internet έτσι ώστε να απολαμβάνουν όλοι καλύτερο last-mile connectivity.

Άλλο ένα σενάριο είναι η ύπαρξη ενός ασύρματου μητροπολιτικού δικτύου σε μια πόλη και πως γίνεται οι χρήστες αυτού του δικτύου να το χρησιμοποιήσουν σε συνδυασμό με την δικιά τους σύνδεση έτσι ώστε πάλι να απολαμβάνουν καλύτερο last-mile[13] connectivity.

Τελευταίο σενάριο είναι η ανάγκη που προκύπτει πολλές φορές να δώσουμε Internet σε πολλούς χρήστες χωρίς να μπορούμε να χρησιμοποιήσουμε καλωδίωση ή κάποια περίτεχνη αρχιτεκτονική για ασύρματο δίκτυο Αυτό μπορεί να συμβεί σε περιπτώσεις όπου έχουμε περιορισμό στο κόστος του εξοπλισμού ή μετά από φυσικές καταστροφές.

Σε αυτήν την περίπτωση θα στήναμε 2-3 "υπέρ-χρήστες" που θα είχαν συνδεσιμότητα στο Internet και μετά όλοι οι υπόλοιποι θα μπορούσαν να πάρουν Internet από αυτούς τους υπέρ-χρήστες, και αυτό θα γινόταν χωρίς να χρειαζόμαστε ακριβές κεραίες για να καλύψουμε μια πολύ μεγάλη περιοχή.

[12] Η και γενικότερα προς μια πύλη (gateway) ενδιαφέροντος
[13] http://en.wikipedia.org/wiki/Last_mile

3.4 Προδιαγραφές και σχεδιασμός του συστήματος

Από τα παραπάνω σενάρια χρήσης μπορούμε να αντλήσουμε και τις προδιαγραφές του συστήματος μας.

- Το σύστημα μας αποτελείται από κόμβους-οντότητες.
- Αυτοί οι κόμβοι, αφού αναφέρονται σε χρήστες, μπορεί να μπαίνουν και να φεύγουν τυχαία, με περιοδικό ή με μη περιοδικό τρόπο.
- Κάποιοι από αυτούς τους κόμβους έχουν συνδεσιμότητα με το Internet, όχι απαραίτητα όλοι.
- Οι κόμβοι είναι ετερογενείς σε υπολογιστικές δυνατότητες καθώς και στην σύνδεση που έχουν με το Internet. Παράλληλα δεν είναι απαραίτητο να έχουν την ίδια διάθεση να μοιραστούν τους πόρους τους (είτε από υπολογιστική πλευρά είτε την σύνδεση τους).
- Το σύστημα μας αποσκοπεί στον διαμοιρασμό ενός πόρου. Ο πόρος που διαμοιράζεται είναι το εύρος ζώνης κάποιων οντοτήτων. Δεν είναι απαραίτητο όλες οι οντότητες να έχουν συνδεσιμότητα με το διαδίκτυο.
- Η κίνηση που θα διαχειρίζεται το σύστημα μας, αποτελείται από ροές με διαφορετικά χαρακτηριστικά.
- Ο σκοπός μας είναι να βελτιώσουμε την χρήση του συνολικού bandwidth να μειώσουμε το κόστος να βελτιώσουμε την συνδεσιμότητα που έχει κάθε χρήστης.

Δεδομένων των παραπάνω προδιαγραφών έχουμε αρκετές επιλογές στη φάση του σχεδιασμού.

Πρώτα χρειάστηκε να αποφασίσουμε αν η υλοποίηση θα γινότανε σε επίπεδο λειτουργικού συστήματος ή σε επίπεδο εφαρμογής. Επιλέξαμε να υλοποιήσουμε σε επίπεδο εφαρμογής για 2 βασικούς λόγους. Ο βασικότερος λόγος είναι ότι θέλαμε το σύστημα να είναι cross-platform για να δοκιμαστεί σε πραγματικές συνθήκες στο Patras Wireless Metropolitan Network[14]. Ο δεύτερος λόγος είναι ότι θέλαμε το σύστημα να μπορεί να εγκατασταθεί εύκολα και γρήγορα από απλούς χρήστες χωρίς να χρειάζονται πολλά dependencies. Από την ανάπτυξη του PERM είδαμε ότι κάτι τέτοιο δεν είναι ιδιαίτερα εύκολο σε επίπεδο συστήματος. Άλλος ένας λόγος για να θέλει κανείς ένα τέτοιο σύστημα να μπορεί να εγκατασταθεί γρήγορα και εύκολα είναι ο εξής. Η όλη επιτυχία του συστήματος στηρίζεται στο να υπάρχουν αρκετοί χρήστες. Αν για να τρέξει το σύστημα ο χρήστης αρκεί να τρέξει ένα πρόγραμμα και να ρυθμίσει το σύστημα του ώστε να χρησιμοποιεί τον proxy

[14] http://en.wikipedia.org/wiki/Patras_wireless_metropolitan_network

server του συστήματος μας έχει πολλές πιθανότητες να το κάνει. Στην αντίπερα όχθη είχαμε το PERM το οποίο ζητούσε από τον χρήστη να εγκαταστήσει ένα ενδιάμεσο driver που έμπαινε ανάμεσα στον πυρήνα και το socket interface ενώ παράλληλα έπρεπε να εγκαταστήσει και τροποποιημένο firmware στον router. Και φυσικά ο router του έπρεπε να είναι ο συγκεκριμένος για τον οποίο είχε γραφτεί το "πειραγμένο" firmware.

Βέβαια η επιλογή που κάναμε δεν έρχεται χωρίς κόστος. Το κόστος είναι οτι είμαστε πιο αργοί από ένα καλογραμμένο driver. Παράλληλα αναγκαζόμαστε να χρησιμοποιήσουμε proxy server για το tunneling της κίνησης μας. Αυτό μας περιορίζει σε συγκεκριμένα πρωτόκολλα για τα οποία έχουμε γράψει τον server μας να καταλαβαίνει. Αυτό δεν είναι τόσο περιοριστικό όσο ακούγεται αφού και στο PERM που είχαν δυνατότητες για όλα τα πρωτόκολλα, όλη η δουλειά γινότανε για την http κίνηση. Τελευταίο μειονέκτημα είναι ότι δεν μπορούμε να επιλέξουμε εμείς ποιό interface του συστήματος μας θα χρησιμοποιηθεί, την επιλογή αυτή την κάνει το λειτουργικό σύστημα.

Στη συνέχεια έπρεπε να λύσουμε το δεύτερο μεγάλο πρόβλημα. Τι θα σημαίνει στην πράξη το δίκτυο οντοτήτων? Θα κάνουμε κάποια υπόθεση για την τοπολογία του δικτύου μας, αυτή (η τοπολογία) θα είναι δυναμική ή στατική. Πως θα επικοινωνούν οι κόμβοι μεταξύ τους? Επίσης πως θα γίνεται η διευθυνσιοδότηση των κόμβων? Αν υλοποιήσουμε ένα overlay network πως θα γίνεται το bootstrapping των νέων κόμβων? Η κίνηση που θα έχουμε θα περνάει πάνω από το overlay network?

Παρατηρούμε ότι τα ερωτήματα δεν είναι όλα ανεξάρτητα το ένα από το άλλο. Επίσης η απάντηση δεν είναι προφανής.

Για παράδειγμα μια απάντηση θα ήταν να δημιουργήσουμε ένα κατανεμημένο overlay network και να περνάμε όλη την επικοινωνία από πάνω από αυτό το δίκτυο. Στη συνέχεια με αυτή τη βάση θα απαντάγαμε και τα υπόλοιπα ερωτήματα.

Μια άλλη λύση θα ήταν να κινηθούμε σε ένα καθαρά συγκεντρωτικό(centralized) σύστημα αλλά αυτό θα απαιτούσε την ύπαρξη κάποιου server και δεδομένου ότι το σύστημα θέλουμε να το τρέχουν χρήστες με οικιακούς υπολογιστές αυτό θα ανάγκαζε κάποιον χρήστη να αφιερώσει αρκετούς υπολογιστικούς πόρους

Εμείς επειδή θέλαμε τη μέγιστη απόδοση αποφασίσαμε να ξεχωρίσουμε (=αποσυνδέσουμε - decouple) την μεταφορά των ροών από την επικοινωνία που απαιτείται μεταξύ των κόμβων για να γίνει η μεταφορά των ροών.

Με αυτό τον τρόπο για την επικοινωνία των κόμβων χρησιμοποιούμε ένα σύστημα MOM[15] (Message Oriented Middleware). Συγκεκριμένα χρησιμοποιήσαμε το πρωτόκολλο AMQP[16] (Advanced Message Queuing Protocol) με broker τον RabbitMQ[17]. Λεπτομέρειες για αυτό το σύστημα παρουσιάζουμε σε ξεχωριστή ενότητα. Με αυτό τον τρόπο η επικοινωνία γίνεται με σχεδόν βέλτιστο τρόπο και όλες οι μεταφορές των ροών γίνεται σε IP, χωρίς κανένα επιπλέον overhead,

εκτός από αυτό που βάζει ο proxy server μας.

Η διευθυνσιοδότηση δεν αλλάζει από τη γνωστή με τις IP διευθύνσεις που γνωρίζουμε, ενώ για bootstraping αρκεί ο κάθε κόμβος να γνωρίζει την IP διεύθυνση του broker.

Το δίκτυο που δημιουργείται είναι ομότιμων, υπό την έννοια ότι δεν έχουμε client και server, καθώς όλοι οι κόμβοι παίζουν και τους 2 ρόλους.

Επιλέγοντας λύση που δεν περιλαμβάνει κάποιο overlay network απλοποιούμε και την διαδικασία ελέγχου, αφού επί της ουσίας αρκεί να χρησιμοποιήσουμε τα network metrics που χρησιμοποιούμε στα IP δίκτυα.

Για να γίνει το scheduling οι κόμβοι αλλάζουν μεταξύ τους και χρησιμοποιώντας το MOM, όλη την απαραίτητη πληροφορία.

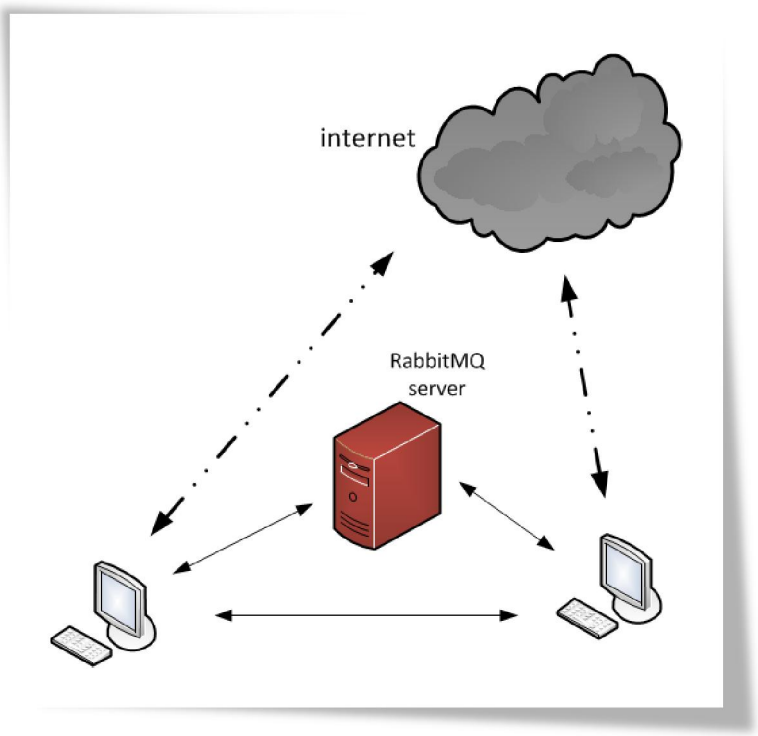
Ζητήματα ασφαλείας δεν θίγονται στη διπλωματική αυτή, παρόλα αυτά δεν είναι τραγικά δύσκολο να κάνει κάποιος κρυπτογράφηση πάνω στις ροές που περνάνε από τους proxy servers. Γενικότερα ισχύει ακριβώς ότι θα ίσχυε αν θα θέλαμε να προστατέψουμε την κίνηση μας, αν τη περνάγαμε σε ένα proxy server που δεν ελέγχεται από εμάς.

Στη συνέχεια θα ακολουθήσει μια λειτουργική περιγραφή των μονάδων και θα προσπαθήσουμε να δείξουμε γιατί θεωρούμε πως το τελικό σύστημα είναι προσαρμόσιμο , επεκτάσιμο και ελέγξιμο.

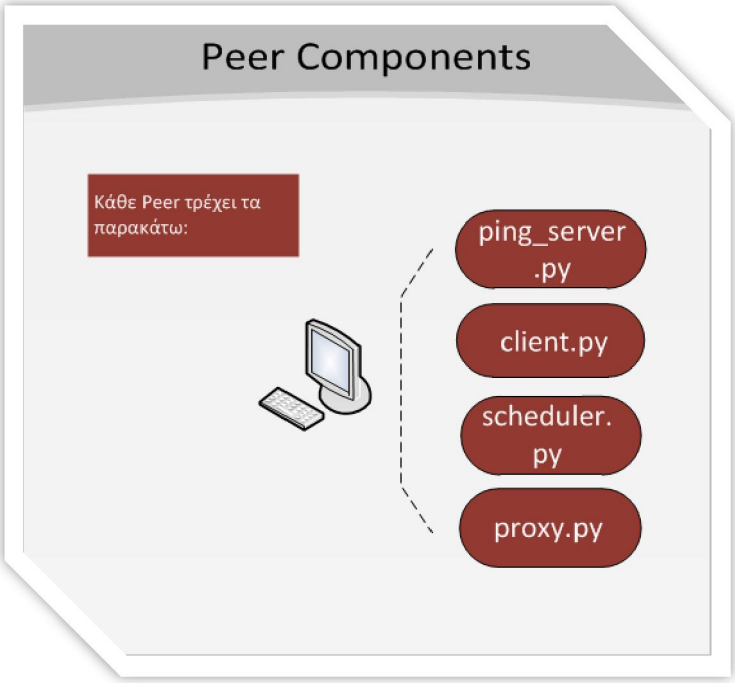
[15] http://en.wikipedia.org/wiki/Message-oriented_middleware
[16] http://en.wikipedia.org/wiki/AMQP
[17] http://en.wikipedia.org/wiki/RabbitMQ

3.5 Περιγραφή της Αρχιτεκτονικής - Οι Λειτουργικές Μονάδες

Έχοντας δει τον τρόπο με τον οποίο οργανώνεται το δίκτυο του συστήματος, για να οριστεί πλήρως η λειτουργία του μένει να περιγραφούν οι βασικές γραμμές λειτουργικότητας που το πλαισιώνουν.



Θεωρούμε ότι κάπου στο δίκτυο των οντοτήτων μας υπάρχει στημένος ένας RabbitMQ server. Κάθε οντότητα τρέχει ένα client. Ο client αυτός αποτελείται από 4 components που επιτελούν ισάριθμο πλήθος από λειτουργίες. Επί της ουσίας το σύστημα μας επιτελεί τις παρακάτω λειτουργίες. Τρέχει ένα proxy server ο οποίος διαχειρίζεται ροές που προκύπτουν. Κρατάει κάποια μετρική για την κατάσταση της σύνδεσης του και τρέχει ένα δρομολογητή ο οποίος με βάση την κατάσταση της σύνδεσης του client και των συνδέσεων των υπόλοιπων clients μπορεί να αποφασίσει που θα στείλει τις εισερχόμενες ροές.



