ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

# PAWS: ΕΝΑΣ ΔΙΑΚΟΜΙΣΤΗΣ ΙΣΤΟΥ ΥΠΟΒΟΗΘΟΥΜΕΝΟΣ ΑΠΟ ΟΜΟΤΙΜΟΥΣ ΧΡΗΣΤΕΣ

Διπλωματική Εργασία

του

Παναγιωτίδη Αθανάσιου

Θεσσαλονίκη, Νοέμβριος 2019

PAWS: ΕΝΑΣ ΔΙΑΚΟΜΙΣΤΗΣ ΙΣΤΟΥ ΥΠΟΒΟΗΘΟΥΜΕΝΟΣ
ΑΠΟ ΟΜΟΤΙΜΟΥΣ ΧΡΗΣΤΕΣ

Παναγιωτίδης Αθανάσιος

Πτυχίο Λογιστικής και Χρηματοοικονομικής, Πανεπιστήμιο Μακεδονίας, 2013

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ
ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής
Κασκάλης Θεόδωρος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 01/11/2019

| Κασκάλης Θεόδωρος | Μαργαρίτης Κωνσταντίνος | Μαμάτας Ελευθέριος |
|---|---|---|
| ..... | ..... | ..... |

Παναγιωτίδης Αθανάσιος

.....

# Περίληψη

Κάθε χρόνο, ο όγκος των δεδομένων που διακινούνται στον παγκόσμιο ιστό αυξάνεται ραγδαία, χωρίς να διαφαίνονται σημάδια κάμψης αυτής της τάσης. Είναι σύνηθες το φαινόμενο, ένας εξυπηρετητής ιστού να μη μπορεί να καλύψει τις ανάγκες μιας ιστοσελίδας για την διανομή του περιεχομένου της. Έτσι συχνά μισθώνονται υπηρεσίες από παρόχους δικτύων διανομής περιεχομένου. Η λύση αυτή θεωρούμε δεν είναι ικανοποιητική, αφού οι υπηρεσίες αυτές είναι ακριβές και δυσπρόσιτες για μια μεγάλη μερίδα ιστοσελίδων. Για το λόγο αυτό σχεδιάσαμε και αναπτύξαμε το PAWS, έναν εξυπηρετητή ιστού υποβοηθούμενο από ομότιμους χρήστες. Καθώς οι χρήστες του διαδικτύου πλοηγούνται σε ιστοσελίδα που φιλοξενείται από το PAWS, συμμετέχουν σε ένα ομότιμο δίκτυο με στόχο τη διανομή του περιεχομένου της. Παρόμοιες ιδέες έχουν προταθεί ήδη από τις αρχές του 21$^{ου}$ αιώνα, όμως μόνο τα τελευταία χρόνια, χάρη στην εισαγωγή της διεπαφής WebRTC στους φυλλομετρητές ιστού, είναι δυνατή μια πρακτική υλοποίηση, που θα είναι διάφανη στους χρήστες. Προκειμένου να αναπτύξουμε το PAWS, μελετήσαμε τη βιβλιογραφία πάνω στα ομότιμα δίκτυα και τα δίκτυα διανομής περιεχομένου. Επίσης ερευνήσαμε και παρουσιάζουμε άλλες προσπάθειες, παρόμοιες με τη δική μας.

**Λέξεις Κλειδιά:** Ομότιμα δίκτυα διανομής περιεχομένου, Διακομιστής Ιστού Υποβοηθούμενος από Ομότιμους Χρήστες, WebRTC

# Abstract

Every year, the total world wide web traffic increases dramatically, with no signs of decline. It is common for a web server to struggle trying to meet the content distribution needs of a single web page. Often, a content delivery network (CDN) is utilized to deal with the problem. We do not consider such a solution satisfactory, as CDNs are usually too expensive for smaller projects. This is the reason why we designed and developed PAWS, a peer assisted web server. As Internet users navigate to a website hosted by a PAWS instance, they join a private peer-to-peer network in order to assist in the distribution of its content. Similar projects have been proposed since the beginning of the 21st century, but it wasn't until recently, with the addition of the WebRTC API, that a practical and transparent to the user, solution, is feasible. In order to develop PAWS, we first review the existing literature on peer-to-peer networks and CDNs. We also research and present, other similar solutions that have been proposed throughout the years.

**Keywords:** Peer-to-peer content delivery networks, Peer Assisted Web Server, WebRTC

# Περιεχόμενα

# Κατάλογος Εικόνων

# Κατάλογος Πινάκων

# 1 Εισαγωγή

## 1.1 Πρόβλημα – Σημαντικότητα του θέματος

Τη Δευτέρα 23 Οκτωβρίου 2017, το πανεπιστήμιο του Cambridge έκανε ελεύθερα διαθέσιμη στο κοινό τη διδακτορική διατριβή του δημοφιλή επιστήμονα Stephen Hawking. Σύμφωνα με το πανεπιστήμιο, η ιστοσελίδα του ιδρυματικού αποθετηρίου κατακλύστηκε από δεκάδες χιλιάδες άτομα που επιθυμούσαν να αποκτήσουν ένα ηλεκτρονικό αντίγραφό της. Η πρωτοφανής σε μέγεθος εκδήλωση ενδιαφέροντος – άλλες σημαντικές διατριβές δέχονται περίπου εκατό λήψεις μηνιαίως – είχε ως επακόλουθο την, ανά διαστήματα, κατάρρευση του υπεύθυνου διακομιστή [1].

Το φαινόμενο της συρροής μεγάλου πλήθους επισκεπτών μέσα σε σύντομο χρονικό διάστημα σε έναν ιστότοπο, αναφέρεται στη βιβλιογραφία με όρους όπως «flash crowd» [2], «slashdot effect» [3] και «reddit hug of death» [4] και αποτελεί απειλή για την εύρυθμη λειτουργία του, αφού δημιουργεί έντονο φόρτο εργασίας στο σύστημα που τον φιλοξενεί.

Όταν ο ρυθμός με τον οποίον ένας διακομιστής ιστού λαμβάνει αιτήματα εξυπηρέτησης, είναι μεγαλύτερος από τον ρυθμό με τον οποίο μπορεί να τα επεξεργαστεί και να τα απαντήσει, τότε σύντομα τα αιτήματα αρχίζουν να καθυστερούν ή και να λήγουν αναπάντητα (timeout). Ως αποτέλεσμα, παρουσιάζεται το φαινόμενο όπου μία ιστοσελίδα φορτώνει αργά, μερικώς ή και καθόλου – ανάλογα με τη βαρύτητα του προβλήματος και το ρόλο του εκάστοτε διακομιστή. Δεδομένου ότι ένα flash crowd είναι επιθυμητό, υπό την έννοια ότι περισσότεροι άνθρωποι εκδηλώνουν ενδιαφέρον για το περιεχόμενο του ιστοτόπου, γίνεται αντιληπτό ότι για να αμβλύνουμε τις συνέπειές του, θα πρέπει να εστιάσουμε στη βελτίωση της ταχύτητας εξυπηρέτησης των αιτημάτων.

Αναφερόμαστε επομένως σε μία προσπάθεια κλιμάκωσης, η οποία μπορεί να επιτευχθεί με δύο γενικές μεθοδολογίες. Από την μία πλευρά, έχουμε την επιλογή για κάθετη κλιμάκωση, η οποία αναφέρεται στην αναβάθμιση των πόρων του συστήματος, όπως είναι η κεντρική μονάδα επεξεργασίας, η διαθέσιμη μνήμη, ο αποθηκευτικός χώρος και η σύνδεση με το διαδίκτυο. Η διαδικασία αυτή είναι συνήθως απλή, όμως το κόστος αυξάνει κατά κανόνα γρηγορότερα από τις επιδόσεις, ενώ υπάρχει και ανώτατο όριο στο βαθμό κλιμάκωσης – περιορισμός που επιβάλλεται από την τεχνολογία της εποχής.

Από την άλλη πλευρά, υπάρχει η επιλογή της οριζόντιας κλιμάκωσης. Εκεί, αντί για αναβάθμιση του μηχανήματος, προσθέτουμε ένα ή περισσότερα όμοιά του, προκειμένου να μοιραστεί ο φόρτος ανάμεσά τους. Η οριζόντια κλιμάκωση προσφέρει μεγαλύτερο άνω όριο κλιμάκωσης σε συνδυασμό με λογικό κόστος, όμως απαιτεί ενδεχομένως, αλλαγές στον κώδικα ή/και στη γενικότερη δομή του ιστοτόπου.

Απαραίτητη προϋπόθεση προτού σχεδιαστεί ένα πλάνο δράσης για την αύξηση της απόδοσης και την αντιμετώπιση των flash crowds, είναι ο εντοπισμός του σημείου συμφόρησης (bottleneck). Η λειτουργία κάθε ιστοχώρου έχει διαφορετικές απαιτήσεις από το σύστημα που το υποστηρίζει και επόμενο είναι, ότι δεν υπάρχει ένας σταθερός παράγοντας που περιορίζει την ταχύτητα εξυπηρέτησης. Για παράδειγμα, ένας ιστότοπος που επιτρέπει τη μετατροπή αρχείων από έναν τύπο σε έναν άλλο, όπως είναι το Zamzar[1], πιθανώς να χρειαστεί περισσότερη επεξεργαστική ισχύ απ' ό,τι ταχύτητα αποθηκευτικού χώρου, ενώ μία υπηρεσία διαμοιρασμού εικόνων όπως το Imgur[2] έχει συχνά ανάγκη για μεγαλύτερο εύρος ζώνης στη διασύνδεσή της με το διαδίκτυο.

Ασχέτως με το είδος του και τις υπηρεσίες που παρέχει, η λειτουργία κάθε ιστοτόπου περιλαμβάνει την αποστολή δεδομένων προς τον πελάτη που τον επισκέπτεται. Στην πιο απλή περίπτωση, η παροχή περιεχομένου αναφέρεται στα πηγαία αρχεία μιας ιστοσελίδας, ενώ σε άλλες, μιλάμε συχνά για πολυμεσικά και άλλου είδους ψηφιακά αρχεία. Είναι επομένως χρήσιμο, η αποστολή τους να γίνεται με τον πλέον αποτελεσματικό τρόπο, ούτως ώστε να εξοικονομούνται πόροι από το σύστημα.

Μία δημοφιλής και αποτελεσματική πρακτική που εφαρμόζεται στις μέρες μας είναι η ανάθεση της ευθύνης για το σωστό διαμοιρασμό του περιεχομένου σε μία εταιρία παροχής υπηρεσιών Δικτύου Διανομής Περιεχομένου (Content Delivery Network, πλέον CDN). Αυτή διαθέτει τις απαραίτητες υποδομές για την υποστήριξη του ιστοτόπου, απαλλάσσοντας το σύστημα που τον φιλοξενεί από ένα σημαντικό κομμάτι της διαδικασίας εξυπηρέτησής του και απελευθερώνοντας πόρους του.

Ένα CDN στην πιο απλή μορφή του, μπορούμε να το σκεφτούμε ως ένα δίκτυο από γεωγραφικά διεσπαρμένους υπολογιστές που λειτουργούν σαν αντίστροφοι διακομιστές μεσολάβησης (reverse proxies), αποθηκεύοντας περιεχόμενο και διανέμοντάς το όπου ζητηθεί. Όμως, η δημιουργία και η λειτουργία ενός τέτοιου ιδιοκτήτου δικτύου σε μεγάλη κλίμα είναι ένα πολύπλοκο και κοστοβόρο εγχείρημα. Γι'

[1] https://www.zamzar.com/

[2] https://imgur.com/

αυτό το λόγο, λίγες μόνο εταιρείες κατέχουν δικά τους, παγκοσμίου βεληνεκούς CDN και ακόμη λιγότερες τα διαθέτουν προς χρήση σε τρίτους. Οι τελευταίες λέγονται πάροχοι υπηρεσιών CDN. Σήμερα, οι γνωστότεροι εξ αυτών είναι η Akamai Technologies[1], η Limelight Networks[2], η Cloudflare[3] και η Amazon CloudFront[4].

Όπως είναι ακριβή η δημιουργία ενός CDN, έτσι είναι ακριβή και η ενοικίαση ενός από αυτές τις εταιρείες. Το υψηλό κόστος μίσθωσης περιορίζει τη χρήση τους μόνο από μεγάλες εταιρείες που δέχονται εκατοντάδες χιλιάδες επισκέπτες στις ιστοσελίδες τους. Μικρότεροι οργανισμοί που θα μπορούσαν να επωφεληθούν από τις υπηρεσίες ενός CDN, δυστυχώς μένουν εκτός. Χαρακτηριστικά, η Akamai, η μεγαλύτερη εταιρία παροχής υπηρεσιών CDN από άποψη κύκλου εργασιών, δεν δημοσιεύει το κοστολόγιο των υπηρεσιών της και απαιτεί τη σύναψη συμβολαίου κατόπιν διαπραγμάτευσης. Άλλοι πάροχοι όπως η Cloudflare και η CloudFront, που διαθέτουν δωρεάν προγράμματα για μικρές ή προσωπικές ιστοσελίδες, δεν εμπνέουν εμπιστοσύνη ότι, σε μία στιγμή που το δίκτυό τους υπερφορτωθεί, δε θα σταματήσουν απροσδόκητα την υπηρεσία τους για να δώσουν προτεραιότητα στους επί πληρωμή πελάτες τους.

Τίθεται επομένως το ερώτημα: τι άλλη επιλογή έχουν μικροί οργανισμοί, αλλά και μεγαλύτεροι που επιθυμούν να έχουν την καλύτερη δυνατή απόδοση της επένδυσής τους, προκειμένου να παρέχουν το περιεχόμενό τους στον παγκόσμιο ιστό;

## 1.2 Σκοπός – Στόχοι

Στην παρούσα διπλωματική θα ασχοληθούμε με την αξιοποίηση της ομότιμης αρχιτεκτονικής συστημάτων (peer-to-peer system architecture), ώστε να προτείνουμε έναν εναλλακτικό τρόπο για τη διανομή περιεχομένου στον παγκόσμιο ιστό. Τέτοιου είδους συστήματα παρουσιάζουν επιθυμητά χαρακτηριστικά, που τα κάνουν ιδανικά γι' αυτού του είδους τη χρήση, όπως είναι το χαμηλό κόστος και οι υψηλές δυνατότητες κλιμάκωσης και ανοχής σε σφάλματα. Όμως, ο ιστός σχεδιάστηκε με την αρχιτεκτονική πελάτη-εξυπηρετητή και δεν υπάρχει προφανής τρόπος εφαρμογής του ομότιμου μοντέλου δικτύωσης.

[1] https://www.akamai.com

[2] https://www.limelight.com

[3] https://www.cloudflare.com

[4] https://aws.amazon.com/cloudfront

Συστήματα που προσπαθούν να συνδυάσουν τις τεχνικές των CDN με αυτές των ομότιμων δικτύων, στοχεύοντας να αποκομίσουν τα πλεονεκτήματα και των δύο τεχνολογιών, καλούνται συχνά στη βιβλιογραφία με όρους όπως P2P-CDN ή Hybrid CDN (ομότιμα ή υβριδικά CDN) και έχουν προταθεί ήδη από τις αρχές του 21ου αιώνα, παρουσιάζοντας μεγάλο ενδιαφέρον μέχρι και σήμερα. Μάλιστα, ο φρενήρης ρυθμός αύξησης των ροών δεδομένων στο διαδίκτυο την τελευταία δεκαετία, κάνει την έρευνα πάνω σε αυτά, ακόμη σημαντικότερη.

Σκοπός της παρούσας εργασίας είναι η αναγνωριστική μελέτη των δύο τεχνολογιών και η παρουσίαση των χαρακτηριστικότερων προσπαθειών συνδυασμού τους από υβριδικά CDN, στο πλαίσιο του παγκόσμιου ιστού. Θα διαπιστώσουμε πως ο τρόπος που σχεδιάζονται τα υβριδικά CDN, αλλάζει με βάση τις διαθέσιμες τεχνολογίες των προγραμμάτων περιήγησης και θα αποτολμήσουμε την ανάπτυξη της δικής μας λύσης, του PAWS (Peer Assisted Web Server), ενός εξυπηρετητή ιστού υποβοηθούμενου από τους πελάτες του. Στόχος είναι να αποτελέσει μία εναλλακτική μέθοδο εξυπηρέτησης για τα στατικά αρχεία ενός ιστοτόπου, δημιουργώντας ένα ιδιωτικό ομότιμο δίκτυο διανομής περιεχομένου αποτελούμενο από τους χρήστες που τον επισκέπτονται. Με αυτόν τον τρόπο, ευελπιστούμε να ελαττώσουμε τις απαιτήσεις του ιστοτόπου σε εύρος ζώνης, ενισχύοντας την αξιοπιστία του σε απότομες διακυμάνσεις του αριθμού των ταυτόχρονων χρηστών και μειώνοντας, παράλληλα, το συνολικό κόστος εξυπηρέτησης.

## 1.3 Συνεισφορά

Προτού αναπτύξουμε το PAWS, προηγήθηκε η απαραίτητη βιβλιογραφική έρευνα πάνω στα ομότιμα συστήματα και στα δίκτυα διανομής περιεχομένου. Έπειτα ακολούθησε η αναζήτηση άλλων προσπαθειών σαν τη δική μας, προκειμένου να κατανοήσουμε πώς προσέγγισαν άλλοι ερευνητές το συνδυασμό των δύο τεχνολογιών και τι αποτελέσματα είχαν.

Έχοντας κατανοήσει σε βαθύτερο επίπεδο το πρόβλημα και τις υπάρχουσες λύσεις, στη συνέχεια επιλέγουμε τις απαραίτητες τεχνολογίες ιστού που θα μας επιτρέψουν να σχεδιάσουμε και να αναπτύξουμε τη δική μας πρόταση, τις οποίες παρουσιάζουμε λεπτομερώς. Τέλος, παρουσιάζουμε το αποτέλεσμα της εργασίας μας, τον PAWS, έναν εξυπηρετητή ιστού υποβοηθούμενο από ομότιμους χρήστες, τον οποίο αντιπαραθέτουμε με τις ήδη υπάρχουσες προτάσεις ώστε να εξάγουμε τα συμπεράσματά μας γι’ αυτόν και τις τεχνολογίες ιστού.

## 1.4 Διάρθρωση της μελέτης

Η διάρθρωση της εργασίας μας έχει ως εξής:

Στο Κεφάλαιο 2 εισάγουμε τον αναγνώστη στις βασικές έννοιες των ομότιμων συστημάτων και των CDN. Οι έννοιες αυτές θα χρησιμοποιηθούν σε επόμενο κεφάλαιο, προκειμένου να προσδιορίσουμε τη λύση μας με βάση χαρακτηριστικά γνωρίσματα των δύο τεχνολογιών. Στο τελευταίο μέρος του κεφαλαίου, αναφερόμαστε στα υβριδικά CDN, τα σημαντικότερα εκ των οποίων παρουσιάζουμε με χρονολογική σειρά.

Στο Κεφάλαιο 3 γίνεται η παρουσίαση των τεχνολογικών υποδομών που κάνουν δυνατή την ανάπτυξη και λειτουργία της εφαρμογής μας, όπως είναι το WebRTC και τα WebSockets στην πλευρά του πελάτη και η πλατφόρμα Node.js στην πλευρά του διακομιστή.

Στο Κεφάλαιο 4 ασχολούμαστε με την εφαρμογή που αναπτύχθηκε στα πλαίσια της εργασίας μας. Εκεί παρουσιάζονται οι δυνατότητες της και παρατίθενται σημαντικά τμήματα του πηγαίου της κώδικα. Στο τέλος, πραγματοποιείται συμπερασματική αξιολόγηση της υλοποίησής μας.

Το Κεφάλαιο 5 συνοψίζει τα βασικά μέρη της εργασίας μας και αναφέρει τις προτάσεις μας σχετικά με τις τεχνολογίες ιστού αλλά και πιθανές μελλοντικές επεκτάσεις της εργασίας μας.

# 2 Βιβλιογραφική Επισκόπηση – Θεωρητικό Υπόβαθρο

Το παρόν κεφάλαιο αρχικά παρέχει στον αναγνώστη το απαραίτητο θεωρητικό υπόβαθρο που χρειάζεται πάνω στις δημοφιλέστερες μεθόδους παροχής περιεχομένου: μέσω της χρήσης εφαρμογών ομότιμης αρχιτεκτονικής και μέσω των δικτύων διανομής περιεχομένου, ενώ στη συνέχεια επικεντρώνεται στα υβριδικά CDN παρουσιάζοντας την εξέλιξή τους μέχρι σήμερα.

Τα δίκτυα διανομής περιεχομένου αποτελούν ακρογωνιαίο λίθο του σημερινού διαδικτύου, βελτιώνοντας την εμπειρία χρήσης του και αποσυμφορίζοντας τις κεντρικές γραμμές των ISPs (Internet Service Providers, Πάροχοι Υπηρεσιών Διαδικτύου). Χαρακτηρίζονται από υψηλό βαθμό αξιοπιστίας και διαθεσιμότητας.

Από την άλλη, τα ομότιμα συστήματα δε τυγχάνουν της ίδιας εκτίμησης αφού στο μυαλό πολλών ανθρώπων συνδέονται με παράνομο διαμοιρασμό υλικού πνευματικής ιδιοκτησίας αλλά και κακόβουλου λογισμικού. Εντούτοις, συστήματα που βασίζονται στην ομότιμη αρχιτεκτονική συχνά είναι οικονομικότερα και παρουσιάζουν περισσότερες δυνατότητες κλιμάκωσης της απόδοσής τους.

Τα υβριδικά CNDs φιλοδοξούν να παρέχουν το κατάλληλο μείγμα χαρακτηριστικών των δύο τεχνολογιών. Όπως θα δούμε, έχουν επιχειρηθεί διάφοροι τρόποι για τον συνδυασμό των δύο τεχνολογιών, ο καθένας με διαφορετικούς στόχους και με διαφορετικές προϋποθέσεις ως προς το περιβάλλον χρήσης.

Στην πρώτη ενότητα του παρόντος κεφαλαίου μελετάμε την ομότιμη αρχιτεκτονική ως μία ακόμη αρχιτεκτονική κατανεμημένων συστημάτων.

Στην επόμενη ενότητα αναφερόμαστε στη θεωρία των δικτύων διανομής περιεχομένου, παρουσιάζοντας τα βασικά θέματα έρευνας γύρω από αυτά.

Τέλος, η τελευταία ενότητα του κεφαλαίου αφιερώνεται στα ομότιμα δίκτυα διανομής περιεχομένου, όπου οι δύο τεχνολογίες συνδυάζονται με στόχο την αποτελεσματικότερη μετάδοση δεδομένων.

## 2.1 Ομότιμη αρχιτεκτονική και συστήματα

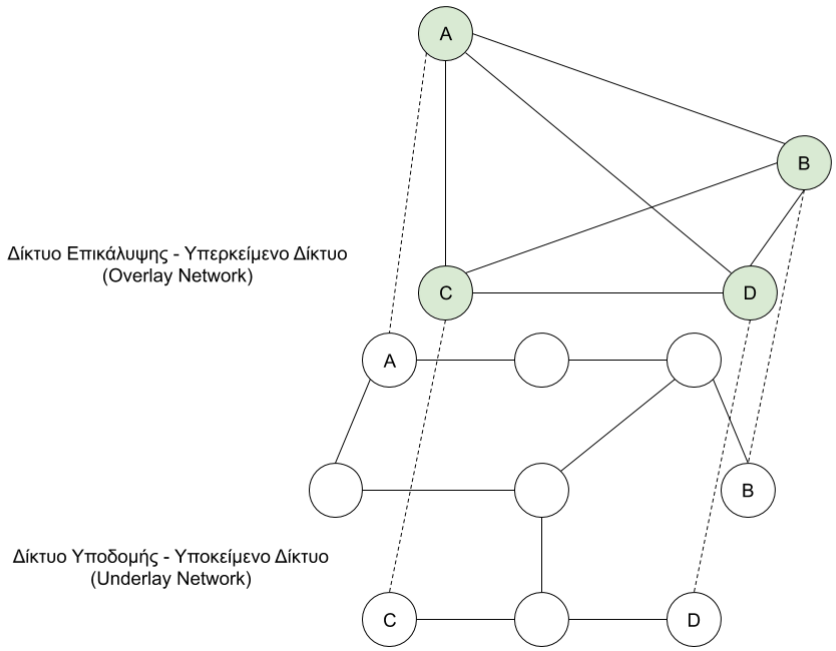
### *2.1.1 Κατανεμημένα συστήματα*

Στη θεωρία των υπολογιστών, όταν μία συλλογή από ανεξάρτητους υπολογιστές εμφανίζεται στους χρήστες ως ένα ενιαίο σύνολο, τότε μπορούμε να χαρακτηρίσουμε τη συλλογή αυτή ως ένα κατανεμημένο σύστημα [5]. Ένα τέτοιο σύστημα έχει στόχο την

εκτέλεση συγκεκριμένου έργου, για το οποίο οι υπολογιστές είναι απαραίτητο να συνεργαστούν μεταξύ τους, μέσω ανταλλαγής μηνυμάτων πάνω από ένα δίκτυο-κανάλι επικοινωνίας.

Ο χρήστης ενός κατανεμημένου συστήματος δεν είναι απαραίτητο να έχει γνώση για τους υπολογιστές που απαρτίζουν το σύστημα. Από την πλευρά του, το σύστημα λειτουργεί διαφανώς, σαν να αποτελείται από ένα μόνο υπολογιστή με την επικοινωνία των στοιχείων του συστήματος να γίνεται στο παρασκήνιο. Κάθε υπολογιστής του κατανεμημένου συστήματος έχει πρόσβαση στους τοπικούς του πόρους, αλλά και σε πόρους που παρέχουν οι άλλοι υπολογιστές μέσω του δικτύου.

Πέρα από το φυσικό δίκτυο που ανάλογα την περίπτωση, αποτελείται από καλώδια χαλκού, οπτικές ίνες, ασύρματα δίκτυα κ.α., στα κατανεμημένα συστήματα συναντάμε και τον όρο δίκτυο επικάλυψης (overlay network). Ο όρος αυτός περιγράφει ένα ιδεατό δίκτυο που χτίζεται πάνω σε ένα άλλο (δίκτυο υποδομής) και για αυτό το λόγο συχνά αναφέρεται και ως υπερκείμενο δίκτυο. Λειτουργεί ως μία αφαιρετική έννοια (abstraction) που απλουστεύει τον σχεδιασμό και τη μελέτη του συστήματος, αφού μας επιτρέπει να σκεφτόμαστε για αυτό, χωρίς να λαμβάνουμε υπόψη λεπτομέρειες του δικτύου υποδομής όπως για παράδειγμα, το πρωτόκολλο IP στην περίπτωση του διαδικτύου.




**Εικόνα 2-1: Δίκτυο Υποδομής και Δίκτυο Επικάλυψης**

Ανάλογα με την τοπολογία του δικτύου επικάλυψης, ένα σύστημα είναι πλήρως κατανεμημένο, όταν κάθε υπολογιστής διατηρεί απευθείας ζεύξη σημείο προς σημείο με όλους τους υπολογιστές του συστήματος και μερικώς κατανεμημένο, όταν διατηρεί ζεύξη μόνο με ένα υποσύνολο αυτών [6].

Στην περίπτωση των πλήρως κατανεμημένων συστημάτων, το σύστημα σχηματίζει πλήρη γράφο, όπου για κάθε ζευγάρι κορυφών (υπολογιστών), υπάρχει μία ακμή που τις συνδέει. Ο αριθμός των υπολογιστών του δικτύου περιορίζεται από την ικανότητα κάθε υπολογιστή-κόμβου να διατηρεί μεγάλο πλήθος ταυτόχρονων ανοιχτών συνδέσεων. Παρ' όλα αυτά, η επικοινωνία μεταξύ των κόμβων είναι αποδοτικότερη αφού τα μηνύματα αποστέλλονται απευθείας στον παραλήπτη τους, χωρίς τη διαμεσολάβηση άλλων κόμβων. Ένα πλήρως συνδεδεμένο δίκτυο είναι λοιπόν επιθυμητό, όταν ο αριθμός των στοιχείων από τα οποία απαρτίζεται είναι σχετικά μικρός.

Σε περιπτώσεις που ο αριθμός των κόμβων δεν επιτρέπει την πλήρη σύνδεση μεταξύ τους, είναι δυνατό το δίκτυο να λειτουργήσει με μόνο μερικές απευθείας συνδέσεις, αρκεί να σχηματίζουν συνεκτικό γράφο. Αυτό σημαίνει ότι για κάθε ζευγάρι κόμβων, θα πρέπει να υπάρχει τουλάχιστον ένα μονοπάτι που τους ενώνει. Οι κόμβοι μπορεί να συνδέονται τυχαία μεταξύ τους ή να σχηματίζουν δομημένες τοπολογίες δικτύων όπως τοπολογία αστέρα, δακτυλίου, δένδρου κ.α. Όπως είναι λογικό, τέτοιου είδους συστήματα είναι λιγότερο αποδοτικά, γιατί οι συνδέσεις του υποκείμενου δικτύου μοιράζονται για την επικοινωνία περισσότερων του ενός ζευγαριού κόμβων. Τέλος, τα συστήματα αυτά είναι περισσότερο επιρρεπή σε μεταβολές της τοπολογίας του δικτύου που συμβαίνουν κατά την αποσύνδεση κόμβων, αφού ένας κόμβος μπορεί να είναι απαραίτητο τμήμα του μονοπατιού με το οποίο κάποιο άλλο ζευγάρι κόμβων επικοινωνεί.

### *2.1.2 Αρχιτεκτονικές Κατανεμημένων Συστημάτων*

Η αρχιτεκτονική ενός συστήματος περιγράφει τον τρόπο με τον οποίο τοποθετούνται και συνδέονται τα επί μέρους στοιχεία του. Τα κατανεμημένα συστήματα, μπορούμε να τα κατατάξουμε με βάση την αρχιτεκτονική τους σε κεντρικοποιημένα, μη κεντρικοποιημένα και υβριδικά [7].

### *2.1.2.1 Κεντρικοποιημένη αρχιτεκτονική*

Στην απλούστερή της έκφανση, μιλάμε για την αρχιτεκτονική πελάτη-εξυπηρετητή (client-server). Σύμφωνα με αυτή, ο πελάτης είναι το στοιχείο του συστήματος που συνεχώς αιτείται υπηρεσίες, τις οποίες παρέχει κατά αποκλειστικότητα ο εξυπηρετητής. Συχνά σε έναν εξυπηρετητή αντιστοιχούν πολλοί πελάτες. Όταν ο πελάτης θέλει να χρησιμοποιήσει τις υπηρεσίες του εξυπηρετητή, κατασκευάζει και του στέλνει ένα αίτημα (request). Αυτός με τη σειρά του, αφού λάβει το αίτημα εκτελεί τις απαραίτητες διαδικασίες και επιστρέφει εφόσον απαιτείται, το αποτέλεσμα ως απόκριση (response). Εξυπακούεται πως ο τρόπος με τον οποίο γίνεται η σειριακοποίηση του αιτήματος και της απόκρισης πρέπει να είναι γνωστός και κατανοητός από τον εξυπηρετητή και τον πελάτη αντίστοιχα.

Μία γενίκευση της αρχιτεκτονικής πελάτη-εξυπηρετητή είναι οι πολυεπίπεδες αρχιτεκτονικές (multi-tiered architectures). Εδώ, οι λειτουργίες του συστήματος αναλύονται σε ξεχωριστά επίπεδα και ανατίθενται σε διαφορετικές οντότητες του συστήματος. Πλέον υπάρχουν περισσότεροι εξυπηρετητές, οι οποίοι δεν είναι αυτάρκεις αλλά συχνά αναλαμβάνουν ρόλο πελάτη, κάνοντας οι ίδιοι αιτήματα σε άλλους προκειμένου να εκτελέσουν το έργο τους.

### *2.1.2.2 Μη κεντρικοποιημένη αρχιτεκτονική*

Στα μη κεντρικοποιημένα συστήματα, κατά κανόνα αποφεύγεται η ανάθεση μίας λειτουργίας σε έναν μόνο υπολογιστή. Είναι σύνηθες κάθε κόμβος του συστήματος να μπορεί να εκτελέσει όλες τις απαιτούμενες λειτουργίες.

Τα ομότιμα συστήματα, με τα οποία ασχολούμαστε σε αυτή την ενότητα, αποτελούν μία υποκατηγορία των μη κεντρικοποιημένων κατανεμημένων συστημάτων τα οποία χαρακτηρίζονται από αυτό-οργάνωση, κοινή χρήση πόρων, ικανότητα κλιμάκωσης, αυτονομία χρηστών και ανθεκτικότητα [8]. Το επίθετο ομότιμος υποδηλώνει την ισότητα και τη συμμετρία των ρόλων που έχουν τα στοιχεία του συστήματος.

Ένας ορισμός που δίνεται στο [9] αναφέρει χαρακτηριστικά ότι ένα σύστημα είναι ομότιμο αν οι υπολογιστές-κόμβοι που το αποτελούν, μοιράζονται τους πόρους τους για να παρέχουν τις υπηρεσίες τις οποίες το σύστημα έχει σχεδιαστεί να παρέχει – ένας κόμβος δύναται να προσφέρει υπηρεσίες στους υπόλοιπους κόμβους αλλά και να αιτείται υπηρεσίες από αυτούς. Ο ορισμός αυτός δεν προβλέπει απαραίτητα την ισότητα

των στοιχείων του συστήματος, όμως τονίζει τη συνεργασία και την αμοιβαιότητα που υπάρχει μεταξύ τους.