3.5.1 Proxy Server

Σε κάθε κόμβο, οι χρήστες δημιουργούν νέες εξερχόμενες ροές δεδομένων με κάποιον απομακρυσμένο προορισμό. Η μονάδα Proxy παρεμβάλλεται ανάμεσα στην εφαρμογή του χρήστη που δημιουργεί τις ροές και στο επίπεδο δικτύου, συλλέγει έγκαιρα τις νέες ροές και τις κατευθύνει προς έναν κατάλληλο προορισμό. Έτσι όλες οι νέες ροές πλέον διέρχονται μέσω του Proxy.

Ο κατάλληλος προορισμός επιλέγεται σύμφωνα με τη δρομολόγηση που εκτελείται από το component scheduler.py. Έτσι, ανάλογα βέβαια και με το σχήμα δρομολόγησης που το σύστημα χρησιμοποιεί, ο Proxy κατευθύνει τις ροές είτε προς τον εξωτερικό σύνδεσμο του ίδιου κόμβου, είτε προς έναν κόμβο του P2P δικτύου. Είναι πολύ πιθανό ανάλογα και με την κατάσταση του δικτύου σε μερικά σχήματα δρομολόγησης παρόλο που ένας κόμβος μπορεί να διαθέτει εξωτερικό σύνδεσμο, η δρομολόγηση του επιβάλλει να εκτρέψει τις ροές του προς άλλο κόμβο με εξωτερικό σύνδεσμο. Κάτι τέτοιο μπορεί να συμβαίνει για παράδειγμα σε ένα σχήμα απληστίας (greedy scheme) όπου κάθε κόμβος επιλέγει εγωιστικά το σύνδεσμο που αποδίδει καλύτερα, είτε είναι ο δικός του είτε όχι.

Ζητήματα που προκύψαν από την υλοποίηση του Proxy και πιστεύουμε ότι είναι άξια σχολιασμού ακολουθούν σε επόμενο κεφάλαιο.

3.5.2 Ping Server

Η καταγραφή στατιστικών στοιχείων και μετρήσεων που σχετίζονται με την δικτυακή κυκλοφορία των συνδέσμων αλλά και με την κατάστασή τους, είναι απαραίτητη σε ένα σύστημα όπως το py-p2plnet. Η ανάγκη αυτή οφείλεται αρχικά στη διεξαγωγή πειραμάτων. Προκειμένου να διερευνηθεί και να ποσοτικοποιηθεί η απόδοση ενός τέτοιου συστήματος και το όφελος που προσφέρει σε μια κοινωνία οντοτήτων που συναθροίζουν τη συνδεσιμότητά τους.

Το συγκεκριμένο component αναλαμβάνει να χρησιμοποιήσει την όποια μετρική επιλέξουμε και να ενημερώσει τους υπόλοιπους κόμβους του δικτύου για τα αποτελέσματα της μέτρησης αυτής. Ο ουσιαστικός του ρόλος είναι να δώσει την πληροφορία που τελικά θα βοηθήσει τον scheduler να καταλήξει σε σωστή απόφαση κάθε φορά που χρειάζεται να γίνει δρομολόγηση.

Η Μονάδα Παρακολούθησης επίσης εκτελεί διερεύνηση για αποτυχίες στους υπάρχοντες συνδέσμους. Αυτό συνίσταται στην περιοδική προσπάθεια για σύνδεση με κάποιον απομακρυσμένο προορισμό. Επιτυχείς προσπάθειες συνεπάγονται ότι ο σύνδεσμος βρίσκεται σε ενεργή κατάσταση, ενώ αποτυχίες οδηγούν στο συμπέρασμα ότι ο σύνδεσμος έχει αποτύχει λόγω σφαλμάτων.

Η περιοδικότητα με την οποία εκτελούνται οι ενεργητικές μετρήσεις ρυθμίζεται στο σύστημά μας. Η ρύθμιση που θα διαλέξουμε εξαρτάται κυρίως από τον scheduler και το πόσο ακριβείς μετρήσεις χρειάζεται.

Εδώ να σημειώσουμε ότι ο ping_server.py δουλεύει τελείως ασύγχρονα από τα υπόλοιπα components αφού οι ότι μετρήσεις κάνει αναλαμβάνει να τις στείλει στον rabbitmq server. Αυτό γίνεται κατανεμημένα από όλους τους κόμβους του συστήματος.

3.5.3 Client

Αυτό το component αναλαμβάνει να πάρει την πληροφορία που στέλνουν όλοι οι peers στον broker και να την μετατρέψει στο κατάλληλο format ώστε ο scheduler να μπορεί να χρησιμοποιήσει την πληροφορία για να καταλήξει και σε απόφαση.

3.5.4 Scheduler

Η μονάδα που κάνει το scheduling αναλαμβάνει να πάρει τα δεδομένα , που δίνουν οι υπόλοιποι κόμβοι για το δίκτυο και να αναλάβει να επιλέξει προορισμό για την επόμενη ροή που θα χρειαστεί ο proxy να δρομολογήσει.

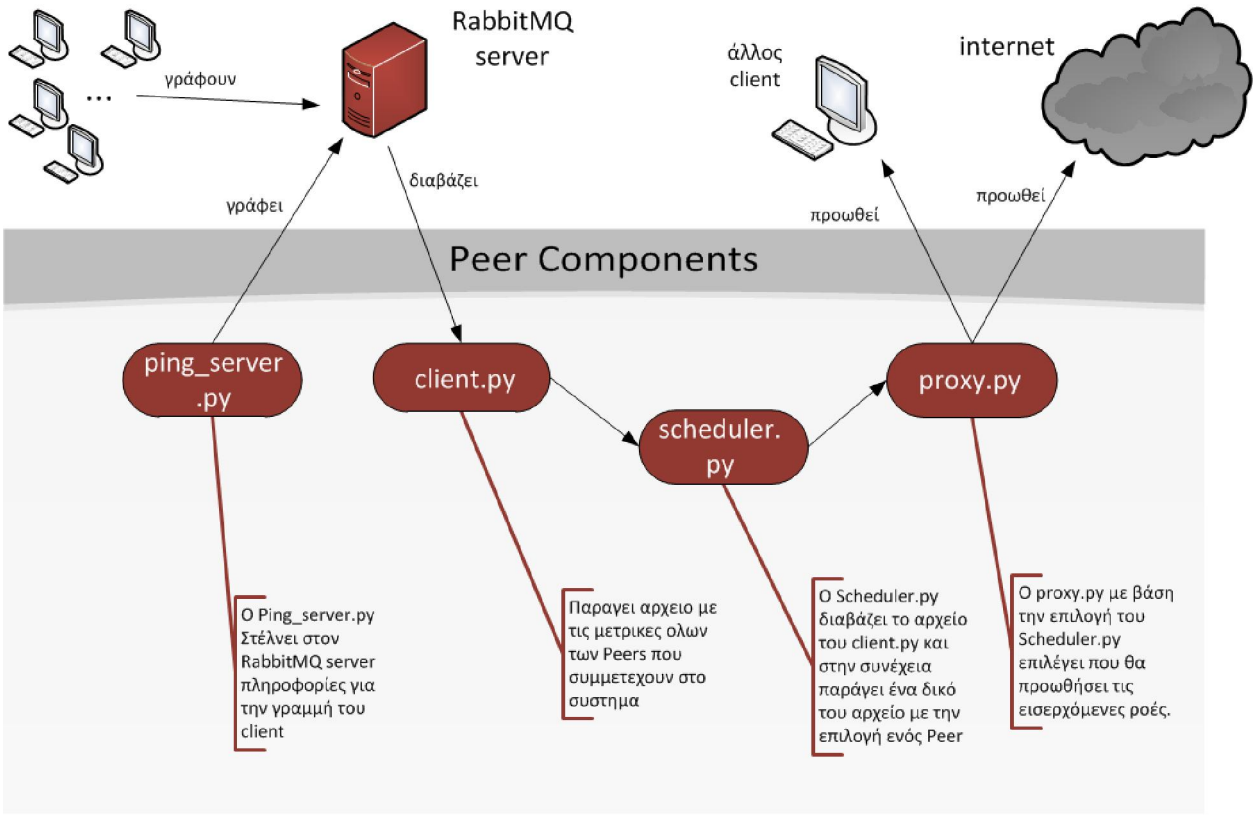
Στην περίπτωση του συστήματος έχουμε υλοποιήσει ένα greedy αλγόριθμο ο οποίος επιλέγει το γρηγορότερο link για να δρομολογήσει τη ροή. Βέβαια είναι αρκετά ευθύ να υλοποιήσουμε και κάποιον άλλο αλγόριθμο.

3.6 Σχολιασμός py-p2pinet

3.6.1 Για την δομή του συστήματος

Θέλαμε ένα σύστημα που να είναι εύκολα deployable σε πολλές πλατφόρμες, χωρίς να έχει ιδιαίτερα dependencies. Επειδή ο σκοπός του συστήματος είναι χρησιμοποιηθεί σε διάφορες δοκιμές πρέπει αυτό το σύστημα πρέπει να είναι παραμετροποιήσιμο και επεκτάσιμο. Για να το πετύχουμε αυτό όλο το σύστημα αποτελείται από components τα οποία δουλεύουν ασύγχρονα μεταξύ τους. Η μοναδική υπόθεση που κάνει το ένα component για το άλλο είναι ότι περιμένει για είσοδο αρχείο που παράγεται από άλλο component με συγκεκριμένη μορφή. Παράλληλα όλο το σύστημα παίρνει σε runtime όλες τις ρυθμίσεις από ένα configuration αρχείο.

Ακολουθεί σχηματικά η δομή του συστήματος.



Η παραπάνω δομή βοηθάει και στον έλεγχο της ορθότητας του συστήματος αφού μας επιτρέπει να ελέγξουμε το κάθε component ξεχωριστά.

3.6.2 Επικοινωνία μεταξύ των clients

Για την κατανόηση των παρακάτω καλό θα ήταν ο αναγνώστης να έχει υπόψη του τα βασικά του AMQP, τα οποία τα παραθέτουμε στο ξεχωριστό κεφάλαιο που αναφέρεται γενικότερα στο messaging.

Το σύστημά μας χρησιμοποιεί ένα topic exchange με όνομα **'events'**. Κάθε οντότητα αναλαμβάνει να στήλει στο exchange events μια μετρική που απεικονίζει την ποιότητα της σύνδεσης της.

Η αποστολή αυτή γίνεται με routing key που φτιάχνεται έτσι

```
"p2p.info." + sysconf['short_name']
```

δηλαδή ο client1 με shortname c1 θα στέλνει με κλειδί **p2p.info.c1**

Παράλληλα όλοι οι clients έχουν μια προσωπική queue με όνομα

sysconf['short_name'].events

Δηλαδή στην περίπτωση του client2 θα είχαμε μια ουρά με όνομα c2.events.

Στις ci.events ουρές φτάνουν όλα τα μηνύματα με routing key 'p2p.info.#' άρα μηνύματα με κλειδιά όπως

```
p2p.info.c1, p2p.info.c2, .....
```

3.6.3 Εξαρτήσεις από εξωτερικά προγράμματα

Τα μοναδικά dependencies που θέλει το σύστημα μας είναι ένα αντίστοιχο με το ping εργαλείο, ένα στημένο rabbitmq server και μια βιβλιοθήκη που να μας επιτρέπει την επικοινωνία με το rabbitmq server.

Αν τώρα θέλουμε να τρέξουμε τα αυτοματοποιημένα test, θα χρειαστούμε ένα σύστημα που να έχει πρόσβαση στο εργαλείο curl.

3.6.4 Γραμμένο σε python

Το όλο σύστημα επιλέξαμε να το υλοποιήσουμε με python. Ανάμεσα στους λόγους για αυτήν μας την επιλογή είναι ότι θέλαμε το όλο σύστημα να είναι cross-platform, η python τρέχει σχεδόν σε όλες τις

πλατφόρμες. Από μόνο του αυτό δεν είναι ιδιαίτερα σημαντικό, το σημαντικό είναι ότι η python έρχεται με μια μεγάλη βιβλιοθήκη, η οποία είναι και αυτή cross-platform. Παράλληλα η python βολεύει πολύ για rapid-prototyping, ενώ συνδυάζει χαρακτηριστικά κλασσικής γλώσσας προγραμματισμού με γλώσσα για scripting.

Αυτό μας βοήθησε ιδιαίτερα όταν πχ χρειαζόταν να τρέξουμε κάποια εξωτερική εντολή και μετά να επεξεργαστούμε το αποτέλεσμα που επέστρεφε με regular expressions ή όταν χρειάστηκε να κάνουμε parsing σε log files του squid. Τέλος ένας ακόμη λόγος για αυτήν μας την επιλογή είναι οτι η βιβλιοθήκη της python έχει ένα module για την ανάπτυξη network servers(λεπτομέρειες για αυτό σε επόμενο κεφάλαιο της διπλωματικής).

3.6.5 Οδηγός εγκατάστασης

Ακολουθεί ένας ενδεικτικός οδηγός εγκατάστασης για το ubuntu 10.04

Αρχικά πρέπει να εγκαταστήσουμε τον **rabbitmq server**. Για να το κάνουμε αυτό χρησιμοποιούμε την παρακάτω εντολή:

```
sudo apt-get install rabbitmq-server
```

Η εγκατάσταση γίνεται σε αυτό το path

```
/usr/lib/rabbitmq
```

Για να τροποποιήσει κανείς τις ρυθμίσεις με τις οποίες ξεκινάει ο server πειράζει αυτό το αρχειο

```
/etc/rabbitmq/rabbitmq.conf
```

Ενω για το management του broker βοηθούν οι παρακάτω εντολές:

```
sudo rabbitmq-activate-plugins
sudo /etc/init.d/rabbitmq-server restart/stop/start
```

τα logs του server βρίσκονται εδώ:

```
/var/log/rabbitmq
```

Στη συνέχεια θα χρειαστούμε την python βιβλιοθήκη **py-amqplib**.

Ένας από τους τρόπους που μπορούμε να την εγκαταστήσουμε είναι ο εξής:

```
apt-get install python-setuptools
sudo easy_install amqplib
```

χρήσιμο βρήκαμε και το **rabbitmq-status plugin** το οποίο αναλαμβάνει να κάνει ένα web based interface για το monitoring του server μας. Το εργαλείο θα το βρούμε εδώ:

```
http://dev.lshift.net/majek/rabbitmq-status_1.7.0.tar.gz
```

Οδηγίες για την εγκατάσταση θα βρούμε στον administration guide του rabbitmq. Μετά την εγκατάσταση θα έχουμε πρόσβαση σε αυτό στην παρακάτω διεύθυνση

```
http://guest:guest@127.0.0.1:55672/
```

3.6.6 Οδηγίες για την εκτέλεση του py-p2pinet

Για να τρέξουμε το πρόγραμμα πρέπει να εκτελέσουμε τα παρακάτω βήματα.

Αρχικά πρέπει να βάλουμε τις σωστές ρυθμίσεις στο settings.conf. Τις ρυθμίσεις από εδώ θα τις χρησιμοποιήσουν και τα υπόλοιπα components.

Στη συνέχεια και με δεδομένο ότι κάπου τρέχει ο rabbitmq server εκτελούμε το script rabbitmq_create.py.

Αυτό αναλαμβάνει να στήσει τις απαραίτητες ουρές στον broker μας. Εδώ να σημειώσουμε ότι στην έκδοση που έρχεται μαζί με τη διπλωματική

το script στήνει τις απαραίτητες δομές για 2 μόνο clients. Για παραπάνω θα πρέπει να το τροποποιήσουμε.

Μαζί με το rabbitmq_create.py δίνουμε και το rabbitmq_destroy.py για να σβήσουμε τις δομές που δημιουργήσαμε από τον server μας.

Στη συνέχεια μπορούμε αν είμαστε σε περιβάλλον unix να τρέξουμε το script start_p2p.sh που θα αναλάβει να κάνει το bootstrap του συστήματος.