#### *2.1.2.3 Υβριδική αρχιτεκτονική*

Πολλές φορές δεν είναι σαφές εάν ένα κατανεμημένο σύστημα ακολουθεί κεντρικοποιημένη ή μη, αρχιτεκτονική. Είναι δυνατόν ένα σύστημα να έχει ορατά χαρακτηριστικά στοιχεία και των δύο αρχιτεκτονικών. Για παράδειγμα, μπορεί να υπάρχει ένας κεντρικός κόμβος ο οποίος αναλαμβάνει τον συντονισμό των υπολοίπων που λειτουργούν με ομότιμο τρόπο. Ή, μία ομάδα κόμβων που διαθέτουν περισσότερες ευθύνες από τους υπόλοιπους και καλούνται υπερκόμβοι (supernodes). Τέτοιου είδους συστήματα καλούνται υβριδικά και στόχο έχουν να παρέχουν τα καλύτερα χαρακτηριστικά των δύο άλλων αρχιτεκτονικών.

### *2.1.3 Αντιπαράθεση ομότιμης και αρχιτεκτονικής πελάτη-εξυπηρετητή*

Για να κατανοήσουμε πότε ενδείκνυται η κάθε αρχιτεκτονική, πρέπει να δούμε τα χαρακτηριστικά της κάθε μίας και να τα αντιπαραβάλουμε. Στην περίπτωση της αρχιτεκτονικής πελάτη-εξυπηρετητή μπορούμε να αναγνωρίσουμε τα εξής πλεονεκτήματα:

- Όλοι οι πόροι για κάθε λειτουργία του συστήματος είναι συγκεντρωμένοι σε ένα κόμβο. Με αυτό τον τρόπο αποφεύγονται προβλήματα συγχρονισμού της κατάστασης του συστήματος και το δίκτυο δεν επιβαρύνεται με συχνές και πιθανώς αργές μεταδόσεις μηνυμάτων.
- Αυξημένη ασφάλεια, αφού οι εξυπηρετητές μπορούν να ελέγξουν ότι μόνο εξουσιοδοτημένοι χρήστες μπορούν να αποκτήσουν πρόσβαση σε ευαίσθητα δεδομένα.
- Η ύπαρξη κεντρικού σημείου διαχείρισης επιτρέπει την ευκολότερη πραγματοποίηση διαχειριστικών εργασιών, όπως είναι η δημιουργία αντιγράφων ασφαλείας.

Από την άλλη πλευρά όμως, έχουν και σημαντικά μειονεκτήματα. Ενδεικτικά αναφέρουμε:

- Αυξημένο κόστος λειτουργίας λόγω της περιπλοκότητας του εξυπηρετητή.
- Ύπαρξη μοναδικού σημείου αστοχίας – αν ένας εξυπηρετητής βρεθεί εκτός λειτουργίας, καθιστά συχνά αδύνατη τη λειτουργία του συστήματος.

- Στην περίπτωση που ένας πελάτης δημιουργήσει μεγάλο φόρτο εργασίας στο σύστημα, αυτό επηρεάζει και τους υπόλοιπους χρήστες.

Η ομότιμη αρχιτεκτονική συστημάτων δίνει λύση σε σημεία που υστερεί αυτή του πελάτη-εξυπηρετητή. Πιο συγκεκριμένα τα ομότιμα συστήματα παρουσιάζουν:

- Αυτόματη κλιμάκωση της απόδοσή τους με την εισαγωγή νέων κόμβων.
- Ανθεκτικότητα σε σφάλματα και κακόβουλες επιθέσεις, αφού δεν υπάρχει κεντρικός κόμβος που μπορεί να αποτελέσει στόχο.
- Μικρό κόστος λειτουργίας μέσω της συνεισφοράς πόρων από τρίτους χρήστες.

Αντίστοιχα, τα ομότιμα συστήματα εμφανίζουν κι αυτά προβλήματα. Τα σημαντικότερα είναι:

- Η ετερογένεια των υπολογιστών που συμμετέχουν στο δίκτυο δυσκολεύει την πλήρη και αποδοτική εκμετάλλευση των πόρων του συστήματος.
- Η συχνή φυγή κόμβων (churn) οδηγεί σε προβλήματα διαχείρισης του δικτύου και σπατάλη πόρων.
- Ο υψηλός λανθάνων χρόνος λόγω της μεγάλης απόστασης των υπολογιστών στο δίκτυο υποδομής που έχει ως αποτέλεσμα, το έργο του συστήματος να εκτελείται ασύγχρονα.

### *2.1.4 Ομότιμα συστήματα και οι λειτουργίες τους*

Όπως είδαμε, τα ομότιμα συστήματα προσφέρουν σημαντικά πλεονεκτήματα. Για το λόγο αυτό, αλλά και επειδή τα θέματα που προκύπτουν για το σχεδιασμό τους είναι ποικίλα, έχουν αποτελέσει δημοφιλές αντικείμενο έρευνας.

Στο RFC 5694 του IETF (Internet Engineering Task Force) [9] γίνεται μία προσπάθεια αναγνώρισης των βασικών λειτουργιών που επιτελούνται μέσα σε ένα ομότιμο σύστημα. Το αποτέλεσμα είναι η εύρεση δύο λειτουργιών που συναντώνται σε κάθε σύστημα και άλλων τεσσάρων, που η ύπαρξή τους εξαρτάται από το είδος και το σκοπό του συστήματος. Πιο συγκεκριμένα, οι δύο καθολικές λειτουργίες είναι οι εξής:

- Λειτουργία ένταξης (Enrollment function): ένας κόμβος του συστήματος πρέπει να αναγνωριστεί και να εξουσιοδοτηθεί πριν συνδεθεί στο σύστημα.

- Λειτουργία εύρεσης ομότιμων κόμβων (Peer discovery function): επιτρέπει σε ένα κόμβο να εντοπίσει άλλους κόμβους του συστήματος και να συνδεθεί μαζί τους.

Οι άλλες τέσσερις λειτουργίες που ενδεχομένως παρουσιάζονται σε ομότιμα συστήματα είναι:

- Λειτουργία ευρετηρίασης δεδομένων (Data indexing function): δημιουργεί ένα ευρετήριο για τα δεδομένα που αποθηκεύονται στο σύστημα.
- Λειτουργία αποθήκευσης δεδομένων (Data storage function): αφορά την αποθήκευση και ανάκτηση των αποθηκευμένων δεδομένων.
- Λειτουργία υπολογισμού (Computation function): αφορά την επεξεργασία δεδομένων μέσα στο σύστημα.
- Λειτουργία μεταφοράς μηνυμάτων (Message transport function): αναφέρεται στην ανταλλαγή μηνυμάτων μεταξύ των κόμβων του συστήματος μέσω της αποτελεσματικής δρομολόγησής τους.

### *2.1.5 Δίκτυα επικάλυψης ομότιμων συστημάτων*

Η εφαρμογή της ομότιμης αρχιτεκτονικής σε ένα σύστημα δεν είναι μία απλή διαδικασία. Ανάλογα με το είδος της εφαρμογής και τα απαιτούμενα χαρακτηριστικά της, θα πρέπει να παρθούν συγκεκριμένες αποφάσεις σχετικά με τη δομή του δικτύου επικάλυψης και τον βαθμό αποκέντρωσής του. Στην πράξη, παρατηρούνται πολλά διαφορετικά είδη ομότιμων συστημάτων, τα οποία παρουσιάζουν διαφορετικές ιδιότητες. Για την διευκόλυνση της μελέτης τους, η κύρια κατηγοριοποίηση των ομότιμων συστημάτων λαμβάνει ως παράγοντα την ύπαρξη λογικής δομής στο δικτύου επικάλυψης. Έτσι ξεχωρίζουμε τα μη δομημένα και τα δομημένα ομότιμα συστήματα [10].

❖ **Μη δομημένα ομότιμα συστήματα**

Στα μη δομημένα ομότιμα συστήματα, δεν υπάρχει λογική στη δημιουργία των συνδέσεων του δικτύου επικάλυψης – η τοπολογία τους είναι τυχαίος γράφος. Σε ένα τέτοιο σύστημα, συχνά υπάρχουν κόμβοι οι οποίοι διαθέτουν περισσότερες λειτουργίες από άλλους. Τέτοιου είδους δομή είναι γενικά κατάλληλη, όταν το σύστημα εκτείνεται

σε παγκόσμια κλίμακα και σε αυτό συμμετέχει μεγάλος αριθμός κόμβων με υψηλό βαθμό ετερογένειας μεταξύ τους.

Η διάδοση και η αναζήτηση πληροφοριών μέσα στο σύστημα πραγματοποιείται με τεχνικές ερωτημάτων όπως είναι η πλημμύρα (flooding) και ο τυχαίος περίπατος (random walking) [11]. Αυτές οι τεχνικές αναζήτησης επιβαρύνουν το δίκτυο με μεγάλο αριθμό μηνυμάτων και για αυτό το λόγο συχνά επιβάλλονται περιορισμοί για το πόσες φορές θα ανακατευθυνθούν μέσα στο σύστημα. Η ντετερμινιστική ανάκτηση δεδομένων δεν είναι εφικτή σε αυτήν την περίπτωση.

Στα κύρια πλεονεκτήματα αυτών των συστημάτων είναι η εύκολη υλοποίησή τους και η ανθεκτικότητά τους στις μεταβολές της τοπολογίας του δικτύου.

Χαρακτηριστικά παραδείγματα μη δομημένων ομότιμων συστημάτων είναι το Napster, το Gnutella, το FastTrack, το Publius, το BitTorrent κ.α.

❖ **Δομημένα ομότιμα συστήματα**

Στα δομημένα ομότιμα συστήματα, το δίκτυο επικάλυψης αναπτύσσεται βάσει συγκεκριμένων κανόνων. Με αυτό τον τρόπο επιδιώκεται η αποτελεσματικότερη εύρεση και ανάκτηση δεδομένων, πράγμα στο οποίο υστερούν τα μη δομημένα συστήματα. Η ύπαρξη αυστηρής δομής εξασφαλίζει ότι η αναζήτηση μπορεί να ολοκληρωθεί ύστερα από ένα μικρό αριθμό αναπηδήσεων στο δίκτυο (hops).

Η ανάπτυξη δομημένων συστημάτων παρουσιάζει πληθώρα θεμάτων που αφορούν τη διατήρηση της δομής με την είσοδο και τη φυγή κόμβων. Για το λόγο, τέτοια συστήματα είναι συνήθως μικρότερα σε μέγεθος.

Πολύ συχνή στην περίπτωση των δομημένων συστημάτων, είναι η χρήση κατανεμημένων πινάκων κατακερματισμού (DHT) για την ευρετηρίαση των δεδομένων. Κάθε κόμβος αναλαμβάνει να διατηρεί ένα πίνακα με απαραίτητες πληροφορίες προκειμένου να διευκολύνει την αναζήτηση δεδομένων. Διάφορα πρωτόκολλα έχουν αναπτυχθεί για την ανάπτυξη δομημένων ομότιμων συστημάτων με DHT, τα κυριότερα εκ των οποίων είναι τα Chord, Pastry, Tapestry και CAN.

Παραδείγματα δομημένων συστημάτων ομότιμης αρχιτεκτονικής αποτελούν το OceanStore, το Mnemosyne, το PAST και το Freenet.

Ένας άλλος παράγοντας ταξινόμησης των ομότιμων συστημάτων που χρησιμοποιείται συχνά στη βιβλιογραφία, είναι ο βαθμός αποκέντρωσής τους. Με βάση αυτόν έχουμε [10], [12], [13]:

- **Υβριδικά αποκεντρωμένα συστήματα (Hybrid decentralized systems)**

Στα υβριδικά αποκεντρωμένα συστήματα υπάρχει συνήθως κάποιος εξυπηρετητής που αναλαμβάνει διάφορες από τις λειτουργίες που λαμβάνουν χώρα σε ένα ομότιμο σύστημα. Τέτοιες λειτουργίες συνήθως είναι οι λειτουργίες της ένταξης, της εύρεσης ομότιμων κόμβων και της ευρετηρίασης των αποθηκευμένων δεδομένων.

Τα κεντρικοποιημένα συστήματα είναι δυνατόν να προσφέρουν υψηλή ταχύτητα αναζήτησης λόγω της ύπαρξης κεντρικού καταλόγου αρχείων – δεν απαιτείται η χρήση του δικτύου για αποστολή ερωτημάτων – ταυτόχρονα όμως, παρουσιάζουν θέματα αξιοπιστίας και περιορισμένης κλιμάκωσης εξαιτίας αυτού. Αν ο κεντρικός εξυπηρετητής δεν είναι διαθέσιμος ή υπολειτουργεί, είναι δυνατόν να καταστήσει μη αξιοποιήσιμο όλο το σύστημα.

Συστήματα τέτοιου είδους είναι απλούστερα στην ανάπτυξή τους. Μάλιστα, μία από τις πρώτες και δημοφιλέστερες εφαρμογές για τον διαμοιρασμό αρχείων με ομότιμο τρόπο, το Napster, ήταν τέτοιου είδους.

- **Πλήρως αποκεντρωμένα συστήματα (Purely decentralized)**

Στη βιβλιογραφία αναφέρονται και ως αμιγή ομότιμα συστήματα (pure peer-to-peer systems). Σε αυτά, όλες οι λειτουργίες του συστήματος μπορούν να εκτελεστούν από όλους τους κόμβους, οι οποίοι συχνά καλούνται με τον όρο servent (συνδυασμός των όρων client και server που υπογραμμίζει τον διπλό ρόλο του κάθε κόμβου).

Η απουσία κεντρικού εξυπηρετητή κάνει τα αποκεντρωμένα συστήματα ιδιαιτέρως ανθεκτικά και αξιόπιστα. Ακόμα κι αν κάποιος κόμβος βγει εκτός λειτουργίας, υπάρχουν πάντα άλλοι που μπορούν να αναλάβουν στη θέση του.

Επίσης, αφού κάθε κόμβος συνεισφέρει με τον ίδιο τρόπο, τέτοια συστήματα συχνά χαρακτηρίζονται από ικανότητα υποστήριξης μεγάλου αριθμού κόμβων και απόδοση που κλιμακώνεται αναλογικά.

Κύρια μειονεκτήματα των αμιγώς ομότιμων συστημάτων είναι ο υψηλός χρόνος που χρειάζεται για την αναζήτηση μέσα στο σύστημα, η οποία απαιτεί, είτε την ύπαρξη δομής στο σύστημα, είτε τη σπατάληση πόρων με τη δημιουργία πολλών ερωτημάτων.

Τέτοιου είδους συστήματα είναι συνήθως τα δομημένα συστήματα, όπως τα γνωρίσαμε παραπάνω, αλλά και άλλα όπως για παράδειγμα το Gnutella.

❖ **Μερικώς κεντρικοποιημένα συστήματα (Partially centralized)**

Σε αυτή την κατηγορία ανήκουν συστήματα, μερικοί κόμβοι των οποίων, αναλαμβάνουν περισσότερες ή σημαντικότερες υπηρεσίες σε σχέση με άλλους. Αυτοί οι κόμβοι καλούνται υπερκόμβοι (supernodes) και παίζουν καθοριστικό ρόλο στη λειτουργία του συστήματος. Συνήθως είναι υπολογιστές που μπορούν να παρέχουν πολλούς πόρους στο σύστημα (επεξεργαστική ισχύ, αποθηκευτικό χώρο, εύρος ζώνης δικτύου κ.α.).

Η κατηγορία επιδιώκει να προσφέρει ότι οι άλλες δύο δεν μπορούν, ακολουθώντας μία μέση λύση. Έτσι, προσφέρει γρηγορότερη αναζήτηση από ένα μη κεντρικοποιημένο σύστημα, χωρίς να εμφανίζει μοναδικό σημείο αποτυχίας λόγω ύπαρξης κεντρικού εξυπηρετητή.

Τέτοια συστήματα που αναφέρονται στην βιβλιογραφία είναι η έκδοση 06 της Gnutella και το FastTrack.

### *2.1.6 Εφαρμογές ομότιμων συστημάτων*

Συγκρινόμενη με την αρχιτεκτονική πελάτη-εξυπηρετητή, η αρχιτεκτονική των ομότιμων συστημάτων προσφέρει σημαντικά χαρακτηριστικά που την κάνουν ιδανική για μερικές κατηγορίες λογισμικού. Παρατηρούμε συγκεκριμένα τη δημοφιλία της στα παρακάτω είδη εφαρμογών [8]–[10], [14]:

- Εφαρμογές επικοινωνίας και συνεργασίας: Η ομότιμη αρχιτεκτονική είναι ιδανική για τέτοιου είδος εφαρμογές γιατί προσφέρει υψηλή αποκρισιμότητα και ιδιωτικότητα έναντι μιας κεντρικοποιημένης αρχιτεκτονικής. Τέτοιες εφαρμογές περιλαμβάνουν το Voice over IP (VoIP) και άλλες εφαρμογές Instant Messaging (IM) που επιτρέπουν την πολυμεσική επικοινωνία μέσω Διαδικτύου.
- Εφαρμογές κατανεμημένης υπολογιστικής: Η χρήση τέτοιου είδους εφαρμογών επιτρέπει την αξιοποίηση των επεξεργαστικών πόρων ενός μεγάλου αριθμού υπολογιστικών συστημάτων για την επίλυση ιδιαιτέρως δύσκολων

προβλημάτων. Τέτοια προβλήματα συνήθως σχετίζονται με επιστημονικές έρευνες. Το γεγονός ότι οποιοδήποτε μηχάνημα μπορεί να συμμετέχει σε μία τέτοια εφαρμογή, δημιούργησε την έννοια του «εθελοντικού υπολογισμού» (volunteer computing), όπου όποιος το επιθυμεί μπορεί να δωρίζει πόρους της συσκευής του όταν δεν τη χρησιμοποιεί. Η πρώτη τέτοιου είδους εφαρμογή ήταν το SETI@home το 1999, ενώ λίγα χρόνια αργότερα δημιουργήθηκε το BOINC, ένα σύστημα για τη διευκόλυνση ανάπτυξης τέτοιου είδους εφαρμογών. Σήμερα, θα μπορούσαμε να εντάξουμε σε αυτήν την κατηγορία, την τεχνολογία blockchain.

- Εφαρμογές δημοσίευσης και αποθήκευσης: Η αποκεντροποιημένη φύση των ομότιμων συστημάτων τα κάνει κατάλληλα για δημοσίευση πληροφοριών που ίσως γίνουν αντικείμενο λογοκρισίας. Η δημοκρατικότητα τέτοιων συστημάτων – όλοι οι κόμβοι είναι ίσοι – κάνει δύσκολη την παρεμπόδιση της διάχυσης της πληροφορίας σε αυτά. Επίσης, η ύπαρξη αντιγράφων σε πολλούς ανεξάρτητους κόμβους εξασφαλίζει σε μεγάλο βαθμό την αντοχή της στο χρόνο. Τέτοιου είδους εφαρμογές είναι το Freenet, ένα σύστημα για ανώνυμη δημοσίευση περιεχομένου και το IPFS, που στοχεύει να κάνει τον ιστό ταχύτερο, ασφαλέστερο και κυρίως, πραγματικά ανοικτό.
- Εφαρμογές παροχής περιεχομένου: Ίσως η δημοφιλέστερη χρήση της ομότιμης αρχιτεκτονικής. Σε τέτοιου είδους εφαρμογές, το διαθέσιμο εύρος ζώνης των ομότιμων κόμβων αξιοποιείται για το διαμοιρασμό περιεχομένου, όπως κείμενα, πολυμέσα και εκτελέσιμα αρχεία. Σε αυτή την κατηγορία εντάσσονται διάφορες εφαρμογές ανταλλαγής αρχείων (file sharing) όπως το Napster και το Kazaa, και άλλες υποστηρικτικές εφαρμογές παροχής περιεχομένου όπως το LiveSky.

Μία ενδιαφέρουσα κατηγοριοποίηση των ομότιμων συστημάτων βάσει της χρήσης τους εμπεριέχεται στο [13], όπου αναγνωρίζονται τρεις κατηγορίες:

- Εφαρμογές για την ανταλλαγή δεδομένων και πληροφοριών (Data-sharing)
- Εφαρμογές για την κοινή αξιοποίηση του εύρους ζώνης (Bandwidth-sharing)
- Εφαρμογές για την κοινή αξιοποίηση υπολογιστικών πόρων (CPU-sharing)

## 2.2 Δίκτυα διανομής περιεχομένου (CDN)

Σε έκθεση που δημοσίευσε η CISCO [15], μεγάλη εταιρία υψηλής τεχνολογίας στο τομέα των προϊόντων και υπηρεσιών δικτύωσης, προβλέπεται ότι έως το 2022, 396EB (exabytes) δεδομένων θα διακινούνται μηνιαίως στο διαδίκτυο – μία αύξηση της τάξης του 325% έναντι του 2017. Κύρια αιτία για το γεγονός αυτό μοιάζει να είναι η συνεχής βελτίωσης των υποδομών του διαδικτύου, που δίνει πρόσβαση σε περισσότερο κόσμο και επιτρέπει παράλληλα, νέες υπηρεσίες και κατανάλωση πολυμεσικού περιεχομένου υψηλότερης ποιότητας. Ειδικά η παρακολούθηση βίντεο παρουσιάζεται ως η δραστηριότητα για την οποία μετακινείται ο μεγαλύτερος όγκος δεδομένων, με πρόβλεψη να ξεπεράσει το 80% του συνολικού όγκου το 2022. Το αντίστοιχο ποσοστό για τον παγκόσμιο ιστό αναμένεται να κυμανθεί κοντά στο 12%.

Από την ίδια έκθεση προκύπτει πως το 2017, για το 56% των δεδομένων του διαδικτύου χρησιμοποιήθηκαν CDN, ενώ αναμένεται αύξηση του ποσοστού στο 72% έως το 2022. Παράλληλα, έρευνα αγοράς που πραγματοποίησε η Zion Market Research αναφέρει ότι το μέγεθος της αγοράς των CDN άγγιξε τα 10,9 δις USD το 2018, με πρόβλεψη για μέσο ρυθμό ετήσιας ανάπτυξης (CAGR) της τάξης του 12,5% έως το 2025 [16].

Σύμφωνα και με τα παραπάνω, η σημασία των CDN και η επιτυχία τους στην υποστήριξη των εφαρμογών του διαδικτύου και του παγκόσμιου ιστού είναι προφανής. Η τεχνολογία των CDN είναι πλέον ώριμη με εκτενή έρευνα γύρω από τα ζητήματά της.

Η ανάλυση των δικτύων διανομής περιεχομένου θα μας επιτρέψει να κατανοήσουμε τα διάφορα θέματα που άπτονται της διανομής περιεχομένου

### *2.2.1 Ορισμός*

Προτού συνεχίσουμε είναι χρήσιμο να δώσουμε ένα ορισμό για το τι είναι ένα δίκτυο διανομής περιεχομένου. Όπως υποδεικνύει το όνομα, αναφερόμαστε σε ένα δίκτυο, δηλαδή ένα σύνολο από κατανεμημένα υπολογιστικά συστήματα, όπου μερικά από αυτά συνδέονται μεταξύ τους και που στόχο έχουν την αποστολή περιεχομένου σε όποιον το αιτείται. Βασικός στόχος είναι η αποδοτικότερη και αποτελεσματικότερη αποστολή περιεχομένου. Αυτό επιτυγχάνεται με την τοποθέτηση αντιγράφων του περιεχομένου σε γεωγραφικά διασκορπισμένους διακομιστές, όσο το δυνατόν πλησιέστερα στις άκρες του Διαδικτύου. Σύμφωνα με το [17] , τα CDN και συγκεκριμένα αυτό της Akamai που ιδρύθηκε την περίοδο 1998-1999, αποτελούν το πρώτο δείγμα της Υπολογιστικής στα Άκρα του Δικτύου (Edge computing).

Κάθε φορά που το δίκτυο λαμβάνει αίτημα προς εξυπηρέτηση, έχει τη δυνατότητα να επιλέξει το βέλτιστο διακομιστή που θα αποστείλει το περιεχόμενο στον χρήστη. Η εγγύτητα ενός διακομιστή με τον χρήστη του CDN ελαχιστοποιεί τον χρόνο εξυπηρέτησης, ενώ το πλήθος των διακομιστών κάνει τις απότομες μεταβολές στο φόρτο εργασίας, διαχειρίσιμες.

### *2.2.2 Βασικά μέρη ενός CDN*

Στο [18] παρουσιάζεται η γενική αρχιτεκτονική ενός συστήματος CDN όπου ξεχωρίζουν 7 επιμέρους οντότητες: οι πελάτες, οι διακομιστές αντιγράφων (replica servers), ο πηγαίος διακομιστής, το σύστημα τιμολόγησης, το σύστημα δρομολόγησης αιτημάτων, το σύστημα διανομής περιεχομένου και το λογιστικό σύστημα. Κάθε μία από τις οντότητες παρουσιάζει επιπλέον ζητήματα. Ως σημαντικότερα χαρακτηρίζονται τα θέματα της τοποθέτησης διακομιστών αντιγράφων στο δίκτυο, του διαμοιρασμού των αρχείων σε αυτούς καθώς και του τρόπου δρομολόγησης των αιτημάτων. Τέλος, παρουσιάζεται μία μέθοδος για τον εντοπισμό υπηρεσιών σε ομότιμα δίκτυα, τα οποία χαρακτηρίζονται ως εν δυνάμει «δίκτυα διανομής περιεχομένου χωρίς την χρήση επίσημων υποδομών».

### *2.2.3 Σημαντικά θέματα των CDN*

Το πρόβλημα της τοποθέτησης replica servers στο δίκτυο είναι πολύπλευρο. Από τη μία πλευρά υπάρχει η επιθυμία οι replica servers να είναι όσο το δυνατόν κοντύτερα στον τελικό χρήστη, ώστε να προσφέρουν την καλύτερη δυνατή απόδοση και να μειώνουν το κόστος που πηγάζει από τη χρήση του δικτύου. Από την άλλη, οι servers θα πρέπει να τοποθετηθούν με τέτοιο τρόπο, ώστε ο συνολικός φόρτος εργασίας να επιμερίζεται ισότιμα ανάμεσά τους. Μία συγκριτική μελέτη αλγορίθμων για την εύρεση των ιδανικότερων σημείων τοποθέτησής παρουσιάζεται στο [19].

Το πρόβλημα του διαμοιρασμού των αρχείων στους εξυπηρετητές του δικτύου, αναφέρεται στην τοποθέτηση αντιγράφων των αρχείων σε αυτούς [12]. Μία πρόταση είναι η ευκαιριακή τοποθέτηση αντιγράφων μόνο όταν τα αρχεία αιτούνται από έναν πελάτη. Τέτοιες μέθοδοι αναφέρονται στη βιβλιογραφία ως «pull-based». Ο replica server έχει τις ιδιότητες μίας κρυφής μνήμης που τραβάει νέο περιεχόμενο μόνο όταν απαιτείται, διαγράφοντας παράλληλα άλλο σύμφωνα με μία πολιτική ανανέωσης της μνήμης του. Ανάλογα με το πως αντιμετωπίζεται η αποτυχία εύρεσης ενός αρχείου, κατηγοριοποιούμε τις μεθόδους αυτές σε «cooperative pull-based» και «non-cooperative

pull-based». Άλλου είδους τεχνική είναι η «cooperative push-based», όπου τα αρχεία προωθούνται στους replica servers από τον κεντρικό εξυπηρετητή προκαταβολικά, προτού χρειαστούν.

Τέλος, το πρόβλημα δρομολόγησης των αιτημάτων αναφέρεται στην επιλογή του καταλληλότερου replica server που θα εξυπηρετήσει ένα αίτημα. Τα κύρια κριτήρια για την επιλογή του διακομιστή συνήθως είναι: η μικρότερη γεωγραφική απόσταση από τον χρήστη, η μικρότερη δικτυακή απόσταση, δηλαδή ο διακομιστής που παρουσιάζει τον μικρότερο λανθάνοντα χρόνο (latency) και η διαθεσιμότητά του, δηλαδή ο φόρτος εργασίας του και η ύπαρξη διαθέσιμων πόρων, μια δεδομένη χρονική στιγμή.

### *2.2.4 Τύπος περιεχομένου και μέθοδοι διανομής*

Τα δίκτυα διανομής περιεχομένου μπορούν να χρησιμοποιηθούν για τη μεταφορά κάθε είδους ψηφιακού περιεχομένου. Ανάλογα το είδος του περιεχομένου ξεχωρίζουμε στατικά και δυναμικά αρχεία [20]. Η διαφορά τους δεν έγκειται στο τύπο του πολυμέσου, αλλά στον τρόπο παραγωγής του. Έτσι αν ένα έγγραφο HTML δημιουργείται για κάθε χρήστη ξεχωριστά (για παράδειγμα, με τη χρήση PHP), το έγγραφο θεωρείται δυναμικό, ενώ αν επαναχρησιμοποιείται απαράλαχτο για κάθε πελάτη, το έγγραφο είναι στατικό. Αντίστοιχα, αν ένα βίντεο αποτελεί ζωντανή μετάδοση τότε το αρχείο είναι δυναμικό, ενώ αν αποτελεί ολοκληρωμένη ταινία, στατικό.

Η φύση του περιεχομένου μπορεί να επηρεάζει και τη μέθοδο διανομής του. Σε αυτή την περίπτωση διακρίνουμε πάλι δύο μεθόδους, την ολοκληρωτική λήψη (downloading) και τη λήψη με ροή (streaming) [8]. Στην πρώτη περίπτωση το περιεχόμενο πρέπει να αποσταλεί ολόκληρο, προκειμένου να έχει νόημα στην πλευρά του χρήστη. Ένα τέτοιο παράδειγμα είναι ένα συμπιεσμένο αρχείο όπου το περιεχόμενό του δεν μπορεί να εξαχθεί χωρίς την πλήρη λήψη του. Από την άλλη πλευρά, η λήψη με ροή επιτρέπει την κατανάλωση του περιεχομένου καθώς αυτό αποστέλλεται. Χαρακτηριστικό παράδειγμα είναι το διαδικτυακό ραδιόφωνο. Η λήψη με ροή, είναι σαφώς πιο περίπλοκη διαδικασία, αφού συχνά απαιτεί σωστό συγχρονισμό στην ταχύτητα λήψης και κατανάλωσης του περιεχομένου.

## 2.3 Ομότιμη διανομή περιεχομένου πάνω στον παγκόσμιο ιστό

Κατά την έρευνα μας στη βιβλιογραφία, εντοπίσαμε τα πρώτα δείγματα επιθυμίας αξιοποίησης της τεχνολογίας των ομότιμων δικτύων για την διανομή περιεχομένου, στις αρχές του 21ου αιώνα. Είναι η εποχή όπου η φρενίτιδα για τις dot-com επιχειρήσεις – επιχειρήσεις που δραστηριοποιούνται κυρίως μέσω του παγκόσμιου ιστού – αρχίζει να καταλαγιάζει και τη θέση της παίρνει μια πιο ώριμη αντίληψη για το διαδίκτυο. Σε αυτή την ενότητα θα παρουσιάσουμε με χρονολογική σειρά διάφορες προσπάθειες χρήσης της ομότιμης αρχιτεκτονικής στον παγκόσμιο ιστό. Δε θα αναφερθούμε σε ομότιμα δίκτυα διανομής περιεχομένου που σχεδιάστηκαν για την υποστήριξη μίας διαδικτυακής εφαρμογής, όπως είναι, ή υπήρξαν κατά καιρούς, το Spotify[1], το PPTV[2] (πρώην PPLive) και το LiveSky[3].

Μια πρώτη προσέγγιση στην ομότιμη παροχή περιεχομένου φαίνεται στο [21], στο οποίο σχεδιάζεται και αξιολογείτε μέσω προσομοίωσης το CoopNet, ένα σύστημα για τη συμμετοχή των πελατών μιας ιστοσελίδας στη διανομή των εγγράφων της. Στόχος είναι η σωστή διαχείριση των flash crowds, όπως αυτό που δέχτηκε το MSNBC την 11η Σεπτεμβρίου του 2001. Κάθε πελάτης που επιθυμεί να συνεισφέρει, εγκαθιστά το απαραίτητο λογισμικό, το οποίο ενημερώνει τον εξυπηρετητή μέσω επικεφαλίδας HTTP. Αυτός με τη σειρά του κρατάει για κάθε αρχείο, μία λίστα FIFO των πελατών συγκεκριμένου μεγέθους. Σε περιόδους υψηλής κίνησης, καθώς νέοι πελάτες ζητούν αρχεία που ήδη υπάρχουν σε proxies άλλων πελατών, τους επιστρέφει τη λίστα προτρέποντάς τους να τα αναζητήσουν εκεί. Στο ίδιο άρθρο προτείνεται επίσης, η επιλογή του χρήστη από τον οποίο θα ληφθεί το αρχείο να γίνεται με ταίριασμα των προθεμάτων των IPs πομπού-δέκτη. Αναγνωρίζεται ότι συχνά, όταν δύο IPs ξεκινούν με το ίδιο πρόθεμα, τα υπολογιστικά συστήματα βρίσκονται κοντά δικτυακά ή/και γεωγραφικά. Η αξιολόγηση του συστήματος αναφέρει ότι: πρώτον, το σύστημα μειώνει επιτυχώς τον φόρτο του εξυπηρετητή, δεύτερον, η πιθανότητα δύο χρήστες με κοντινές IP να ζητήσουν το ίδιο αρχείο είναι μεγάλη και τρίτον, το σύστημα χρειάζεται κάποιο μηχανισμό εύλογης χρήσης, αφού είναι πιθανό να προκαλέσει υψηλό φόρτο σε έναν

[1] https://www.spotify.com

[2] https://www.pptv.com

[3] Yin, Hao, et al. "Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky." Proceedings of the 17th ACM international conference on Multimedia. ACM, 2009.