Αν δεν είμαστε σε unix ή αν θέλουμε να σηκώσουμε μόνοι μας τα διάφορα components, θα πρέπει να τρέξουμε τα παρακάτω:

1. client.py
2. scheduler.py
3. ping_server.py
4. proxy.py

3.6.7 Testing py-p2pinet

Για τον έλεγχο της ορθότητας του συστήματος χρησιμοποιήσαμε τις παρακάτω μεθόδους:

- χρησιμοποιήσαμε τον browser μας και το πρόγραμμα curl

  http://curl.haxx.se/

  για να ελέγξουμε την ορθή λειτουργία του proxy server.

- στη συνέχεια, χρησιμοποιήσαμε το εργαλείο "ab - Apache HTTP server benchmarking tool"

  για να μπορέσουμε να βεβαιώσουμε ότι το σύστημά μας αντέχει να εξυπηρετήσει πολλές χιλιάδες requests, και πολλές φορές παράλληλα.

  πχ τρέχαμε το εργαλείο έτσι, για να εξυπηρετήσουμε 80000 requests τα οποία γίνονται ανά 100 παράλληλα.

  ```
  ab -c 100 -n 80000 -X $proxy_addr $location
  ```

- τέλος φτιάξαμε ένα parser για τα log files του squid server[18] έτσι ώστε να πάρουμε logs από το Patras Wireless Network (PWN) και να τρέξουμε πραγματική κίνηση πάνω από το σύστημα μας.

Για να αυτοματοποιήσουμε την διαδικασία του ελέγχου ενσωματώσαμε όλη αυτή τη λειτουργικότητα στα παρακάτω 3 scripts:

- stress_tester.py
- test_02.sh
- test_proxy.sh

[18] http://www.squid-cache.org/

4 Messaging

4.1 Message-oriented middleware (MOM)

Σε αυτό το κεφάλαιο θα εξετάσουμε το πρωτόκολλο AMQP και γενικότερα έννοιες που άπτονται των Message-oriented middleware[19] (MOM) συστημάτων.

Μιλώντας για **Message-oriented middleware (MOM)** μιλάμε για υποδομές που επικεντρώνονται στην αποστολή και λήψη μηνυμάτων και που μας επιτρέπουν να δημιουργήσουμε μια εφαρμογή που κομμάτια της να είναι κατανεμημένα σε ετερογενείς πλατφόρμες. Ένα MOM περιορίζει την πολυπλοκότητα της ανάπτυξης εφαρμογών που εκτείνονται σε πολλαπλά λειτουργικά συστήματα και πρωτόκολλα δικτύου απομονώνοντας τον προγραμματιστή από τις λεπτομέρειες των διάφορων συστημάτων.

4.1.1 Πλεονεκτήματα

4.1.1.1 Αποθήκευση (Storage)

Τα περισσότερα MOM συστήματα παρέχουν τρόπο για την «μόνιμη» αποθήκευση των μηνυμάτων έτσι ώστε ο αποστολέας και ο παραλήπτης να μην χρειάζεται να είναι συνδεδεμένοι την ίδια ώρα στο δίκτυο. Έτσι έχουμε ασύγχρονη παράδοση των μηνυμάτων.

Αυτό γίνεται ιδιαίτερα χρήσιμο όταν έχεις διαλείπουσες συνδέσεις ή αναξιόπιστα δίκτυα. Σημαίνει επίσης ότι δίνεται η δυνατότητα στους αποστολείς να συνεχίσουν ανεπηρέαστα την αποστολή μηνυμάτων, καθώς τα μηνύματα που αποστέλλουν απλά θα συγκεντρώνονται στο χώρο αποθήκευσης μηνυμάτων για μετέπειτα επεξεργασία όταν ο δέκτης θα είναι έτοιμος να τα πάρει.

4.1.1.2 Δρομολόγηση (Routing)

Το MOM έχει ένα ακόμα πλεονέκτημα αφού έχει την δυνατότητα να δρομολογεί τα μηνύματα μέσα στο messaging layer. Αυτό του δίνει την δυνατότητα να μπορεί να παραδώσει ένα μήνυμα σε περισσότερους από έναν παραλήπτες (broadcast or multicast).

4.1.2 Μειονεκτήματα

Το κύριο μειονέκτημα της προσέγγισης με MOM είναι ότι απαιτούν ένα επιπλέον στοιχείο στην αρχιτεκτονική του συστήματος, συγκεκριμένα τον message transfer agent ή message broker.

[19] http://en.wikipedia.org/wiki/Message-oriented_middleware

Αυτό αυξάνει την πολυπλοκότητα και πιθανώς βλάπτει και την απόδοση. Παράλληλα όσο περισσότερα στοιχεία έχει ένα σύστημα τόσο πιο δύσκολη και δαπανηρή είναι η συντήρησή του.

Παράλληλα τα MOM δεν βολεύουν στις περιπτώσεις που χρειάζεται σύγχρονη επικοινωνία, με τον αποστολέα να περιμένει απάντηση από τον παραλήπτη πριν συνεχίσει.

Τέλος σοβαρό πρόβλημα δημιουργεί και η έλλειψη προτύπων (standards). Η έλλειψη προτύπων που διέπουν τη χρήση MOM προκαλεί προβλήματα. Οι περισσότερες μεγάλες εταιρείες λογισμικού έχουν τις δικές τους υλοποιήσεις, το καθένα με τη δική διεπαφή προγραμματισμού εφαρμογών του (API) και εργαλεία διαχείρισης.

4.1.3 Advanced Message Queuing Protocol (AMQP)

Το Advanced Message Queuing Protocol (AMQP) είναι ένα ανερχόμενο και ανοιχτό πρότυπο που προσπαθεί να λύσει τα προβλήματα που έχουν τα proprietary MOM.

Το AMQP καθορίζει το πρωτόκολλο και το format που χρησιμοποιούνται για την ανταλλαγή μηνυμάτων μεταξύ του διακομιστή και του πελάτη, με αποτέλεσμα να κάνει τις διάφορες υλοποιήσεις πλήρως διαλειτουργικές (interoperable).

Το AMQP είναι ορισμένο έτσι ώστε να παρέχει ευέλικτη δρομολόγηση, και εύκολη υλοποίηση κάποιων μοτίβων επικοινωνίας(messaging paradigms) όπως point-to-point, fanout, publish/subscribe, and request-response. Υποστηρίζει ακόμη διαχείριση των συναλλαγών, ουρές, κατανεμημένη υποστήριξη και υποστήριξη πολλών πλατφόρμων.

Το AMPQ είναι ένα ανοιχτό πρωτόκολλο, το οποίο επιτρέπει σε συμβατές εφαρμογές να επικοινωνούν με συμβατούς messaging middleware servers τους οποίους ονομάζουμε brokers.

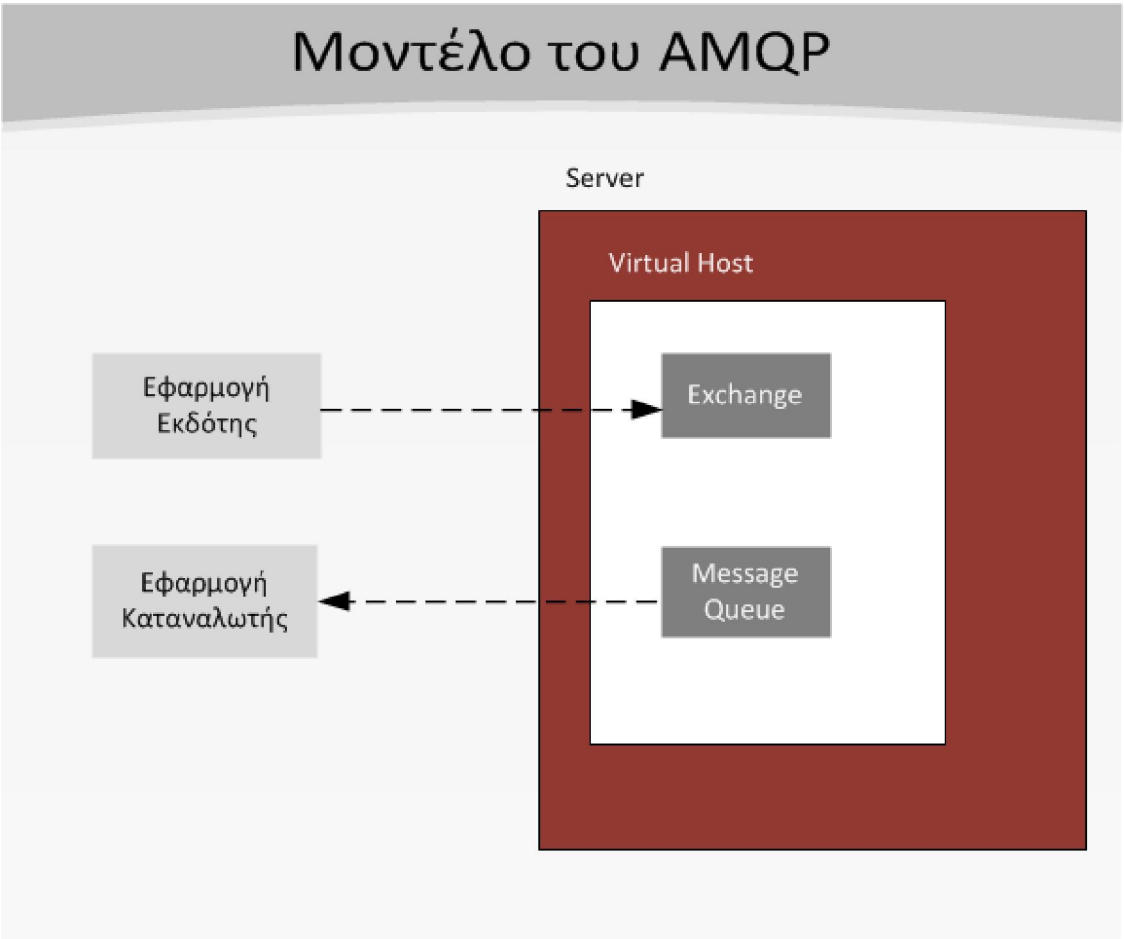
4.1.3.1 Γιατί AMPQ

Στόχος της ανάπτυξης του AMPQ είναι αρχικά να καταστεί δυνατή η ανάπτυξη τυποποιημένων MOM που θα είναι διαλειτουργικά μεταξύ τους ενώ σε δεύτερο επίπεδο οι διάφορες λειτουργίες των MOM να περάσουν στο ίδιο το δίκτυο και να οδηγήσουν στην ανάπτυξη νέων εφαρμογών.

4.2 Το μοντέλο του AMPQ

Το μοντέλο του AMPQ περιλαμβάνει 3 κύρια είδη από modular components τα οποία συνδέονται με διάφορους τρόπους μέσα στον server για να δημιουργηθεί η επιθυμητή(για τον χρήστη) λειτουργικότητα.

Η παρακάτω εικόνα, παρουσιάζει συνοπτικά το μοντέλο του AMQP. Θα ξεκινήσουμε με μια εισαγωγή στα βασικότερα components του μοντέλου, για να δούμε μετά και περισσότερες λεπτομέρειες. Ολόκληρο το κομμάτι έχει βασιστεί στο specification του AMQP και για αυτό ακολουθεί και την δομή του.



Βασικά components του μοντέλου είναι:

1. Exchanges (ανταλλαγές)
2. Message Queues (ουρές μηνυμάτων)
3. Bindings (δεσίματα)

Συνοπτικά, ο χρήστης ορίζει exchanges τα οποία παίρνουν μηνύματα από τις εφαρμογές-εκδότες (publishing applications) και τα δρομολογεί στις κατάλληλες ουρές μηνυμάτων(Message Queues). Η δρομολόγηση αυτή γίνεται με αυθαίρετα κριτήρια τα οποία ορίζονται στα bindings.

Οι ουρές μηνυμάτων αναλαμβάνουν να αποθηκεύσουν τα μηνύματα μέχρι αυτά να «καταναλωθούν»- επεξεργαστούν από 1 ή περισσότερες εφαρμογές-καταναλωτή (consuming applications)

Με τη χρήση των παραπάνω μπορούμε μεταξύ άλλων να προσομοιώσουμε τις κλασσικές MOM έννοιες για store-and-forward queues και topic subscriptions πολύ εύκολα.

Το μοντέλο δημιουργήθηκε για να καλύψει τους παρακάτω στόχους:

1. να είναι εύκολα προσαρμόσιμο σε νέους τρόπους δρομολόγησης και χρήσης των ουρών
2. να επιτρέπει στην ίδια την εφαρμογή να προγραμματίσει τον server μέσω του πρωτοκόλλου
3. είναι επεκτάσιμο αλλά ταυτόχρονα και απλό

4.2.1 Εισαγωγή στο AMQP

Ίσως κάνει εντύπωση στην αρχή αλλά στο μοντέλο καθορίζεται και η λειτουργία του server η οποία πρέπει να ναι κι αυτή τυποποιημένη για να μπορεί να εγγυηθεί κανείς την διαλειτουργικότητα μεταξύ των διαφόρων AMPQ υλοποιήσεων.

Γενικότερα μπορούμε να συνοψίσουμε τη λειτουργία ενός middleware server ως εξής:

Είναι ένας data server που δέχεται μηνύματα και κάνει δύο πράγματα με αυτά

- Ø τα δρομολογεί σε διαφορετικούς καταναλωτές βασιζόμενος σε διάφορα κριτήρια
- Ø και τα αποθηκεύει στη μνήμη ή στον δίσκο όταν οι καταναλωτές δεν είναι έτοιμοι να τα αποδεχθούν

Πριν από την χρήση του AMPQ αυτές λειτουργίες γινόντουσαν από μονολιθικές εφαρμογές που υλοποιούσανε και κάνανε συγκεκριμένου είδους δρομολόγηση και αποθήκευση. Με το AMPQ και το καινούργιο μοντέλο που προτείνει, μπορούμε να σπάσουμε την προσέγγισή μας σε μικρότερα και πιο αρθρωτά κομμάτια τα οποία μπορούν να συνδυαστούν με ποικίλους τρόπους και να δώσουν πιο εύρωστες (robust) υλοποιήσεις.

Το AMPQ ορίζει μια ξεκάθαρη διεπαφή που την λέμε binding , μεταξύ ενός exchange και μιας message queue.

Η χρησιμότητά του μοντέλου έρχεται από τα παρακάτω τρία κύρια χαρακτηριστικά

1. Την ικανότητα να δημιουργούμε αυθαίρετες exchanges και message queues διαφόρων τύπων (κάποιες ορίζονται μέσα από το AMPQ standard , άλλες μπορούν να προστεθούν σαν επεκτάσεις στον server)
2. Την ικανότητά του να ενώνουμε exchanges και message queues μεταξύ τους για να δημιουργούμε οποιοδήποτε σύστημα επεξεργασίας μηνυμάτων
3. Την ικανότητα να ελέγχουμε-να προγραμματίζουμε όλα τα παραπάνω, προγραμματιστικά μέσω του πρωτοκόλλου σε πραγματικό χρόνο εκτέλεσης

4.2.2 Η ουρά μηνυμάτων - theMessage Queue

Μια ουρά μηνυμάτων αποθηκεύει μηνύματα στη μνήμη ή τον δίσκο και παραδίδει τα διάφορα μηνύματα σε μια ή περισσότερες εφαρμογές-καταναλωτές. Έτσι οι ουρές μηνυμάτων είναι οντότητες που χρησιμοποιούνται για την αποθήκευση των μηνυμάτων και για την εύκολη διανομή αυτών. Κάθε ουρά είναι τελείως ανεξάρτητη από τις υπόλοιπες.

Μια ουρά μήνυμα έχει διάφορες ιδιότητες: Μπορεί να είναι ιδιωτική ή κοινόχρηστη, ανθεκτική-σταθερή ή παροδική, μόνιμη ή προσωρινή.

Επιλέγοντας τον κατάλληλο συνδυασμό που ιδιότητες μπορούμε να δημιουργήσουμε ουρές που να καλύπτουν όλες τις ανάγκες που μπορεί να έχουμε από μια τέτοια οντότητα

1. Μια κλασική store-and-forward ουρά, κρατάει μηνύματα και τα διανείμει ανάμεσα στους συνδρομητές σε μια round robin βάση. Αυτές οι ουρές είναι συνήθως ανθεκτικές-σταθερές και διαμοιραζόμενες-κοινόχρηστες ανάμεσα σε πολλαπλούς συνδρομητές.
2. Μια προσωρινή ουρά απάντησης(temporary reply queue) η οποία κρατάει τα μηνύματα και τα προωθεί-διαβιβάζει σε ένα μόνο συνδρομητή. Οι ουρές απάντησης είναι συνήθως προσωρινές και ιδιωτικές σε ένα συνδρομητή

3. Μια "pub-sub" ουρά, η οποία κατέχει μηνύματα που συλλέγονται από διάφορες πηγές στις οποίες έχει γραφεί αν ο συνδρομητής και τα προωθεί σε αυτόν. Αυτές οι μέρες είναι συνήθως προσωρινές και ιδιωτικές σε έναν συνδρομητή

Οι κατηγορίες αυτές δεν είναι τυπικά ορισμένες στο πρότυπο του AMQP, είναι απλά παραδείγματα του τρόπου και της ευκολίας με την όποια μπορούμε να δημιουργήσουμε ουρές μηνυμάτων. Το μοντέλο του AMQP κάνει τετριμμένη την δημιουργία νέων οντοτήτων για την κάλυψη σχεδόν όλων των επικοινωνιακών αναγκών μιας εφαρμογής.

4.2.3 Η ανταλλαγή - the exchange

Ένα exchange δέχεται μηνύματα από μια εφαρμογή-παραγωγό και τα δρομολογεί σε ουρές μηνυμάτων σύμφωνα με προκαθορισμένα κριτήρια. Αυτά τα κριτήρια τα λέμε bindings. Επί της ουσίας ένα exchange είναι ένα component που κάνει το «διάβασμα» της routing πληροφορίας και το routing των μηνυμάτων. Για να γίνει αυτό, το exchange διαβάζει το μήνυμα και χρησιμοποιεί τους δικούς του routing πίνακες. Τα exchanges δεν αποθηκεύουν μηνύματα.

Η λέξη exchange αναφέρεται και στην κλάση των αλγορίθμων αλλά και στα instances της κλάσης αυτής. Δηλαδή αναφέρεται και σε “exchange types” και σε “exchange instances”.

Το AMQP ορίζει κάποια exchange types που καλύπτουν τις πιο συνηθισμένες ανάγκες για δρομολόγηση των μηνυμάτων. Για αυτά τα exchange types δίνονται υλοποιημένα default exchange instances από τον εκάστοτε AMQP server που χρησιμοποιούμε. Βέβαια δίνεται η δυνατότητα στις εφαρμογές που χρησιμοποιούν το AMQP να δημιουργήσουν τις δικές τους exchange instances.

Τα exchange types έχουν το καθένα δικό τους όνομα έτσι ώστε οι εφαρμογές που δημιουργούν τα δικά τους να μπορούν να ορίσουν στον server ποιο type να χρησιμοποιήσει αντίστοιχα και τα exchange instances έχουν δικό τους όνομα έτσι ώστε οι εφαρμογές-καταναλωτές να μπορούν να ορίσουν τους binding κανόνες και οι εφαρμογές-παραγωγοί να ορίζουν που θα στείλουν τα μηνύματά τους.

Περιληπτικά η έννοια του exchange έχει μπει στο AMQP ώστε να μπορεί να υποστηριχθεί επεκτάσιμη routing συμπεριφορά στον server.

4.2.4 Η δρομολόγηση - the routing key

Είδαμε ότι στην γενική περίπτωση ένα exchange εξετάζει τις ιδιότητες ενός μηνύματος, τα κλειδιά που βρίσκονται στην επικεφαλίδα ενός μηνύματος καθώς και το περιεχόμενο του ίδιου του μηνύματος και χρησιμοποιώντας αυτές τις πληροφορίες και πιθανώς πληροφορίες από άλλες πηγές αποφασίζει που θα το δρομολογήσει.

Στην πλειονότητα των περιπτώσεων το exchange εξετάζει ένα απλό πεδίο, το οποίο το ονομάζουμε routing key δηλαδή κλειδί δρομολόγησης, και από την τιμή αυτού του πεδίου αποφασίζει πως θα το δρομολογήσει.

- Για point-to-point δρομολόγηση το κλειδί αντιστοιχεί σε μια message queue
- Για topic pub-sub δρομολόγηση το κλειδί αντιστοιχεί σε μια ιεραρχία ενός θέματος (θα το δούμε με παράδειγμα πιο κάτω τι σημαίνει αυτό)
- Σε πιο περίπλοκες περιπτώσεις η δρομολόγηση βασίζεται στην επικεφαλίδα και/η στο σώμα του μηνύματος.

4.2.5 Analogy to Email

Για να γίνει πιο κατανοητό το τι περιλαμβάνει το AMPQ πολλές φορές χρησιμοποιείται αναλογία με το σύστημα του email.

1. Ένα μήνυμα του AMPQ είναι ανάλογο με ένα email μήνυμα
2. Μια ουρά μηνυμάτων είναι ανάλογη με ένα mailbox
3. Ένας καταναλωτής είναι σαν ένα email-client που παίρνει και διαγράφει μηνύματα
4. Ένα exchange είναι σαν ένα MTA (mail transfer agent) που εξετάζει ένα email και αποφασίζει στηριζόμενο στη διεύθυνση που να στείλει το email
5. Ένα κλειδί δρομολόγησης αντιστοιχεί στα πεδία του email Cc: / To: / Bcc: / χωρίς την πληροφορία για τον server αφού η δρομολόγηση γίνεται εσωτερικά σε έναν AMQP server.
6. Κάθε exchange instance είναι σαν ένα ξεχωριστό MTA process, που διαχειρίζεται κάποιο subdomain ή κίνηση συγκεκριμένου τύπου.
7. Ένα binding είναι σαν μια γραμμή σε πίνακα δρομολόγησης ενός MTA

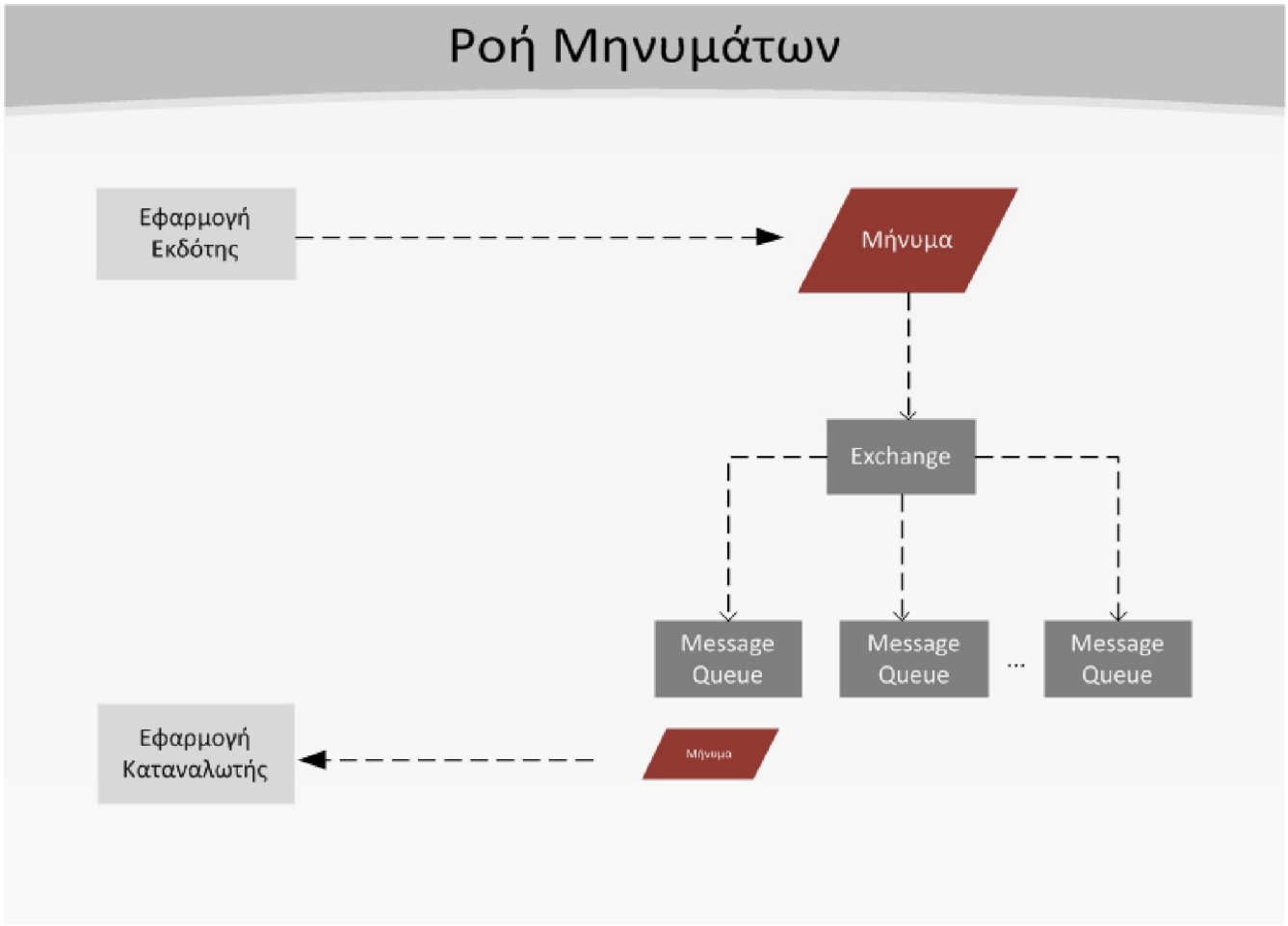
Η δύναμη του AMQP προέρχεται από την δυνατότητα που μας δίνει να δημιουργούμε ουρές (mailboxes), exchanges(MTA processes) και bindings (routing entries) δυναμικά, στο χρόνο εκτέλεσης και να τα ενώνουμε αυτά μαζί με τρόπους που ξεπερνάνε την απλή αντιστοίχηση που έχουμε στην περίπτωση του email.

Βέβαια οι ομοιότητες με το email δεν είναι και πάρα πολλές. Το AMQP διαχωρίζει ανάμεσα στο routing που γίνεται μέσα στον ίδιο server (αυτό κάνουν τα exchanges) και στο routing που γίνεται ανάμεσα σε διαφορετικούς servers. Ο διαχωρισμός γίνεται γιατί τα θεωρούν διαφορετικά προβλήματα με διακριτή λύση.

Ενδεικτικά αναφέρουμε ότι για τη δρομολόγηση μηνυμάτων ανάμεσα σε διαφορετικούς AMQP servers πρέπει να στηθούν σαφείς γέφυρες και ο ένας server να πάρει τον ρόλο του client.

4.2.6 Message Flow – Ροή Μηνυμάτων

Ακολουθεί διάγραμμα που δείχνει την ροη των μηνυμάτων μέσα από ένα AMPQ server



4.2.6.1 Ο κύκλος ζωής ενός μηνύματος

Ένα μήνυμα αποτελείται από ιδιότητες-επικεφαλίδας (header properties) και το σώμα του μηνύματος.

Η εφαρμογή-παραγωγός δημιουργεί το μήνυμα χρησιμοποιώντας το AMPQ API. Όλη η πληροφορία βρίσκεται μέσα στο σώμα του μηνύματος ενώ η εφαρμογή μπορεί να θέσει και κάποιες ιδιότητες στην επικεφαλίδα. Στην συνέχεια ο παραγωγός βάζει στο μήνυμα την πληροφορία δρομολόγησης και το στέλνει σε κάποιο exchange.

Όταν το μήνυμα φτάσει στον server τότε το exchange δρομολογεί το μήνυμα σε ένα σύνολο από ουρές οι οποίες επίσης υπάρχουν στον server. Εάν το μήνυμα δεν μπορεί να δρομολογηθεί τότε το exchange μπορεί να το «πετάξει», να το απορρίψει ή να το δρομολογήσει σε κάποιο άλλο exchange,

ανάλογα με το τι έχει ζητήσει ο παραγωγός. Ένα μήνυμα λοιπόν μπορεί να υπάρχει πανομοιότυπο σε πολλές ουρές.

Ο AMQP server προσπαθεί να προωθήσει το μήνυμα αμέσως στον καταναλωτή και μόνο αν αυτό δεν σταθεί δυνατό το αποθηκεύει στην αντίστοιχη ουρά, μέχρι ο καταναλωτής να είναι έτοιμος. Όταν ο καταναλωτής πάρει το μήνυμα , το επεξεργαστεί και το «αποδεχθεί» τότε ο server μπορεί να το διαγράψει από την ουρά. Ο καταναλωτής είναι αυτός που επιλέγει αν, πως και πότε θα «αποδεχθεί» ένα μήνυμα. Ενδεικτικά αναφέρουμε ότι ο καταναλωτής έχει την δυνατότητα να επιστρέψει ένα μήνυμα πίσω στην ουρά ή να το απορρίψει σαν μη επεξεργάσιμο.

4.2.6.2 Από την μεριά του παραγωγού

Βλέπουμε ότι ο παραγωγός δεν στέλνει απευθείας το μήνυμα σε κάποια ουρά αλλά σε κάποιο exchange. Αν ο παραγωγός μπορούσε να στείλει το μήνυμα απευθείας σε μια ουρά θα χάλαγε το abstraction του μοντέλου αφού θα γινότανε αδύνατη η οποιαδήποτε περαιτέρω επεξεργασία (πχ εισαγωγή κάποιου φίλτρου) ενός μηνύματος πριν φράσει στον καταναλωτή.

Το μοντέλο του AMQP μοιάζει σε αυτό το σημείο με το email: αφού τελικά όλα τα μηνύματα στέλνονται σε ένα κεντρικό σημείο, το exchange, και από εκεί με τη χρήση κανόνων και πληροφορίας που είναι κρυμμένη από τον παραγωγό-αποστολέα το μηνύματα φτάνουν στον τελικό προορισμό τους.

4.2.6.3 Από την μεριά του καταναλωτή

Η αναλογία μας με το σύστημα του email δεν λειτουργεί όταν αρχίζουμε να μιλάμε για τον καταναλωτή. Συγκεκριμένα ένας email-client είναι αρκετά παθητικός αφού μπορεί μόνο να διαβάσει κάποιο mailbox και δεν έχει καμιά επιρροή στο πως θα γεμίσει (το mailbox). Ένας AMQP-καταναλωτής μπορεί να είναι και αυτός παθητικός σαν τον email-client αν αυτό θέλει, και απλά να κάθεται και να επεξεργάζεται μηνύματα που έρχονται στην message queue του. Όμως αν θέλει έχει την δυνατότητα:

1. Να δημιουργεί ή να καταστρέφει ουρές μηνυμάτων.
2. Να καθορίσει τον τρόπο με τον οποίο γεμίζουν οι message queues, αφού έχει την δυνατότητα να ορίζει αυτός τα bindings που ελέγχουν τα exchanges.
3. Να επιλέγει διαφορετικά exchanges

Παρατηρούμε ότι το AMQP είναι περισσότερο σαν γλώσσα για να συνδέεις διαφορετικά components μεταξύ τους, παρά ένα σύστημα. Αυτός εξάλλου ήταν και ο σκοπός των δημιουργών του AMQP αφού θέλανε να κάνουνε την συμπεριφορά του server προγραμματίσιμη μέσω του πρωτοκόλλου.