εθελοντή-πελάτη. Τέλος, το άρθρο διαπιστώνει πως τα προβλήματα των flash crowds δημιουργούνται λόγω περιορισμένης ταχύτητας σύνδεσης του εξυπηρετητή με το διαδίκτυο κι όχι λόγω ανεπαρκούς επεξεργαστικής ισχύος. Σήμερα, με την πολυπλοκότητα των εφαρμογών ιστού και τη συχνή παραγωγή δυναμικού περιεχομένου, ο ισχυρισμός αυτός δεν επιβεβαιώνεται πάντα.

Άλλο ένα σύστημα ομότιμης παροχής περιεχομένου που εντοπίζεται στη βιβλιογραφία και έχει επίσης στόχο την αντιμετώπιση των flash crowds είναι το Backslash [22]. Σκοπός του είναι η συνεργασία μικρών οργανισμών, όπως σχολεία και μη κερδοσκοπικά ιδρύματα, για τη συνεργατική εξυπηρέτηση εγγράφων ιστού όταν ο πηγαίος εξυπηρετητής αντιμετωπίζει υψηλό φόρτο. Κάθε οργανισμός παρέχει το δικό του εξυπηρετητή που έχει πλήρη αντίγραφα του ιστοτόπου του και αρκετό κενό αποθηκευτικό χώρο προκειμένου να λειτουργήσει ως προσωρινή μνήμη για το περιεχόμενο άλλων συνεργαζόμενων ιστοτόπων. Οι συμμετέχοντες εξυπηρετητές στο Backslash σχηματίζουν ένα δομημένο δίκτυο επικάλυψης βασισμένο στο CAN (Content Addressable Network) . Όταν ο φόρτος εργασίας σε έναν εξυπηρετητή ξεπεράσει ένα προκαθορισμένο όριο, αλλάζει τα ενσωματωμένα URLs των εγγράφων του έτσι ώστε να παραπέμπουν σε άλλους συνεργαζόμενους εξυπηρετητές. Τα αντικείμενα στα οποία αναφέρονται τα URLs, πρέπει να έχουν ήδη τοποθετηθεί σε αυτούς. Δυστυχώς, η αρχιτεκτονική δεν παρουσιάζει με ποιο τρόπο μπορεί να γίνει υποστήριξη δυναμικού περιεχομένου, ή πώς προφυλάσσεται το σύστημα από κακόβουλους ή προβληματικούς κόμβους.

Δύο συστήματα που δεν έχουν στόχο την προστασία από flash crowds αλλά την μείωση του κόστους που πηγάζει από την αυξημένη χρήση εύρους ζώνης σε πύλες που συνδέουν τοπικά δίκτυα με το διαδίκτυο, είναι τα BuddyWeb [23] και Squirrel [24]. Σε αυτά προτείνεται η συνεργασία των υπολογιστών σε ένα τοπικό δίκτυο, προκειμένου να σχηματίσουν μία αποκεντρωμένη προσωρινή μνήμη ιστού (web cache). Κάθε υπολογιστής απαιτείται να εγκαταστήσει ειδικό λογισμικό που λειτουργεί ως ενδιάμεσος πληρεξούσιος εξυπηρετητής (forward proxy) και να ρυθμίσει αντίστοιχα τον φυλλομετρητή του για να κάνει χρήση αυτού. Οι proxies αυτοί αποθηκεύουν τα έγγραφα που προσπελάζουν οι χρήστες τους και συμμετέχουν σε ομότιμο δίκτυο με άλλους proxies σε υπολογιστές του τοπικού δικτύου. Το ομότιμο δίκτυο του BuddyWeb είναι μη δομημένο και πλήρως αποκεντρωμένο και χρησιμοποιεί τεχνικές πλημμύρας για την αναζήτηση σε άλλους κόμβους. Από την άλλη, το δίκτυο επικάλυψης του Squirrel

βασίζεται στη μέθοδο ευρετηρίασης Pastry για να αναζητήσει έγγραφα στις μνήμες των ομοτίμων. Και οι δύο αυτές μέθοδοι απαιτούν αρκετές αναπηδήσεις δικτύου (hops) προκειμένου να ολοκληρωθεί η αναζήτηση. Αν και η αναζήτηση γίνεται σε επίπεδο τοπικού δικτύου – πράγμα που σημαίνει μικρότερο λανθάνων χρόνο – τέτοιες τεχνικές θα ήταν απαγορευτικές στον σημερινό ιστό όπου κάθε δευτερόλεπτο είναι κρίσιμο [25].

Άλλα δύο ομότιμα δίκτυα διανομής περιεχομένου είναι και τα CoralCDN [26], [27] και CoDeeN [28], τα οποία αποτελούν ακαδημαϊκές προσπάθειες δημιουργίας CDN. Αυτά αποτελούν εθελοντικά δίκτυα εξυπηρετητών που αναπτύχθηκαν στο PlanetLab[1], ένα παγκόσμιο ερευνητικό δίκτυο που υποστηρίζει τον πειραματισμό πάνω στις τεχνολογίες δικτύων. Το CoralCDN φιλοδοξεί να δώσει λύση στο πρόβλημα των flash crowds και να προσφέρει συγκρίσιμες επιδόσεις με το απλό μοντέλο πελάτη-εξυπηρετητή. Θεωρεί υψίστης σημασίας την αποφυγή δημιουργίας hot spots σε μεμονωμένους εθελοντές, έτσι ώστε να μην αποθαρρύνονται και να αποχωρούν. Οι εξυπηρετητές του CoralCDN οργανώνονται με τη χρήση DSHT, μίας πολυστρωματικής παραλλαγής των κλασσικών DHTs που προτείνουν οι δημιουργοί του CoralCDN. Σύμφωνα με αυτό, οι εξυπηρετητές ανήκουν σε πολλά ομότιμα δίκτυα επικάλυψης, καθένα από τα οποία αντιστοιχεί σε όλο και μεγαλύτερο γεωγραφικό χώρο. Έτσι μπορούν να υπάρχουν δίκτυα που αντιστοιχούν σε πόλη, σε χώρα κ.ο.κ. Στην αναζήτηση ενός αρχείου, πρώτα ερωτάται το μικρότερο, σε γεωγραφικό μέγεθος, δίκτυο και προχωρά στα ευρύτερα μόνο αν το αρχείο δε βρεθεί. Σημαντική προσπάθεια έγινε για την εύκολη χρήση του CoralCDN, αφού το μόνο που απαιτεί είναι η αλλαγή της διεύθυνσης ενός αρχείου με την προσθήκη του επιθέματος «nyud.net». Με αυτόν τον τρόπο, ο καθένας μπορεί να επιλέξει να αποθηκεύσει ένα αντικείμενο ιστού στο CDN – είτε είναι ο πάροχος του περιεχομένου είτε ένας πελάτης που θέλει να κοινοποιήσει την σελίδα. Ένα πρόβλημα που παρουσιάζεται με αυτή την τεχνική είναι ότι υπάρχει υψηλή πιθανότητα το αντικείμενο να μην προλάβει να αποθηκευτεί στους servers του CDN προτού ο πηγαίος εξυπηρετητής κορεσθεί. Δυστυχώς, το έργο δεν έτρεξε ποτέ έξω από την δοκιμαστική του πλατφόρμα στο PlanetLab, αφού απαιτούσε την ύπαρξη πλήρους εμπιστοσύνης μεταξύ των εξυπηρετητών του συστήματος. Επίσης, η δημοφιλία του HTTPS, τεχνολογία που δεν υποστήριζε το CoralCDN, οδήγησε στην πτώση της χρήσης του [29].

[1] https://www.planet-lab.org

Το CoDeeN, μία παρόμοια προσπάθεια με το CoralCDN, επιστρατεύει διαμεσολαβητές για την προσωρινή αποθήκευση και διαμοιρασμό αρχείων ιστοσελίδων. Για την χρήση του είναι απαραίτητη η κατάλληλη ρύθμιση στον φυλλομετρητή του χρήστη, πράγμα που εμποδίζει την καθολική του υιοθέτηση. Όταν ο φυλλομετρητής αιτηθεί ένα αρχείο, οι διαμεσολαβητές ελέγχουν αν το διαθέτουν τοπικά. Αν όχι, τότε προωθούν το αίτημα σε κάποιον άλλο διαμεσολαβητή του συστήματος, λαμβάνοντας υπόψη μετρικές όπως είναι ο φόρτος εργασίας, ο βαθμός αξιοπιστίας και η απόσταση. Το CoDeeN προτείνει τη χρήση μηνυμάτων UDP και TCP heartbeats για τον έλεγχο της κατάστασης των ομότιμων διαμεσολαβητών. Παράλληλα, κάθε διαμεσολαβητής συλλέγει διάφορες πληροφορίες για την κατάστασή του και ενημερώνει περιοδικά το σύστημα. Τέτοιες πληροφορίες είναι η: διαθέσιμη επεξεργαστική ισχύς, τα ανοιχτά αρχεία, η κατάσταση του συστήματος DNS, η χρήση του δικτύου, ο χρόνος λειτουργίας (uptime) κ.α. Το CoDeeN προτείνεται και για την αποφυγή προσπαθειών λογοκρισίας.

Μια διαφορετική οπτική στο θέμα της διανομής περιεχομένου με ομότιμο τρόπο στον ιστό έγινε με την ανάπτυξη του συστήματος Web2Peer [30]. Σκοπός του είναι να επιτρέψει τη δημοσίευση και ανάκτηση ιστοσελίδων με πλήρως ομότιμο τρόπο, παρέχοντας υψηλή διαθεσιμότητα και ανοχή σε σφάλματα. Οι δημιουργοί του ισχυρίζονται ότι η κεντρικοποιημένη αρχιτεκτονική του διαδικτύου, δημιουργεί εξάρτηση από λίγες μεγάλες εταιρείες και ανεβάζει τα κόστη φιλοξενίας μικρών ιστοσελίδων. Η λύση τους, ένας φυλλομετρητής με δυνατότητες ομότιμης επικοινωνίας και εξυπηρέτησης ιστοσελίδων, επιτρέπει σε κάθε χρήστη να δημοσιεύσει τη δική του ιστοσελίδα στο δίκτυο χωρίς κάποιο έξοδο. Όταν ένας χρήστης θέλει να ανεβάσει τη δική του ιστοσελίδα, χρησιμοποιεί τον επεξεργαστή κειμένου που ενσωματώνει ο φυλλομετρητής. Η σελίδα δημοσιεύεται σε ειδική URL που κάνει χρήση του σχήματος πρωτοκόλλου «p2p». Γράφεται σε HTML και μπορούν να χρησιμοποιηθούν ειδικές «meta» ετικέτες προκειμένου να βοηθήσουν αργότερα στην ανεύρεση των αρχείων μέσω λειτουργίας αναζήτησης που επίσης παρέχεται από το σύστημα. Για κάθε ετικέτα, ένα ζευγάρι κλειδιού-τιμής αποθηκεύεται στο DHT του δικτύου (Pastry). Για την αντιγραφή της ιστοσελίδας σε άλλου κόμβους, χρησιμοποιείται παθητική αντιγραφή. Αυτό σημαίνει ότι, προκειμένου ένας κόμβος να αποκτήσει αντίγραφο της σελίδας, θα πρέπει πρώτα να τον έχει επισκεφθεί οικειοθελώς. Συμπερασματικά θα λέγαμε ότι αυτή ήταν, μέχρι στιγμής, η πιο ουσιώδης προσπάθεια για την αλλαγή της αρχιτεκτονικής του παγκόσμιου ιστού. Δυστυχώς όμως, τα οφέλη δεν ήταν προφανή για τους χρήστες και το γεγονός ότι

η πρόσβαση στις ιστοσελίδες γινόταν μόνο μέσω του συγκεκριμένου φυλλομετρητή, αποτέλεσε εμπόδιο στην υιοθέτηση της τεχνολογίας.

Σε αυτό το σημείο, θέλουμε να παρατηρήσουμε ότι το πρόβλημα της κινητοποίησης των χρηστών αποτελεί συχνά ισχυρό εμπόδιο στην υιοθέτηση ομότιμων λύσεων. Κατά καιρούς έχουν προταθεί διάφορες τεχνικές προκειμένου να δοθούν κίνητρα σε χρήστες ώστε να συμμετέχουν ενεργά σε ένα ομότιμο δίκτυο. Διάφορες λύσεις έχουν προταθεί κατά καιρούς, από τη μέθοδο «μία σου και μία μου» (tit-for-tat) του πρωτοκόλλου BitTorrent, μέχρι χρηματικά κίνητρα, όπως προτείνουν τα [31], [32]. Μία κατηγοριοποίηση κινήτρων δίνεται στο [33].

Συνεχίζοντας την παρουσίαση διαφόρων αποπειρών δημιουργίας ομότιμων CDN, πρέπει να κάνουμε αναφορά στην Akamai και το NetSession. Η Akamai, ως ηγέτιδα δύναμη στο χώρο των παρόχων υπηρεσιών CDN, εξαγόρασε το 2007 την εταιρεία RedSwoosh που εξειδικεύονταν σε λύσεις P2P streaming και δύο χρόνια αργότερα εισήγαγε στην αγορά το NetSession, μία προσπάθεια δημιουργίας ενός παγκόσμιου υβριδικού CDN. Το NetSession ήταν εκτελέσιμη εφαρμογή που απαιτούσε εγκατάσταση και έτρεχε στο παρασκήνιο όσο ο υπολογιστής του χρήστη ήταν ανοιχτός. Για να πετύχει ευρεία υιοθέτηση, το NetSession εγκαθίστατο παράλληλα με άλλα προγράμματα σαν bundleware. Το γεγονός αυτό δε βοήθησε τη φήμη του. Αν επισκεφθούμε σήμερα το διαδικτυακό τόπο του προγράμματος, θα παρατηρήσουμε ότι μία από τις συχνότερες ερωτήσεις των χρηστών είναι αν το πρόγραμμα αποτελεί κακόβουλο λογισμικό ή άλλου είδους απειλή [34].

Λίγα χρόνια μετά την ανάπτυξη του CoralCDN, οι δημιουργοί του επανέρχονται με μία καινούργια πρόταση για τη δημιουργία ομότιμων δικτύων διανομής περιεχομένου, την οποία ονομάζουν Firecoral [35]. Πλέον επιδιώκουν, αντί να συμμετέχουν εξυπηρετητές στο δίκτυο, αυτό να απαρτίζεται από τους φυλλομετρητές των ίδιων των πελατών. Το νέο του σύστημα αποτελείται από τέσσερα μέρη: το πρόγραμμα περιήγησης του πελάτη στο οποίο απαιτείται η προσθήκη νέων λειτουργιών μέσω της εγκατάστασης επέκτασης (extension), τον πηγαίο εξυπηρετητή που μένει αμετάβλητος, τους ανιχνευτές (trackers) που διαχειρίζονται πληροφορίες κατάστασης και τις υπηρεσίες υπογραφής (signing services) του περιεχομένου. Η επέκταση του φυλλομετρητή αναλαμβάνει ταυτόχρονα ρόλο forward proxy και εξυπηρετητή ιστού με τρόπο παρόμοιο με το Web2Peer, αξιοποιώντας τη δυνατότητα των επεκτάσεων του Firefox να δημιουργούν απευθείας TCP συνδέσεις στο διαδίκτυο. Ο tracker από την

άλλη, διαχειρίζεται μια λίστα με τα κομμάτια των αρχείων που διαθέτει ο κάθε φυλλομετρητής, ενώ οι υπηρεσίες υπογραφής – που μπορεί να είναι στο ίδιο σύστημα με τον tracker – αναλαμβάνουν να δημιουργήσουν τιμές κατατεμαχισμού (hashes) για τμήματα των αρχείων, ώστε να είναι δυνατή η επιβεβαίωση του περιεχομένου τους. Παρατηρείστε ότι αναφερθήκαμε σε κομμάτια και όχι σε ολόκληρα αρχεία. Το FireCoral είναι μία από τις λίγες υλοποιήσεις που προσπαθούν να μεγιστοποιήσουν την ταχύτητα μετάδοσης ενός αρχείο μέσω παράλληλης λήψης τμημάτων από πολλούς ομότιμους χρήστες. Η νέα αυτή λύση προσφέρει το σημαντικό πλεονέκτημα του χαμηλότερου κόστους έναντι του CoralCDN. Όμως παρουσιάζει προβλήματα στην κλιμάκωση της απόδοσης του tracker και στη διάσχιση των NAT, ενώ χαρακτηρίζεται και από έλλειψη κινήτρων για τον χρήστη ώστε να εγκαταστήσει το απαραίτητο extension. Τέλος, θα θέλαμε να αναφέρουμε ότι σήμερα δεν υπάρχει η δυνατότητα ανάπτυξης μίας επέκτασης με παρόμοιες δυνατότητες, αφού το XPCOM API της Mozilla δεν παρέχεται πλέον σαν επιλογή, μιας και έχει αντικατασταθεί από το WebExtensions API [36] που είναι σημαντικά πιο περιορισμένο σε δυνατότητες.

Την ίδια εποχή με το Firecoral άλλο ένα παρόμοιο σύστημα βασισμένο σε επέκταση φυλλομετρητή ανακοινώνεται. Το FlowerCDN [37], [38] φιλοδοξεί να δημιουργήσει ένα ομότιμο δίκτυο διανομής περιεχομένου το οποίο οργανώνεται βάσει τοποθεσίας και ενδιαφέροντος (locality and interest based). Σαν ένα ομότιμο CDN, ορίζει τέσσερις σημαντικές μετρικές στις οποίες θα πρέπει να αποδίδει καταλλήλως: στον χρόνο απόκρισης, στη δυνατότητα κλιμάκωσης, στο hit ratio (ποσοστό αιτήσεων που ικανοποιούνται από το CDN) και στη διαθεσιμότητα. Για το λόγο αυτό επιλέγει ένα συνδυασμό δομημένης και μη δομημένης δομής δικτύου επικάλυψης. Κάθε χρήστης που μοιράζεται τα ίδια ενδιαφέροντα (ιστοσελίδες) και βρίσκεται στην ίδια περιοχή με άλλους, συμμετέχει σε ένα μικρό, μη δομημένο δίκτυο επικάλυψης που χρησιμοποιεί μεθόδους gossip για την οργάνωσή του και καλείται πέταλο (petal). Τα πέταλα στη συνέχεια οργανώνονται με δομημένο τρόπο (DHT), προκειμένου κάθε χρήστης να μπορεί να αναζητήσει με αποδοτικό τρόπο πέταλα για την τοποθεσία του και την ιστοσελίδα που θέλει να επισκεφθεί. Ένα σημαντικό μειονέκτημα του συστήματος είναι η ακαταλληλότητά του για μη δημοφιλείς ιστοσελίδες.

Άλλη μια προσπάθεια που εντοπίζει τη σημασία της συσχέτισης των πελατών με βάση τα ενδιαφέροντα τους προκειμένου να σχηματίσουν αποδοτικό ομότιμο CDN είναι το WebCloud [39]. Οι ερευνητές παρουσιάζουν τη διαπίστωσή τους ότι τα κλασσικά

CDN δεν αποδίδουν ικανοποιητικά όταν χρησιμοποιούνται για περιεχόμενο κοινωνικών δικτύων όπως το Facebook[1]. Συγκεκριμένα εντοπίζουν πως το υλικό πλατφορμών κοινωνικής δικτύωσης παράγεται στις άκρες του διαδικτύου και παρουσιάζει ομοιόμορφη ζήτηση η οποία περιορίζεται γεωγραφικά σε ένα τόπο. Για την λειτουργία του συστήματος που προτείνουν, απαιτείται η εγκατάσταση πληρεξούσιων εξυπηρετητών από τον ιδιοκτήτη του ιστοτόπου και η συμπερίληψη ενός σεναρίου JavaScript σε κάθε ιστοσελίδα. Το σενάριο επικοινωνεί με τον πληρεξούσιο εξυπηρετητή, προκειμένου να λάβει το περιεχόμενο της ιστοσελίδας, το οποίο το αποθηκεύει στο LocalStorage του φυλλομετρητή. Ο πληρεξούσιος εξυπηρετητής αναλαμβάνει να διατηρεί συνδέσεις με όλους τους πελάτες μιας περιοχής, καθώς και λίστες με τα αρχεία που αυτοί διαθέτουν στο LocalStorage τους. Όταν ζητείται κάποιο αρχείο, ο πληρεξούσιος εξυπηρετητής αναζητά στις λίστες του άλλους πελάτες που έχουν προηγουμένως προσπελάσει και αποθηκεύσει το αρχείο, το λαμβάνει από αυτούς και το προωθεί στον αιτούντα πελάτη. Αν και αυτή η τεχνική καταφέρνει να περιορίσει την κίνηση δεδομένων μεταξύ AS (autonomous system), στην ουσία διπλασιάζει τη συνολική κίνηση μέσα σε αυτό λόγω της διαμεσολάβησης του πληρεξούσιου εξυπηρετητή. Η εφαρμογή αυτής της τεχνικής μπορεί να έχει νόημα μόνο σε πολύ ιδιάζουσες περιπτώσεις. Θα λέγαμε ότι η αρχιτεκτονική του συστήματος είναι μία ιδιαίτερη μορφή έμμεσα ομότιμης δικτύωσης.

Το επόμενο σύστημα που θα μελετήσουμε, το οποίο μάλιστα μοιάζει περισσότερο με τη δική μας υλοποίηση, είναι το Maygh [40]. Το Maygh εκμεταλλεύεται τα πρωτόκολλα RTMFP και WebRTC, προκειμένου να επιτύχει άμεση και πραγματικού χρόνου επικοινωνία μεταξύ των πελατών μιας ιστοσελίδας. Στόχος είναι η ελαχιστοποίηση του κόστους εξυπηρέτησης μίας σελίδας. Οι δυνατότητες του Maygh δύναται να αξιοποιηθούν μέσω της βιβλιοθήκης JavaScript στον φυλλομετρητή του χρήστη. Για τη λήψη περιεχομένου, ο φυλλομετρητής στέλνει αίτημα σε ένα κεντρικό εξυπηρετητή που λειτουργεί σαν tracker, προκειμένου να του προταθούν χρήστες από τους οποίους μπορεί να πραγματοποιήσει τη λήψη. Μόλις λάβει τους προτεινόμενους χρήστες, ο φυλλομετρητής αιτείται σύνδεση με έναν από αυτούς, στέλνοντας το αίτημα του σε αυτόν μέσω του εξυπηρετητή που αναλαμβάνει το ρόλο καναλιού σηματοδότησης. Σε αυτό το σημείο αν δεν υπάρξει πρόβλημα, η σύνδεση θα επιτευχθεί και ο φυλλομετρητής θα στείλει αίτημα απευθείας στον ομότιμό του για να ξεκινήσει η

[1] https://www.facebook.com

μεταφορά περιεχομένου. Αυτή η υλοποίηση είναι η πιο διαφανής στο χρήστη από όσες εξετάσαμε, αφού δεν απαιτεί κάποιου είδους εγκατάσταση λογισμικού.

Το τελευταίο σύστημα που μελετήσαμε είναι το BemTV [41], ένα υβριδικό σύστημα CDN με στόχο την υποστήριξη live streaming βίντεο – συγκεκριμένα με τη χρήση του πρωτοκόλλου HLS[1] που αναπτύσσει η Apple. Το BemTV, όπως το Maygh, χρησιμοποιεί την τεχνολογία WebRTC προκειμένου να δημιουργήσει ένα χαμηλού κόστους CDN με υψηλές δυνατότητες κλιμάκωσης. Η δυναμικότητα των ζωντανών ροών βίντεο περιπλέκει την ανάπτυξη ενός τέτοιου συστήματος, όμως τα δυνητικά οφέλη μπορεί να είναι ιδιαιτέρως μεγάλα λόγω της φύσης του περιεχομένου. Το BemTV έχει εξαγοραστεί από την Streamroot, μία από τις μεγαλύτερες εταιρείες στον χώρο των ομότιμων CDN.

Κλείνοντας αυτή την ενότητα, θα θέλαμε να αναφερθούμε στην αγορά των εμπορικών ομότιμων δικτύων διανομής περιεχομένου. Σύμφωνα με έρευνα αγοράς [42] της MarketsandMarkets Analysis, η αγορά των επαγγελματικών P2P CDN αναμένεται να διπλασιαστεί μεταξύ 2018 και 2023 και θα αγγίξει τα 2,6 δις USD (Εικόνα 2-2).


**Εικόνα 2-2: Μέγεθος αγοράς ομότιμων δικτύων διανομής περιεχομένου. (πηγή: MarketsandMarkets Analysis)**

Σύμφωνα με την έρευνα, η άνοδος οφείλεται στη διαρκώς αυξανόμενη ζήτηση για υψηλής ποιότητας βίντεο σε φορητές συσκευές. Ως σημαντικότερους παρόχους αναφέρει τις εξής εταιρείες:

- Streamroot Inc.

[1] https://developer.apple.com/streaming

- Peer5 Inc.
- Viblast
- Globecast
- Qumu Corp.
- CDN Video
- Play2Live
- Kollective
- PeerApp
- Akamai Technologies
- Alibaba group Holding Ltd.
- StriveCDN

# 3 Τεχνολογικές Υποδομές

Η ανάπτυξη του PAWS με τις δυνατότητες και τα χαρακτηριστικά που διαθέτει, δε θα ήταν δυνατή χωρίς τις τεχνολογίες λογισμικού που παρουσιάζουμε σε αυτό το κεφάλαιο.

Αν και η ιστορία του Παγκόσμιου Ιστού ξεκινά περίπου 30 χρόνια πριν τη συγγραφή αυτής της διπλωματικής εργασίας, το 1989, τα τελευταία χρόνια ειδικότερα, παρατηρείται μία ραγδαία ανάπτυξη όσο αφορά τα είδη των εφαρμογών και των υπηρεσιών που μπορούν να αναπτυχθούν σε αυτόν. Μάλιστα, φαίνεται να υπάρχει μία ισχυρή ώθηση στην κοινότητα των προγραμματιστών, προκειμένου οι τεχνολογίες ιστού να καθιερωθούν ως ανταγωνιστική επιλογή για τη δημιουργία desktop εφαρμογών που τρέχουν τοπικά σε ένα υπολογιστή.

Δημοφιλής desktop εφαρμογές που έχουν χτιστεί βασιζόμενες σε τεχνολογίες ιστού, περιλαμβάνουν: επεξεργαστές κειμένου όπως το Visual Studio Code1 της Microsoft και τον Atom2 του GitHub, πλατφόρμες επικοινωνίας όπως το Slack3 και το Discord4, υπηρεσίες streaming όπως το Spotify5 και το TIDAL6.

Βλέπουμε ότι το μοντέλο ανάπτυξης (developing paradigm) του ιστού επεκτείνεται έξω από τα στενά όρια του φυλλομετρητή. Ξεχωρίζουμε τρεις λόγους που παρατηρείται η τάση αυτή:

- Η ύπαρξη ήδη έτοιμου τμήματος του κώδικα που απαιτείται για την ανάπτυξη μίας native εφαρμογής και η επιθυμία να επαναχρησιμοποιηθεί – για παράδειγμα υπάρχων κώδικας για την έκδοση ιστού της εφαρμογής (reusable code).
- Η ανάπτυξη με τη χρήση τεχνολογιών ιστού είναι ανεξάρτητη πλατφόρμας (cross-platform). Έτσι αποφεύγεται η κατασκευή διαφορετικών εκδόσεων για κάθε πλατφόρμα-λειτουργικό, διευκολύνοντας την ανάπτυξη και μειώνοντας την απαιτούμενη επένδυση σε κόστος και χρόνο.

[1] https://code.visualstudio.com

[2] https://atom.io

[3] https://slack.com

[4] https://discordapp.com

[5] https://www.spotify.com

[6] https://tidal.com

- Είναι ευκολότερη η πρόσληψη προσωπικού που γνωρίζει καλά ένα περιορισμένο σετ τεχνολογιών και το οποίο, λόγω της ομοιογένειας του συνόλου του πηγαίου κώδικα (code base), μπορεί να απασχοληθεί σε όλο το φάσμα της εφαρμογής.


**Εικόνα 3-1: Electron και nw.js - Δύο frameworks που επιτρέπουν την ανάπτυξη desktop εφαρμογών με τη χρήση τεχνολογιών ιστού**

Πέρα από τον προγραμματισμό εφαρμογών desktop, την ίδια τάση για επέκταση των δυνατοτήτων των τεχνολογιών ιστού παρατηρούμε και στους φυλλομετρητές. Στον ιστό, η ανάπτυξη μίας εφαρμογής περιορίζεται από τις διαθέσιμες τεχνολογίες που επιλέγει να υποστηρίξει κάθε κατασκευαστής φυλλομετρητή (browser vendor). Συνήθως αυτές οι τεχνολογίες, ξεκινούν ως ανοιχτά πρότυπα (open standards) από οργανισμούς-κοινοπραξίες τεχνολογικών κολοσσών, που σκοπό έχουν την ανάπτυξη του παγκόσμιου ιστού μακροχρόνια και εισάγονται σταδιακά στους διάφορους φυλλομετρητές.

Ο όρος “Web 1.0” περιγράφει τον ιστό όταν πρωτο-επινοήθηκε από τον Tim Burners Lee ως ένα σύστημα για την προσπέλαση αρχείων – ένα σύστημα αυστηρά για ανάγνωση. Το “Web 2.0” ήρθε στις αρχές της χιλιετίας για να προσφέρει μία προσωποποιημένη εμπειρία χρήσης που επιτρέπει την ανάπτυξη εφαρμογών ιστού – σε αντίθεση με τις απλές στατικές ιστοσελίδες – όπως, μεταξύ άλλων, ηλεκτρονικά καταστήματα, κοινωνικά δίκτυα και εφαρμογές ηλεκτρονικής διακυβέρνησης. Πλέον ο ιστός αποκτά μνήμη και λογική, είναι ένα σύστημα εγγραφής και ανάγνωσης.

Επεκτείνοντας το παραδοσιακό μοντέλο δικαιωμάτων αρχείων (read-write-execute), θα μπορούσαμε να εικάσουμε ότι στο επόμενο στάδιο το web γίνεται εκτελέσιμο. Κατά μία έννοια, αυτό ήδη εξελίσσεται σε πραγματικότητα, αρκεί να κοιτάξουμε μερικές από τις τελευταίες τεχνολογίες που έχουν εισαχθεί στους περιηγητές.

Αναφερόμαστε στα λεγόμενα PWAs (progressive web apps), που επιτρέπουν σε μία ιστοσελίδα να συμπεριφέρεται σαν εγκατεστημένη εφαρμογή. Αυτό επιτυγχάνεται με τη χρήση προτύπων όπως είναι: 1) το αρχείο manifest που περιλαμβάνει τα μεταδεδομένα της, 2) οι Service Workers που επιτρέπουν τη λήψη ειδοποιήσεων και ενημερώσεων της εφαρμογής καθ' όσο είναι στο παρασκήνιο, 3) η νέα προγραμματιστική διεπαφή (API) Cache που δίνει τη δυνατότητα σε εφαρμογές να λειτουργούν ακόμη κι όταν δεν υπάρχει πρόσβαση στο διαδίκτυο, 4) τα IndexedDB και WebStorage που επιτρέπουν την αποθήκευση της κατάστασης της εφαρμογής, 5) η WebAssembly που κάνει δυνατή την εκτέλεση κώδικα σαν να τρέχει εγγενώς, 6) το πρότυπο WebGL που δίνει πρόσβαση στη κάρτα γραφικών του συστήματος και το WebXR που υποστηρίζει την κατανάλωση υλικού εικονικής/επαυξημένης πραγματικότητας, 7) το File API και το WebUSB που δίνουν πρόσβαση σε αρχεία του συστήματος και σε συνδεδεμένες συσκευές USB αντιστοίχως.

Βέβαια, όλες αυτές οι δυνατότητες που προστέθηκαν και συνεχίζουν να προστίθενται στους περιηγητές ιστού έχουν κι ένα μεγάλο αρνητικό. Μετατρέπουν τους περιηγητές σε υπερβολικά περίπλοκα συστήματα λογισμικού που μόνο λίγες μεγάλες εταιρείες είναι δυνατό να εξελίξουν και να συντηρήσουν. Αποτέλεσμα είναι η μείωση του πλήθους επιλογών που διαθέτει ο κάθε χρήστης, με ό,τι συνεπάγεται αυτό για την υγιή πορεία του Παγκόσμιου Ιστού.

Η πρώτη ενότητα του παρόντος κεφαλαίου πραγματεύεται εκείνες τις τεχνολογίες ιστού που έκαναν δυνατή την ανάπτυξη της εφαρμογής μας. Αφορούν ανοιχτά πρότυπα του παγκόσμιου ιστού, που υποστηρίζονται ευρέως από ενημερωμένους φυλλομετρητές.

Στη δεύτερη ενότητα, σειρά παίρνει το backend της εφαρμογής μας. Εκεί θα μιλήσουμε για την επιλογή μας να χρησιμοποιήσουμε το, ανοιχτού κώδικα περιβάλλον προγραμματισμού, Node.js που μετατρέπει στην ουσία, τη γλώσσα προγραμματισμού JavaScript, από γλώσσα σεναρίου για χρήση σε ιστοσελίδες, σε μία γενική γλώσσα προγραμματισμού κατάλληλη για μια πληθώρα χρήσεων, όπως είναι οι γλώσσες Python και Ruby.

## 3.1 Τεχνολογίες στην πλευρά του πελάτη (HTML5)

### *3.1.1 WebRTC*

Το WebRTC (Web Real-Time Communications, Επικοινωνίες Ιστού Πραγματικού Χρόνου) αποτελεί ένα σύνολο τεχνολογιών που επιτρέπουν τη σύγχρονη

επικοινωνία δύο ή περισσοτέρων χρηστών του διαδικτύου με ήχο, εικόνα και δεδομένα, χωρίς τη διαμεσολάβηση, όπου είναι αυτό δυνατό, ενός εξυπηρετητή Ιστού (Web server). Με τον όρο "σύγχρονη" ή "πραγματικού χρόνου" επικοινωνία αναφερόμαστε στο είδος της επικοινωνίας που είναι ευαίσθητη ως προς τον χρόνο διαβίβασης των δεδομένων. Το χαρακτηριστικό αυτό καθιστά απαραίτητη τη χρήση εξειδικευμένων πρωτοκόλλων επικοινωνίας και μεθόδων κωδικοποίησης πολυμέσων ώστε να εξασφαλίζεται η έγκαιρη λήψη των μηνυμάτων.