4.3 Virtual Hosts

Ένα virtual host περιλαμβάνει το δικό του χώρο ονομάτων (namespace), τις δικές του ουρές και exchanges. Κάθε σύνδεση πρέπει να είναι συνδεδεμένη με ένα virtual host.

Ο client επιλέγει τον virtual host μετά το authentication.

4.4 Exchanges

Ένα exchange είναι ένας routing agent μέσα σε ένα virtual host. Το exchange δέχεται μηνύματα και πληροφορία δρομολόγησης και προωθεί το μήνυμα σε κάποια ουρά.

Οι εφαρμογές μπορούν ελεύθερα να δημιουργούν, μοιράζονται, χρησιμοποιούν και να καταστρέφουν exchange instances μέσα στα πλαίσια των ελευθεριών που τους δίνονται από τον admin του server.

Γενικότερα ένα exchange μπορεί να έχει τις παρακάτω δυνατότητες:

- Durable

  Ένα durable exchange διαρκεί μέχρι να διαγραφεί

- Transient

  Ένα Transient exchange διαρκεί μέχρι να κλείσει ο server

- Auto-deleted

  Ένα auto-deleted exchange διαρκεί μέχρι να σταματήσει να χρησιμοποιείται

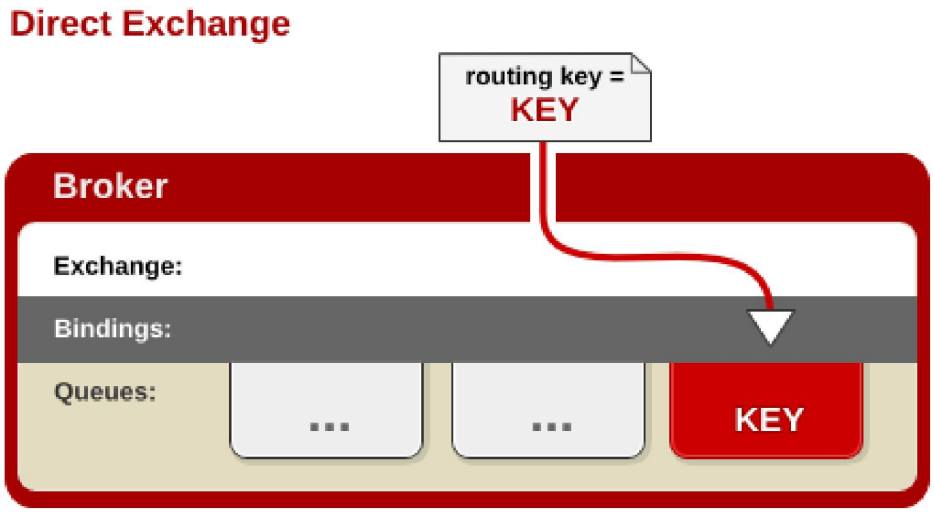
Το πρότυπο ορίζει ένα σύνολο από exchange types, τα οποία πρέπει να υλοποιεί οποιοσδήποτε AMQP server. Κάθε exchange type υλοποιεί ένα συγκεκριμένο routing αλγόριθμο. Τα exchange types στα οποία αναφερόμαστε είναι:

- direct exchange type (υποχρεωτική υλοποίηση)
- fanout exchange type (υποχρεωτική υλοποίηση)

- topic exchange type (προαιρετική υλοποίηση)
- headers exchange type
- system exchange type

4.4.1 direct exchange type

Με το direct exchange έχουμε δρομολόγηση βασισμένη στο ταίριασμα του κλειδιού δρομολόγησης και του binding key, δηλαδή του κλειδιού που έχουμε χρησιμοποιήσει στο binding του exchange με την ουρά.



Με αυτό τον τύπο μπορούμε να κατασκευάσουμε point-to-point messaging μοντέλα.

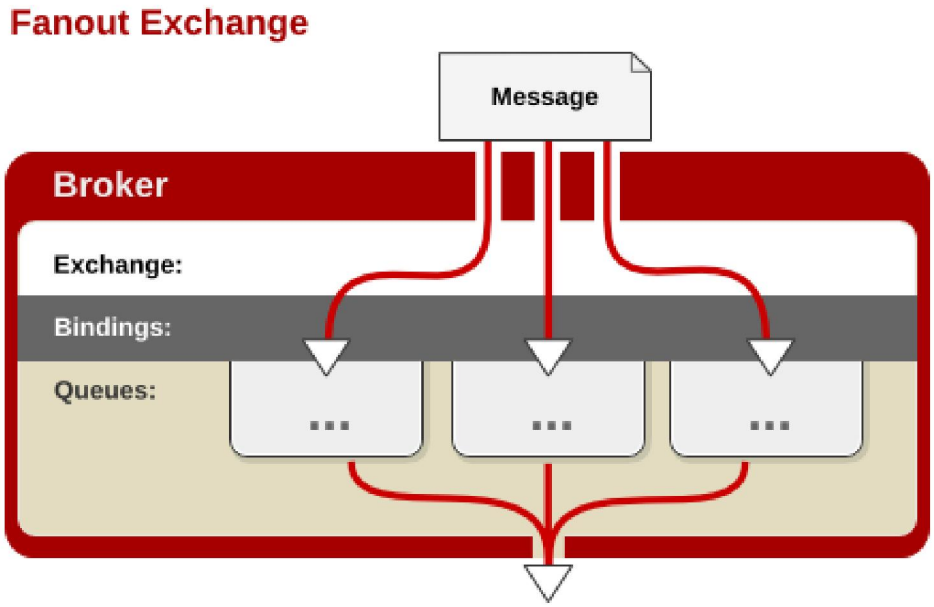
Ο τρόπος λειτουργίας του συγκεκριμένου exchange είναι:

1. Μια ουρά «δένεται» σε ένα exchange με τη χρήση ενός κλειδιού K(binding key, K).
2. Ένας παραγωγός στέλνει στο exchange ένα μήνυμα με κλειδί δρομολόγησης R
3. Το μήνυμα προωθείται σε όλες τις ουρές που είναι δεμένες στο exchange και έχουν κλειδί K όπου K=R

με τον όρο «δένεται» αναφερόμαστε στην διαδικασία του binding που αναφέρουμε πιο πάνω.

4.4.2 fanout exchange type

Η fanout exchange δρομολογεί μηνύματα σε όλες τις ουρές που είναι «δεμένες» μαζί της, άσχετα από το routing key του μηνύματος.

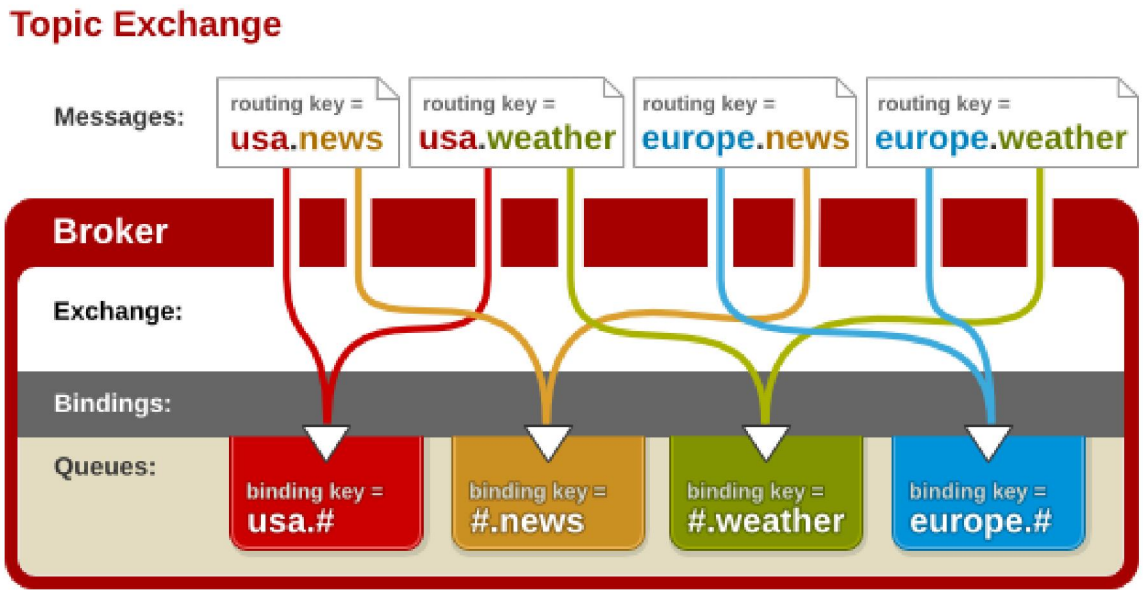


Ο τρόπος λειτουργίας του συγκεκριμένου exchange είναι:

1. Μια ουρά δένεται σε ένα exchange χωρίς κάποιο κλειδί
2. Ένας παραγωγός στέλνει στο exchange ένα μήνυμα
3. Το μήνυμα προωθείται σε όλες τις ουρές που έχουν «δεθεί» με το συγκεκριμένο exchange, χωρίς να είναι ανάγκη να ισχύει κάποια συνθήκη.

4.4.3 topic exchange type

Με το topic exchange έχουμε δρομολόγηση μηνυμάτων βασισμένη σε κάποιο μοτίβο ανάμεσα στο binding key και στο routing key του μηνύματος. Αυτός ο τύπος χρησιμοποιείται για να υποστηρίξει το κλασσικό **public/subscribe paradigm** χρησιμοποιώντας το namespace σαν μοντέλο διευθυνσιοδότησης για να επιλέξει μηνύματα και να τα δρομολογήσει ανάμεσα σε πολλαπλούς καταναλωτές βασιζόμενο σε μερικό ή ολικό ταίριασμα του κλειδιού δρομολόγησης με ένα topic pattern.



Ο τρόπος λειτουργίας του συγκεκριμένου exchange είναι:

1. Μια ουρά δένεται σε ένα exchange χρησιμοποιώντας ένα κλειδί K (binding key, K)
2. Ένας παραγωγός στέλνει ένα μήνυμα σε μια ουρά με κλειδί δρομολόγησης R
3. Το μήνυμα περνιέται σε όλες τις ουρές όπου το K ταιριάζει με το R.

Το binding key σχηματίζεται από την συνένωση πολλών tokens όπου το κάθε token ξεχωρίζει από το άλλο από τον χαρακτήρα '.' (τελεία – dot). Παράλληλα στο κλειδί υποστηρίζονται και wild-chars:

- '*' : αντιστοιχεί σε μια λέξη
- '#' : αντιστοιχεί σε καμμία ή περισσότερες λέξεις

για παράδειγμα το binding key "*.stock.#" ταιριάζει με τα κλειδιά δρομολόγησης "usd.stock", "eur.stock.db" αλλά όχι με το "stock.nasdaq".

4.4.4 headers exchange type

Με το headers exchange μας δίνεται η δυνατότητα για πιο σύνθετη δρομολόγηση μηνυμάτων βασισμένη στα header properties του μηνύματος. Επειδή δεν την χρειαστήκαμε για την διπλωματική μας, δεν θα την αναλύσουμε περαιτέρω.

4.4.5 system exchange type

Ο τρόπος λειτουργίας του συγκεκριμένου exchange είναι:

1. ο παραγωγός στέλνει ένα μήνυμα στο exchange με κλειδί S
2. το system exchange βασισμένο σε αυτό το κλειδί καλεί ένα service του συστήματος S

4.5 Ο κύκλος ζωής ενός exchange instance

Οι AMQP εφαρμογές δημιουργούν τα δικά τους exchanges ή χρησιμοποιούν τα προκαθορισμένα exchanges του server. Το AMQP δεν χρησιμοποιεί κάποια «create» command αλλά μια «declare», η οποία σημαίνει : «αν δεν υπάρχει δημιούργησε, αλλιώς προχώρησε».

Είναι πιθανό εφαρμογές να δημιουργούν τις δικές τους exchanges και μετά να τις καταστρέφουν όταν τελειώσει η χρήση τους.

4.6 Message Queues (ουρές μηνυμάτων)

Μια ουρά είναι ένα buffer με συγκεκριμένο όνομα το οποίο κρατάει μηνύματα εκ μέρους ενός συνόλου εφαρμογών-καταναλωτών (το σύνολο μπορεί να είναι περιλαμβάνει και μόνο μια εφαρμογή).

Οι εφαρμογές στα πλαίσια τις ελευθερίας που τους δίνει ο administrator τις ουράς μπορούν να δημιουργήσουν, καταστρέψουν, μοιραστούν και να χρησιμοποιήσουν κάποια ουρά.

Μια ουρά μπορεί να έχει τα παρακάτω χαρακτηριστικά:

- Durable

  Μια durable ουρά διαρκεί μέχρι να διαγραφεί ,ακόμα και αν ο server κάνει restart. Μια τέτοια ουρά μπορεί να χάσει μηνύματα που δεν είναι persistent στην περίπτωση που ο server κάνει restart.

- Transient

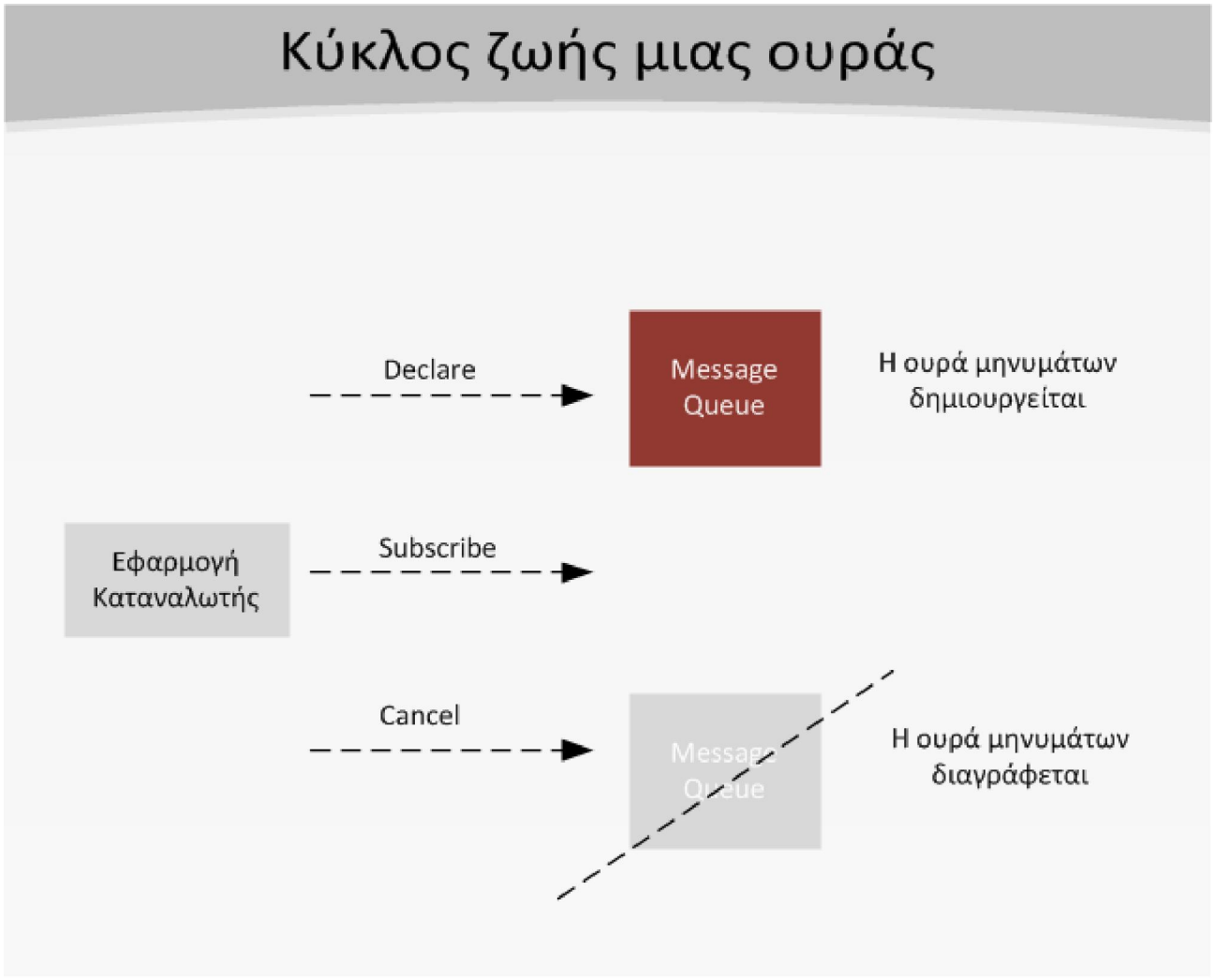
  Μια transient ουρά διαρκεί μέχρι να κλείσει ο server

- Auto-deleted

  Μια auto-deleted ουρά διαρκεί μέχρι να σταματήσει να χρησιμοποιείται

4.6.1 Κύκλος ζωής μιας ουράς (Queue Life-cycle)

Από τα παραπάνω προκύπτει και ο κύκλος ζωής μιας ουράς.