Πριν το WebRTC, η ανάπτυξη μίας εφαρμογής πραγματικού χρόνου όπως για παράδειγμα μίας εφαρμογής συνδιαλέξεων, απαιτούσε ιδιαίτερη προσπάθεια και πόρους από την πλευρά των προγραμματιστών της, ενώ συχνά γινόταν χρήση κλειστών ιδιόκτητων τεχνολογιών. Το 2011 η Google, έχοντας ήδη εργαστεί πάνω σε δικές της εφαρμογές πραγματικού χρόνου, όπως το Google Hangouts, και διαπιστώσει πόσο προβληματική ήταν η ανάπτυξή τους, αποφασίζει να διαθέσει με μορφή ελεύθερου ανοικτού λογισμικού το WebRTC, το οποίο περιλαμβάνει θεμελιώδεις τεχνολογίες για την κατασκευή τέτοιων εφαρμογών [43]. Έκτοτε η Google συνεργάζεται με αρχές προτυποποίησης όπως η IETF (Internet Engineering Task Force, Επιχειρησιακή Ομάδα Τεχνολογίας του Διαδικτύου) και η W3C (World Wide Web Consortium, Κοινοπραξία του Παγκοσμίου Ιστού) για την τυποποίησή του και προτρέπει την κοινότητα να το υιοθετήσει ως το προκαθορισμένο πρότυπο για πολυμεσικές εφαρμογές πραγματικού χρόνου στον παγκόσμιο ιστό.

Σήμερα το WebRTC υποστηρίζεται πλήρως ή εν μέρει, από τους σημαντικότερους φυλλομετρητές. Συνολικά εκτιμάται ότι περίπου το 87% των χρηστών παγκοσμίως χρησιμοποιούν φυλλομετρητή με υποστήριξη WebRTC [44]. Οι πιο δημοφιλείς φυλλομετρητές που δεν υποστηρίζουν στην τελευταία τους έκδοση, καμία από τις δυνατότητες του WebRTC είναι ο UC για Android, ο Internet Explorer και ο Opera Mini. Προηγουμένως, όποιος ήθελε να αναπτύξει μία εφαρμογή πραγματικού χρόνου που να λειτουργεί στον παγκόσμιο ιστό, έπρεπε να κάνει χρήση τεχνολογιών όπως Flash, Silverlight ή Java Applets και ενδεχομένως να ζητήσει από τον χρήστη την εγκατάστασή τους στη συσκευή του. Αυτό αποτελούσε αποθαρρυντικό παράγοντα στην υιοθέτηση μιας εφαρμογής τέτοιου είδους. Πλέον, με την έλευση του WebRTC οι περιηγητές ιστού διαθέτουν όλες τις απαραίτητες διεπαφές προγραμματισμού εφαρμογών (APIs) που καθιστούν δυνατή την ανάπτυξη τέτοιων εφαρμογών χωρίς την ανάγκη τρίτων προσθέτων (plugins).

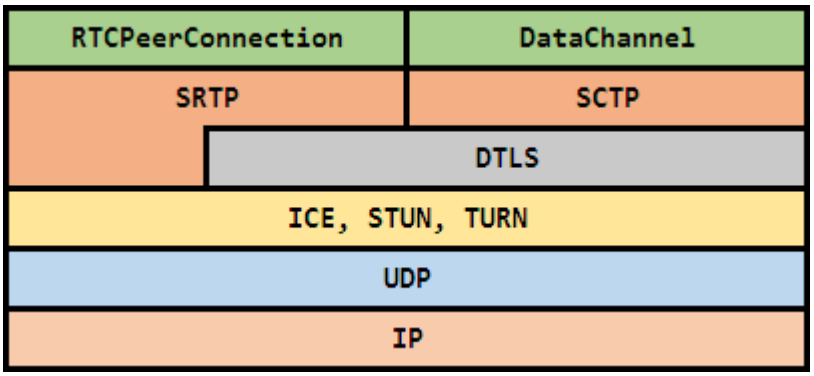
Το WebRTC αποτελεί αναντικατάστατο κομμάτι της υλοποίησής μας. Λόγω της ανάγκης από εφαρμογές πραγματικού χρόνου για δυνατότητα έγκαιρης παράδοσης των δεδομένων τους, ήταν απαραίτητο να δοθεί στους φυλλομετρητές η δυνατότητα να συνδεθούν απευθείας μεταξύ τους, χωρίς τη διαμεσολάβηση εξυπηρετητών ιστού, έτσι ώστε να μειωθεί ο λανθάνων χρόνος (latency) και να βελτιωθεί η αποκρισιμότητα των εφαρμογών. Αυτή τη δυνατότητα εκμεταλλευόμαστε στο τελευταίο κομμάτι της εργασίας μας, για να δημιουργήσουμε ένα δίκτυο ομότιμων χρηστών, οπού ο διακομιστής έχει όσο το δυνατόν μικρότερο ρόλο και η εξυπηρέτηση των αντικειμένων της ιστοσελίδας αναλαμβάνεται από τους ίδιους τους χρήστες.

#### *3.1.1.1 Προτυποποίηση*

Στην προτυποποίηση του WebRTC συμμετέχουν δύο διεθνείς φορείς, η IETF σε επίπεδο πρωτοκόλλου και η W3C σε επίπεδο διεπαφών προγραμματισμού. Μέσα στο IETF έχει συγκροτηθεί η ομάδα εργασίας RTCWEB, η οποία είναι υπεύθυνη για τον ορισμό των βασικών λειτουργιών που υποστηρίζει το WebRTC όπως για παράδειγμα οι κωδικοποιητές ροών ήχου και εικόνας, η διάσχιση δικτύων πίσω από NAT (NAT traversal), η δημιουργία συνεδριών και η διατερματική κρυπτογράφηση και ασφάλεια. Από την άλλη, στην W3C, η ομάδα εργασίας WEBRTC είναι αρμόδια για τις διεπαφές προγραμματισμού εφαρμογών που διατίθενται στην πλευρά των περιηγητών Ιστού. Είναι σημαντικό οι διεπαφές να μην αποτελούν σημαντική αφαίρεση (abstraction) των πρωτοκόλλων, ώστε να μη χάνεται η πρόσβαση σε απαραίτητες δυνατότητες και να είναι εφικτή η ανάπτυξη πολύπλοκων εφαρμογών με καινοτόμες λειτουργίες. Στην πράξη οι γνώμες διίστανται, αφού από τη μία πλευρά υπάρχει απαίτηση για έλεγχο σε χαμηλότερο επίπεδο και από την άλλη το υπάρχον περιβάλλον ανάπτυξης θεωρείται ήδη περίπλοκο [45], [46].

#### *3.1.1.2 Το WebRTC ως πρωτόκολλο επικοινωνίας*

Η ανάπτυξη του WebRTC δεν ξεκίνησε από το μηδέν. Αντιθέτως, το WebRTC σαν τεχνολογία επικοινωνίας βασίζεται εν μέρει σε ήδη υπάρχοντα πρωτόκολλα, τα οποία εκμεταλλεύεται προκειμένου να λειτουργεί απρόσκοπτα και συμβατά με ήδη υπάρχουσες δικτυακές εφαρμογές. Στην Εικόνα 3-2 βλέπουμε τα βασικά πρωτόκολλα που χρησιμοποιούνται για την επίτευξη μίας WebRTC σύνδεσης.




**Εικόνα 3-2: Τα πρωτόκολλα του WebRTC**

Θα ξεκινήσουμε την περιγραφή τους, προσεγγίζοντάς τα από κάτω προς τα πάνω – από το γενικό προς το ειδικό.

***IP και διάσχιση NAT***

Στη βάση του σχήματος, βλέπουμε το πρωτόκολλο IP. Το IP είναι το βασικό πρωτόκολλο πάνω στο οποίο στηρίζεται σήμερα το διαδίκτυο. Αποτελεί ένα μοναδικό αναγνωριστικό για μία συσκευή δικτύου – μία διεύθυνση. Ο χώρος διευθύνσεων IP ορίζεται ως το σύνολο των δυνατών τιμών που υποστηρίζει το πρωτόκολλο και είναι διαχωρισμένος σε δύο τμήματα, στο χώρο των δημόσιων και στο χώρο των ιδιωτικών διευθύνσεων. Αντίστοιχα οι διευθύνσεις, ανάλογα με το χώρο στον οποίο ανήκουν χαρακτηρίζονται ως δημόσιες και ιδιωτικές.

Για τη σύνδεση με το διαδίκτυο απαιτείται μία δημόσια IP, όμως ο αριθμός τους ήταν περιορισμένος μέχρι την ανάπτυξη της έκτης έκδοσης του πρωτοκόλλου (IPv6). Προκειμένου να αποφευχθεί η εξάντληση του χώρου δημοσίων διευθύνσεων, η τεχνολογία NAT (Network Address Translation – Μετάφραση Διευθύνσεων Διαδικτύου) αναπτύχθηκε. Η NAT φροντίζει να αντιστοιχεί μία ή περισσότερες ιδιωτικές διευθύνσεις σε μία δημόσια.

Μία πύλη NAT (NAT gateway) είναι μία δικτυακή συσκευή που αναλαμβάνει να διατηρεί ένα πίνακα μετάφρασης διευθύνσεων (IP translation table). Όταν λάβει ένα αίτημα σύνδεσης από μία συσκευή του ιδιωτικού δικτύου η οποία προορίζεται για μία απομακρυσμένη διαδικτυακή συσκευή, αλλάζει την πηγαία διεύθυνση του IP πακέτου με τη δική της δημόσια IP, σημειώνοντας τις απαραίτητες πληροφορίες σύνδεσης (τις διευθύνσεις IP των δύο συσκευών και τις θύρες (ports) που χρησιμοποιούν) στον πίνακα μετάφρασης. Με αυτόν τον τρόπο, όταν η απομακρυσμένη συσκευή απαντήσει, η πύλη γνωρίζει που να ανακατευθύνει τα δεδομένα της σύνδεσης.

Η τεχνολογία NAT έχει εφαρμοστεί κατά κόρον στο διαδίκτυο. Η δημοφιλία της έγκειται στο γεγονός ότι είναι απλή στη χρήση της, καθυστερώντας παράλληλα αποτελεσματικά, την εξάντληση των δημόσιων διευθύνσεων IP. Όμως, παρά τα θετικά της στοιχεία, η NAT ως τεχνολογία εμποδίζει τη δημιουργία εισερχόμενων συνδέσεων από το διαδίκτυο, πράγμα που δυσκολεύει την ανάπτυξη ομότιμων δικτύων.

Μία απομακρυσμένη διαδικτυακή συσκευή που επιδιώκει να συνδεθεί με μία τοπική συσκευή σε ιδιωτικό δίκτυο πίσω από NAT, γνωρίζει μόνο την δημόσια διεύθυνση της πύλης. Όταν επομένως στείλει ένα IP πακέτο προς τη τοπική συσκευή, αυτό θα φτάσει πρώτα στην πύλη. Μη γνωρίζοντας σε ποιον απευθύνεται το πακέτο, αφού δε θα έχει προηγουμένως σημειώσει στον πίνακα μετάφρασης τις απαραίτητες πληροφορίες αντιστοίχισης, η πύλη θα το απορρίψει.

Για τη διάσχιση της δικτυακής επικοινωνίας σε δίκτυα που χρησιμοποιούν NAT συσκευές (NAT Traversal), έχουν προταθεί ποικίλες μέθοδοι. Ανάλογα με το είδος του NAT που εφαρμόζεται στο δίκτυο, η επιτυχία αυτών των μεθόδων δεν είναι πάντα εγγυημένη. Παρακάτω θα δούμε πώς το WebRTC χειρίζεται τη διάσχιση των NAT.

***ICE, STUN και TURN***

Το ICE (Interactive Connectivity Establishment) είναι το πρωτόκολλο που επιλέχθηκε από το WebRTC προκειμένου να κάνει εφικτή την ομότιμη σύνδεση δύο απομακρυσμένων συσκευών, ακόμα κι όταν παρεμβάλλονται πύλες NAT. Το ICE [47] φιλοδοξεί να αποτελέσει τον καθολικό τρόπο διάσχισης NAT, αποδίδοντας ικανοποιητικά σε κάθε τοπολογία δικτύου.

Σημαντική προϋπόθεση για τη χρήση του ICE είναι οι δύο συσκευές που θέλουν να συνδεθούν, να έχουν τη δυνατότητα να επικοινωνήσουν μεταξύ τους με έμμεσο τρόπο, μέσα από ένα κανάλι σηματοδότησης. Συνήθως αυτό επιτυγχάνεται μέσω κάποιου εξυπηρετητή που διαμεσολαβεί – αν και το ICE δεν ορίζει συγκεκριμένο τρόπο. Η έμμεση επικοινωνία των προς σύνδεση συσκευών, έχει σκοπό την ανταλλαγή απαραίτητων πληροφοριών σύνδεσης μέσα από μηνύματα SDP (Session Description Protocol). Η δομή των μηνυμάτων SDP, ορίζεται στο [48] και περισσότερες πληροφορίες για τον τρόπο με τον οποίο χρησιμοποιούνται στο WebRTC περιγράφεται στο [49].

Στα μηνύματα αυτά, το ICE προσθέτει συνδυασμούς πρωτοκόλλων μεταφοράς – διευθύνσεων IP και θυρών επικοινωνίας – με τους οποίους είναι πιθανώς προσβάσιμη μία συσκευή και στους οποίους αναφερόμαστε με τον όρο ICE Candidate.

Ο πράκτορας ICE (ICE Agent) του φυλλομετρητή αναλαμβάνει την συλλογή όλων των ICE Candidate. Για το σκοπό αυτό, συνεργάζεται με το λειτουργικό σύστημα της συσκευής καθώς επίσης με τρίτους εξυπηρετητές τύπου STUN και TURN.

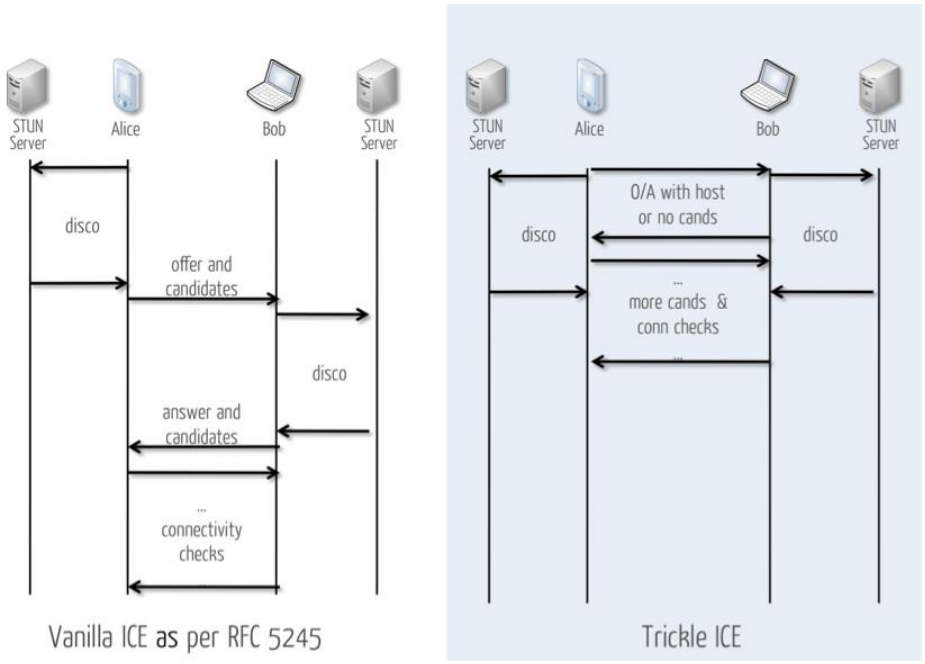
Οι ICE Candidate θα σταλούν στη συνέχεια, μέσω του καναλιού σηματοδοσίας, στην απομακρυσμένη συσκευή. Όταν οι δύο συσκευές ολοκληρώσουν τη συλλογή των τοπικών τους ICE Candidate και λάβουν τους αντίστοιχους των ομότιμών τους, θα ξεκινήσει ο έλεγχος συνδεσιμότητας. Για κάθε πιθανό συνδυασμό τοπικών και απομακρυσμένων ICE Candidate, κάθε συσκευή στέλνει ένα αίτημα τύπου STUN στην άλλη. Αν η λήψη του αιτήματος πραγματοποιηθεί επιτυχώς, η απομακρυσμένη συσκευή πρέπει να το επιβεβαιώσει με δική της απόκριση. Αμέσως μετά δοκιμάζει τον ίδιο συνδυασμό από την πλευρά της. Εφόσον και η δεύτερη συνδιαλλαγή είναι επιτυχής, αυτό το ζευγάρι των ICE Candidate αποτελεί έναν τρόπο με τον οποίο οι δύο συσκευές μπορούν να συνδεθούν και να επικοινωνήσουν απευθείας μεταξύ τους.

Παραπάνω αναφερθήκαμε σε εξυπηρετητές STUN και TURN, χωρίς να εξηγήσουμε το ρόλο τους. Το STUN (Session Traversal Utilities for NAT) είναι ένα πρωτόκολλο που έχει διττή λειτουργία [50]. Πρώτον, επιτρέπει την εύρεση της δημόσιας IP και θύρας μίας συσκευής που βρίσκεται πίσω από ένα NAT, μέσω της ερώτησης ενός τρίτου εξυπηρετητή (STUN server). Με αυτόν τον τρόπο μπορεί να δημιουργήσει ICE Candidate που εμπεριέχουν τη δημόσια IP της συσκευής, η οποία είναι απαραίτητη για συνδέσεις μέσω διαδικτύου. Δεύτερον, χρησιμοποιείται για τον έλεγχο της συνδεσιμότητας των δύο συσκευών, μέσω χειραψίας τεσσάρων σταδίων, όπως περιγράφηκε παραπάνω.

Από την άλλη πλευρά, το πρωτόκολλο TURN [51] επιλέγεται όταν δεν είναι δυνατή η απευθείας σύνδεση των δύο συσκευών. Δυστυχώς υπάρχουν περιπτώσεις, όπως η ύπαρξη συμμετρικού NAT και στα δύο άκρα του δικτύου, που δεν επιτρέπουν την ολοκλήρωσή της ομότιμης σύνδεσης. Όταν ισχύει κάτι τέτοιο, είναι δυνατόν να αξιοποιήσουμε τις υπηρεσίες ενός διακομιστή TURN. Αυτός είναι ένας δημόσια προσβάσιμος υπολογιστής, που αναλαμβάνει να μεταβιβάσει διαφανώς τα μηνύματα των δύο συσκευών. Καθώς όλα τα δεδομένα περνούν μέσα από αυτόν, η σύνδεση δεν είναι

άμεση με αποτέλεσμα το (υψηλό) κόστος επικοινωνίας να βαραίνει και τον διαχειριστή του και η σύνδεση να χαρακτηρίζεται από υψηλή καθυστέρηση διαβίβασης μηνυμάτων.

Για την επιτυχή και ευρεία υιοθέτηση ενός πρωτοκόλλου, αυτό δεν αρκεί μόνο να προσφέρει λύση στο πρόβλημα για το οποίο σχεδιάστηκε. Θα πρέπει, επίσης, η λύση να είναι αποδοτική. Βλέπουμε ότι το ICE απαιτεί τη δημιουργία αιτήματος σε εξυπηρετητή STUN, προτού κατασκευαστούν και διανεμηθούν οι ICE Candidate. Αυτή η διαδικασία δεν απαιτεί συνήθως, παραπάνω από μερικά δευτερόλεπτα. Όμως, έστω και αυτός ο λίγος χρόνος καθυστερεί την εγκαθίδρυση της ομότιμης σύνδεσης και δρα αρνητικά στην εμπειρία του χρήστη. Η καθυστέρηση εντείνεται από το γεγονός ότι η απομακρυσμένη συσκευή, δεν έχει λάβει ακόμη το αίτημα σύνδεσης.




**Εικόνα 3-3: Σύγκριση ICE και Trickle ICE - Διάγραμμα ακολουθίας μηνυμάτων**

Μία παραλλαγή του ICE, το Trickle ICE [52], επιτρέπει την εγκαθίδρυση ομότιμων συνδέσεων σε σημαντικά λιγότερο χρόνο. Στην εικόνα γίνεται σύγκριση των διαγραμμάτων ακολουθίας στις δύο εκδοχές του ICE. Παρατηρούμε στην ουσία ότι το Trickle ICE μετατρέπει τη διαδικασία ICE από σύγχρονη σε ασύγχρονη. Κατά το Trickle ICE, το αίτημα της ομότιμης σύνδεσης αποστέλλεται χωρίς ICE Candidate ή μόνο με ICE Candidate που περιέχουν τοπικές διευθύνσεις IP. Ενημερώνοντας την απομακρυσμένη συσκευή για την πρόθεση σύνδεσης νωρίτερα, επιτυγχάνουμε την

παράλληλη αναζήτηση για ICE Candidate από τις δύο συσκευές. Καθώς αυτοί συγκεντώνονται, αποστέλλονται κανονικά και ο έλεγχος συνδεσιμότητας γίνεται με δυναμικό τρόπο.

***SCTP και SRTP, με DTLS πάνω σε UDP***

Στην προηγούμενη ενότητα είδαμε πώς επιτυγχάνεται η διάσχιση NAT για την άμεση σύνδεση δύο μηχανών. Αφότου βρούμε, μέσω ICE, το μονοπάτι του διαδικτύου με το οποίο θα συνδεθούμε, μένει να δούμε με ποιόν τρόπο οι ομότιμες συσκευές μπορούν να ανταλλάξουν μηνύματα.

Στο διαδίκτυο δύο είναι τα κατεξοχήν πρωτόκολλα μεταφοράς, το TCP και το UDP. Στην περίπτωση του WebRTC, επιλέχθηκε η αξιοποίηση του πρωτοκόλλου UDP. Αιτία για αυτή την απόφαση, ήταν η απαίτηση για όσο το δυνατόν μικρότερη καθυστέρηση στη μετάδοση των δεδομένων – απαίτηση που πηγάζει από την επιθυμία υποστήριξης εφαρμογών πραγματικού χρόνου [53].

Το UDP δεν προσφέρει υπηρεσίες όπως παραλαβή δεδομένων με σωστή σειρά, επαναμετάδοση σε περίπτωση σφάλματος, έλεγχο συμφόρησης κ.α. Σε αντίθεση με το TCP, το UDP είναι ένα stateless (χωρίς κατάσταση) και άρα connectionless (χωρίς σύνδεση) πρωτόκολλο μεταφοράς. Με αυτό τον τρόπο αποφεύγονται χρονοβόρες διαδικασίες σύνδεσης και η μεταφορά των δεδομένων μπορεί να ξεκινάει άμεσα.

Τα μηνύματα που χρησιμοποιεί το πρωτόκολλο UDP ονομάζονται datagrams. Σε ένα datagram περιέχονται δεδομένα επικεφαλίδας απαραίτητα για τη λειτουργία του πρωτοκόλλου. Το συνολικό τους μέγεθος είναι μόλις 8 bytes, επιβαρύνοντας ελάχιστα το συνολικό μέγεθος των δεδομένων αποστολής. Μετά την επικεφαλίδα, ακολουθεί το αντικείμενο της επικοινωνίας. Δυστυχώς, στο UDP υπάρχει περιορισμός στο μέγεθος του datagram, ίσος με 65.536 bytes συνολικά ή 65.527 bytes ωφέλιμου φορτίου. Αν ένα αντικείμενο προς αποστολή είναι μεγαλύτερο, θα πρέπει να διασπαστεί σε μικρότερα τμήματα στο επίπεδο της εφαρμογής.

Όπως γίνεται κατανοητό, η απλότητα του UDP αποτελεί εμπόδιο για την ανάπτυξη εφαρμογών με υψηλότερες απαιτήσεις. Το WebRTC επιλέγει τα πρωτόκολλα SRTP (Secure Real-Time Transport Protocol – Ασφαλές Πρωτόκολλο Μεταφοράς Δεδομένων Πραγματικού Χρόνου) και SCTP (Stream Control Transport Protocol – Πρωτόκολλο Ελέγχου Μεταφοράς Ροών) τα οποία χρησιμοποιεί πάνω από κανάλια

UDP. Έτσι συνδυάζεται η ευρεία υποστήριξη που γνωρίζει το πρωτόκολλο UDP στο διαδίκτυο, με τις πιο εξειδικευμένες υπηρεσίες που παρέχουν τα SRTP και SCTP.

Για την ασφάλεια των δεδομένων που απαιτεί το WebRTC, γίνεται χρήση του πρωτοκόλλου DTLS (Datagram Transport Layer Security). Το DTLS σχεδιάστηκε για να μεταφέρει τις δυνατότητες του TLS, που χρησιμοποιείται στο HTTP (TCP), στο UDP. Για να λειτουργήσει το TLS πάνω από UDP, απαιτείται αξιόπιστη και με σειρά εκτέλεση της διαδικασίας χειραψίας του TLS. Το DTLS χρησιμοποιεί σειριακή κωδικοποίηση μηνυμάτων και χρονομετρητές για την υλοποίηση μίας απλής μορφής του TCP [53], προκειμένου να γίνει σωστά η εκτέλεση της χειραψίας, ενώ την δημιουργία των πιστοποιητικών αναλαμβάνει αυτόματα ο φυλλομετρητής.

Πάνω από το DTLS, θα χρησιμοποιηθεί, ανάλογα με τις ανάγκες της εφαρμογής το SRTP ή το SCTP. Αν πρόκειται για πολυμεσική εφαρμογή σε πραγματικό χρόνο, τότε το πρωτόκολλο SRTP επιλέγεται. Το SRTP [54] αποτελεί εξέλιξη του RTP και παρέχει επιπλέον: κρυπτογράφηση μηνυμάτων, πιστοποίηση ταυτότητας μηνυμάτων, ακεραιότητα και προστασία από επιθέσεις επανάληψης (replay attacks). Παρόμοια με το RTP, απαιτεί ξεχωριστό πρωτόκολλο ελέγχου. Το SRTCP (Secure Real-time Control Transport Protocol) επεκτείνει αντίστοιχα το RTCP. Ενώ το SRTP περιγράφει τα αρχεία πολυμέσων τα οποία μεταφέρει, το SRTCP διαχειρίζεται τα μεταδεδομένα της σύνδεσης, καταγράφοντας διάφορα στατιστικά για τη βέλτιστη μετάδοση του περιεχομένου.

Από την άλλη πλευρά, αν η εφαρμογή μας δεν σχετίζεται με πολυμεσικό υλικό, πιθανόν να χρειαζόμαστε το πρωτόκολλο SCTP. Το SCTP χρησιμοποιείται για τη δημιουργία καναλιών δεδομένων (data channels) πάνω από τα DTLS και UDP, και υποστηρίζει την μετάδοση οποιουδήποτε τύπου περιεχομένου. Σε αντίθεση με τα πρωτόκολλα UDP και TCP, η έκδοση SCTP που χρησιμοποιείται στο WebRTC είναι παραμετροποιήσιμη όσον αφορά τις υπηρεσίες που μπορεί να παρέχει.

Η παραμετροποίηση γίνεται σε επίπεδο καναλιού δεδομένων. Μία σύνδεση SCTP [55] είναι ένας μηνυματοκεντρικός τρόπος επικοινωνίας που υποστηρίζει, μέσω πολυπλεξίας (multiplexing), πολλά ταυτόχρονα κανάλια, τα οποία λειτουργούν ανεξάρτητα μεταξύ τους και μπορούν να παρουσιάζουν διαφορετικές ιδιότητες. Ανάλογα με τη φύση των πληροφοριών προς μετάδοση, μπορούμε να επιλέξουμε αν επιθυμούμε αξιόπιστη επικοινωνία που διορθώνει αυτόματα σφάλματα μετάδοσης, καθώς και αν θέλουμε τα μηνύματα να λαμβάνονται με την ίδια σειρά που αποστάλθηκαν.

Η μικρότερη αυτοτελής οντότητα δεδομένων στο SCTP ονομάζεται chunk. Σε ένα chunk προστίθενται 28 bytes για τις πληροφορίες των επικεφαλίδων, έναντι μόλις 8 στο πρωτόκολλο UDP. Αν και το SCTP δε θέτει περιοριστικούς κανόνες για το μέγεθος των μηνυμάτων, στους φυλλομετρητές ιστού υπάρχει ένα άγραφο όριο της τάξης των 64 KiB. Το όριο αυτό είναι αναγκαίο γιατί κάποιοι εξ αυτών δεν υποστηρίζουν την επικεφαλίδα EOR της επέκτασης ndata του SCTP. Η επικεφαλίδα EOR επιτρέπει την αποφυγή του λεγόμενου head-of-line blocking, κατά το οποίο η αποστολή ενός μεγάλου μηνύματος εμποδίζει την αποστολή άλλων [56]. Αν δύο φυλλομετρητές υποστηρίζουν την επικεφαλίδα, τότε το μέγιστο μέγεθος αυξάνεται δραματικά – πάλι όμως περιοριζόμενο από τη διαθέσιμη μνήμη του υποκείμενου υπολογιστικού συστήματος.

#### *3.1.1.3 Το WebRTC ως διεπαφή προγραμματισμού εφαρμογών*

Όπως αναφέραμε, υπεύθυνη για την προτυποποίηση των διεπαφών προγραμματισμού που προσφέρει το WebRTC μέσα στους φυλλομετρητές, είναι η W3C. Αυτή ορίζει ποιες δυνατότητες του πρωτοκόλλου WebRTC είναι διαθέσιμες για χρήση από τους προγραμματιστές ιστού. Στην πλειοψηφία τους οι παρεχόμενες διεπαφές είναι ασύγχρονες και οδηγούμενες από συμβάντα.

##### *RTCPeerConnection*

Η κυριότερη διεπαφή του WebRTC είναι ο κατασκευαστής `RTCPeerConnection`. Ένα στιγμιότυπο αυτού αναπαριστά μία ομότιμη σύνδεση μεταξύ δύο φυλλομετρητών. Με τα πεδία και τις μεθόδους που παρέχει είναι δυνατή η διαχείριση της σύνδεσης, από την εγκαθίδρυσή της μέχρι τον τερματισμό της. Παράλληλα, παρακολουθεί την κατάσταση της σύνδεσης και φροντίζει για τη διάσχιση NAT, την ανάλυση των μηνυμάτων SDP, την επαναδιαπραγμάτευση των συνδέσεων όταν απαιτείται κ.α.

Ως όρισμα στον κατασκευαστή `RTCPeerConnection` μπορούμε να δώσουμε ένα αντικείμενο `RTCConfiguration`, στο οποίο ορίζονται πεδία-πληροφορίες για τη σύνδεση. Κυριότερες εξ αυτών είναι το πεδίο `iceServers` που περιέχει μία λίστα από STUN και TURN εξυπηρετητές, που θα χρησιμοποιηθούν για τη διάσχιση NAT και το `iceCandidatePoolSize`, που επιτρέπει να αποσταλούν αιτήματα στους `iceServers` προτού να ζητηθεί η σύνδεση, βοηθώντας στην γρηγορότερη επίτευξή της. Ακολουθεί ένα παράδειγμα δημιουργίας ενός αντικειμένου `RTCPeerConnection`.

```
// Ρυθμίσεις σύνδεσης.
const config = {
  iceServers: {
    urls: [
      'stun:stun1.l.google.com:19302',
      'stun:stun2.l.google.com:19302'
    ]
  },
  iceCandidatePoolSize: 255
}
// Δημιουργία στιγμιότυπου.
const connection = new RTCPeerConnection(config);
```

Παρακάτω βλέπουμε πώς χρησιμοποιείται το αντικείμενο `RTCPeerConnection` προκειμένου να πραγματοποιηθεί σύνδεση. Παρατηρούμε τη χρήση ενός τρίτου καναλιού επικοινωνίας (`signalChannel`) που αναλαμβάνει να διαμεσολαβήσει προκειμένου να γίνει η ανταλλαγή των απαραίτητων μηνυμάτων SDP (διαδικασία σηματοδότησης). Συνήθως για αυτό το σκοπό χρησιμοποιούνται συνδέσεις WebSockets, αν και οποιοσδήποτε τρόπος αμφίδρομης επικοινωνίας θα ήταν κατάλληλος. Στο παράδειγμα, για λόγους απλότητας, χρησιμοποιούμε μία φανταστική διεπαφή επικοινωνίας.

```
(async () => {
  // Δημιουργία SDP offer.
  const offer = await connection.createOffer();
  connection.setLocalDescription(offer);
  // Αποστολή SDP offer μέσω καναλιού σηματοδότησης.
  signalChannel.send({
    offer: offer.sdp
  });
  // Trickle ICE, τα ICE Candidates στέλνονται σε
  // δεύτερο χρόνο, όταν ανακαλύπτονται.
  connection.addEventListener('icecandidate', e => {
    if (e.candidate) {
      signalChannel.send(e.candidate);
    }
  });
  // Εγκατάσταση χειριστή για μηνύματα SDP answer
  // και για μηνύματα ICE Candidates από τον
  // απομακρυσμένο φυλλομετρητή.
  signalChannel.addEventListener('message', msg => {
    if (msg.answer) {
      connection.setRemoteDescription(msg.answer);
    }
    if (msg.iceCandidate) {
      connection.addIceCandidate(msg.iceCandidate);
    }
  });
})();
```

Η κλήση της μεθόδου `createOffer()` δημιουργεί ένα μήνυμα SDP τύπου offer, όπως αυτό περιγράφεται στο [49], προκειμένου να αιτηθεί τη σύνδεση με τον απομακρυσμένο φυλλομετρητή. Αμέσως αρχίζει η συλλογή ICE Candidate. Στο

παραπάνω παράδειγμα, χρησιμοποιούμε τη μέθοδο την παραλλαγή του ICE, Trickle ICE. Έτσι δεν περιμένουμε μέχρι να ολοκληρωθεί η διαδικασία συλλογής, αλλά στέλνουμε κατευθείαν το αίτημα σύνδεσης, ώστε να αρχίσει η αντίστοιχη διαδικασία και από την πλευρά του απομακρυσμένου πελάτη. Οι ICE Candidate αποστέλλονται με ξεχωριστά μηνύματα σε δεύτερο χρόνο. Αν και η τεχνική αυτή δημιουργεί περισσότερα μηνύματα, είναι συνήθως προτιμότερη. Τέλος, εγκαθιστούμε ένα χειριστή για τα μηνύματα που θα λάβουμε από τον άλλο φυλλομετρητή. Αυτά είναι δύο ειδών, ένα μήνυμα με SDP τύπου answer που δείχνει την αποδοχή της σύνδεσης και ένα ή περισσότερα μηνύματα με ICE Candidate που θα πρέπει να δοκιμαστούν συνδυαστικά με τους ICE Candidate που εντόπισε ο τοπικός φυλλομετρητής, προκειμένου να επιτευχθεί η σύνδεση ή να βρεθεί το καλύτερο δυνατό μονοπάτι (path) στο δίκτυο. Ο έλεγχος των ICE Candidate γίνεται με διαφανή, για τον προγραμματιστή, τρόπο. Αυτός αρκεί να καλέσει σωστά το `setLocalDescription()` και το `setRemoteDescription()`, καθώς και να ενημερώνει τη σύνδεση με νέα ICE Candidates (`addIceCandidate()`) που λαμβάνει μέσα από το κανάλι σηματοδότησης.