Έτσι έχουμε:

Durable message queues: αυτές υπάρχουν και μαζεύουν μηνύματα ανεξάρτητα με το αν έχουμε subscribers να τα πάρουν.

Temporary message queues: αυτές οι ουρές είναι ιδιωτικές σε κάποιον subscriber και υπάρχουν όσο υπάρχει και αυτός.

Το πρωτόκολλο παρέχει την δυνατότητα σε ένας χρήστη με τα κατάλληλα permissions να μπορεί να διαγράψει μια ουρά, οπότε το θελήσει . Τέλος σημειώνουμε ότι οι ουρές μηνυμάτων υπάρχουν μέσα σε κάποιο virtual host.

4.7 Bindings

Ένα binding είναι μια σχέση ανάμεσα σε μια ουρά μηνυμάτων και ένα exchange . Πιο συγκεκριμένα το binding ορίζει πια μηνύματα πρέπει να πάνε σε κάποια ουρά. Οι εφαρμογές δημιουργούν και καταστρέφουν bindings όπως θέλουν έτσι ώστε να κατευθύνουν την ροή μηνυμάτων μέσα στις ουρές τους.

Ο κύκλος ζωής ενός binding εξαρτάται από την ουρά ή το exchange για το οποίο έχει οριστεί, δηλαδή όταν μια ουρά ή ένα exchange καταστρέφεται μαζί του καταστρέφονται και τα bindings του.

Τα bindings δημιουργούνται από την εφαρμογή η οποία έχει τον έλεγχο μιας ουράς.

4.8 Messages

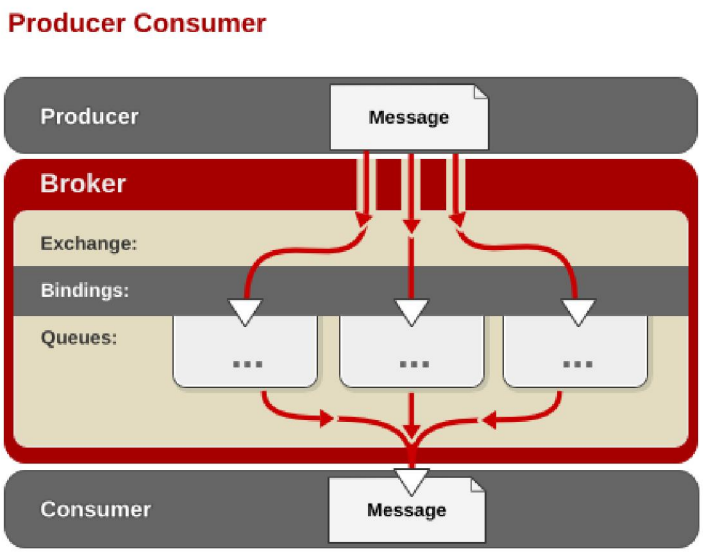
Ένα μήνυμα είναι μια οντότητα που δρομολογείται και αποθηκεύεται μέσα σε μια ουρά. Τα μηνύματα έχουν επικεφαλίδες με συγκεκριμένο σύνολο από ιδιότητες και ένα σώμα που αποτελεί αδιαφανές block από binary δεδομένα, δηλαδή δεν μεταφέρει κείμενο όπως πχ ένα http message .

Ένα μήνυμα μπορεί να είναι persistent δηλαδή να αποθηκεύεται για ασφάλεια μέσα στον δίσκο του broker μας έτσι ώστε να είναι σίγουρο ότι θα παραδοθεί ακόμα και αν έχουμε ένα σοβαρό network failure , ένα server crash , κτλ.

Τα μηνύματα έχουν επίπεδα προτεραιότητας. Έτσι, με δεδομένη μια ουρά, ένα μήνυμα με υψηλή προτεραιότητα θα παραδοθεί πριν από ένα μήνυμα με χαμηλή προτεραιότητα ακόμα και αν αυτό με την χαμηλή προτεραιότητα στάλθηκε πρώτο.

Ο server δεν τροποποιεί το σώμα των μηνυμάτων, αλλά μπορεί να αλλάξει συγκεκριμένες κεφαλίδες πριν προωθήσει το μήνυμα στην εφαρμογή καταναλωτή.

Την όλη ιδέα του πρωτοκόλλου μπορούμε να την συνοψίσουμε με την παρακάτω εικόνα.



4.9 RabbitMQ

Είδαμε μια μικρή περίληψη του AMQP και των λόγων που μας έκαναν να το επιλέξουμε για να καλύψουμε τις messaging ανάγκες του συστήματός μας. Ψάχνοντας τα διάφορα enterprise messaging συστήματα καταλήξαμε οτι το RabbitMQ θα μας βόλευε περισσότερο.

Το RabbitMQ είναι ένας message broker ανοιχτού λογισμικού, γραμμένος σε erlang και βασισμένος πάνω στο Open Telecom Platform framework . Ο πηγαίος κώδικας διατίθεται με άδεια Mozilla Public License.

Υπάρχουν αρκετοί λόγοι για τους οποίους μπορεί κάποιος να επιλέξει το RabbitMQ. Ο server αυτός ξεχωρίζει για το high reliability, availability και scalability που προσφέρει παράλληλα έχει ικανοποιητικές αποδόσεις στους τομείς του throughput και του latency. Είναι cross-platform, έχει ενεργή κοινότητα και χρησιμοποιείται σε αρκετά συστήματα.

Για να μιλήσουμε στον RabbitMQ χρησιμοποιήσαμε την βιβλιοθήκη py-amqplib[20].

[20] http://barryp.org/software/py-amqplib

5 HTTP protocol

Ένα από τα πιο βασικά components του συστήματος μας είναι ο proxy server, παράλληλα το ίδιο το σύστημα μας έχει στόχο να βοηθήσει το collaboration μεταξύ οντοτήτων για πιο αποτελεσματική χρήση των HTTP ροών. Για να την υλοποίηση του συστήματος μας χρειάστηκε να ασχοληθούμε εκτενώς με το HTTP πρωτόκολλο.

Το HTTP ξεκίνησε ως ένα πολύ απλό πρωτόκολλο με μοναδικό σκοπό να επιτρέπει σε κάποιον client να στέλνει ένα request για ένα αρχείο υπερκείμενου (hypertext file) και να το λαμβάνει ως απάντηση από τον server.

Η πιο σύγχρονη μορφή του HTTP παραμένει στην ουσία του ίδιο, δηλαδή ένα request/reply πρωτόκολλο. Βέβαια τώρα περιλαμβάνει πολλά καινούρια χαρακτηριστικά και δυνατότητες με σκοπό να υποστηρίξει το όλο ένα και αυξανόμενο μέγεθος του Web , αλλά και τις διαφορετικές χρήσεις που κάνουμε σε αυτό(το Web). Έτσι ο καλύτερος τρόπος για να το εξηγήσουμε είναι να κοιτάξουμε την λειτουργία του σαν σύνολο και πως η επικοινωνία λαμβάνει χώρα ανάμεσα σε ένα client και ένα server .

Η ανάπτυξη του HTTP έγινε υπό την εποπτεία του World Wide Web Consortium και του Internet Engineering Task Force (IETF). Η πρώτη έκδοση του πρωτοκόλλου αναφέρεται με το όνομα HTTP/0.9 ενώ οι 2 μεταγενέστερες αναφέρονται ως HTTP/1.0 και HTTP/1.1.

5.1 HTTP/0.9

Σε αυτήν την έκδοση το πρωτόκολλο υποστήριζε μόνο την μεταφορά hypertext documents και τίποτα άλλο. Ορίζεται ότι ένας client δημιουργεί (establish) μια σύνδεση με τον HTTP server χρησιμοποιώντας το TCP[21](Transmission Control Protocol).

Στην συνέχεια ο client στέλνει ένα GET request στο οποίο προσδιορίζει ποιο resource θέλει να του απαντήσει ο server. Ο server απαντάει στέλνοντας το ζητηθέν resource σαν ένα stream of text bytes και η σύνδεση τερματίζεται.

5.2 HTTP/1.0

Όπως μεγάλωνε το Web , πολλές καινούριες ιδέες βρήκαν τον δρόμο τους στο HTTP . Το αποτέλεσμα όλης αυτής της ανάπτυξης ήταν η δημιουργία του πρώτο επίσημου HTTP πρότυπου.

[21] http://en.wikipedia.org/wiki/Transmission_Control_Protocol

Το HTTP/1.0 βγήκε το Μάιο του 1996 με το RFC 1945, βέβαια χρησιμοποιούταν αρκετά χρόνια πριν επισημοποιηθεί από το RFC.

Το HTTP/1.0 μεταμόρφωσε το πρωτόκολλο από ένα απλό request/response πρωτόκολλο, σε ένα πραγματικό και πλήρες messaging πρωτόκολλο.

Περιέγραφε ένα πλήρες format για τα μηνύματα του HTTP και εξηγούσε πως πρέπει να χρησιμοποιηθεί για την επικοινωνία ενός client με ένα server.

Μια από της πιο σημαντικές αλλαγές ήταν η αλλαγή του πρωτόκολλου, έτσι ώστε να υποστηρίζει πολλών ειδών resources και όχι μόνο hypertext documents.

Για να διευρύνουν το scope του HTTP , οι δημιουργοί του δανείστηκαν ιδέες και headers από το Multipurpose Internet Mail Extensions[22] (MIME) πρότυπο, το οποίο είχε αναπτυχθεί για το email.

Το πρωτόκολλο όμως είχε ελλείψεις που αφορούσαν κυρίως το scalability. Ενδεικτικά αναφέρουμε ότι κάθε site έπρεπε να φιλοξενείται σε ένα server.

5.3 HTTP/1.1

Το HTTP/1.1 εμφανίστηκε σαν draft στο RFC 2068, η τελική του μορφή δημοσιεύτηκε σαν RFC 2616 τον Ιούνιο του 1999.

- Το πρωτόκολλο διατηρεί προς τα πίσω συμβατότητα με τα HTTP/1.0 και HTTP/0.9.
- Συνοδεύεται από το RFC 2617 που ασχολείται με θέματα authentication και ασφαλείας.
- Το HTTP/1.1 λειτουργεί χρησιμοποιώντας persistent connections (θα δούμε στην συνέχεια τι είναι αυτές).
- Τέλος το πρωτόκολλο φέρνει αρκετές βελτιώσεις σε σχέση με τους προκατόχους του.

5.4 Βασικό μοντέλο επικοινωνίας

Στην πιο απλή του μορφή το HTTP χρειάζεται έναν HTTP client , ο οποίος συνήθως είναι ένας web browser και έναν HTTP server οι οποίοι συνήθως λέγονται web servers. Αφού δημιουργηθεί μια TCP σύνδεση, οι επικοινωνία έχει τα παρακάτω 2 βήματα:

Client Request

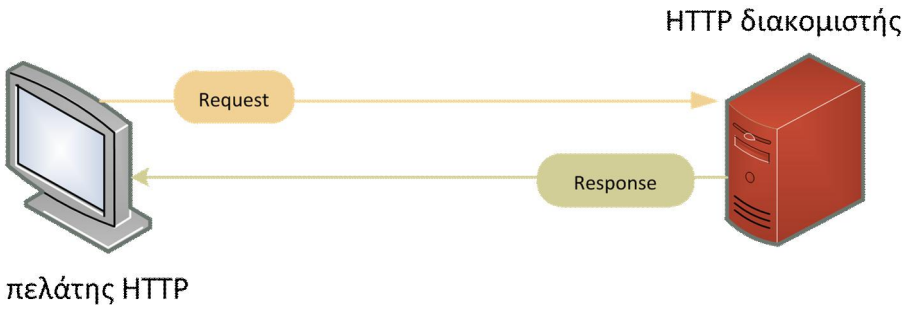
[22] http://en.wikipedia.org/wiki/MIME

Ο HTTP client στέλνει ένα request μήνυμα, το οποίο έχει τη δομή που ορίζεται από το πρωτόκολλο (θα την δούμε παρακάτω).

Το μήνυμα περιλαμβάνει είτε τον πόρο που ο client θέλει να ανακτήσει (retrieve), είτε πληροφορίες για τον ίδιο τον server .

Server Response

Ο server διαβάζει και ερμηνεύει το request μήνυμα του client και δημιουργεί ένα μήνυμα απάντησης - HTTP response μήνυμα το οποίο το στέλνει πίσω στον πελάτη. Το μήνυμα απάντησης περιέχει πληροφορία αν η αίτηση του πελάτη υπήρξε επιτυχής και μπορεί ακόμα να περιέχει το περιεχόμενο του πόρου που ζήτησε ο πελάτης.



Το πρωτόκολλο HTTP είναι η transaction - driven. Αυτό σημαίνει ότι κάθε request message θα οδηγήσουν για μία και μόνο μία response message.

Είδαμε λίγο πιο πάνω πως ο πελάτης θα ανοίξει -"εγκαταστήσει" μια σύνδεση TCP με τον server, ο server θα του απαντήσει και η σύνδεση θα κλείσει. Αν ο πελάτης θέλει να στείλει και άλλο μήνυμα θα χρειαστεί να ανοίξει μια καινούρια σύνδεση.

`[CON1] [REQ1] [RESP1] [CLO1] [CON2] [REQ2] [RESP2] [CLO2] ...`

όπου ισχύουν οι παρακάτω αντιστοιχίσεις:

- `CON: connection`
- `REQ: request`
- `RESP: response`
- `CLO: close connection`

Δηλαδή σε αυτή την λειτουργία του πρωτοκόλλου τα connections είναι όσα τα http transactions. Μια λεπτομέρεια: αφού ο server κλείνει το connection, μετά από κάθε απάντηση, ο πελάτης δεν χρειάζεται να ξέρει το μήκος του περιεχομένου του μηνύματος απάντησης. Αυτός ο τρόπος λειτουργίας ονομάζεται transitory connection.

Το πλεονέκτημα της λύσης αυτής είναι η απλότητα. Το πρόβλημα που δημιουργεί είναι η έλλειψη απόδοσης όταν ο client θέλει να κάνει πολλά requests στον server. Αυτό συμβαίνει αρκετά στα σύγχρονα hypertext documents τα οποία έχουν references σε εικόνες και άλλα media. Έτσι ένα αρχικό request για μια σελίδα συνήθως οδηγεί σε πολλά ακόμα requests στον ίδιο server για σχετικά με την σελίδα αντικείμενα. Κάθε ένα request όμως δημιουργεί και νέα TCP σύνδεση και κάθε νέα σύνδεση κοστίζει σε πόρους στον server και σε network bandwidth.

Η λύση στο παραπάνω πρόβλημα ήρθε με το HTTP/1.1 που επιτρέπει σε ένα HTTP client και server να ανοίξουν μια "ανθεκτική σύνδεση" ή persistent connection.