Από την πλευρά του απομακρυσμένου φυλλομετρητή, η διαδικασία είναι παρόμοια. Με το που λάβει το αίτημα SDP, δημιουργεί ένα αντικείμενο `RTCPeerConnection` όπως είδαμε παραπάνω και καλεί τη μέθοδο `setRemoteDescription()`. Στη συνέχεια, σχηματίζει ένα μήνυμα SDP τύπου answer, καλόντας τη μέθοδο `createAnswer()` και την μεταβιβάζει πίσω στον άλλο φυλλομετρητή. Θέτει κι αυτός από την πλευρά του, χειριστές γεγονότων προκειμένου να ενημερώνει το αντικείμενο `RTCPeerConnection` με τους ICE Candidate που λαμβάνει από το κανάλι σηματοδότησης και φροντίζει να στέλνει τους δικούς του.

Σε αυτό το σημείο, είναι απαραίτητο να γίνει μια αναφορά στο κύκλο ζωής μίας `RTCPeerConnection`. Ο κατασκευαστής `RTCPeerConnection` είναι διαθέσιμος στο αντικείμενο window. Το γεγονός αυτό μαρτυρά ένα σημαντικό περιορισμό του WebRTC: το ότι η διάρκεια ζωής της σύνδεσης είναι άρρηκτα συνδεδεμένη με την τρέχουσα ιστοσελίδα. Σε περίπτωση που ο χρήστης κλείσει την σελίδα ή μεταβεί σε άλλη, αναγκαία επέρχεται η αποσύνδεση.

Ήδη από το 2015, υπάρχει πρόταση για πρόσβαση στη διεπαφή του WebRTC από Workers [57]. Οι ServiceWorkers, οι οποίοι διαθέτουν άλλο κύκλο ζωής – ανεξάρτητο από αυτό της ιστοσελίδας που υποστηρίζουν – θα μπορούσαν να δώσουν νέες δυνατότητες για καλύτερη αξιοποίηση του WebRTC σε καινοτόμες υπηρεσίες όπως

είναι τα ομότιμα δίκτυα διανομής περιεχομένου. Δυστυχώς όμως μέχρι και σήμερα, δεν έχουν γίνει σημαντικά βήματα προόδου για την αποδέσμευση του WebRTC από το παράθυρο μιας ιστοσελίδας.

***DataChannel***

Αφού δημιουργήσουμε ένα στιγμιότυπο `PeerConnection`, μπορούμε να δημιουργήσουμε ένα κανάλι δεδομένων (data channel) με την κλήση της μεθόδου `createDataChannel()`. Ένα κανάλι δεδομένων αντιστοιχεί σε ένα ζευγάρι μίας εισερχόμενης και μίας εξερχόμενης ροής στην υποκείμενη σύνδεση SCTP [58].

Η επικοινωνία για την σύσταση ενός καναλιού δεδομένων γίνεται συνήθως αυτόματα μέσα από την ίδια την ομότιμη σύνδεση (in-band) και δεν απαιτεί τη χρήση τρίτου καναλιού σηματοδότησης. Στη μέθοδο `createDataChannel()` δίνουμε ως ορίσματα ένα αναγνωριστικό όνομα καθώς και ένα αντικείμενο τύπου `RTCDataChannelInit`. Με αυτό θέτουμε ρυθμίσεις όπως: αν επιθυμούμε out-of-band διαπραγμάτευση της σύνδεσης, τη σειρά με την οποία λαμβάνονται τα μηνύματα (σειριακά ή τυχαία), την αξιόπιστη ή μη, αποστολή των μηνυμάτων κ.α. Εφόσον η διαπραγμάτευση του καναλιού γίνεται in-band, η απομακρυσμένη εφαρμογή οφείλει να έχει θέσει χειριστή για συμβάντα τύπου `RTCDataChannelEvent`, προκειμένου να αποκτήσει πρόσβαση στο κανάλι.

Εφόσον συσταθεί ένα κανάλι επικοινωνίας, μπορεί να χρησιμοποιηθεί για την αποστολή οποιουδήποτε τύπου δεδομένων. Εγγενώς, η μέθοδος `send()` του `RTCDataChannel` υποστηρίζει δεδομένα κειμένου και δυαδικά δεδομένα όπως `Blobs` και `ArrayBuffers`. Για να κλείσει το κανάλι, παρέχεται η αντίστοιχη μέθοδος `close()`. Τέλος, η λήψη μηνυμάτων καθώς και η ενημέρωση για σφάλματα και συμβάντα του κύκλου ζωής του καναλιού, γίνονται μέσα από εγκατάσταση ανάλογων χειριστών γεγονότων (event handlers).

Σε γενικές γραμμές, η διεπαφή του `RTCDataChannel` ομοιάζει με αυτή του WebSocket.

### *3.1.2 WebSockets*

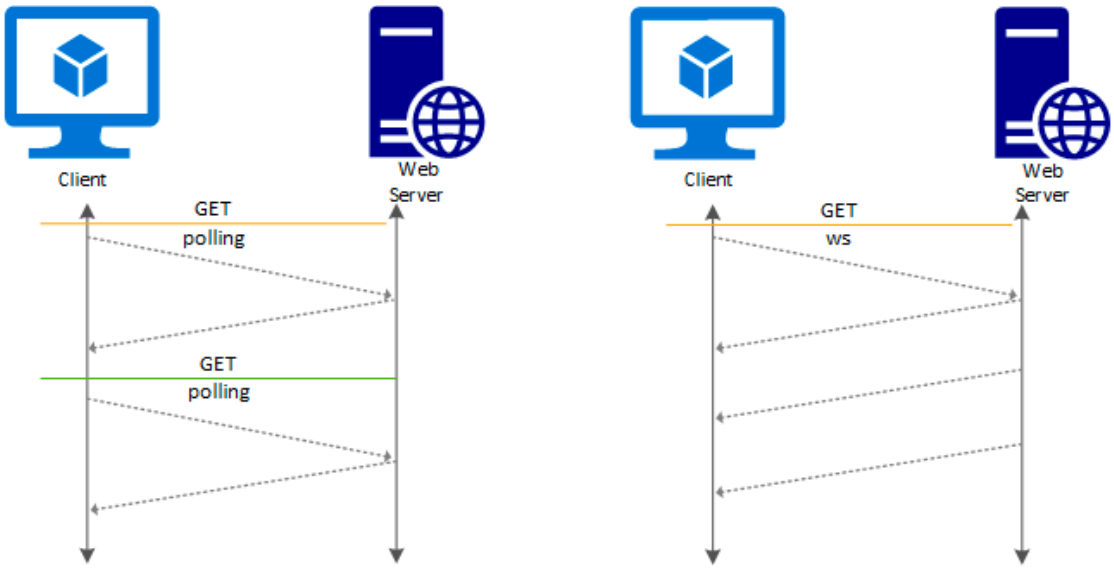
#### *3.1.2.1 Το πρόβλημα*

Από την απαρχή του Παγκόσμιου Ιστού και μέχρι τις αρχές του 21$^{ου}$ αιώνα, είχε καθιερωθεί ότι η περιήγηση στην πληθώρα των εγγράφων που προσφέρει θα γίνεται με

τη χρήση HTTP αιτημάτων τα οποία κατασκευάζονται ως αντίδραση στις ενέργειες του χρήστη. Για παράδειγμα όταν ένας χρήστης συμπληρώνει μία διεύθυνση στη γραμμή διευθύνσεων ή πατάει σε έναν υπερσύνδεσμο, ο περιηγητής αναλαμβάνει να ετοιμάσει ένα αίτημα GET, να το αποστείλει στον υπεύθυνο εξυπηρετητή και να περιμένει να λάβει τη σχετική απόκριση. Παρατηρούμε ότι η φορά της επικοινωνίας ξεκινά πάντα από τον πελάτη – είναι επομένως μονόδρομη.

Η πρακτική αυτή δεν αμφισβητήθηκε μέχρι το 2005 όπου η εισαγωγή της AJAX (Asynchronous JavaScript and XML), μίας τεχνολογίας που επέτρεπε την πραγματοποίηση HTTP αιτημάτων προγραμματιστικά μέσω JavaScript, άλλαξε τον τρόπο με τον οποίο σκεφτόμασταν για τον ιστό και τα όριά του. Πλέον κάθε σελίδα δύναται να φορτώσει δυναμικά επιμέρους τμήματά της και να τα χρησιμοποιήσει με τη βοήθεια της JavaScript. Παρ' όλο που η AJAX υποστηρίζει αποκλειστικά μονόδρομη επικοινωνία, αποτελεί βασική αφορμή για να γίνει αντιληπτή η ανάγκη για αμφίδρομη επικοινωνία μεταξύ πελάτη-εξυπηρετητή.

Το ίδιο κιόλας έτος, υπηρεσίες πραγματικού χρόνου (real time) όπως το GTalk και το Meebo αποδεικνύουν ότι είναι ήδη δυνατή η ανάπτυξη εφαρμογών στις οποίες ένας διακομιστής μπορεί να στείλει πληροφορίες στον φυλλομετρητή, όποτε αυτός το κρίνει σκόπιμο, χάρη σε μία ομάδα τεχνικών που μόλις κάνουν την εμφάνισή τους.




**Εικόνα 3-4: Η τεχνική polling σε σύγκριση με μία σύνδεση WebSocket. (πηγή: https://docs.microsoft.com/en-us/azure/application-gateway/application-gateway-websocket)**

Ο Alex Russell στο [59] εισάγει τον όρο «Comet» για να περιγράψει τη μεταφορά δεδομένων με χαμηλό χρόνο καθυστέρησης που υποκινούνται από την πλευρά του εξυπηρετητή, ασχέτως ποια τεχνική χρησιμοποιείται. Τέτοιου είδους τεχνικές παρουσιάζονται στο RFC 6202 [60] του IETF (Internet Engineering Task Force - Επιχειρησιακή Ομάδα Τεχνολογίας του Διαδικτύου) όπου αναλύονται, παράλληλα, γνωστά προβλήματα και καλές πρακτικές. Επιγραμματικά, τα θέματα που αναφέρονται αφορούν την ακαταλληλότητα του πρωτοκόλλου HTTP, τον μέγιστο λανθάνων χρόνο (maximal latency) μεταξύ των μηνυμάτων, τη συχνή δημιουργία συνδέσεων TCP/IP, την κατανάλωση πόρων του συστήματος υποστήριξης, την ομαλή υποβάθμιση της απόδοσης σε περίπτωση αυξημένου φόρτου, το χρόνο λήξης (timeout) ενός αιτήματος και τον τρόπου που επηρεάζουν ενδιάμεσους διακομιστές όπως διαμεσολαβητές με λειτουργίες κρυφής μνήμης (cache proxies).

#### *3.1.2.2 Η ανάγκη για WebSockets*

Οι τεχνικές αμφίδρομης επικοινωνίας που προτάθηκαν, αν και λειτουργικές, δεν ήταν η βέλτιστη λύση στο πρόβλημα. Η πληθώρα των τεχνικών και η έλλειψη προτυποποίησής τους αποτελούσε τροχοπέδη για την ευρεία υιοθέτησή τους από την κοινότητα των προγραμματιστών.

Δεν υπήρχε αμφιβολία ότι ένα επίσημο ανοιχτό πρότυπο που θα υποστηρίζεται από όλους τους δημοφιλείς φυλλομετρητές, θα διευκόλυνε την ανάπτυξη εφαρμογών που απαιτούν αμφίδρομη ανταλλαγή μηνυμάτων. Μάλιστα, ενώ οι υπάρχουσες τεχνικές αποτελούσαν «ευφάνταστες χρήσεις» του πρωτοκόλλου TCP/IP, τώρα υπήρχε η δυνατότητα σχεδίασης ενός νέου εξειδικευμένου πρωτοκόλλου με βελτιωμένη απόδοση, που θα ταίριαζε κατάλληλα για τη χρήση στην οποία προοριζόταν.

Κάπως έτσι αποφασίστηκε ο σχεδιασμός του WebSocket πρωτοκόλλου. Την πρωτοβουλία για την προτυποποίηση του ανέβαλε η ομάδα εργασίας WHATWG με επικεφαλής τον Michael Carter. Η πρώτη ονομασία που δόθηκε στο νέο πρωτόκολλο ήταν TCPConnection, αφού η αρχική φιλοδοξία του ήταν να δώσει την ικανότητα σε φυλλομετρητές να ανοίγουν πλήρως ικανές συνδέσεις TCP. Κάτι τέτοιο αποδείχτηκε μη εφικτό, αφού θα έδινε τη δυνατότητα σε κακόβουλους ιστοτόπους για καταχρηστική χρήση της τεχνολογίας αυτής. Οι βασικές απαιτήσεις που όρισε ο Carter για πρωτόκολλο ήταν οι ακόλουθες [61]:

- Η δυνατότητα αμφίδρομης επικοινωνίας μίας διεργασίας με ένα σενάριο που εκτελείται σε μία ιστοσελίδα.

- Η ευκολία υλοποίησης συμβατής εφαρμογής στην πλευρά του διακομιστή.
- Η αποτροπή χρήσης της για κακόβουλες ενέργειες.
- Η ικανότητα της να λειτουργεί παρά την ενδεχόμενη ύπαρξη τειχών προστασίας (firewalls).

Η πρώτη υιοθέτηση του πρωτοκόλλου σε φυλλομετρητή έγινε με την τέταρτη έκδοση του Google Chrome τον Ιανουάριο του 2010 [44].

#### *3.1.2.3 Τα WebSockets σήμερα*

Στις μέρες μας, για την προτυποποίηση της τεχνολογίας των WebSockets εμπλέκονται δύο οργανισμοί:

- Η IETF, η οποία έχει αναλάβει την προτυποποίηση των WebSockets ως πρωτόκολλο επικοινωνίας που λειτουργεί στο επίπεδο εφαρμογής (Application Layer) του μοντέλου αναφοράς OSI. Το πρωτόκολλο θεωρείται σταθερό ήδη από το Δεκέμβριο του 2011 και περιγράφεται στο RFC 6455 [62]. Οποιαδήποτε βελτίωση ή αλλαγή γίνεται μόνο με τη χρήση επεκτάσεων πάνω στο ήδη ορισμένο πρωτόκολλο.
- Η WHATWG, η οποία αναπτύσσει τη διεπαφή προγραμματισμού που διατίθεται από τους φυλλομετρητές στους προγραμματιστές ιστού ως τμήμα του εγγράφου HTML Living Standard (ενεργό πρότυπο HTML) [63].

Η κατάσταση υποστήριξης των WebSockets θεωρείται άκρως ικανοποιητική, με το 96% των χρηστών παγκοσμίως να χρησιμοποιούν ένα φυλλομετρητή που τα υποστηρίζει [44]. Στις εφαρμογές που κάνουν χρήση του πρωτοκόλλου περιλαμβάνονται ηλεκτρονικά παιχνίδια πολλών παικτών, ροές περιεχομένου όπως ειδήσεις, ενημερώσεις κοινωνικών δικτύων, συνεργατικές εφαρμογές κ.α.

#### *3.1.2.4 Τα WebSockets ως πρωτόκολλο*

Το πρωτόκολλο WebSocket είναι μία ιδιαιτέρως ευέλικτη μέθοδος για τη μεταφορά δεδομένων μέσω μηνυμάτων ανάμεσα σε ένα πελάτη και ένα διακομιστή. Σκοπός του είναι η υποστήριξη ταυτόχρονης και αμφίδρομης επικοινωνίας (full duplex and bi-directional). Προσπαθεί κι επιτυγχάνει να μοιάζει με ένα απλό TCP socket, στο βαθμό που αυτό είναι δυνατό. Στην πραγματικότητα βρίσκεται ένα επίπεδο υψηλότερα από το πρωτόκολλο μεταφοράς TCP, πάνω από το οποίο λειτουργεί και προσθέτει υπηρεσίες όπως [53]:

- Διαπραγμάτευση συνδέσεων με επιβολή πολιτικής κοινής προέλευσης (same-origin policy).
- Διαλειτουργικότητα με την υπάρχουσα υποδομή του διαδικτύου.
- Επικοινωνία με ανταλλαγή μηνυμάτων και αποδοτική πλαισίωση (framing) τους.
- Διαπραγμάτευση πρωτοκόλλων και επεκτασιμότητα.

Βασική αρχή του πρωτοκόλλου είναι η όσο το δυνατόν μικρότερη επιβάρυνση των μηνυμάτων με πληροφορίες πλαισίωσης για την υποστήριξη των υπηρεσιών του.

Στη διάρκεια μίας σύνδεσης WebSocket μπορούμε να ξεχωρίσουμε δύο βασικά στάδια: το στάδιο της εγκαθίδρυσης της σύνδεσης και το στάδιο της κανονικής λειτουργίας.

***Το στάδιο εγκαθίδρυσης***

Προτού επιτευχθεί η σύνδεση πελάτη-εξυπηρετητή μέσω WebSocket, απαιτείται η ανταλλαγή ορισμένων απαραίτητων πληροφοριών που περιγράφουν την επικείμενη σύνδεση. Για τη διαδικασία αυτή χρησιμοποιείται ο όρος «opening handshake» (χειραψία σύνδεσης).

Σε αυτό το σημείο πρέπει να αναφέρουμε για το πρωτόκολλο WebSocket ότι αν και σχεδιάστηκε με αφορμή την εφαρμογή του σε φυλλομετρητές, τίποτα δεν εμποδίζει τη χρήση του και έξω από αυτούς. Η χειραψία σύνδεσης έτσι όπως περιγράφεται στην παρούσα ενότητα, αφορά τη χρήση του πρωτοκόλλου σε περιηγητές ιστού. Σε άλλες εκφάνσεις χρήσης του είναι πολύ πιθανό να διαφέρει.

Για την επίτευξη της σύνδεσης, χρησιμοποιείται το πρωτόκολλο HTTP. Συνέπεια αυτού είναι ότι η διαδικασία σύνδεσης πρέπει πάντα να ξεκινά από τον πελάτη. Η χρήση του HTTP συνδράμει στη διαλειτουργικότητα των WebSockets με τις ήδη εγκατεστημένες εφαρμογές εξυπηρετητών ιστού.

Αρχικά, ο φυλλομετρητής ετοιμάζει ένα αίτημα HTTP με μέθοδο GET. Η τοποθεσία που ζητείται μέσω του URL καλείται «webSocket endpoint» και δεν έχει ιδιαίτερο ρόλο, αρκεί ο εξυπηρετητής να είναι ρυθμισμένος έτσι ώστε να δέχεται εκεί τα αιτήματα εγκαθίδρυσης. Οι απαραίτητες πληροφορίες σύνδεσης εμπεριέχονται σε επικεφαλίδες HTTP, όπως ορίζονται στο επίσημο έγγραφο τυποποίησης που δημοσιεύει το IETF.

Ο Πίνακας 3-1 παρουσιάζει τις απαραίτητες και μη επικεφαλίδες που συναντάμε σε ένα αίτημα εγκαθίδρυσης. Άλλες επικεφαλίδες μπορούν επίσης να χρησιμοποιηθούν.

**Πίνακας 3-1: Επικεφαλίδες αιτήματος εγκαθίδρυσης σύνδεσης WebSocket**

| Επικεφαλίδα αιτήματος | Περιγραφή |
|---|---|
| `Host` | Δηλώνει τον host (οικοδεσπότη) στον οποίο απευθύνεται το αίτημα. Είναι σημαντικό αφού επιτρέπει τη φιλοξενία πολλών εξυπηρετητών με τη χρήση μίας μόνο εξωτερικής διεύθυνσης IP (πρακτική virtual hosting). |
| `Upgrade` | Η τιμή της επικεφαλίδας πρέπει να περιέχει τον όρο-κλειδί «websocket». Υποδηλώνει την επιθυμία του πελάτη να ανοίξει σύνδεση WebSocket. |
| `Connection` | Η τιμή της επικεφαλίδας πρέπει να περιέχει τον όρο-κλειδί «Upgrade». |
| `Sec-WebSocket-Key` | Αποτελεί μία τυχαία τιμή μεγέθους 16 byte που παράγει ο φυλλομετρητής και κωδικοποιεί με τη μέθοδο base64. Χρησιμεύει προκειμένου να εξακριβώσουμε ότι ο εξυπηρετητής υποστηρίζει πλήρως το πρωτόκολλο και βοηθάει στην αποτροπή επιθέσεων. |
| `Origin` | Περιέχει αναγνωριστικό της σελίδας (origin) που πραγματοποιεί το αίτημα. Με αυτό το τρόπο μπορεί να επιβληθεί η πολιτική ασφαλείας για τη συνεργασία διαφορετικών οντοτήτων του ιστού, CORS (cross-origin resource sharing). |

| | |
|---|---|
| **Sec-WebSocket-Version** | Πρέπει να περιέχει την τιμή 13. Η έκδοση αυτή είναι η τελευταία και σταθερή έκδοση του πρωτοκόλλου WebSocket. |
| **Sec-WebSocket-Protocol** | Περιλαμβάνει λίστα με τα υπό-πρωτόκολλα μηνυμάτων που υποστηρίζει ο πελάτης. Τα πρωτόκολλα αυτά δεν σχετίζονται με τα WebSockets αλλά με την εφαρμογή που τα χρησιμοποιεί. |
| **Sec-WebSocket-Extensions** | Περιλαμβάνει μία λίστα με τις επεκτάσεις του πρωτοκόλλου των WebSockets που υποστηρίζει ο φυλλομετρητής. |
| *** Οι υπογραμμίσεις υποδηλώνουν την υποχρεωτική ύπαρξη των συγκεκριμένων επικεφαλίδων.** | |

Όταν ο εξυπηρετητής λάβει το αίτημα εγκαθίδρυσης WebSocket σύνδεσης πρέπει να πραγματοποιήσει μία χειραψία TLS στην περίπτωση που το αίτημα στάλθηκε με HTTPS και να ζητήσει επιπλέον στοιχεία για την αυθεντικοποίηση του χρήστη αν απαιτείται. Επίσης αν κρίνεται σκόπιμο, μπορεί να ανακατευθύνει το χρήστη σε άλλη διεύθυνση, όπου η διαδικασία χειραψίας θα επαναληφθεί από την αρχή.

Μόλις ολοκληρωθούν οι παραπάνω διαδικασίες και εφόσον γίνει αποδεκτό το αίτημα σύνδεσης, ο εξυπηρετητής απαντά με την ανάλογη HTTP απόκριση. Η απόκριση έχει κωδικό κατάστασης (status code) με τιμή 101 – Switching Protocols – και περιέχει τις παρακάτω επικεφαλίδες (Πίνακας 3-2):

**Πίνακας 3-2: Επικεφαλίδες απόκρισης εγκαθίδρυσης σύνδεσης WebSocket**

| **Επικεφαλίδα απόκρισης** | **Περιγραφή** |
|---|---|

| | |
|---|---|
| **Upgrade** | Η τιμή πρέπει να είναι ίδια με την αντίστοιχη επικεφαλίδα αιτήματος, δηλαδή «websocket». |
| **Connection** | Η τιμή της πρέπει να είναι ίδια με την αντίστοιχη επικεφαλίδα αιτήματος, δηλαδή «Upgrade». |
| **Sec-WebSocket-Accept** | Υπολογίζεται από την τιμή της επικεφαλίδας αιτήματος Sec-WebSocket-Key ύστερα από την επεξεργασία της με μία αυστηρώς ορισμένη διαδικασία που περιγράφεται στο RFC 6455. Χρησιμεύει προκειμένου να εξακριβώσουμε ότι ο εξυπηρετητής υποστηρίζει πλήρως το πρωτόκολλο και βοηθάει στην αποτροπή επιθέσεων. |
| **Sec-WebSocket-Protocol** | Περιέχει ένα ή περισσότερα από τα υπό-πρωτόκολλα που πρότεινε ο φυλλομετρητής του πελάτη και υποστηρίζει ο εξυπηρετητής. Σε περίπτωση που δε βρεθεί κοινά υποστηριζόμενο πρωτόκολλο, η τιμή της πρέπει να είναι null. |
| **Sec-WebSocket-Extensions** | Περιέχει μία ή περισσότερες από τις επεκτάσεις του πρωτοκόλλου WebSocket που πρότεινε ο φυλλομετρητής του πελάτη και υποστηρίζει ο εξυπηρετητής. Σε περίπτωση που δε βρεθεί κοινά υποστηριζόμενη επέκταση, η επικεφαλίδα δε συμπεριλαμβάνεται στην απόκριση. |

*** Οι υπογραμμίσεις υποδηλώνουν την υποχρεωτική ύπαρξη των συγκεκριμένων επικεφαλίδων.**

### *Μηνύματα και πλαισίωση*

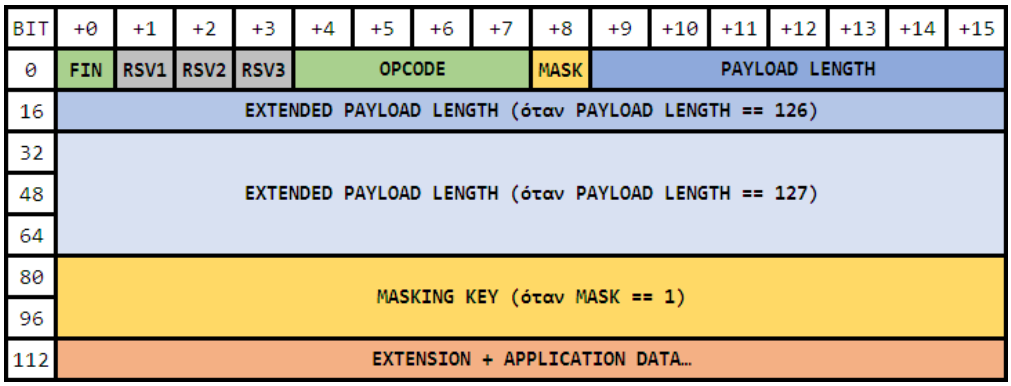
Αφού ολοκληρωθεί επιτυχώς το opening handshake, πλέον μπαίνουμε στο στάδιο της κανονικής λειτουργίας που επιτρέπει την απρόσκοπτη επικοινωνία δύο κατευθύνσεων. Λαμβάνοντας υπόψη τη δαιδαλώδη ιεραρχία του σημερινού διαδικτύου και την ασφυκτική προσέγγιση που επιλέγουν οι διαχειριστές δικτύων, κλείνοντας κάθε περιττή θύρα επικοινωνίας στα συστήματα που ελέγχουν, η απόφαση να χρησιμοποιηθούν οι γνωστές θύρες TCP 80 και 443 – οι θύρες του παγκόσμιου ιστού – ήταν κρίσιμης σημασίας για την επιτυχία του πρωτοκόλλου.

Το πρωτόκολλο μεταφοράς TCP μέσω του οποίου λειτουργούν τα WebSockets είναι ένα πρωτόκολλο βασισμένο σε ροές (stream-oriented) που χρησιμοποιεί πακέτα για την αποστολή δεδομένων. Αυτό σημαίνει ότι καθώς τα πακέτα φτάνουν στον προορισμό τους, θα πρέπει να υπάρχει η απαραίτητη λειτουργία σε επίπεδο εφαρμογής που θα αναλάβει την προσωρινή αποθήκευση (buffering) και συρραφή των πακέτων προκειμένου να αναδομηθεί το αντικείμενο της αποστολής.

Όπως έχει ήδη αναφερθεί, το πρωτόκολλο WebSocket χτίζει πάνω από το TCP προκειμένου να παρέχει επιπρόσθετες υπηρεσίες. Μία εξ αυτών είναι εισαγωγή του μοντέλου επικοινωνίας μέσω μηνυμάτων αντί για πακέτα. Η διαφορά έγκειται στο γεγονός ότι ένα μήνυμα αποτελεί ένα αυτοτελές αντικείμενο που εμπεριέχει τη συνολική πληροφορία προς αποστολή, ενώ ένα πακέτο είναι ένα σταθερού μεγέθους τμήμα της συνολικής πληροφορίας, το οποίο απαιτεί επιπλέον επεξεργασία. Βασικό προτέρημα του μοντέλου μηνυμάτων είναι η ευκολότερη ανάπτυξη εφαρμογών.

Τα μηνύματα μπορούν να φέρουν κάθε είδους ωφέλιμο φορτίο (payload) αφού είναι ικανά να μεταφέρουν δυαδικά δεδομένα (binary data). Στην πράξη, πέρα από δυαδικά δεδομένα, το πρωτόκολλο υποστηρίζει και δεδομένα κειμένου (text data) κωδικοποιημένα σύμφωνα με το σχήμα κωδικοποίησης χαρακτήρων UTF-8.

Κάθε μήνυμα αποτελείται από ένα ή περισσότερα επί μέρους μικρότερα τμήματα που ονομάζονται πλαίσια (frames). Ένα πλαίσιο απαρτίζεται από την επικεφαλίδα που περιλαμβάνει μεταδεδομένα του μηνύματος – απαραίτητα για τη λειτουργία του πρωτοκόλλου – ακολουθούμενη από το ωφέλιμο φορτίο του πλαισίου που περιέχει ένα μέρος ή το σύνολο του αντικειμένου της μεταφοράς.




**Εικόνα 3-5: Η δομή ενός πλαισίου WebSocket.**

Είναι σημαντικό το μέγεθος των επικεφαλίδων να είναι όσο το δυνατό μικρότερο. Όπως μπορούμε να δούμε από την Εικόνα 3-5, το μέγεθος της επικεφαλίδας ενός πλαισίου είναι δυναμικό και κυμαίνεται από 2 bytes στην καλύτερη περίπτωση, έως 14 bytes στην χειρότερη.

Η θέση bit 0 του πλαισίου (FIN) δείχνει στον αποδέκτη του πλαισίου αν το παρόν πλαίσιο είναι το τελευταίο για το τρέχον μήνυμα.

Τα επόμενα τρία bit (RSV1, RSV2, RSV3) είναι κατοχυρωμένα για χρήση από επεκτάσεις του πρωτοκόλλου και υπό κανονικές συνθήκες έχουν τιμή 0. Σχετικά με τις επεκτάσεις, να σημειώσουμε ότι οι δημοφιλείς φυλλομετρητές Firefox και Chromium υποστηρίζουν μόνο την επέκταση permessage_deflate[1], η οποία επιτρέπει την συμπίεση των δεδομένων προς αποστολή.

Στις θέσεις 4 έως 7 δηλώνεται το OPCODE (operation code – κώδικας εργασίας) και παίρνει τιμή σύμφωνα με μία απαρίθμηση (enum) που ορίζεται στο RFC 6455. Δηλώνει το είδος του frame που μπορεί να είναι:

1. Continuation Frame: το πλαίσιο αυτό αποτελεί συνέχεια προηγούμενου πλαισίου.
2. Text Frame: είναι το πρώτο πλαίσιο ενός μηνύματος και μεταφέρει κείμενο.
3. Binary Frame: είναι το πρώτο πλαίσιο ενός μηνύματος και μεταφέρει δυαδικά δεδομένα.
4. Connection Close Frame: το πλαίσιο ζητά το κλείσιμο της σύνδεσης. Στο ωφέλιμο φορτίο του πλαισίου μπορεί να καταχωρηθεί ο λόγος. Το κλείσιμο της σύνδεσης μπορούν να αιτηθούν και οι δύο πλευρές.
5. Ping Frame: το πλαίσιο ελέγχει αν η άλλη πλευρά είναι αποκρίσιμη.
6. Pong Frame: το πλαίσιο ενημερώνει την άλλη πλευρά ότι είναι αποκρίσιμη.

[1] Η επέκταση ορίζεται στο RFC 7692.

Τα 1, 2 και 3 ανήκουν στην ομάδα των πλαισίων δεδομένων (data frames) ενώ τα 4, 5, και 6 στην ομάδα των πλαισίων ελέγχου (control frames).

Στη συνέχεια, στη όγδοη θέση της επικεφαλίδας ακολουθεί το MASK bit. Αυτό υποδηλώνει την ύπαρξη ενός κλειδιού-μάσκας (MASKING KEY) στη συνέχεια της επικεφαλίδας και πρέπει να έχει πάντα την τιμή 1, δηλαδή αληθές, όταν το πλαίσιο αποστέλλεται με κατεύθυνση από τον φυλλομετρητή στον εξυπηρετητή.

Οι επόμενες επτά 7 θέσεις ορίζουν το μέγεθος, σε bytes, του ωφέλιμου φορτίου (PAYLOAD LENGTH) που περιέχει το πλαίσιο. Όταν η τιμή είναι μικρότερη ή ίση του 125, τότε αυτό είναι το μέγεθός του. Αν η τιμή του είναι 126, τότε μας παραπέμπει στα επόμενα 2 bytes της επικεφαλίδας, προκειμένου να διαπιστώσουμε το μέγεθος του ωφέλιμου φορτίου. Αντίστοιχα, αν η τιμή που περιέχει είναι ίση με 127, τότε μας παραπέμπει στα επόμενα 8 bytes.

Στο τέλος, μετά την περιγραφή του μεγέθους του ωφέλιμου φορτίου ακολουθεί το κλειδί-μάσκα (MASKING BIT), αν αυτό είναι απαραίτητο. Αποτελεί μία τιμή 32 bit, η οποία επιλέγεται τυχαία – πρέπει να είναι αδύνατη η πρόβλεψή της. Το κλειδί μεταλλάσσει τα δεδομένα του ωφέλιμου φορτίου, ώστε να μην είναι δυνατόν κάποιος να επηρεάσει τη δυαδική μορφή του μηνύματος με τρόπο τέτοιο, που να μπορεί να χρησιμοποιηθεί σαν τρόπος επίθεσης (attack vector) σε ενδιάμεσες μνήμες αποθήκευσης (proxy caches).

Με το κλειδί-μάσκα κλείνει η επικεφαλίδα του πλαισίου και ακολουθεί το ωφέλιμο φορτίο, το οποίο απαρτίζεται από δεδομένα επεκτάσεων (extension data) στην αρχή και δεδομένα εφαρμογής (application data) στη συνέχεια.

### *Υποστήριξη WebSockets σε ενδιάμεσους πληρεξούσιους διαμεσολαβητές*

Στην ενότητα αυτή εξετάζουμε τη διαλειτουργικότητα των WebSockets με τους ενδιάμεσους πληρεξούσιος εξυπηρετητές (διαμεσολαβητές - proxies).

Αν η εφαρμογή του πελάτη έχει ρυθμιστεί έτσι ώστε να χρησιμοποιεί διαμεσολαβητές με το πρωτόκολλο WebSocket, τότε προτού ξεκινήσει η χειραψία σύνδεσης θα πρέπει να στείλει ένα αίτημα HTTP με τη μέθοδο `CONNECT`. Όταν ο διαμεσολαβητής λάβει το αίτημα και εφόσον το υποστηρίζει, ανοίγει σύνδεση TCP με τον τελικό εξυπηρετητή ιστού. Εφόσον η σύνδεση επιτευχθεί, η εφαρμογή του πελάτη συνεχίζει με τη διαδικασία της χειραψίας, όπως αυτή περιγράφηκε παραπάνω. Το αίτημα

της εγκαθίδρυσης αποστέλλεται στον ενδιάμεσο εξυπηρετητή, ο οποίος μεταβιβάζει τη ροή της υποκείμενης TCP σύνδεσης στον τελικό, δρώντας σαν δίοδος (tunnel).

Αν η εφαρμογή του πελάτη δεν έχει ρυθμιστεί κατάλληλα και επομένως δε γνωρίζει την ύπαρξη του διαμεσολαβητή, τότε δε θα στείλει το σχετικό αίτημα `CONNECT` αλλά θα αρχίσει κατευθείαν τη διαδικασία χειραψίας. Στην περίπτωση αυτή, το αποτέλεσμα εξαρτάται από τον διαμεσολαβητή.

Τέλος, αν χρησιμοποιούνται κρυπτογραφημένες συνδέσεις μεταφοράς δεδομένων με τον εξυπηρετητή, τότε πιθανότατα έχει ήδη πραγματοποιηθεί αίτημα `CONNECT` για τη μετάδοση των HTTPS αιτημάτων και αποκρίσεων και η σύνδεση WebSockets θα επιτευχθεί χωρίς πρόβλημα.

***HTTP/2 και WebSockets***

Με τη δημιουργία της δεύτερης έκδοσης του πρωτοκόλλου μεταφοράς υπερκειμένου (HTTP/2), ο τρόπος λειτουργίας των WebSocket αλλάζει. Η HTTP/2, παρά όλα τα προτερήματα που παρουσιάζει έναντι της HTTP/1.1, είναι ασύμβατη με τον υπάρχοντα τρόπο χρήσης των WebSockets. Σαν παράδειγμα μπορούμε να αναφέρουμε ότι οι επικεφαλίδες αιτημάτων `Upgrade` και `Connection`, δεν υποστηρίζονται στο HTTP/2 αφού δεν έχουν νόημα ύπαρξης σε αυτό.

Αιτία της ασυμβατότητας είναι η φύση του νέου πρωτοκόλλου που χρησιμοποιεί παρασκηνιακά μία μόνο σύνδεση TCP και εφαρμόζει τεχνικές πολυπλεξίας (multiplexing) στα μηνύματα που διακινούνται. Εδώ εμφανίζεται το ερώτημα: πώς θα ζητήσουμε την αναβάθμιση της σύνδεσης HTTP2 σε WebSocket όταν αυτή είναι αποκλειστικά και μόνο, μία;

Η απάντηση που δίνεται το RFC 8441 είναι ότι δεν υπάρχει λόγος να προσπαθήσουμε να αναβαθμίσουμε τη σύνδεση, αφού η σύνδεση WebSocket μπορεί να υλοποιηθεί ως μία ακόμη ροή HTTP/2, αποκομίζοντας, παράλληλα, τα πλεονεκτήματα της χρήσης αυτού του πρωτοκόλλου [64].

Οι επικεφαλίδες `Upgrade` και `Connection` δεν υποστηρίζονται πλέον. Τη θέση τους παίρνει ένα αίτημα με μέθοδο `CONNECT`, όπως αυτή ορίζεται για το HTTP/2 και επεκτείνεται από το RFC 8441. Η επέκταση της μεθόδου εισάγει την ψευδο-κεφαλίδα (pseudo-header) `:protocol` που χρησιμοποιείται για να υποδείξει το υπερκείμενο πρωτόκολλο – websocket στην περίπτωση αυτή. Η χρήση της απαιτεί ο πελάτης να έχει

λάβει προηγουμένως, την παράμετρο SETTINGS_ENABLE_CONNECT_PROTOCOL από τον εξυπηρετητή και αυτή να έχει τιμή 1.

Για την εγκαθίδρυση της σύνδεσης, αποστέλλεται ένα CONNECT αίτημα και απαραίτητα οι ψευδο-επικεφαλίδες :protocol, :scheme και :path. Το :scheme πρέπει να έχει τιμή «https» για ασφαλείς συνδέσεις και «http» για μη ασφαλείς. Το :path δείχνει τη διεύθυνση στην οποία ακούει ο WebSocket εξυπηρετητής. Απαραίτητη είναι και η ψευδο-επικεφαλίδα :authority, η οποία αντικαθιστά την επικεφαλίδα HOST.

Από τις επικεφαλίδες που χρησιμοποιούμε για το HTTP/1.1, οι Origin, Sec-WebSocket-Version, Sec-WebSocket-Protocol και Sec-WebSocket-Extensions συνεχίζουν να ισχύουν και απαιτούνται κανονικά, ενώ οι επικεφαλίδες Sec-WebSocket-Key και Sec-WebSocket-Accept αποσύρονται αφού τις λειτουργίες τους τις αναλαμβάνει η ψευδο-επικεφαλίδα :protocol.

Αφού ο εξυπηρετητής εγκρίνει τη σύνδεση, ανοίγει μία ροή HTTP/2 και όλα τα μηνύματα στέλνονται από εκεί. Για να κλείσει η WebSocket σύνδεση, χρειάζεται να κλείσουμε την υποκείμενη ροή.

#### *3.1.2.5 Τα WebSockets ως διεπαφή προγραμματισμού εφαρμογών*

Η διεπαφή προγραμματισμού WebSocket, όπως αυτή ορίζεται από το HTML Living Standard και παρέχεται από τους εκάστοτε κατασκευαστές φυλλομετρητών, επιτρέπει στις εφαρμογές τη χρήση του πρωτοκόλλου. Σκοπός της είναι να παρέχει ένα απλό και εύκολο τρόπο ανάπτυξης εφαρμογών που αξιοποιούν την τεχνολογία των WebSockets. Είναι ασύγχρονη και καθοδηγούμενη από συμβάντα (event-driven), για τα οποία η εφαρμογή προσθέτει χειριστές γεγονότων.

Πρόσβαση στη διεπαφή προγραμματισμού έχουμε με τον κατασκευαστή WebSocket, που ορίζεται ως πεδίο στο παγκόσμιο αντικείμενο window και χρησιμοποιείται για τη δημιουργία WebSocket συνδέσεων. Σημειώνεται ότι, όπως το WebRTC, δεν είναι δυνατός ο απευθείας χειρισμός των WebSockets μέσα από Workers.

Ένα αντικείμενο WebSocket κατασκευάζεται ως εξής:

```
1.  /* Δημιουργία WebSocket σύνδεσης */
2.
3.  let webSocket = null;
4.
5.  // έλεγχος υποστήριξης WebSocket
6.  if ('WebSocket' in window) {
7.    webSocket = new WebSocket('ws://www.example.com/wsEndpoint');
8.  }
```

Προτού χρησιμοποιήσουμε τον κατασκευαστή και λόγω της ετερογένειας των φυλλομετρητών που δουλεύουν πάνω στον παγκόσμιο ιστό μας, είναι χρήσιμο να εξετάσουμε αν αυτός υπάρχει ως πεδίο στο window. Εφόσον υπάρχει, μπορούμε να δημιουργήσουμε μία νέα σύνδεση καλώντας τον και δίνοντάς του το απαραίτητο όρισμα – μία διεύθυνση στην οποία αναμένουμε τον εξυπηρετητή να παρακολουθεί αιτήματα εγκαθίδρυσης συνδέσεων WebSocket. Προαιρετικά, σαν δεύτερο όρισμα μπορούμε να δώσουμε μία λίστα από υπό-πρωτόκολλα που μπορεί να διαχειριστεί η εφαρμογή μας.

Αν παρατηρήσουμε καλά, θα δούμε ότι η διεύθυνση που περάσαμε στον κατασκευαστή έχει διαφορετικό σχήμα (scheme) από τα http και https που συναντάμε συνήθως. Το πρωτόκολλο WebSocket ορίζει τα σχήματα ws και wss, για να την αναφορά σε WebSocket endpoints.

Καλώντας τον κατασκευαστή, λαμβάνουμε ένα αντικείμενο τύπου `WebSocket`. Επειδή η δημιουργία της σύνδεσης είναι ασύγχρονη, είναι απαραίτητο, αμέσως μετά την κλήση του κατασκευαστή, να εγκαταστήσουμε τους χειριστές γεγονότων που θα καλούνται όταν αλλάξει η κατάσταση της σύνδεσης ή ληφθεί νέο μήνυμα (όπως και στην περίπτωση των data channels στο WebRTC). Ειδάλλως, μπορεί να μην ενημερωθούμε για ένα ή περισσότερα συμβάντα. Ακολουθεί ένα παράδειγμα με τους τέσσερις χειριστές γεγονότων που μπορούμε να ορίσουμε:

```
/* Εγκατάσταση χειριστών γεγονότων */

// Η σύνδεση άνοιξε επιτυχώς
webSocket.addEventListener('open', function () { ... });

// Λάβαμε νέο μήνυμα
webSocket.addEventListener('message', function (messageEvent) {
  // επεξεργασία δεδομένων μηνύματος
  process(event.data);
});

// Υπήρξε σφάλμα σύνδεσης
webSocket.addEventListener('error', function (errorEvent) {
  // τύπωσε κωδικό σφάλματος
  console.error(errorEvent.code);
});

// Η σύνδεση έκλεισε
webSocket.addEventListener('close', function (closeEvent) {
  // τύπωσε κωδικό λήξης σύνδεσης
  console.log(closeEvent.code);
});
```

Παρατηρείστε ότι το μήνυμά μας είναι προσβάσιμο μέσα από το πεδίο `data` του `messageEvent`. Αν υπάρχει σφάλμα κατά την σύνδεση, τότε μπορούμε να εξετάσουμε το `errorEvent` ώστε να βρούμε την αιτία. Μετά από ένα σφάλμα ή ένα αίτημα διακοπής,

καλείται το χειριστής για το γεγονός “close”. Στο `closeEvent` μπορούμε να εντοπίσουμε το λόγο.

Η αποστολή των δεδομένων γίνεται με τη χρήση της μεθόδου `send()` του αντικειμένου WebSocket. Ανάλογα με το όρισμα που δώσουμε, τα πλαίσια που θα κατασκευαστούν μπορεί να είναι δυαδικού τύπου ή τύπου κειμένου. Προσοχή χρειάζεται να βεβαιωθούμε ότι η σύνδεση είναι ανοικτή, ειδάλλως το σενάριο θα εγείρει εξαίρεση (exception). Για τον έλεγχο της κατάστασης σύνδεσης, μπορούμε να εξετάσουμε το πεδίο `readyState` του WebSocket.

```
1.  /* Αποστολή δεδομένων κειμένου */
2.
3.  if (webSocket.readyState === 1) {
4.    webSocket.send('Καλησπέρα!');
5.  } else {
6.    console.warn('Η σύνδεση δεν είναι διαθέσιμη.')
7.  }
```

Η καταστάσεις από τις οποίες διέρχεται μία σύνδεση είναι οι:

- CONNECTING – η σύνδεση ετοιμάζεται,
- OPEN – η σύνδεση είναι ανοικτή,
- CLOSING – η σύνδεση είναι στη διαδικασία κλεισίματος,
- CLOSED – η σύνδεση απέτυχε ή έκλεισε.

Μία δημοφιλής πρακτική είναι η σειριακοποίηση και αποστολή απλών αντικειμένων με τη χρήση της μεθόδους `JSON.stringify()`.

Αν υποπτευόμαστε ότι η εφαρμογή μας θα παράγει ένα μεγάλο όγκο μηνυμάτων, μπορούμε να ελέγχουμε το πεδίο `bufferedAmount` του WebSocket και να στέλνουμε δεδομένα όταν η τιμή μηδενίζεται. Το πεδίο αναφέρει τον κατειλημμένο χώρο στην ενδιάμεση μνήμη όπου αποθηκεύονται τα μηνύματα πριν αποσταλούν. Αυτή είναι απαραίτητη, γιατί ο ρυθμός παραγωγής μηνυμάτων ενδέχεται να είναι μεγαλύτερος από τον ρυθμό μεταφοράς τους από το δίκτυο.

### *3.1.3 Cache*

Η διεπαφή προγραμματισμού Cache επιτρέπει την εύκολη αποθήκευση και ανάκτηση ζευγαριών αιτημάτων-αποκρίσεων (requests-responses). Δε σχετίζεται με την HTTP Cache, η οποία δεν είναι προσβάσιμη από την εφαρμογή και την ευθύνη διαχείρισής της την έχει ο φυλλομετρητής. Αντιθέτως, η Cache σχεδιάστηκε για να

δώσει τη δυνατότητα στους προγραμματιστές ιστού, να διαχειρίζονται οι ίδιοι μία δική τους κρυφή μνήμη.

Η διεπαφή Cache σχεδιάστηκε με αφορμή μία άλλη τεχνολογία, αυτής των ServiceWorker. Μάλιστα η προτυποποίηση της Cache γίνεται στο ίδιο έγγραφο (working draft) [65], για το οποίο είναι υπεύθυνο το W3C. Ο συνδυασμός των Service Worker και Cache φιλοδοξεί να επιτρέψει σε εφαρμογές ιστού να λειτουργούν ακόμα και όταν δεν υπάρχει διαθέσιμη σύνδεση στο διαδίκτυο.

#### *3.1.3.1 Η ApplicationCache ως προπομπός της Cache*

Πριν την ανάπτυξη της διεπαφής, ήταν ήδη διαθέσιμη σε πολλούς φυλλομετρητές η τεχνολογία ApplicationCache, η οποία είχε παρόμοιο σκοπό – την υποστήριξη της offline λειτουργίας στον ιστό. Σε αυτή, ήταν αρκετή η σύνταξη ενός αρχείου manifest, στο οποίο δηλώνονται οι διευθύνσεις των αρχείων προς αποθήκευση στη κρυφή μνήμη και η σύνδεση του αρχείου με συγκεκριμένη ιστοσελίδα μέσα από τον πηγαίο κώδικα της.

Με το που ξεκίνησε η υποστήριξη της τεχνολογίας ApplicationCache σε φυλλομετρητές, κατέστη αμέσως ιδιαιτέρα δημοφιλής, αφού ήταν απλή στη χρήση της χωρίς να απαιτεί περίπλοκα αρχεία σεναρίων. Η απλότητά της έγκειται κυρίως στη δηλωτική (declerative) φύση του αρχείου manifest.

Παρά την ευκολία στη χρήση της, η ApplicationCache θα συναντήσει και αυστηρή κριτική. Χαρακτηριστικό παράδειγμα είναι αυτό του Jake Archibald, ενός εκ των συντακτών του εγγράφου προτυποποίησης των Service Workers που σε διαδικτυακή του ανάρτηση [66] αναφέρει τα εξής προβλήματα γι’ αυτήν:

- Αποκλειστική χρήση της ApplicationCache ακόμα κι όταν υπάρχει σύνδεση στο διαδίκτυο.
- Για να ενημερωθεί η κρυφή μνήμη όταν αλλάζει κάποιο αρχείο, πρέπει να αλλάξει και το manifest.
- Η αλλαγή του manifest δεν εγγυάται την ενημέρωση των αρχείων, γιατί η ApplicationCache συμβουλεύεται και την HTTP cache του φυλλομετρητή, η οποία μπορεί να επιλέξει να επιστρέψει παλαιά έκδοση του αρχείου αν δεν είναι σωστά ρυθμισμένες οι επικεφαλίδες που επηρεάζουν το HTTP caching.
- Αν η διεύθυνση του manifest περιληφθεί στο ίδιο το αρχείο, τότε κλειδώνουμε οποιαδήποτε δυνατότητα ενημέρωσης της εφαρμογής.

- Σε αποθηκευμένες σελίδες, ακόμα και αν διαθέτουμε πρόσβαση στο διαδίκτυο, δε θα φορτώσουν αντικείμενα που δεν περιλαμβάνονται στο manifest. Πρέπει να υπάρχει ειδική ρητή ρύθμιση που να επιτρέπει τη λήψη τους μέσω δικτύου.
- Δεν υπάρχει δυνατότητα για αποθήκευση υπό όρους – οι διευθύνσεις ορίζονται στατικά.
- Δε ενημερώνει για το λόγο αποτυχίας μίας λήψης.
- Προσπάθειες ανακατεύθυνσης (redirect) σε άλλο όνομα τομέα (domain) δεν λειτουργούν.
- Βιβλιοθήκες για Ajax λανθασμένα αναγνωρίζουν αποκρίσεις από την ApplicationCache, όταν είμαστε offline, σαν αποτυχία.

Συμπερασματικά, αναγνωρίζει την αξία που προσφέρει η ApplicationCache, επισημαίνοντας όμως το γεγονός ότι η περιπλοκότητα των σημερινών εφαρμογών ιστού είναι τόσο μεγάλη που είναι αδύνατο να υποστηριχθούν πλήρως από αυτή.

#### *3.1.3.2 Η διεπαφή προγραμματισμού εφαρμογών Cache*

Η νέα διεπαφή Cache έρχεται να δώσει μία πιο ευέλικτη λύση στην προσωρινή αποθήκευση αρχείων ιστού. Με αυτή μία εφαρμογή έχει πλήρη δυνατότητα να δημιουργήσει και να διαχειριστεί πλήρως, όσες κρυφές μνήμες χρειάζεται. Ο προγραμματιστής της εφαρμογής καλείται πλέον να κάνει τις δικές του επιλογές αναφορικά με θέματα όπως είναι η ενημέρωσή της και ο καθαρισμός της.

Για λόγους ασφαλείας, κάθε origin έχει τις δικές του μνήμες, όπου κανένα άλλο origin δεν έχει πρόσβαση. Τα origins χρησιμοποιούνται συχνά για την εξουσιοδότηση ενός σεναρίου να εκτελέσει μία ενέργεια. Ένα origin είναι ένας πίνακας με τις εξής πληροφορίες:

- Σχήμα (scheme),
- Host
- Θύρα (port)
- Όνομα τομέα (domain name)

Η Cache χρησιμοποιεί τον αποθηκευτικό χώρο του συστήματος στο οποίο τρέχει, κάνοντας εγγραφές και ανακτήσεις. Αυτό συνεπάγεται μία λογική καθυστέρηση, όσο ο φυλλομετρητής ζητά από το λειτουργικό σύστημα πρόσβαση στο χώρο της Cache. Για το

λόγο αυτό, η διεπαφή παρέχει ασύγχρονες μεθόδους που χρησιμοποιούν το Promise API της JavaScript.

Πρόσβαση στη διεπαφή Cache παίρνουμε μέσα από το αντικείμενο caches που εντοπίζεται σαν πεδίο στο window και είναι τύπου CacheStorage. Ακόμη, το αντικείμενο είναι διαθέσιμο σε Workers και ServiceWorkers, ως παγκόσμια μεταβλητή (global variable).

Το CacheStorage είναι ένας πίνακας αντιστοίχισης που ταιριάζει ονόματα προσωρινών μνημών με αντικείμενα τύπου Cache. Έτσι μπορούμε να βρίσκουμε όλες τις διαθέσιμες μνήμες για το origin μας. Τις σημαντικότερες μεθόδους ενός CacheStorage περιγράφει συνοπτικά ο Πίνακας 3-3:

**Πίνακας 3-3: Βασικές μέθοδοι του CacheStorage**

| Όνομα μεθόδου | Περιγραφή |
|---|---|
| `match(request, options)` | Αναζητά σε όλες τις διαθέσιμες προσωρινές μνήμες, αν έχουν αποθηκευμένη απόκριση για το παρεχόμενο αίτημα. Στο αντικείμενο options, μπορούμε να δηλώσουμε σε ποια μνήμη να γίνει η αναζήτηση και αν θέλουμε να αγνοήσουμε μεταβλητές που δηλώνονται στο URL (queries), την μέθοδο του αιτήματος και την επικεφαλίδα Vary. |
| `has(cacheName)` | Ελέγχει αν υπάρχει προσωρινή μνήμη με το παρεχόμενο όνομα. |
| `open(cacheName)` | Ανοίγει και επιστρέφει την προσωρινή μνήμη ως αντικείμενο τύπου Cache. |
| `delete(cacheName)` | Διαγράφει την προσωρινή μνήμη, εφόσον αυτή υπάρχει. |
| `keys()` | Δίνει πρόσβαση σε ένα πίνακα με όλα τα ονόματα των προσωρινών μνημών που υπάρχουν για το origin μας. |

Τις σημαντικότερες μεθόδους ενός αντικειμένου Cache περιγράφει ο Πίνακας 3-4:

**Πίνακας 3-4: Βασικές μέθοδοι του ενός Cache στιγμιοτύπου**

| Όνομα μεθόδου | Περιγραφή |
|---|---|
| `match(request, options)` | Επιστρέφει την πρώτη απόκριση που θα βρει στην προσωρινή μνήμη και ταιριάζει στο παρεχόμενο request. Στο αντικείμενο options, μπορούμε να δηλώσουμε σε ποια μνήμη να γίνει η αναζήτηση και αν θέλουμε να αγνοήσουμε μεταβλητές που δηλώνονται στο URL (queries), την μέθοδο του αιτήματος και την επικεφαλίδα Vary. |
| `matchAll(request, options)` | Επιστρέφει όλες τις αποκρίσεις που θα βρει στην προσωρινή μνήμη και ταιριάζουν στο παρεχόμενο request. Στο αντικείμενο options, μπορούμε να δηλώσουμε σε ποια μνήμη να γίνει η αναζήτηση και αν θέλουμε να αγνοήσουμε μεταβλητές που δηλώνονται στο URL (queries), την μέθοδο του αιτήματος και την επικεφαλίδα Vary. |
| `add(request)` | Πραγματοποιεί λήψη της απόκρισης για το παρεχόμενο request και εφόσον είναι επιτυχής (κωδικός κατάστασης αρχίζει από 2), την αποθηκεύει στην προσωρινή μνήμη. |
| `addAll(requests)` | Πραγματοποιεί λήψη των αποκρίσεων για όλα τα request του παρεχόμενου πίνακα και εφόσον είναι επιτυχείς (κώδικας κατάστασης αρχίζει από 2), τις αποθηκεύει στην προσωρινή μνήμη. |

| | |
|---|---|
| `put(request, response)` | Αποθηκεύει στην προσωρινή μνήμη το παρεχόμενο ζευγάρι αιτήματος-απόκρισης. Δεν κάνει έλεγχο αν η απόκριση ήταν επιτυχής (κώδικας κατάστασης αρχίζει από 2). Αντικαθιστά οποιοδήποτε άλλο ζευγάρι έχει το ίδιο request. |
| `delete(request, options)` | Διαγράφει κάθε ζευγάρι αιτήματος-απόκρισης που ταιριάζει στο παρεχόμενο request. Στο αντικείμενο options, μπορούμε να δηλώσουμε σε ποια μνήμη να γίνει η αναζήτηση και αν θέλουμε να αγνοήσουμε μεταβλητές που δηλώνονται στο URL (queries), την μέθοδο του αιτήματος και την επικεφαλίδα Vary. |
| `keys(request)` | Δίνει πρόσβαση σε όλα τα κλειδιά (αντικείμενα αιτήματος) που υπάρχουν στην προσωρινή μνήμη. |

Ακολουθεί ένα παράδειγμα που δείχνει τη χρήση των μεθόδων `CacheStorage.open()`, `Cache.match()` και `Cache.put()`. Στο παράδειγμα ανοίγουμε μία προσωρινή μνήμη και ελέγχουμε αν έχει αποθηκευμένη μία απόκριση για το αίτημά μας. Αν ναι, το χρησιμοποιούμε. Διαφορετικά, δοκιμάζουμε να κάνουμε λήψη του. Εφόσον είναι επιτυχής, αποθηκεύουμε το ζευγάρι request-response στη μνήμη και επιστρέφουμε ένα αντίγραφό του προς χρήση από την εφαρμογή.

```
1.  /* Παράδειγμα αποθήκευσης/ανάκτησης */
2.
3.  const getFile = async () => {
4.    // Ορισμός σταθερών: όνομα μνήμης και διεύθυνση λήψης.
5.    const CACHE_NAME = 'exampleCache';
6.    const FETCH_URL = 'http://www.example.com/';
7.
8.    // Άνοιγμα προσωρινής μνήμης.
9.    const myCache = await caches.open(CACHE_NAME);
10.   // Αναζήτηση διεύθυνσης λήψης στη μνήμη.
11.   let response = await myCache.match(FETCH_URL);
12.
13.   if (response) {
14.     // Βρέθηκε στη μνήμη, η συνάρτηση επιστρέφει.
15.     return response;
16.   } else {
```

```
    // Δεν βρέθηκε στη μνήμη, λήψη μέσω fetch.
    try {
      response = await fetch(FETCH_URL);
      if (response.ok) {
        // Λήψη επιτυχής, αποθήκευση στη μνήμη.
        // Απαραίτητη η κλωνοποίηση της απόκρισης, γιατί
        // αποτελεί ροή και καταναλώνεται όταν χρησιμοποιείται.
        myCache.put(FETCH_URL, response.clone());
        // Η συνάρτηση επιστρέφει.
        return response;
      } else {
        // Λάβαμε απόκριση με μη επιτυχή κωδικό λήψης (200-299).
        throw new TypeError('Απόκριση με σφάλμα.')
      }
    } catch (error) {
      // Αποτυχία λήψης
      throw error;
    }
  }
};
```

### *3.1.3.3 Χώρος Αποθήκευσης και Όρια*

Η ευθύνη της διαχείρισης της προσωρινής μνήμης ανήκει πλέον στην ίδια την εφαρμογή. Κάθε σύστημα έχει ένα περιορισμένο αριθμό πόρων και δεν είναι δυνατό να αφιερώσει απεριόριστο χώρο αποθήκευσης προς χρήση. Η καταχρηστική χρήση των πόρων του συστήματος θα σήμαινε αρνητική εμπειρία χρήσης για τον χρήστη του. Για αυτό το λόγο, οι κατασκευαστές φυλλομετρητών, θέτουν όρια στη χρήση του χώρου αποθήκευσης του υπολογιστικού συστήματος.

Τα όρια αυτά δεν επιβάλλονται – κι ούτε αναφέρονται – στο έγγραφο προτυποποίησης της Cache. Αντιθέτως, κάθε φυλλομετρητής είναι ελεύθερος να χρησιμοποιήσει τα δικά του όρια. Γενικά οι φυλλομετρητές επιλέγουν όρια ανάλογα με τον ελεύθερο χώρο που δηλώνει το λειτουργικό σύστημα, ο οποίος μοιράζεται στα διάφορα APIs που πρέπει να πραγματοποιούν εγγραφές στο δίσκο όπως π.χ. cookies, IndexedDB και φυσικά η διεπαφή Cache. Παρακάτω αναφέρουμε ενδεικτικά την ισχύουσα πρακτική των δημοφιλέστερων, με την υποσημείωση ότι μπορεί να αλλάξουν απροσδόκητα:

- Στον Firefox, υπάρχουν δύο όρια αποθήκευσης: το παγκόσμιο και το όριο ομάδας (group limit) [67]. Το παγκόσμιο όριο ορίζεται ως το 50% του συνολικού χώρου του συστήματος. Αν αυτό ξεπεραστεί, τρέχει μία διαδικασία με όνομα «origin eviction», η οποία διαγράφει το αποθηκευμένο περιεχόμενο ενός origin ολοκληρωτικά (βάσει το πόσο πρόσφατα έγινε χρήση του χώρου). Η διαδικασία συνεχίζει μέχρι ο χρησιμοποιούμενος χώρος να πέσει κάτω από το όριο. Το όριο ομάδας χωρίζει τα origins σε ομάδες με βάση το όνομα τομέα (eTLD+1). Κάθε

ομάδα δεν επιτρέπεται να ξεπερνά το 20% του παγκόσμιου ορίου. Αν αυτό συμβεί, τα δεδομένα δε διαγράφονται αλλά απαγορεύεται στην ομάδα η χρήση επιπλέον χώρου.

- Στον Chrome [68], επιδιώκεται η διατήρηση ενός ελάχιστου διαθέσιμου χώρου της τάξεως των 2GB για συστήματα με συνολικό χώρο αποθήκευσης άνω των 20GB και 10% επί του συνολικού χώρου για συστήματα με λιγότερο. Αν αυτό το όριο ξεπεραστεί, ο φυλλομετρητής δεν επιτρέπει την περαιτέρω αποθήκευση αρχείων. Μάλιστα, σε περίπτωση που ο διαθέσιμος χώρος πέσει κάτω του 1GB για συστήματα με συνολικό χώρο αποθήκευσης άνω των 100GB και κάτω του 1% επί του συνολικού χώρου για συστήματα με λιγότερο, τότε ξεκινά η διαδικασία εκδίωξης αντικειμένων.
- Στον Edge [69], τα συστήματα χωρίζονται σε τέσσερις κλάσεις ανάλογα με το συνολικό χώρο αποθήκευσης που διαθέτουν. Υπάρχει ένα γενικό όριο και ένα όριο ανά όνομα τομέα (domain name). Το γενικό όριο ορίζεται στα 50MB για έως 8GB χώρου, στα 500MB για έως 32GB, στο 4% του χώρου για έως 500GB και τέλος στα 20GB για τις υπόλοιπες περιπτώσεις. Το όριο ανά όνομα τομέα υπολογίζεται ως το 20% του γενικού ορίου.

## 3.2 Τεχνολογίες στην πλευρά του εξυπηρετητή

Στην προηγούμενη ενότητα είδαμε τις απαραίτητες τεχνολογίες που κάνουν δυνατή την αξιοποίηση των πελατών ενός διαδικτυακού τόπου, προκειμένου να συμμετέχουν με δικούς τους πόρους στη διανομή των εγγράφων του. Μιλήσαμε για το WebRTC που επιτρέπει την ομότιμη επικοινωνία των πελατών, για τα WebSockets που επιτρέπουν την άμεση και αμφίδρομη επικοινωνία πελάτη-εξυπηρετητή και της διεπαφής Cache που δίνει τη δυνατότητα να αποθηκεύουμε αποκρίσεις σε μια προσωρινή μνήμη.

Σε αυτή την ενότητα θα μιλήσουμε για τις τεχνολογίες που επιλέξαμε προκειμένου να ετοιμάσουμε το σύστημα υποστήριξης (backend) της εφαρμογής μας. Θα μιλήσουμε για την καθολικότητα της γλώσσας προγραμματισμού σεναρίων JavaScript και το περιβάλλον εκτέλεσης (runtime environment) Node.js που επιτρέπει την εκτέλεσή της σε μία πληθώρα λειτουργικών συστημάτων.

Τέλος, θα κάνουμε μία σύντομη αναφορά στις κυριότερες βιβλιοθήκες που χρησιμοποιήσαμε.