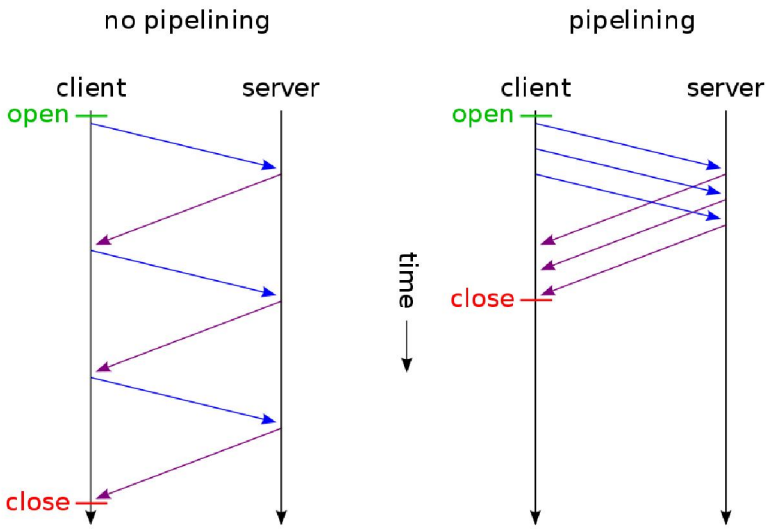
Με τις persistent connections δεν αλλάζει η βασική λειτουργία του HTTP. Η κύρια διαφορά είναι ότι η TCP σύνδεση μένει ανοιχτή μετά από κάθε ζευγάρι request/response, ώστε η επόμενη επικοινωνία να μην χρειαστεί νέα σύνδεση. Το session κλείνει όταν ο client δεν έχει άλλες αιτήσεις.

[CON] [REQ1] ... [RESP1] [REQ2] ... [RESP2] ... [CLO]

Η τελευταία βελτίωση που θα συναντήσουμε είναι το pipelining των requests. Αυτή δουλεύει ως εξής:

Έστω ότι ο client θέλει να ζητήσει από τον server τα αρχεία A, B, C. Δεδομένου ότι τα αρχεία αυτά θα τα ζητήσει στο ίδιο TCP connection, δεν υπάρχει λόγος να περιμένει την απάντηση του server όταν ζητήσει ένα αρχείο και πριν ζητήσει το επόμενο. Έτσι ο client μπορεί να στείλει όλα τα requests μαζί και να περιμένει τον server να απαντήσει και για τα 3.

[CON] [REQ1] [REQ2] ... [RESP1] [RESP2] [CLO] ...



Το μειονέκτημα των persistent connections είναι ότι κάνουν πιο περίπλοκη την επικοινωνία μεταξύ server και client. Με την χρήση transitory connections, ένας client ήξερε ότι όλα τα δεδομένα που παίρνει από ένα server είναι η απάντηση σε ένα μόνο request. Επίσης ξέρει ότι όταν έχει όλα τα bytes που έστειλε ο server και ο server κλείσει το TCP connection, η μεταφορά του αρχείου έχει ολοκληρωθεί.

Με τη χρήση persistent connections και ιδίως με pipelining, ένας server θα στέλνει το ένα αρχείο μετά το άλλο στον client. Ο τελευταίος θα πρέπει να έχει ένα τρόπο να τα ξεχωρίσει, γιατί γνωρίζουμε ότι το TCP στέλνει δεδομένα ως μια ακολουθία από bytes χωρίς καμμία δομή και ότι είναι ευθύνη της εφαρμογής να ορίσει τρόπους για να ξεχωρίζουν τα αρχεία μεταξύ τους.

Με πιο λίγα λόγια η χρήση persistent connections γεννάει ζητήματα data-length τα οποία πρέπει να αντιμετωπιστούν στο HTTP.

Είπαμε πιο πάνω ότι το HTTP/1.1. είναι backwards compatible με τα προηγούμενα HTTP πρωτόκολλα, αυτό πρακτικά σημαίνει ότι υποστηρίζει και transitory connections.

5.5 HTTP Messages

Είδαμε ότι όλο το HTTP πρωτόκολλο στήνεται πάνω στα HTTP messages που ανταλλάσσουν οι clients και οι servers μεταξύ τους. Το πρωτόκολλο ορίζει πολύ συγκεκριμένα το format αυτών των μηνυμάτων.

Τα HTTP μηνύματα χωρίζονται σε 2 τύπους: response messages και request messages. Συνήθως οι clients στέλνουν request messages και οι servers response messages . Οι πιθανοί ενδιάμεσοι κόμβοι (πχ proxies) μπορούν να στέλνουν και τα 2 είδη μηνυμάτων ανάλογα με την περίσταση.

5.5.1 Generic Message Format

Και τα 2 είδη μηνυμάτων προκύπτουν από ένα τύπο μηνύματος που στο standard ονομάζεται generic message format. Εδώ να σημειώσουμε ότι όπως και στα υπόλοιπα πρωτόκολλα της TCP/IP στοίβας, τα μηνύματα είναι text-based και όχι binary.

Το generic message format είναι ως εξής:

1. <start-line>
2. <message-headers>
3. <empty-line>
4. [<message-body>]
5. [<message-trailers>]

Όλες οι γραμμές τερματίζονται με τους carriage return-line feed (CRLF) χαρακτήρες ελέγχου. Η <empty-line> περιέχει μόνο αυτούς τους 2 χαρακτήρες. Το <message-body> μπορεί να είναι είτε κείμενο είτε 8bit binary πληροφορία.

<start-line> Είναι η γραμμή που δείχνει την φύση του μηνύματος. Σε ένα request η γραμμή αυτή περιέχει το όνομα μιας μεθόδου και ένα URI που είναι το αντικείμενο του request.

Από την άλλη ένα response message χρησιμοποιεί αυτήν τη γραμμή για να δώσει πληροφορίες για την έκβαση του request.

<message-headers> Στο πρωτόκολλο ορίζονται αρκετά message-headers. Σχεδόν όλοι είναι προαιρετικοί, μοναδική εξαίρεση είναι ο host header, ο οποίος πρέπει υποχρεωτικά να υπάρχει στο HTTP/1.1. Η σειρά με την οποία στέλνουμε τις κεφαλίδες δεν έχει σημασία, αρκεί να ακολουθείται το παρακάτω format:

<header-name>:<header-value>

[<message-body>] Το σώμα του μηνύματος είναι προαιρετικό, γιατί χρειάζεται σε ορισμένους τύπους μηνυμάτων. Αν υπάρχει, πιθανό να μεταφέρει πληροφορία που θέλουν να ανταλλάξουν ο client και ο server, πχ λεπτομέρειες για κάποιο λάθος.

Άλλη πιθανότητα είναι να μεταφέρει κάποιο αρχείο ή γενικότερα κάποιο resource, το οποίο στην γλώσσα του προτύπου λέγεται οντότητα(entity). Συνήθως entities βρίσκουμε στα σώματα μηνυμάτων απάντησης γιατί οι περισσότεροι clients ζητάνε από τους servers κάποιο αρχείο ή γενικότερα κάποιο resource. Βέβαια μπορούν να βρεθούν και σε συγκριμένα requests messages.

[<message-trailers>] Είδαμε πριν ότι το HTTP/1.1 χρησιμοποιεί persistent connections by default. Αυτό σημαίνει ότι ο server μπορεί στέλνει μηνύματα το ένα μετά το άλλο σε μια σταθερή ροή δεδομένων. Έτσι με κάποιο τρόπο πρέπει να μπορούμε να ξεχωρίσουμε που τελειώνει το ένα μήνυμα και που ξεκινάει το άλλο, για να το καταφέρουμε αυτό έχουμε 2 τρόπους. Ο πρώτος είναι να χρησιμοποιήσουμε ένα ειδικό header που θα μας δείχνει το μήκος του μηνύματος. Ο δεύτερος τρόπος ονομάζεται **chunking** επί της ουσίας είναι όταν ένα μήνυμα χωρίζεται σε κομμάτια για μετάδοση και το μέγεθος των κομματιών αυτών εμφανίζεται μέσα στο message body. Όταν το chunking τελειώσει, συγκεκριμένοι message trailers ακολουθούν το σώμα του μηνύματος.

Στην πράξη οι trailers έχουν ακριβώς την ίδια δομή με τους headers, το μόνο που αλλάζει είναι η θέση τους στο μήνυμα.

5.5.2 HTTP Request Message Format

Ο client ξεκινάει ένα HTTP session ανοίγοντας ένα TCP connection στον HTTP server με τον οποίο επιθυμεί να επικοινωνήσει. Στην συνέχεια στέλνει HTTP Request Messages στον server, κάθε ένα από τα οποία προσδιορίζει ένα τύπο δράσης που ο client θέλει τον server να κάνει. Τέτοια requests μπορούν να γίνουν είτε από συγκεκριμένες πράξεις του χρήστη (πχ κλικ σε έναν υπερσύνδεσμο μιας σελίδας) είτε έμμεσα σαν αποτέλεσμα προηγούμενης δράσης του χρήστη (όπως το request για ένα html document που περιέχει αναφορές σε άλλα στοιχεία).

Το format του HTTP Request Message είναι το εξής:

1. <request-line>
2. <general-headers>
3. <request-headers>
4. <entity-headers>
5. <empty-line>
6. [<message-body>]
7. [<message-trailers>]

ακολουθεί ένα παράδειγμα

GET /path/file.html HTTP/1.1	**Request Line**
Date: Fri, 31 Dec 1999 23:59:59 GMT **Connection: close**	General Headers
Host: www.ceid.upatras.gr **From: someuser@ceid.upatras.gr** **Accept: text/html, text/plain** **User-Agent: Firefox**	Request Headers
	Entity Headers
	Message Body

5.5.2.1 Request Line

Η request-line έχει 3 σκοπούς:

- Να δείχνει την δράση (action) που ο client θέλει να γίνει
- Να δείχνει τον πόρο πάνω στον οποίο πρέπει να γίνει η δράση (action)
- Να δείχνει στον server ποια έκδοση του HTTP ο client χρησιμοποιεί

Το συντακτικό της γραμμής είναι το εξής:

```
<METHOD><request-uri><HTTP-VERSION>
```

ακολουθεί ανάλυση για τα επιμέρους στοιχεία

<METHOD>

προσδιορίζεται πάντα με κεφαλαία. Στο πρότυπο ορίζονται 8 μέθοδοι από τις οποίες συνήθως χρησιμοποιούνται οι εξής: GET, HEAD, POST

<request-uri>

Το URI μπορεί θεωρητικά να είναι είτε Uniform Resource Locator (URL) είτε Uniform Resource Name (URN). Στην πράξη το URI είναι ένα HTTP URL. Είναι ενδιαφέρον ότι η μορφή του URL διαφέρει σε σχέση με αυτή που χρησιμοποιούμε στα HTML documents.

πχ ένα URL που θα γράφαμε έτσι:

```
www.ceid.upatras.gr:8080/path/file.html
```

ο client για να το στείλει στον server θα το έκανε έτσι:

```
GET /path/file.html HTTP/1.1

Host: www.ceid.upatras.gr:8080
```

Εξαίρεση σε αυτόν τον κανόνα αποτελεί όταν ο client θα στείλει το request σε κάποιον proxy. Τότε το request θα γίνει κάπως έτσι:

```
GET www.ceid.upatras.gr:8080/path/file.html HTTP/1.1
```

έτσι ώστε ο proxy να μπορεί να το επεξεργαστεί όπως ο original client.

<HTTP-VERSION>

Αυτό το element το βάζουμε έτσι ώστε ο server να ξέρει πως να διαμορφώσει την απάντηση που θα στείλει.

5.5.2.2 Headers

Πριν χωρίσαμε τους headers σε 3 κατηγορίες:

General Headers

Οι Headers αυτοί αναφέρονται κυρίως στο ίδιο το μήνυμα, και όχι στα περιεχόμενα του. Συνήθως χρησιμοποιούνται για να δώσουν επιπλέον πληροφορία στον αποδέκτη ή να ελέγξουν το πως θα γίνει η επεξεργασία του μηνύματος.

Request Headers

Οι Headers αυτοί δίνουν πληροφορία για τη φύση του request. Πχ σε αυτή την κατηγορία ανήκουν headers που κάνουν ένα conditional request. Άλλοι ενημερώνουν τον server τι encoding μπορεί να επεξεργαστεί ο client.

Entity Headers

Οι Headers αυτοί περιγράφουν την οντότητα που περιέχεται στο μήνυμα, αν περιέχεται κάποια οντότητα.

5.5.3 HTTP Response Message Format

Κάθε request message που φτάνει σε ένα server προκαλεί την δημιουργία ενός response message.

Το format του HTTP Response Message είναι το εξής:

1. <status-line>
2. <general-headers>
3. < response-headers>
4. <entity-headers>
5. <empty-line>
6. [<message-body>]
7. [<message-trailers>]

ακολουθεί ένα παράδειγμα

HTTP/1.1 200 OK	**Status Line**
Date: Fri, 31 Dec 1999 23:59:59 GMT **Connection: close**	General Headers
Server: Apache **Accept-Ranges: bytes**	Response Headers
Content-Type: text/html **Content-Length: 1354** **Last-Modified: Fri, 31 Dec 1999 23:59:59 GMT**	Entity Headers
<html> **<body>** ** Happy Thesis! ** **</body>** **</html>**	Message Body

5.5.3.1 Status Line

Η status-line έχει 2 λειτουργίες:

- Να ενημερώνει τον client τι έκδοση του HTTP χρησιμοποιεί ο server
- Να δίνει στον client μια περίληψη του τι έγινε με το request του.

Το συντακτικό της γραμμής είναι το εξής:

<HTTP-VERSION><status-code><reason-phrase>

ακλουθεί ανάλυση για τα επιμέρους στοιχεία όπου αυτά διαφέρουν από την ανάλυση που κάναμε για τα request messages.

<status-code> και <reason-phrase>

Και τα 2 στοιχεία παρέχουν στον client πληροφορίες σχετικά με τα αποτελέσματα από την επεξεργασία του server.

- Το status-code είναι ένας 3ψήφιος αριθμός που δείχνει το "επίσημο" αποτέλεσμα της επεξεργασίας. Είναι σχεδιασμένο έτσι ώστε να το επεξεργάζεται το λογισμικό του client και να δρα ανάλογα.
- Αντίθετα το reason-phrase είναι φτιαγμένο για να το διαβάζουν οι άνθρωποι-χρήστες του πρωτοκόλλου.

5.6 HTTP Methods

Είναι προφανές ότι ο client επικοινωνεί με κάποιον server, για να πει στον server να κάνει κάτι.

Ο τρόπος με τον οποίο ο client ενημερώνει τον server για το τι θέλει να κάνει είναι μέσω συγκεκριμένων μεθόδων που δίνει το HTTP πρωτόκολλο και έχουν τον ρόλο εντολών.

Θα αναλύσουμε τις πιο συχνές μεθόδους που χρησιμοποιούνται.

GET

Αυτή η μέθοδος ζητάει από τον server να επιστρέψει τον πόρο που δείχνει το URL στο request-line του μηνύματος. Αυτή είναι πιο συχνά χρησιμοποιούμενη μέθοδος, καθώς είναι αυτή που συναντάμε όταν πατάμε σε κάποιο hyperlink ή όταν γράφουμε μια διεύθυνση στον browser.

HEAD

Αυτή η μέθοδος είναι ίδια με την GET με μια μικρή διαφορά. Ο client λέει στον server να ετοιμάσει την απάντηση αλλά να μην συμπεριλάβει σε αυτήν το κυρίως σώμα. Έτσι ο server στέλνει στον client μόνο τους headers.

Πολλές φορές αυτή η μέθοδος χρησιμοποιείται για να ελέγξει ο client την ύπαρξη ενός resource ή το μέγεθος του, πριν αποφασίσει ότι το θέλει.