### *3.2.1 JavaScript*

Η ανάπτυξη του backend της εφαρμογής μας έγινε στη γλώσσα προγραμματισμού EcmaScript, κοινά γνωστή ως JavaScript. Στην παρούσα εργασία προτιμούμε τη χρήση της δεύτερης ονομασίας, αφού είναι η δημοφιλέστερη.

Μέχρι το 1995, όταν και ξεκίνησε η ανάπτυξη της JavaScript, οι σελίδες υπερκειμένου ήταν στατικές και αποτελούνταν κυρίως από μεγάλα κείμενα. Η χρήση της JavaScript φιλοδοξούσε να προσδώσει σε αυτές δυναμικότητα, παρέχοντας δυνατότητες προγραμματισμού τους. Με την ενσωμάτωσή της στον τότε δημοφιλή περιηγητή ιστού Netscape Navigator, η JavaScript απέδειξε τη χρησιμότητά της. Έκτοτε, οποιαδήποτε άλλη προσπάθεια έγινε για την εισαγωγή άλλων γλωσσών προγραμματισμού, δεν αντίκρισε την ίδια επιτυχία [70]. Σήμερα πλέον, θεωρείται η κατ' εξοχήν γλώσσα προγραμματισμού στον παγκόσμιο ιστό, ενώ η χρήση της έχει επεκταθεί πέρα από τα στενά όρια των φυλλομετρητών.

#### *3.2.1.1 Μηχανές JavaScript*

Ως γλώσσα σεναρίων, η JavaScript είναι μία συναρτησιακού τύπου γλώσσα δομημένου προγραμματισμού, που δίνει περισσότερη βαρύτητα στις διαδικασίες και λιγότερη στα δεδομένα. Χρησιμοποιεί δυναμικό και ασθενές σύστημα τύπων, αν και υπάρχουν εργαλεία που βοηθούν στον στατικό έλεγχο, όπως η TypeScript[1] και το Flow[2]. Υποστηρίζει πρωτοτυπική κληρονομικότητα, ξεχωρίζοντας έτσι από άλλες γλώσσες που βασίζονται σε κλάσεις.

Για την εκτέλεση του κώδικά της, απαιτείται η χρήση μίας μηχανής JavaScript, η οποία ερμηνεύει τις εντολές και τις εκτελεί. Στις μέρες μας, οι αποδοτικότερες μηχανές χρησιμοποιούν τεχνικές, όπως just-in-time μεταγλώττιση στα κυριότερα μέρη του σεναρίου (hot-paths), προκειμένου να εκτελέσουν το σενάριο όσο το δυνατόν αποδοτικότερα.

Στις σημαντικότερες μηχανές JavaScript περιλαμβάνονται η V8[3] της Google, η SpiderMonkey[4] και η Rhino[5] της Mozilla και η JavaScriptCore[6] της Apple.

[1] https://www.typescriptlang.org

[2] https://flow.org

[3] https://v8.dev

[4] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey

[5] https://mozilla.github.io/rhino

[6] https://developer.apple.com/documentation/javascriptcore

### *3.2.2 Node.js*

Η Node.js δημοσιεύθηκε το 2009 ως ένα ανοικτού λογισμικού έργο που επιτρέπει την ανάπτυξη εφαρμογών με JavaScript έξω από προγράμματα περιήγησης. Αν και αρχικά κύριο κίνητρο της ήταν η ανάπτυξη αποδοτικών δικτυακών εφαρμογών, χάρη στη δημοφιλία της, σήμερα υπάρχει μία μεγάλη ποικιλία από εργαλεία και βιβλιοθήκες που κάνουν δυνατή την ανάπτυξη κάθε είδους εφαρμογής.

Το γεγονός ότι το backend και το frontend ενός λογισμικού είναι πλέον δυνατό να αναπτυχθούν με μία κοινή γλώσσα προγραμματισμού, δίνει τη δυνατότητα σε πολλούς προγραμματιστές να ασχοληθούν και με τις δύο πλευρές (full-stack), αφού η εναλλαγή από το ένα περιβάλλον στο άλλο, δεν μεταβάλλει το γενικό πλαίσιο σκέψης τους. Στο [71] αναγνωρίζεται ότι οι full-stack προγραμματιστές είναι μία κατηγορία T-Shaped ατόμων, δηλαδή ατόμων που έχουν λειτουργικές γνώσεις σε πολλά επίπεδα ενός συστήματος/εφαρμογής/εταιρίας και είναι πολύτιμα για επιχειρήσεις.

#### *3.2.2.1 Υποσυστήματα της Node.js*

Μπορούμε να σκεφτούμε τη Node.js σαν ένα λογισμικό που απαρτίζεται από πολλά μικρότερα τμήματα [72]. Στη συνέχεια θα δούμε τα σημαντικότερα.

##### *Μηχανή V8*

Είναι η μηχανή JavaScript της Google, η οποία χρησιμοποιείται στον Chrome. Η V8 γράφεται σε C++, χρησιμοποιεί just-in-time μεταγλώττιση και θεωρείται γενικά μία από τις γρηγορότερες και ικανότερες μηχανές. Η Node.js χρησιμοποιεί την V8 προκειμένου να επιτρέψει στον προγραμματιστή να χρησιμοποιήσει την JavaScript στην πλευρά του εξυπηρετητή.

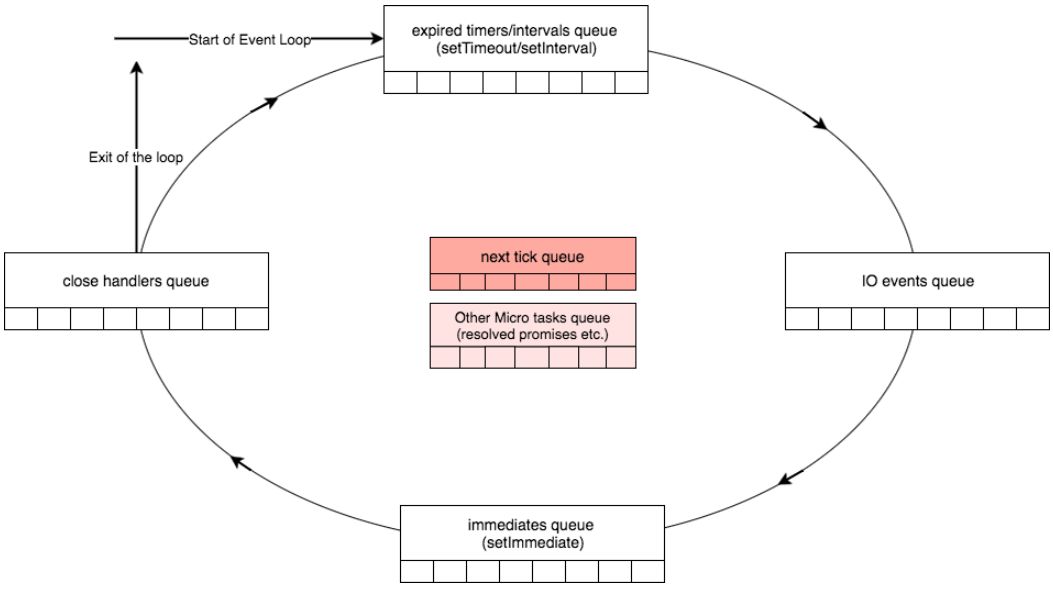
Θα πρέπει να σημειώσουμε ότι η JavaScript σαν γλώσσα διαθέτει ένα περιορισμένο αριθμό διεπαφών, πράγμα που έρχεται σε αντίθεση με άλλες γλώσσες προγραμματισμού όπως η Python και η PHP. Αυτό είναι συνειδητή απόφαση των κατασκευαστών της. Ο μεγαλύτερος αριθμός διεπαφών που συναντάμε στον προγραμματισμό για φυλλομετρητές, αποτελεί μέρος του Web API και του DOM, κι όχι της ίδιας της γλώσσας. Αυτό σημαίνει ότι οι έτοιμες διεπαφές που μας παρέχει η JavaScript στη Node.js είναι περιορισμένες σε αριθμό και λειτουργίες. Θα δούμε σε λίγο, πώς οι χρήστες της γλώσσας ξεπερνούν αυτό τον περιορισμό με τη χρήση ενός κοινού ανοικτού καταλόγου βιβλιοθηκών, του NPM.

### *Libuv*

Είναι γεγονός ότι η JavaScript δεν υποστηρίζει το μοντέλο της πολυνηματικής επεξεργασίας (multi-threaded programming). Σε αυτό το μοντέλο, όταν απαιτείται η εκτέλεση μίας συνάρτησης που απαιτεί αυξημένο χρόνο (συνήθως διαδικασίες εισόδου-εξόδου), δημιουργείται ένα ξεχωριστό νήμα το οποίο περιμένει την ολοκλήρωσή της και την επιστροφή του αποτελέσματός της. Χωρίς αυτήν την τεχνική, ο επεξεργαστής θα έπρεπε να αναμένει αδρανής (block) και το πρόγραμμα εκτέλεσης θα έμοιαζε μη αποκρίσιμο.

Οι δικτυακές εφαρμογές, όπως αυτές που τρέχουν σε έναν εξυπηρετητή, αποτελούν μία κατηγορία εφαρμογών στις οποίες απαιτείται συχνά η δημιουργία νημάτων. Για κάθε πελάτη που προσπαθεί να συνδεθεί, ο εξυπηρετητής θα πρέπει να αφιερώσει για τη σύνδεση, ένα αποκλειστικό νήμα. Η χρήση όμως πολλών νημάτων είναι επιβαρυντική για την εύρυθμη λειτουργία του, αφού καταλαμβάνει πόρους από το σύστημα (κυρίως μνήμη). Συνεπάγεται επομένως ότι, κάθε εξυπηρετητής έχει ένα ανώτατο όριο ταυτόχρονων συνδέσεων που μπορεί να υποστηρίξει, ανάλογο των πόρων που διαθέτει.

Από την άλλη πλευρά, η JavaScript χειρίζεται το πρόβλημα της καθυστέρησης των συναρτήσεων εισόδου-εξόδου, εφαρμόζοντας το μοντέλο της ασύγχρονης επεξεργασίας. Σύμφωνα με αυτό, υπάρχουν σύγχρονες και ασύγχρονες συναρτήσεις. Καθώς το πρόγραμμα εκτελείται σειριακά, όταν κληθεί μία ασύγχρονη συνάρτηση, αυτή, καθώς και όλες οι μεταβλητές που είναι προσβάσιμες (in scope), τοποθετούνται στον βρόγχο συμβάντων (event loop) προκειμένου η συνάρτηση να εκτελεστεί σε δεύτερο χρόνο, όταν θα αδειάσει η στοίβα κλήσεων (call stack). Το αποτέλεσμα της κλήσης αξιοποιείται με τη βοήθεια χειριστών γεγονότων (callback functions).




**Εικόνα 3-6: Ο βρόγχος συμβάντων του Node.js**
**(πηγή: https://jsblog.insiderattack.net/event-loop-and-the-big-picture-nodejs-event-loop-part-1-1cb67a182810)**

Το libuv αποτελεί ένα αφαιρετικό επίπεδο πάνω από το λειτουργικό σύστημα, που διαχειρίζεται παρασκηνιακά τις συναρτήσεις εισόδου-εξόδου και συνεργάζεται με το βρόγχο συμβάντων, χάρη στον οποίο επιστρέφει τα αποτελέσματα στην JavaScript. Προσφέρει μια κοινή διεπαφή για όλα τα λειτουργικά που υποστηρίζει, πράγμα που οδήγησε στην υιοθέτησή του και από άλλες γλώσσες προγραμματισμού. Μερικές από τις λειτουργίες που αναλαμβάνει το libuv είναι διαχείριση του συστήματος αρχείων, δικτυακές αποστολές και λήψεις δεδομένων, χειρισμός θυγατρικών διεργασιών (child processes) κ.α.

Το libuv έχει αναπτυχθεί με τη γλώσσα C ώστε να είναι όσο το δυνατόν αποδοτικότερο. Χάρη σε αυτό, η Node.js απαλλάσσει τους προγραμματιστές από τη δύσκολη και χρονοβόρα ανάπτυξη πολυνηματικών αρχιτεκτονικών, μεταθέτοντας εν μέρει την ευθύνη για βέλτιστη αξιοποίηση των πόρων του συστήματος στο libuv.

***C/C++ βιβλιοθήκες***

Δυστυχώς, η JavaScript δεν έχει κάτι αντίστοιχο της βιβλιοθήκης stdlib της C. Η Node.js επιχειρεί να εμπλουτίσει τις διαθέσιμες λειτουργίες της JavaScript, κάνοντας διαθέσιμες βιβλιοθήκες για κρυπτογράφηση, συμπίεση δεδομένων, επίλυση ερωτημάτων DNS και ανάλυση του πρωτοκόλλου μεταφοράς υπερκειμένου. Η Node.js δεν προσπαθεί

να εφεύρει ξανά τον τροχό. Οι βιβλιοθήκες αυτές είναι ελεύθερο λογισμικό ανοικτού κώδικα που δεν έχουν αναπτυχθεί αποκλειστικά για χρήση από τη Node.js.

#### *3.2.2.2 Ο Διαχειριστής Πακέτων NPM*

Σε ένα κόσμο που οι λύσεις λογισμικού γίνονται συνεχώς όλο και περισσότερο περίπλοκες, δεν είναι δυνατόν να ζητείται από κάθε προγραμματιστή η ανάπτυξη κώδικα για ήδη επιλυμένα προβλήματα. Παρά τις προσθήκες της Node.js στις διεπαφές της JavaScript, είναι κοινά αποδεκτό ότι σαν πλατφόρμα ανάπτυξης λογισμικού, η Node.js προσφέρει εγγενώς μόνο τα απολύτως απαραίτητα. Για κάθε άλλη λειτουργία, προτρέπει τον προγραμματιστή να χρησιμοποιήσει τρίτες βιβλιοθήκες. Για το σκοπό αυτό, κατά την εγκατάστασή της, πέρα από το περιβάλλον εκτέλεσης (runtime environment) εγκαθίσταται και το npm (node package manager), ένα εργαλείο διαχείρισης πακέτων λογισμικού που επικοινωνεί με την ομώνυμη δημόσια κεντρική αποθήκη[1] (repository). Τα πακέτα αυτά μπορεί να έχουν διάφορες λειτουργίες, κυρίως όμως χρησιμοποιούνται για τη διανομή βιβλιοθηκών προς χρήση στη Node.js.

Η χρήση της αποθήκης npm για την εύρεση και χρήση βιβλιοθηκών διαθέτει μια σειρά από πλεονεκτήματα, τα σημαντικότερα εξ αυτών είναι:

- Διευκολύνει τη γρήγορη προτυποποίηση (prototyping) αφού σε αυτή υπάρχουν βιβλιοθήκες για οποιοδήποτε πρόβλημα.
- Επιτρέπει στους προγραμματιστές να επιλέγουν βιβλιοθήκες που έχουν δοκιμαστεί σε επίπεδο παραγωγής (production-ready).
- Συχνά η υλοποίηση που παρέχει μία βιβλιοθήκη είναι ασφαλέστερη, αφού προβλέπει ακραίες περιπτώσεις (edge cases). Σε περίπτωση κενού ασφαλείας, η αναβάθμιση σε ασφαλή έκδοση είναι απλή διαδικασία.
- Παρέχει βιβλιοθήκες που επικεντρώνονται στη μεγιστοποίηση της ταχύτητας εκτέλεσης.

### *3.2.3 Βιβλιοθήκες*

Στην ανάπτυξη της εφαρμογής μας χρησιμοποιήσαμε βιβλιοθήκες από την αποθήκη npm. Θεωρητικά θα μπορούσαμε να αναπτύξουμε τις λειτουργίες που μας ενδιαφέρουν από το μηδέν, όμως μία τέτοια προσπάθεια θα απαιτούσε να ληφθούν

[1] https://www.npmjs.com/

υπόψη πολλές πληροφορίες χαμηλού επιπέδου και θα αποσπούσε την προσοχή μας από τον κύριο στόχο. Παρακάτω αναφερόμαστε συνοπτικά στα σημαντικότερα πακέτα-βιβλιοθήκες που αξιοποιήσαμε για την ανάπτυξη του PAWS.

#### *3.2.3.1 rollup*

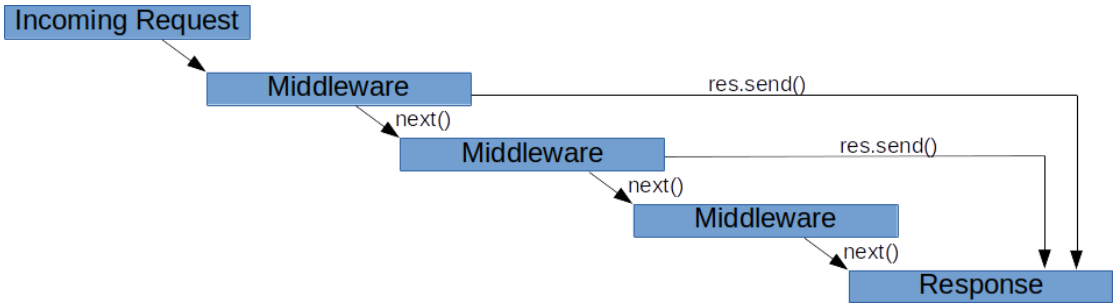
Το rollup είναι ένα module bundler για τη JavaScript που αναλαμβάνει τη συνένωση αρχείων πηγαίου κώδικα για τη δημιουργία μίας εφαρμογής ή βιβλιοθήκης [73]. Αν και συνήθως χρησιμοποιείται σαν εκτελέσιμο πρόγραμμα στο στάδιο της μεταγλώττισης (build step), μπορεί να χρησιμοποιηθεί και σαν βιβλιοθήκη, προκειμένου μία τρίτη εφαρμογή να αποκτήσει παρόμοιες λειτουργίες.

Το rollup είναι επεκτάσιμο με τη χρήση προσθέτων[1] που δίνουν επιπλέον δυνατότητες, όπως η μετατροπή κώδικα ES6 σε ES5 προκειμένου να είναι συμβατός με παλαιότερους φυλλομετρητές και εκδόσεις της Node.js, η ελαχιστοποίηση του μεγέθους των παραγόμενων αρχείων μέσω μεθόδων όπως το uglification και η συμπίεση αρχείων, η ενσωμάτωση εικόνων στον κώδικα ενός σεναρίου κ.α.

Στην εφαρμογή μας χρησιμοποιούμε το rollup για να ετοιμάσουμε το σενάριο που θα τρέξει στην πλευρά του πελάτη.

#### *3.2.3.2 express*

Το express είναι ένα framework για την ανάπτυξη εφαρμογών ιστού με JavaScript και Node.js. Κύρια χαρακτηριστικά του είναι η ευελιξία, η ταχύτητα και η απλή διεπαφή προγραμματισμού του [74]. Δεν στοχεύει να παρέχει κάθε λειτουργία που θα χρειαστεί ο χρήστης του. Αντιθέτως, φροντίζει ώστε να είναι εύκολη η προσθήκη επιπλέον λειτουργικότητας με τη χρήση τρίτων βιβλιοθηκών.




**Εικόνα 3-7: Η πορεία ενός αιτήματος μέσα από ενδιάμεσες συναρτήσεις του Express**

[1] https://github.com/rollup/awesome

**(πηγή: https://developer.okta.com/blog/2018/09/13/build-and-understand-express-middleware-through-examples)**

Το express προωθεί το μοντέλο της επεξεργασίας αιτημάτων με τη χρήση ενδιάμεσων συναρτήσεων (middleware functions). Σύμφωνα με αυτό, καθώς ένα αίτημα φτάνει στον εξυπηρετητή, περνάει ως όρισμα σε μία σειρά από συναρτήσεις μέχρις ότου μία από αυτές να επιστρέψει μία απόκριση (Εικόνα 3-7). Πιο συγκεκριμένα, μία ενδιάμεση συνάρτηση λαμβάνει ως ορίσματα τα αντικείμενα `request` και `response` και τη συνάρτηση `next()` που αποτελεί την επόμενη συνάρτηση προς κλήση, και μπορεί να:

- απαντήσει χρησιμοποιώντας κατάλληλες μεθόδους στο αντικείμενο `response` ή
- να υπολογίσει ενδιάμεσες πληροφορίες μέσω της ανάλυση του σώματος του αιτήματος ή της αυθεντικοποίησης του χρήστη. Αυτές οι πληροφορίες αποθηκεύονται σε πεδία των `request` και `response`, προκειμένου οι επόμενες συναρτήσεις να έχουν πρόσβαση σε αυτές. Στο τέλος εκτελεί την next προκειμένου να συνεχίσει η επεξεργασία.

Οι ενδιάμεσες συναρτήσεις διαχείρισης σφαλμάτων είναι μια ειδική κατηγορία ενδιάμεσων συναρτήσεων που δέχονται ως πρώτο όρισμα ένα αντικείμενο σφάλματος (`error`). Αν κατά το διάστημα επεξεργασίας σε μία ενδιάμεση συνάρτηση προκύψει σφάλμα, η `next()` πρέπει να κληθεί με αυτό ως πρώτο όρισμα, έτσι ώστε οι υπόλοιπες ενδιάμεσες συναρτήσεις, μέχρι την πρώτη συνάρτηση διαχείρισης σφαλμάτων, να παρακαμφθούν.

Το express χρησιμοποιείται από την εφαρμογή μας για την αποστολή στατικών αρχείων.

#### *3.2.3.3 ws*

Το ws παρουσιάζεται ως μία απλή, γρήγορη και ενδελεχώς δοκιμασμένη βιβλιοθήκη για την ανάπτυξη εξυπηρετητών WebSockets [75]. Η διεπαφή προγραμματισμού που παρέχει είναι οδηγούμενη από συμβάντα για τα οποία πρέπει να εγκαταστήσουμε χειριστές.

Σε αντίθεση με άλλες βιβλιοθήκες για WebSockets όπως η Socket.io, δεν περιλαμβάνει κάποιο σενάριο για εκτέλεση στην πλευρά του χρήστη, αλλά υποστηρίζει την εγγενή WebSocket διεπαφή του φυλλομετρητή.

Χρησιμοποιούμε το ws για να δημιουργήσουμε συνδέσεις WebSocket μεταξύ εξυπηρετητή και φυλλομετρητή.

#### *3.2.3.4 mime*

Το mime είναι μία απλή βιβλιοθήκη που αντιστοιχεί ονόματα αρχείων σε τύπους αρχείων σύμφωνα με το πρότυπο MIME (Multipurpose Internet Mail Extensions) [76]. Οι τύποι αρχείων MIME χρησιμοποιούνται εκτενώς από το πρωτόκολλο μεταφοράς υπερκειμένου (HTTP), προκειμένου να είναι δυνατό το άνοιγμα διαφόρων ειδών αρχείων.

Το mime μας επιτρέπει να ενημερώνουμε τον φυλλομετρητή για τον τύπο κάθε αρχείου, ώστε να αναγνωρίζεται σωστά.

#### *3.2.3.5 uuid*

Το uuid παράγει καθολικώς μοναδικά αναγνωριστικά (Universally Unique Identifiers - UUID) σύμφωνα με το RFC 4122[1] [77]. Τα UUID έχουν μήκος 128 bits και εγγυόνται τη μοναδικότητά τους χωρίς να απαιτείται κάποιο κεντρικό μητρώο.

Χρησιμοποιείται για την απόδοση μοναδικών αναγνωριστικών στους ομότιμους χρήστες.

#### *3.2.3.6 yargs*

Η βιβλιοθήκη yargs επιτρέπει τη δημιουργία εργαλείων γραμμής εντολών (command line tools), αναλαμβάνοντας την ανάλυση των ορισμάτων και την αυτόματη δημιουργία μενού βοήθειας που περιλαμβάνει υποδείξεις για τον τρόπο χρήσης τους [78].

[1] https://www.ietf.org/rfc/rfc4122.txt

# 4 PAWS – A Peer Assisted Web Server

Για την καλύτερη κατανόηση των ζητημάτων των ομότιμων δικτύων διανομής περιεχομένου και για την εξοικείωσή μας με τις παρούσες τεχνολογίες ιστού, αναπτύξαμε τη δική μας εφαρμογή. Το PAWS είναι ένας εξυπηρετητής ιστού για στατικά αρχεία, που αναπτύχθηκε με στόχο να αναλάβει τη διανομή περιεχομένου μιας ιστοσελίδας συνεργατικά με τους πελάτες της. Με αυτόν τον τρόπο επιδιώκουμε να προσφέρουμε χαμηλότερο κόστος εξυπηρέτησης σε σύγκριση με ένα κοινό εξυπηρετητή που λειτουργεί βάσει κεντρικοποιημένης αρχιτεκτονικής.

## 4.1 Απαιτήσεις και σχεδιασμός

Σε αυτή την ενότητα περιγράφουμε τα επιθυμητά χαρακτηριστικά του PAWS, σύμφωνα με τα οποία έγιναν οι επιλογές μας στη φάση του σχεδιασμού.

Πρώτη μας επιθυμία ήταν η αυτόματη λειτουργία του στη πλευρά του χρήστη, χωρίς αλλαγές στο σύστημά του. Όπως αποδείχθηκε από άλλες προσπάθειες δημιουργίας ομότιμων δικτύων διανομής περιεχομένου, η κινητοποίηση των πελατών προκειμένου να εγκαταστήσουν νέο λογισμικό, είτε ακόμη και για να αλλάξουν τις ρυθμίσεις του φυλλομετρητή τους, είναι πολύ δύσκολη. Το [40] απέδειξε ότι είναι δυνατή η δημιουργία ενός τέτοιου συστήματος με την αξιοποίηση των τεχνολογιών RTMFP και WebRTC.

Δεύτερη μας επιθυμία ήταν η ευκολία υιοθέτησης της λύσης μας από τους παρόχους περιεχομένου – τους ιδιοκτήτες των ιστοσελίδων. Έτσι δώσαμε τη δυνατότητα στο PAWS να λειτουργεί σαν μία οντότητα – ένα εκτελέσιμο πρόγραμμα – στην οποία τρέχουν όλα τα επιμέρους υποσυστήματά της (web server, tracker, signal channel).

Τρίτη και τελευταία μας επιθυμία ήταν να είναι ταχύς. Αναφερόμαστε τόσο στην ταχύτητα εύρεσης και σύνδεσης με άλλους ομότιμους peers, όσο και στην ταχύτητα λήψης του περιεχομένου – σημείο που λίγα από τα ομότιμα CDN, που παρουσιάσαμε στο Κεφάλαιο 2 επικεντρώθηκαν. Για το λόγο αυτό, κάναμε τις εξής τρεις επιλογές:

- σχεδιάσαμε την ακολουθία των βημάτων σύνδεσης και αιτήματος με τέτοιον τρόπο ώστε να μειώνεται ο χρόνος αδράνειας και επιλέξαμε το WebRTC έναντι του RTMFP, αφού παρουσίασε καλύτερες επιδόσεις σύμφωνα με το [40],
- αναθέσαμε την ευρετηρίαση των αρχείων και τη δρομολόγηση σε κεντρικό εξυπηρετητή. Με αυτό τον τρόπο επιτυγχάνεται η μικρότερη

δυνατή καθυστέρηση στην αναζήτηση περιεχομένου και στην εύρεση κατάλληλων ομότιμων χρηστών και

- επιλέξαμε τη διαίρεση των αρχείων σε μικρότερα τμήματα σταθερού μεγέθους, ώστε να είναι δυνατή η παράλληλη λήψη τους από πολλές πηγές.

Όσον αφορά στην ταχύτητα λήψης, πρέπει να παρατηρήσουμε ότι η τεχνολογία πρόσβασης πολλών παρόχων διαδικτυακής πρόσβασης (ISP) είναι ασύμμετρη όσον αφορά τη εύρος ζώνης της. Έτσι συχνά, το κανάλι αποστολής (uplink) ενός πελάτη είναι σημαντικά πιο αργό από το κανάλι λήψης (downlink). Αυτό αποτελεί σημαντικό περιορισμό που πρέπει να ληφθεί υπόψη στο σχεδιασμό οποιασδήποτε ομότιμης εφαρμογής. Μάλιστα συχνά οι πάροχοι επιλέγουν να τιμωρήσουν χρήστες που κάνουν υπερβολική χρήση τέτοιων εφαρμογών [79]. Πολλές έρευνες έχουν γίνει για να διαπιστωθεί αν αυτοί οι χρήστες αποτελούν αρνητική εξωτερικότητα, αν επηρεάζουν την κατάσταση των δικτύων των ISP [80] και πώς μπορούμε να ελαχιστοποιήσουμε τις επιπτώσεις τους στο δίκτυο [81]. Για το λόγο αυτό, ο εξυπηρετητής που σχεδιάσαμε μπορεί να χρησιμοποιήσει ως κριτήριο επιλογής για τη σύνδεση των ομότιμων χρηστών την IP τους, επιδιώκοντας τη σύνδεση πελατών που ανήκουν σε ίδιο AS (autonomous system), κρατώντας την κίνηση δεδομένων μέσα σε αυτό και ελαχιστοποιώντας τις συνέπειες για τους παρόχους.

Σύμφωνα με τα παραπάνω ορίσαμε τις λειτουργικές απαιτήσεις της εφαρμογής ως εξής:

- Δημιουργία server για διανομή αρχείων με το μοντέλο πελάτη-εξυπηρετητή.
- Δημιουργία server για διανομή αρχείων με ομότιμο τρόπο.
- Παραμετροποίηση με αρχείο ρυθμίσεων.
- Παροχή διεπαφής προγραμματισμού στη πλευρά του πελάτη.
- Αυτόματη αναζήτηση στην HTML για φόρτωση ενσωματωμένων αντικειμένων.

Ενώ μη λειτουργικές απαιτήσεις είναι οι εξής:

- Λειτουργία χωρίς εγκατάσταση στη πλευρά του πελάτη.
- Εύκολη εγκατάσταση και χρήση στη πλευρά του εξυπηρετητή.
- Υψηλή ταχύτητα σύνδεσης και λήψης.
- Δυναμική δημιουργία σεναρίου JavaScript και διανομή του.

- Αυτόματη δημιουργία και διαχείριση συνδέσεων.
- Διατήρηση ιστορικού λήψεων για κάθε πελάτη.
- Ταξινόμηση προτεινόμενων χρηστών με βάση το AS που ανήκουν.

## 4.2 Περιγραφή υλοποίησης στην πλευρά του εξυπηρετητή

Για τη λειτουργία του συστήματός μας απαιτείται η εκτέλεση διαδικασιών τόσο στην πλευρά του εξυπηρετητή όσο και στην πλευρά του χρήστη. Στην ενότητα αυτή περιγράφουμε τις ευθύνες και τον τρόπο λειτουργίας του PAWS από την πλευρά του εξυπηρετητή.

Κατά την εκκίνηση του προγράμματος εξυπηρέτησης, αυτό διέρχεται από τα εξής στάδια:

1. Ανάλυση ορισμάτων γραμμής εντολών
2. Ανάλυση αρχείου ρυθμίσεων
3. Δημιουργία σεναρίου JavaScript για την πλευρά του πελάτη
4. Εκκίνηση του HTTP(S) server
5. Εκκίνηση του WebSocket server

### *4.2.1 Ορίσματα γραμμής εντολών*

Το PAWS δέχεται τα εξής δύο ορίσματα γραμμής εντολών:

- `-c` ή `--config` επιτρέπει τον ορισμό του αρχείου ρυθμίσεων προς χρήση. Αν δεν οριστεί, το PAWS διαβάζει το αρχείο pawsrc.json ή το αρχείο paws.json, αν υπάρχουν, στον τρέχοντα κατάλογο εργασίας.
- `--only-build-client` δημιουργεί μόνο το αρχείο σεναρίου JavaScript. Χρησιμοποιείται κυρίως στην ανάπτυξη της εφαρμογής.