POST

Αυτή η μέθοδος επιτρέπει στον client να στείλει στον server μια entity που να περιέχει arbitrary δεδομένα. Χρησιμοποιείται συνήθως όταν θέλουμε ένας client να στείλει πληροφορία που απαιτείται πχ για την συμπλήρωση κάποιας φόρμας.

Το URL στο request-line του μηνύματος χρησιμοποιείται για να καθορίσει το όνομα του προγράμματος στον server που θα δεχθεί τα δεδομένα.

5.7 HTTP Messages

Οι HTTP headers είναι αρκετοί για να τους αναλύσουμε εδώ. Σε αυτό το εδάφιο θα περιγράψουμε το connection header, τον οποίο τις περισσότερες φορές τον συναντάμε με αυτήν τη μορφή.

```
Connection: close
```

Ανήκει στην κατηγορία των general headers. Στο HTTP/1.1 έχουμε την εισαγωγή των hop-by-hop headers, δηλαδή των headers που ισχύουν για μια μόνο connection και όχι για ολόκληρο το μονοπάτι. Κάτι που έχει σημασία για την υλοποίηση ενός proxy.

Ο connection:close header μπαίνει σε ένα μήνυμα για να αλλάξει την default συμπεριφορά του HTTP που θέλει να χρησιμοποιεί persistent connections, έτσι ώστε να έχουμε transitory connection.

6 Proxy Server

Έχοντας δει κάποια πράγματα για το HTTP πρωτόκολλο είμαστε έτοιμοι να περάσουμε στην υλοποίηση του proxy.

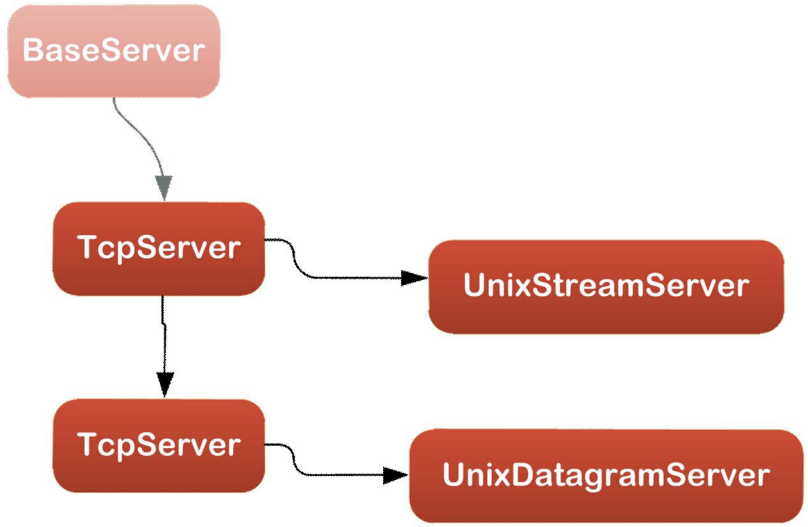
Ένας από τους λόγους που επιλέξαμε να κάνουμε υλοποίηση σε python είναι ότι παρέχει ένα πλήρες framework για την υλοποίηση network servers.

Για την υλοποίηση του Proxy χρησιμοποιήσαμε το SocketServer module της Python, ένα framework που απλοποιεί την δημιουργία network servers. Το SocketServer module ορίζει κλάσεις για τη διαχείριση ασύγχρονων network requests πάνω από TCP, UDP, Unix streams, και Unix datagrams. Παράλληλα παρέχει τις λεγόμενες mix-in κλάσεις, ώστε να μπορούμε εύκολα να δημιουργήσουμε servers που να χρησιμοποιούν ξεχωριστά process ή ξεχωριστά thread, ανάλογα με το τι προτιμάμε.

Η διαχείριση ενός request χωρίζεται ανάμεσα σε μια **server class** και μια **request handler class**. Η server κλάση αναλαμβάνει να διαχειριστεί τα ζητήματα επικοινωνίας (πχ listening on a socket, accepting connections) ενώ η κλάση που κάνει το request handling αναλαμβάνει να ασχοληθεί με ζητήματα που έχουν σχέση με το πρωτόκολλο που θέλουμε να υλοποιήσουμε, πχ την ερμηνεία ή την επεξεργασία των εισερχόμενων δεδομένων. Αυτός ο διαχωρισμός των ευθυνών σε 2 είδη κλάσεων, σημαίνει ότι πολλές φορές μπορείς να χρησιμοποιήσεις κάποιον από τους έτοιμους servers και να υλοποιήσεις μόνο μια handler class για ζητήματα που άπτονται του πρωτοκόλλου.

6.1 Server Types

Υπάρχουν 5 διαφορετικά είδη κλάσεων που ορίζονται στο SocketServer module.



- η κλάση BaseServer ορίζει το API και δεν είναι για να χρησιμοποιείται άμεσα
- η κλάση TCPServer χρησιμοποιεί TCP/IP sockets για να επικοινωνήσει.
- η κλάση UDPServer χρησιμοποιεί datagram sockets.
- οι κλάσεις UnixStreamServerand UnixDatagramServer χρησιμοποιούν Unix-domain sockets και είναι διαθέσιμες μόνο στις πλατφόρμες Unix.

6.2 Server Objects

Για να υλοποιήσουμε ένα server, χρησιμοποιούμε μια server class (από αυτές που αναφέραμε πιο πάνω ή δική μας) στην οποία της περνάμε μια διεύθυνση στην οποία θέλουμε ο server μας να ακούει για requests και μια request handler class (τονίζουμε ότι πρέπει να class και όχι instance).

```
server_class(server_address, handler_class)
```

Μόλις δημιουργήσουμε ένα instance του server μπορούμε να χρησιμοποιήσουμε είτε την handle_request() είτε την serve_forever() (Η δεύτερη απλά καλεί την πρώτη σε ένα infinite loop).

6.3 Implementing a Server

Για την υλοποίηση ενός server τις περισσότερες φορές αρκεί να χρησιμοποιήσουμε κάποιον από τους έτοιμους. Αν δεν μας κάνει μπορούμε να χρησιμοποιήσουμε σαν βάση την BaseServer κλάση, και να κάνουμε override κάποια από τις παρακάτω μεθόδους:

- **verify_request(request, client_address)** - επιστρέφει True για να επεξεργαστούμε ένα request και false για να το αγνοήσουμε. Πχ μπορούμε να την χρησιμοποιήσουμε για να αρνηθούμε requests από ένα συγκεκριμένο εύρος από IPs.
- **process_request(request, client_address)** - Συνήθως αυτή η μέθοδος καλεί την finish_request() που κάνει όλη την δουλειά. Η μέθοδος αυτή χρησιμοποιείται όταν θέλουμε να έχουμε ξεχωριστά threads ή processes για κάθε requst.
- **finish_request(request, client_address)** - Αυτή η μέθοδος δημιουργεί ένα request handler instance χρησιμοποιώντας την κλάση που δώσαμε στον constructor του server object μας. Τέλος η μέθοδος αυτή καλεί την μέθοδο handle() του request handler για να κάνει process το request.

6.4 Request Handlers

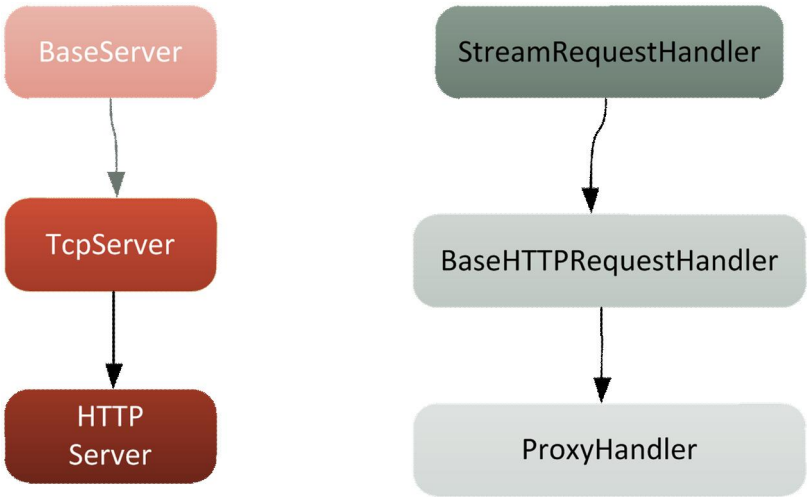
Το τελευταίο συστατικό για να έχουμε έναν ολοκληρωμένο server είναι να δημιουργήσουμε ένα Request handler. Οι Request handlers κάνουν όλοι τη δουλειά όσον αφορά τα incoming requests και το τι πρέπει να γίνει με αυτά. Συνοπτικά θα γράφαμε ότι ο Request handler διαβάζει από το κανάλι δεδομένα-requests, τα επεξεργάζεται και στη συνέχεια γράφει πίσω την απάντηση.

Οι παρακάτω 3 μέθοδοι είναι διαθέσιμοι για να γίνουν overriden.

- **setup()** - Ετοιμάζει τον request handler για το εισερχόμενο request. Για παράδειγμα ο StreamRequestHandler, χρησιμοποιεί αυτήν τη μέθοδο για να δημιουργήσει file-like αντικείμενα objects που τα χρησιμοποιεί για να διαβάζει και να γράφει στο socket.
- **handle()** - Αυτή η μέθοδος αναλαμβάνει να κάνει parse το εισερχόμενο request, να το επεξεργαστεί και να στείλει πίσω μια απάντηση.
- **finish()** - Μέθοδος που αναλαμβάνει το cleanup μετά την επεξεργασία του request, πχ να καθαρίσει ότι δημιουργήθηκε στο cleanup.

Στις περισσότερες των περιπτώσεων αρκεί να κάνουμε override την handle() method.

Ακολουθεί η ιεραρχία των κλάσεων μας, κάποιες από την βιβλιοθήκη της python και κάποιες δικές μας.



Εμείς δημιουργούμε τον server μας με την παρακάτω μέθοδο:

```
def runHTTPserver(server_class=BaseHTTPServer.HTTPServer,
                  handler_class=ProxyHandler):
server_address = (HOST, PORT)
httpd = server_class(server_address, handler_class)
#httpd.serve_forever()
try:
     print 'Use Control-C to exit'
     httpd.serve_forever()
except KeyboardInterrupt:
     print 'Exiting'
```

την οποία την καλούμε με αυτόν τον τρόπο:

```
runHTTPserver(ThreadedHTTPServer, Proxy2ProxyHandler)
```

Η ThreadedHTTPServer είναι υλοποιημένη έτσι:

```
class ThreadedHTTPServer(SocketServer.ThreadingMixIn,
BaseHTTPServer.HTTPServer):
    # This means the main server will not do the equivalent of a
    # pthread_join() on the new threads.  With this set, Ctrl-C will
    # kill the server reliably.
    daemon_threads = True

    # By setting this we allow the server to re-bind to the address by
    # setting SO_REUSEADDR, meaning you don't have to wait for
    # timeouts when you kill the server and the sockets don't get
    # closed down correctly.
    allow_reuse_address = True
    pass
```

ενώ από την υλοποίηση της Proxy2ProxyHandler αξίζει να δούμε την παρακάτω λεπτομέρεια:

Είδαμε στο σημείο που μιλάγαμε για HTTP Request Message Format οτι ο client στέλνει διαφορετικά ένα μήνυμα που έχει σαν τελικό προορισμό ένα web server από ένα μήνυμα που έχει σαν τελικό προορισμό ένα proxy.

Στο py-p2pinet έχουμε proxy που ανάλογα με συγκεκριμένα κριτήρια, επιλέγει αν θα στείλει το μήνυμα στον τελικό server ή θα τον στείλει σε ενδιάμεσο proxy. Στην υλοποίησή μας την δουλειά αυτή αναλαμβάνει να την κάνει η παρακάτω συνάρτηση.

```
def pic_location(netloc, scm):
    f = open(p2pinet.scedulingDecisionFile)
    selectedIP=f.readline().rstrip()

    if selectedIP == myIP:
        print "I am *not* forwarding to another proxy"
        destination_server = netloc
        resource_location  = ''
        resource_scheme    = ''
    else:
        print "Forwarding to another proxy"
        destination_server = "%s:%s"  % ( selectedIP, proxy_port) #now the
destinatation server is a proxy
        resource_location  = netloc
        resource_scheme    = scm

    f.close()
    return destination_server, resource_location, resource_scheme
```

Η pic_location() θα αναλάβει να επιστρέψει ένα tuple με τις εξής πληροφορίες destination_server, resource_location, resource_scheme. Αυτές θα τις χρησιμοποιήσει η Proxy2ProxyHandler για να δημιουργήσει το request ως εξής:

```
str = '%s %s %s\r\n' % (
        self.command,
        urlparse.urlunparse((resourceScheme, resourceLoc, path, params,
query, '')),
        self.request_version)
```

έτσι στην μια περίπτωση θα στείλουμε

```
GET www.ceid.upatras.gr:8080/path/file.html HTTP/1.1
```

ενώ στην άλλη

```
GET /path/file.html HTTP/1.1
```

έτσι ώστε ο επόμενος proxy να μπορεί να χειριστεί το request όπως ο original proxy.

7 Βιβλιογραφία

[1] PERM: A Collaborative System for Residential Internet Access, Nathanael Thompson, Guanghui He, and Haiyun Luo, Dept. of Computer Science, University of Illinois at Urbana-Champaign

[2] Flow Scheduling for End-host Multihoming, Nathanael Thompson, Guanghui He, and Haiyun Luo

Dept. of Computer Science, University of Illinois at Urbana-Champaign

[3] R. Chandra, P. Bahl, and P. Bahl, MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card, in Proceedings of IEEE INFOCOM, 2004.

[4] L. Magalhaes and R. Kravets, Transport level mechanisms for bandwidth aggregation on mobile hosts, in Proceedings of IEEE ICNP, 2001.

[5] H.-Y. Hsieh and R. Sivakumar, A transport layer approach for achieving aggregate bandwidths on multi-home mobile hosts, in Proceedings of ACM MobiCom, 2002.

[6] K. Chebrolu and R. Rao, Communication using multiple wireless interfaces, in Proceedings of IEEE WCNC, March 2002.

[7] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, Resilient overlay networks,in Proceedings of ACM SOSP, 2001.

[8] F. Guo, J. Chen, W. Li, and T. cker Chiueh, Experiences in building a multihoming load balancing system, in Proceedings of IEEE INFOCOM, 2004.

[9] Andersen, D. G., Balakrishnan, H., Kaashoek, M. F., and Morris, R. Resilient overlay networks. In Proceedings of ACM SOSP (2001).

[10] H. Adiseshu, G. Parulkar, and G. Varghese, A reliable and scalable striping protocol, in Proceedings of ACM SIGCOMM, August 1996.

[11] A. Akella, B. Maggsy, S. Seshan, A. Shaikh, and R. Sitaraman, A measurement-based analysis of multihoming, in Proceedings of ACM SIGCOMM, 2003.

[12] Exploring the performance benets of end-to-end path switching, in Proceedings of IEEE ICNP, 2004.

[13] Y. Gao, G. He, and J. C. Hou, On exploiting trafc predictability in active queue management, in Proceedings of IEEE INFOCOM, 2002.

[14] Y. Azar, A. Z. Broder, and A. R. Karlin, On-line load balancing, in Proceedings of ACM Symposium on Foundations of Computer Science, 1992.

[15] Akella, A., Pang, J., Maggsy, B., Seshan, S.,and Shaikh, A. A comparison of overlay routing and multihoming route control. In Proceedings of ACM SIGCOMM (2004).

[17] Akella, A., Seshan, S., and Shaikh, A. Multihoming performance benets: An experimental evaluation of practical enterprise strategies. In Proceedings of USENIX Annual Technical Conference (2004).