### *4.2.2 Αρχείο ρυθμίσεων*

Η παραμετροποίηση του PAWS, όπως είδαμε παραπάνω επιτυγχάνεται μέσω του σχετικού αρχείου. Ο Πίνακας 4-1 παρουσιάζει τις δυνατές επιλογές:

**Πίνακας 4-1: Επιλογές ρυθμίσεων PAWS**

| Όνομα ρύθμισης | Προκαθορισμένη τιμή | Περιγραφή |
|---|---|---|
| `httpPort` | 8080 | Η TCP θύρα στην οποία αναμένει για αιτήματα ο HTTP server. Σε περίπτωση που η ρύθμιση redirect είναι αληθής, τότε ο HTTP server θα κάνει ανακατεύθυνση (κωδικός 301) στον HTTPS server. Σε περίπτωση που η θύρα είναι <1000, τότε πιθανώς να απαιτούνται δικαιώματα υπερχρήστη. |
| `httpsPort` | - | Η TCP θύρα στην οποία αναμένει για αιτήματα εξυπηρέτησης ο HTTPS server. Σε περίπτωση που η θύρα είναι <1000, τότε πιθανώς να απαιτούνται δικαιώματα υπερχρήστη. |
| `index` | “index.html” | Το αρχείο HTML που αποτελεί την προκαθορισμένη αρχική σελίδα του ιστοτόπου. |
| `mountPoint` | | Ένα json αντικείμενο με δύο κλειδιά: το urlPath που δηλώνει τη σχετική διεύθυνση ιστού για την οποία είναι υπεύθυνος να εξυπηρετεί ο server μας καθώς και το localPath που αναφέρει την τοποθεσία στον κατάλογο αρχείων του server όπου βρίσκονται τα αρχεία για διαμοιρασμό. |
| `redirect` | false | Θα πρέπει να έχει τιμή true ή false. Σε περίπτωση που η τιμή του είναι true, μη ασφαλή αιτήματα (HTTP) ανακατευθύνονται (301) σε ασφαλή (HTTPS). Απαιτεί να έχει οριστεί η ρύθμιση httpsPort. |

| | | |
|---|---|---|
| **trustProxy** | false | Σε περίπτωση που ο server δε λαμβάνει απευθείας τα αιτήματα των χρηστών, αλλά διαμεσολαβεί κάποιος άλλος server, όπως αντίστροφοι διακομιστές μεσολάβησης και εξισορροπητές φορτίου, πρέπει να οριστεί η ρύθμιση αυτή, είτε παρέχοντας την τιμή true, είτε μία ή περισσότερες IP διευθύνσεις, είτε ένας αριθμός που δηλώνει τα βήματα (hops) από τον πρώτο μεσολαβητή. Περισσότερες πληροφορίες στο https://expressjs.com/en/guide/behind-proxies.html. |
| **webSocketPath** | "/ws-paws" | Το endpoint που θα χρησιμοποιηθεί για την δημιουργία καναλιού σηματοδότησης πελάτη-εξυπηρετητή (WebSockets). Η επιλογή αυτή παρέχεται ώστε να είναι εφικτή η χρήση περισσότερων της μίας σύνδεσης WebSockets, σε περίπτωση που ο ιστότοπος κάνει ήδη χρήση αυτών. Το path δεν χρησιμοποιεί ως αναφορά το mountpoint αλλά τη ριζική διεύθυνση του ιστοτόπου. |

| | | |
|---|---|---|
| **scanElements** | true | Θα πρέπει να έχει τιμή true ή false. Ενεργοποιεί τη διαδικασία ανίχνευσης του ειδικού HTML attribute data-resource-url. Με αυτό τον τρόπο ενεργοποιείται η λήψη των δεδομένων μέσα από το δίκτυο των ομότιμων χρηστών. Υπάρχει η δυνατότητα απενεργοποίησης της διαδικασίας αυτής στην περίπτωση που ο διαχειριστής του ιστοτόπου επιθυμεί να κάνει αποκλειστικά χρήση της διεπαφής προγραμματισμού (API) που παρέχει η εφαρμογή μας. Στην περίπτωση αυτή, ο σχετικός κώδικας αφαιρείται από το σενάριο που τρέχει στον φυλλομετρητή για εξοικονόμηση πόρων. |
| **exportName** | “paws” | Δίνει τη δυνατότητα να οριστεί συγκεκριμένο όνομα για το αντικείμενο με το οποίο παρέχεται πρόσβαση στο API της εφαρμογής μας. Είναι χρήσιμη στην περίπτωση που ο διαχειριστής του ιστοτόπου επιθυμεί να κάνει χρήση της διεπαφής προγραμματισμού (API) που παρέχει η εφαρμογή μας και υπάρχει ενδεχόμενο να προκύψει σύγκρουση ονομάτων στο global namespace του παραθύρου του φυλλομετρητή (window). |
| **bundleName** | “paws.js” | Το όνομα του αρχείου JavaScript το οποίο δημιουργείται δυναμικά κατά την αρχικοποίησή του server. Η πλήρης διεύθυνση του αρχείου προκύπτει με συνένωση του «domain», του «mountPoint.urlPath» και του «bundleName». |

| | | |
|---|---|---|
| **timeoutTimeMs** | 5000 | Το χρονικό διάστημα, κατά τη διάρκεια του οποίου αν δεν επιτευχθεί η λήψη κάποιου τμήματος, η λήψη μέσω ομότιμων χρηστών ακυρώνεται. |
| **chunkSize** | 65536 | Το μέγεθος των τμημάτων σε bytes στα οποία σπάει ένα αρχείο με στόχο να επιτευχθεί η όσο το δυνατό καλύτερη παραλληλοποίηση της λήψης του από πολλούς ομότιμους χρήστες. |
| **maxPeers** | 5 | Ο μέγιστος αριθμός ομότιμων χρηστών από τους οποίους λαμβάνει ένα αρχείο. Ο αριθμός είναι ανά αρχείο. |
| **minPeers** | 2 | Ο ελάχιστος αριθμός ομότιμων χρηστών που είναι απαραίτητος προκειμένου ο server να προτείνει τη λήψη του με ομότιμο τρόπο. |
| **iceServers** | [{ urls: [ 'stun:stun1.l.google.com:19302', 'stun:stun2.l.google.com:19302' ]} ] | Πίνακας json με RTCIceServers όπως ορίζεται στο https://www.w3.org/TR/webrtc/#rtciceserver-dictionary. |
| **asn** | false | Ενεργοποιεί τον έλεγχο του αριθμού AS, ώστε να δίνεται προτεραιότητα στη δημιουργία Intra-AS συνδέσεων. |

### *4.2.3 Δυναμική δημιουργία σεναρίου με τη χρήση του πακέτου rollup*

Η δυναμική δημιουργία του σεναρίου JavaScript έχει τριπλή χρησιμότητα. Πρώτον, μας επιτρέπει να δίνουμε στον διαχειριστή του ιστοτόπου τη δυνατότητα να επιλέξει το όνομα με το οποίο η διεπαφή μας θα είναι διαθέσιμη στην πλευρά του πελάτη. Δεύτερον, μας επιτρέπει να περάσουμε διάφορες απαραίτητες μεταβλητές που περιγράφουν τον server, ώστε να μην χρειάζεται να επέμβει ο διαχειριστής του ιστοτόπου στον πηγαίο κώδικα του σεναρίου. Τέλος, μας δίνει τη δυνατότητα να απορρίπτουμε μέρος του κώδικα (tree shake), εφόσον ο διαχειριστής δεν επιθυμεί κάποιες λειτουργίες.

Το σενάριο θα διανέμεται αυτόματα από τον εξυπηρετητή μας σε διεύθυνση που προκύπτει από το αρχείο ρυθμίσεων. Για τη χρήση του σεναρίου από τον φυλλομετρητή, απαιτείται η προσθήκη της ετικέτας script στις σελίδες HTML.

### *4.2.4 Εκκίνηση HTTP(S) server*

Ανάλογα με τις απαιτήσεις του διαχειριστή, το PAWS μπορεί να ρυθμιστεί προκειμένου να δημιουργήσει ένα HTTP server, έναν HTTPS server ή έναν HTTP και έναν HTTPS server. Στην τελευταία περίπτωση, ο μη ασφαλής εξυπηρετητής θα ανακατευθύνει τα αιτήματα που λαμβάνει στον ασφαλή. Οι servers δημιουργούνται με τη χρήση του middlware «static» που έρχεται με την εγκατάσταση του express framework. Παράλληλα, οι servers ρυθμίζονται έτσι ώστε να περιμένουν αιτήματα για αναβάθμιση της σύνδεσης σε WebSockets σε συγκεκριμένο endpoint.

### *4.2.5 Εκκίνηση WebSocket server*

Καθώς οι πελάτες μεταβαίνουν σε μία ιστοσελίδα, ανοίγουν μία σύνδεση WebScoket με το PAWS, η οποία χρησιμοποιείται προκειμένου:

- να γνωρίζουμε ποιοι πελάτες βρίσκονται στην ιστοσελίδα μας κάθε στιγμή και να παρακολουθούμε τα αρχεία τα οποία διαθέτουν,
- να διευκολύνει την ομότιμη σύνδεση δύο πελατών λειτουργώντας ως κανάλι σηματοδότησης για το WebRTC.

Για να ταυτοποιούμε τους πελάτες, αναθέτουμε σε καθένα μία αναγνωριστική τιμή, το `peerId`, που είναι μία τυχαία αλλά μοναδική τιμή που λαμβάνουμε με χρήση του πακέτου «uuid». Αν ο πελάτης μάς επισκέπτεται για πρώτη φορά, δε θα έχει `peerId`, οπότε φροντίζουμε να του αναθέσουμε ένα, το οποίο αποθηκεύεται στο LocalStorage

του και μας το αναφέρει σε κάθε μας συνδιαλλαγή (σαν να ήταν cookie). Αν ο πελάτης έχει ήδη `peerId`, φροντίζει να μας ενημερώσει σχετικά, μέσω παραμέτρου στο URL που θα γίνει η WebSocket σύνδεση. Για παράδειγμα, ένας πελάτης που δεν έχει `peerId`, θα συνδεθεί στο endpoint `ws://example.com/ws-paws` ενώ κάποιος που έχει από παλαιότερη μας συνδιαλλαγή θα συνδεθεί στο endpoint `ws://example.com/ws-paws?peerId=1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed`. Με αυτό τον τρόπο αποφεύγουμε ένα round trip.

Ο WebSocket server είναι ρυθμισμένος να διαχειρίζεται τα εξής μηνύματα:

- FILE_REQUEST: ο πελάτης αιτείται ένα αρχείο. Εφόσον το αρχείο που ζητείται υπάρχει και ανήκει στη δικαιοδοσία του PAWS, αναζητούνται άλλοι χρήστες που έχουν το αρχείο μέσα στον κατάλογο που διατηρούμε. Αν δεν βρεθούν όσοι χρήστες απαιτούνται, ο server απαντά με σφάλμα και προτρέπει τον χρήστη να δοκιμάσει να αιτηθεί το αρχείο με τον παραδοσιακό τρόπο (μέσω `fetch()`). Αν βρεθούν υποψήφιοι ομότιμοι χρήστες, τότε ενημερώνει τον πελάτη για τις απαραίτητες πληροφορίες του αρχείου όπως το μέγεθός του και τον τύπο του, και παράλληλα, στέλνει εντολή στους χρήστες που βρήκε προκειμένου να δημιουργήσουν WebRTC κανάλι δεδομένων με αυτόν.
- FILE_RESPONSE: σε περίπτωση που ένας από τους ομότιμους χρήστες δε διαθέτει το αρχείο, γιατί για παράδειγμα διαγράφτηκε από τον φυλλομετρητή, θα ενημερώσει με μήνυμα FILE_RESPONSE που θα περιέχει σφάλμα (μη διαθέσιμο αρχείο).
- GOT_FILE: όταν η λήψη και αποθήκευση ενός αρχείου τελειώσει επιτυχώς, ο φυλλομετρητής θα ενημερώσει τον εξυπηρετητή ότι πλέον διαθέτει το αρχείο για να τον προσθέσει στον αντίστοιχο κατάλογο.
- DELETED_FILE: αν ο χειριστής της διεπαφής του PAWS επιθυμεί να διαγράψει ένα αρχείο από τη μνήμη, μπορεί να καλέσει τη σχετική μέθοδο που θα ενημερώσει τον εξυπηρετητή με ένα μήνυμα αυτού του τύπου.
- OFFER: ένα αρχείο SDP τύπου offer προκειμένου να πραγματοποιηθεί η ομότιμη σύνδεση μεταξύ δύο φυλλομετρητών. Το offer στέλνεται από τον κάτοχο του αρχείου σε αυτόν που το ζητάει.

- ANSWER: αντίστοιχα, πρόκειται για SDP αρχείο τύπου answer, από τον χρήστη που ζητά το αρχείο σε έναν κάτοχο αυτού.

Καθώς οι πελάτες επικοινωνούν με τον εξυπηρετητή, αυτός καταγράφει ποιοι από αυτούς είναι συνδεδεμένοι και τι αρχεία έχουν στην κατοχή τους. Όταν ο πελάτης φύγει από την σελίδα, η μνήμη Cache του φυλλομετρητή του δεν αδειάζει. Για το λόγο αυτό επιλέγουμε να μην διαγράφουμε τη λίστα των αρχείων του κατά την έξοδό του. Έτσι την επόμενη φορά που θα μας επισκεφθεί, θα γνωρίζουμε χάρη στο `peerId` του, ποια αρχεία έχει ήδη.

Προκειμένου να προτείνουμε peers για ένα αρχείο, για κάθε συνδεδεμένο πελάτη γίνεται αναζήτηση στα αρχεία που διαθέτει. Η σειρά με την οποία ελέγχονται οι πελάτες είναι ίδια με τη σειρά σύνδεσής τους. Αυτό πρακτικά σημαίνει ότι όσο περισσότερη ώρα παραμένει ένας πελάτης συνδεδεμένος, τόσο πιο πιθανό είναι να προταθεί. Το σύστημα αυτό δεν είναι δίκαιο, όμως πιστεύουμε ότι βοηθάει να μειωθούν τα προβλήματα από τη συχνή φυγή των ομότιμων χρηστών (churn), αφού όσο μεγαλώνει η διάρκεια της σύνδεσης, τόσο πιο απίθανο είναι ο χρήστης να αποχωρήσει στο επόμενο διάστημα.

Επειδή ο κατάλογος των αρχείων χρησιμοποιεί τη δομή δεδομένων `Set` της JavaScript, η αναζήτηση στα αρχεία που διαθέτει κάθε ομότιμος χρήστης γίνεται ιδιαιτέρως γρήγορα (O(1) πολυπλοκότητα). Η αναζήτηση σταματά μόλις βρούμε τον αριθμό των peers που χρειαζόμαστε σύμφωνα με το αρχείο ρυθμίσεων.

Παρακάτω παραθέτουμε τον πηγαίο κώδικα του αντικειμένου tracker, το οποίο αναλαμβάνει την παρακολούθηση των ομότιμων χρηστών και την αναζήτηση σε αυτούς.

**Απόσπασμα πηγαίου κώδικα 1: Υλοποίηση Tracker.**

```
1.  // Αρχείο tracker.js.
2.  // Για κάθε peerId, αποθηκεύουμε ένα Set με τα URLs που διαθέτει ο πελάτης.
3.  const trackedUserFiles = new Map();
4.  // Για κάθε peerId, αποθηκεύουμε ένα αντικείμενο με την κατάστασή του και
5.  // προαιρετικά το νούμερο του AS που ανήκει.
6.  const trackedUsersMetadata = new Map();
7.  // Set με όλους τους συνδεδεμένους χρήστες. Διατηρεί σειρά εισαγωγής.
8.  const connectedUsersId = new Set();
9.
10. // Καταγράφει ότι ένας χρήστης διαθέτει ένα συγκεκριμένο URL.
11. exports.setUserHasFile = (userId, fileUrl) => {
12.   if (!trackedUserFiles.has(userId)) {
13.     trackedUserFiles.set(userId, new Set());
14.   }
15.   trackedUserFiles.get(userId).add(fileUrl);
16. };
17.
```

```
18. //  Διαγράφει ένα URL από τα αρχεία που διαθέτει ο χρήστης.
19. exports.unsetUserHasFile = (userId, fileUrl) => {
20.   trackedUserFiles.get(userId).delete(fileUrl);
21. };
22.
23. // Ενημερώνει τα μεταδεδομένα του ομότιμου χρήστη, προσθέτοντας ή
24. // διαγράφοντάς τον από την λίστα των συνδεδεμένων. Το αντικείμενο
25. // options διαθέτει το πεδίο connected και προαιρετικά το asn.
26. exports.updateUserInfo = (userId, options) => {
27.   const currentRecord = trackedUsersMetadata.get(userId) || {};
28.   if (options.connected) {
29.     console.log('adding user ' + userId + ' to connected peers.');
30.     connectedUsersId.add(userId);
31.   } else {
32.     console.log('deleting user ' + userId + ' from connected peers.');
33.     connectedUsersId.delete(userId);
34.   }
35.   const newRecord = Object.assign(currentRecord, options);
36.   trackedUsersMetadata.set(userId, newRecord);
37. };
38.
39. // Εύρεση των peerIds των χρηστών που έχουν συγκεκριμένο URL. Το
40. // όρισμα howMany δηλώνει πόσους χρήστες θέλουμε. Το όρισμα asn
41. // δηλώνει αν πρέπει να λάβουμε υπόψη το νούμερο AS των χρηστών.
42. exports.getUsersForUrl = (fileUrl, myId, howMany = 5, asn) => {
43.   // Ελέγχουμε τους χρήστες, με τη σειρά που συνδέθηκαν.
44.   const connectedUsersIdArray = Array.from(connectedUsersId);
45.   // Αναζήτηση αν δεν απαιτείται έλεγχος AS.
46.   if (!asn) {
47.     const usersThatHaveFile = [];
48.     for (let i = 0; i < connectedUsersIdArray.length; i++) {
49.       const connectedUserId = connectedUsersIdArray[i];
50.       if (connectedUserId !== myId) {
51.         const userFiles = trackedUserFiles.get(connectedUserId);
52.         if (userFiles && userFiles.has(fileUrl)) {
53.           usersThatHaveFile.push(connectedUserId);
54.           if (usersThatHaveFile.length >= howMany) {
55.             // Βρήκαμε τον απαιτούμενο αριθμό. Γρήγορη έξοδος.
56.             break;
57.           }
58.         }
59.       }
60.     }
61.     return usersThatHaveFile;
62.   }
63.   // Αναζήτηση αν απαιτείται έλεγχος AS.
64.   const usersThatHaveFileInSameAsn = [];
65.   const usersThatHaveFileInOtherAsn = [];
66.
67.   let done = false;
68.   for (let i = 0; i < connectedUsersIdArray.length; i++) {
69.     const connectedUserId = connectedUsersIdArray[i];
70.     if (connectedUserId !== myId) {
71.       const userFiles = trackedUserFiles.get(connectedUserId);
72.       if (userFiles && userFiles.has(fileUrl)) {
73.         if (trackedUsersMetadata.get(connectedUserId).asn === asn) {
74.           usersThatHaveFileInSameAsn.push(connectedUserId);
75.         } else {
76.           usersThatHaveFileInOtherAsn.push(connectedUserId);
77.         }
78.       }
79.       if (usersThatHaveFileInSameAsn.length >= howMany) {
80.         // Γρήγορη έξοδος.
81.         done = true;
82.         break;
```

```
      }
    }
  }
  if (done) {
    return usersThatHaveFileInSameAsn;
  }
  const missing = howMany - usersThatHaveFileInSameAsn.length;
  return usersThatHaveFileInSameAsn.concat(usersThatHaveFileInOtherAsn.slice(0,
 missing));
};

// Εφόσον τα δεδομένα των χρηστών δε διαγράφονται με τη φυγή του,
// η μέθοδος αυτή καλείται περιοδικά για την απελευθέρωση μνήμης.
exports.clearTrackedUserFiles = () => {
  const userIds = Array.from(trackedUserFiles.keys());
  for (let i = 0; i < userIds.length; i++) {
    const userId = userIds[i];
    if (!connectedUsersId.has(userId)) {
            trackedUserFiles.delete(userId);
            trackedUsersMetadata.delete(userId);
          }
        }
      };
```

## 4.3 Περιγραφή υλοποίησης στην πλευρά του πελάτη

Στην ενότητα αυτή περιγράφουμε τις ευθύνες και τον τρόπο λειτουργίας του PAWS από την πλευρά του πελάτη.

Η λειτουργικότητα του PAWS υλοποιείται στην ομώνυμη κλάση. Πρώτο καθήκον του κατασκευαστή της κλάσης δεν είναι άλλο από τον έλεγχο του φυλλομετρητή ώστε να διαπιστωθεί αν αυτός υποστηρίζει τις απαραίτητες τεχνολογίες. Τον έλεγχο αναλαμβάνει η συνάρτηση `isClientSupported()` που υλοποιείται ως εξής:

**Απόσπασμα πηγαίου κώδικα 2: Συνάρτηση isClientSupported().**

```
const isClientSupported = () => {
  return (
    'WebSocket' in window &&
    'RTCPeerConnection' in window &&
    'RTCDataChannel' in window &&
    'localStorage' in window &&
    'caches' in window &&
    'Request' in window &&
    'Response' in window &&
    'TextEncoder' in window &&
    'TextDecoder' in window
  );
};
```

Στη συνέχεια, επιδιώκεται η δημιουργία του καναλιού επικοινωνίας με τον εξυπηρετητή. Για κανάλι επικοινωνίας χρησιμοποιήσαμε τη δική μας κλάση

`SignalChannel`, η οποία διαχειρίζεται μία σύνδεση `WebSocket`. Αποφύγαμε να χρησιμοποιήσουμε απευθείας ένα αντικείμενο `WebSocket`, αφού ο τρόπος χρήσης του δεν ταίριαζε στην περίπτωσή μας. Συγκεκριμένα, το `SignalChannel` που δημιουργήσαμε προσθέτει πάνω από τη διεπαφή των WebSockets:

- δυνατότητα για προσωρινή αποθήκευση μηνυμάτων αν η σύνδεση δεν έχει ανοίξει και μαζική αποστολή τους σε δεύτερο χρόνο,
- δυνατότητα λήψης απάντησης σε συγκεκριμένο μήνυμα. Για να επιτευχθεί αυτό, προσθέτει ένα αναγνωριστικό `messageId` στο μήνυμα και περιμένει απάντηση με το ίδιο `messageId`,
- αυτόματη κωδικοποίηση μηνυμάτων σε JSON,
- μία αυτόματη προσπάθεια επανασύνδεσης σε περίπτωση που κλείσει η υποκείμενη σύνδεση WebSocket.

Το σημαντικότερο τμήμα του πηγαίου κώδικα της κλάσης `SignalChannel` παρατίθεται παρακάτω:

**Απόσπασμα πηγαίου κώδικα 3: Απόσπασμα κλάσης SignalChannel.**

```
1.  // Απόσπασμα από την κλάση SignalChannel.
2.  // Τοποθετεί ένα μήνυμα σε στοίβα και εφόσον η σύνδεση
3.  // είναι ανοιχτή, στέλνει όλα τα μηνύματα που βρίσκει εκεί.
4.  send(message) {
5.    if (this.isDead) {
6.      console.error("Can't send message. WebSocket is dead.");
7.      return;
8.    }
9.
10.   if (message) {
11.     this.messageQueue.push(message);
12.   }
13.
14.   if (this.webSocket.readyState === this.webSocket.OPEN) {
15.     for (let i = 0; i < this.messageQueue.length; i++) {
16.       const currentMessageJsonString = JSON.stringify(
17.         this.messageQueue.shift()
18.         );
19.       this.webSocket.send(currentMessageJsonString);
20.     }
21.   }
22. }
23.
24. // Χρησιμοποιεί το Promises API και μας δίνει τη
25. // δυνατότητα να περιμένουμε απάντηση στο μήνυμά μας.
26. sendForReply(message) {
27.   return new Promise((resolve, reject) => {
28.     const messageId = Date.now();
29.     message.msgId = messageId;
30.     const that = this;
```

```
31.     const handle = function handle({ detail: { data } }) {
32.       resolve(data);
33.       that.removeEventListener(messageId, handle);
34.     };
35.     this.addEventListener(messageId, handle);
36.     this.send(message);
37.   });
38. }
39.
40. // Χειρισμός εισερχόμενων μηνυμάτων.
41. onWebSocketMessage(message) {
42.   let data;
43.   try {
44.     data = JSON.parse(message.data);
45.   } catch (err) {
46.     console.error("Received message can't be parsed. Error: %o", err);
47.     return;
48.   }
49.   if (data.msgId !== undefined) {
50.     this.dispatchEvent(new CustomEvent(data.msgId, { detail: { data } }));
51.   } else {
52.     this.dispatchEvent(new CustomEvent('message', { detail: { data } }));
53.   }
54. }
55.
56. // Προσπάθεια επανασύνδεσης σε περίπτωση σφάλματος.
57. onWebSocketClose(e) {
58.   if (e.code === 1000) {
59.     console.log('WebSocket closed normally.');
60.     return;
61.   }
62.   if (this.retries > 0) {
63.     this.retries -= 1;
64.     this.webSocket = this.createWebSocket();
65.     return;
66.   }
67.   this.isDead = true;
68. }
```

Αφού δημιουργηθεί ένα στιγμιότυπο του `SignalChannel`, εγκαθίσταται ο απαραίτητος χειριστής γεγονότων για τα μηνύματα που στέλνει ο εξυπηρετητής. Αυτά είναι ένα από τα παρακάτω:

- SET_PEER_ID: αν ο πελάτης δε διαθέτει `peerId`, τότε ο εξυπηρετητής θα επιλέξει ένα και θα του το στείλει.
- FILE_REQUEST: αποτελεί αίτημα που έχει στείλει άλλος χρήστης για ένα αρχείο. Αν ο παρών χρήστης διαθέτει πρόσφατη έκδοση του αρχείου, χρησιμοποιεί ένα αντικείμενο `PeerManager` προκειμένου να λάβει ένα αντικείμενο `Peer` και να δημιουργήσει ένα `RTCDataChannel`, μέσω του οποίου θα στείλει τμήματα του αρχείου.

- ANSWER: ένας χρήστης έστειλε μέσω του εξυπηρετητή μήνυμα SDP τύπου answer. Τέτοιου είδους μηνύματα λαμβάνονται από χρήστες που αιτούνται αρχεία, αφού προηγουμένως έχουμε στείλει offer.
- OFFER: ένας χρήστης έστειλε μέσω του εξυπηρετητή μήνυμα SDP τύπου offer. Τέτοιου είδους μηνύματα λαμβάνονται από χρήστες που προσφέρουν αρχεία.

Πιο πάνω αναφερθήκαμε σε αντικείμενα τύπου `PeerManager` και `Peer`. Η κλάση `Peer` διαχειρίζεται μία σύνδεση `RTCPeerConnection` με έναν άλλο ομότιμο χρήστη. Αναλαμβάνει τη διαχείριση του κύκλου ζωής της, από την επίτευξή μέχρι το κλείσιμό της. Παρέχει μέθοδο δημιουργίας `RTCDataChannel`, προκειμένου να γίνει η αποστολή των τμημάτων ενός αρχείου και εγκαθιστά τον απαραίτητο χειριστή μηνυμάτων. Ο χρήστης που θέλει να λάβει ένα αρχείο, στέλνει ένα αίτημα μέσω του καναλιού αναφέροντας το αρχείο και τον αριθμό του τμήματος που ζητά. Ο χειριστής μηνυμάτων, αφού λάβει το αίτημα, ψάχνει στη μνήμη του και εφόσον διαθέτει το αρχείο, στέλνει το τμήμα που του ζητήθηκε.

Ο `PeerManager` αναλαμβάνει τη δημιουργία των αντικειμένων `Peer` και τη διατήρησή τους. Επιτρέπει την επαναχρησιμοποίηση μιας υπάρχουσας ομότιμης σύνδεσης. Παρακάτω παρατίθεται ενδεικτικά, ο χειριστής μηνυμάτων που εξυπηρετεί τα αιτήματα για τμήματα αρχείων.

**Απόσπασμα πηγαίου κώδικα 4: Απόσπασμα κλάσης Peer.**

```
1.  // Απόσπασμα από την κλάση Peer.
2.  handleMessage(message, dataChannel) {
3.    // Διαβάζουμε το αίτημα του ομότιμου χρήστη.
4.    const {url, chunkIndex, chunkSize} = parseArrayBufferMessage(message.data);
5.    caches
6.      .open(cacheConstants.CACHE_NAME)
7.      .then(cache => {
8.        // Αναζήτηση στη μνήμη.
9.        return cache.match(url);
10.     })
11.     .then(res => {
12.       if (res) {
13.         return res.arrayBuffer();
14.       }
15.       throw new Error("File not found in peer's cache");
16.     })
17.     .then(buffer => {
18.       // Αν βρεθεί το αρχείο στη μνήμη, στέλνουμε το τμήμα που ζητείται.
19.       return dataChannel.send(
20.         buffer.slice(chunkIndex * chunkSize, (chunkIndex + 1) * chunkSize)
21.       );
```

```
22.     })
23.     .catch(err => {
24.       // Αν δεν βρεθεί, στέλνουμε σφάλμα και κλείνουμε το data channel.
25.       // Ένα data channel χρησιμοποιείται αποκλειστικά για ένα αρχείο,
26.       // οπότε αν δεν διαθέτουμε το αρχείο, μπορούμε να το κλείσουμε.
27.       dataChannel.send(makeArrayBufferMessage({ error: err.message }));
28.       dataChannel.close();
29.     });
30. }
```

Είδαμε πώς απαντά ένας χρήστης σε ένα αίτημα. Πώς όμως στέλνει ο ίδιος τα δικά του αιτήματα; Η δυνατότητα αυτή παρέχεται μέσω της μεθόδου `fetchFromPeers()` που ορίζεται στην κλάση `Paws`. Η διαδικασία έχει ως:

1. Αρχικά στέλνουμε μέσω του `SignalChannel` αίτημα για ένα αρχείο και περιμένουμε απάντηση από τον εξυπηρετητή. Ο εξυπηρετητής θα αναζητήσει άλλους χρήστες που διαθέτουν το αρχείο και θα τους προτρέψει να δημιουργήσουν ένα κανάλι δεδομένων μαζί μας.
2. Μόλις λάβουμε την απάντηση, αν έχει προκύψει κάποιο σφάλμα, για παράδειγμα δεν υπάρχουν άλλοι ομότιμοι χρήστες με το αρχείο, αιτούμαστε ξανά το αρχείο με τη χρήση της συνάρτησης `fetch()` που διαθέτουν οι φυλλομετρητές, αποθηκεύουμε το αρχείο στη μνήμη και ενημερώνουμε τον φυλλομετρητή ώστε να μπορεί να μας προτείνει σε άλλους χρήστες που θα θελήσουν το αρχείο.
3. Αν δεν υπάρχει σφάλμα, η απάντηση θα περιέχει βασικές πληροφορίες για το αρχείο που αιτηθήκαμε όπως είναι το μέγεθός του, ο τύπος του (mimetype) και ο χρόνος της τελευταίας τροποποίησής του. Με αυτές τις πληροφορίες σχηματίζουμε ένα αντικείμενο `Response`.
4. Σε αυτό το σημείο, ο `PeerManager` μας ενημερώνει ότι νέα κανάλια δεδομένων έχουν ανοίξει (βήμα 1). Τα κανάλια αυτά, τα προσθέτουμε σε ένα αντικείμενο τύπου `DownloadQueue`, προκειμένου να τα διαχειριστεί για τη λήψη του αρχείου.
5. Παράλληλα, για κάθε τμήμα του αρχείου, δημιουργείται ένα αντικείμενο `ChunkRequest` προκειμένου να διαχειριστεί τη λήψη του τμήματος. Και αυτά τα αντικείμενα στο `DownloadQueue`.
6. Η `DownloadQueue` αρχίζει τη λήψη των τμημάτων. Για κάθε αντικείμενο `ChunkRequest` ορίζει ένα κανάλι δεδομένων το οποίο θα αναλάβει τη μεταφορά του τμήματος. Καθώς τα διάφορα τμήματα καταφθάνουν, ένας χειριστής γεγονότων φροντίζει να τα προσθέτει στο σώμα του `Response`.

7. Το αντικείμενο `Response` επιστρέφει ως αποτέλεσμα κλήσης της μεθόδου `fetchFromPeers()`.

Στη συνέχεια παραθέτουμε απόσπασμα της μεθόδου `download()` του `DownloadQueue`, που καλείται για την έναρξη της ομότιμης λήψης. Αμέσως μετά, ακολουθεί ο πηγαίος κώδικας του αρχείου chunk-request.js στο οποίο ορίζεται η κλάση `ChunkRequest`.

**Απόσπασμα πηγαίου κώδικα 5: Η μέθοδος download().**

```
1.  // Μέθοδος download της κλάσης DonwloadQueue.
2.  // Η μέθοδος καλείται αναδρομικά.
3.  download() {
4.    if (!this.lastReceived) {
5.      this.lastReceived = Date.now();
6.    }
7.
8.    // Κατέβηκαν όλα τα τμήματα του αρχείου;
9.    if (this.completedChunks.size === this.numberOfChunks) {
10.     const buffers = new Array(this.completedChunks.size);
11.     this.completedChunks.forEach((value, key) => {
12.       buffers[key] = value;
13.     });
14.     this.dispatchEvent(new CustomEvent('complete', { detail: { buffers } }));
15.     clearTimeout(this.timeoutId);
16.     return;
17.   }
18.
19.   // Πόσος χρόνος πέρασε από τη τελευταία επιτυχή λήψη ενός τμήματος;
20.   if (Date.now() - this.lastReceived > this.msBeforeError) {
21.     this.dispatchEvent(
22.       new CustomEvent('error', { detail: {
23.         error: `${this.msBeforeError}ms passed without receiving a chunk.`
24.       } })
25.     );
26.     clearTimeout(this.timeoutId);
27.   }
28.
29.   // Κάνω λήψη με full concurrency;
30.   if (this.downloadingChunks.length === this.concurrency) {
31.     this.timeoutId = setTimeout(() => this.download(), 10);
32.     return;
33.   }
34.
35.   // Υπάρχουν αδρανή κανάλια;
36.   if (this.idleChannels.length === 0) {
37.     this.timeoutId = setTimeout(() => this.download(), 10);
38.     return;
39.   }
40.
41.   // Όσο υπάρχουν αδρανή κανάλια, ανάθεσέ τους τη λήψη ενός τμήματος.
42.   while (this.idleChannels.length > 0 && this.chunkRequests.length > 0) {
43.     const chunkRequest = this.chunkRequests.shift();
44.     const dataChannel = this.idleChannels.pop();
45.     this.busyChannels.push(dataChannel);
46.
```

```
    chunkRequest.onComplete = ({ index, data }) => {
      this.lastReceived = Date.now();
      this.completedChunks.set(index, data);
      this.idleChannels.push(dataChannel);
      removeObjectFromArray(this.busyChannels, dataChannel);
      removeObjectFromArray(this.downloadingChunks, chunkRequest);
      this.flushCompleted();
      this.timeoutId = setTimeout(() => this.download(), 10);
    };
    // Αποτυχία λήψης τμήματος, τοποθέτησέ το πάλι στο πίνακα για λήψη.
    chunkRequest.onError = () => {
      removeObjectFromArray(this.busyChannels, dataChannel);
      this.chunkRequests.unshift(chunkRequest);
    };
    chunkRequest.onTimeout = () => {
      removeObjectFromArray(this.busyChannels, dataChannel);
      this.chunkRequests.unshift(chunkRequest);
    };
    this.downloadingChunks.push(chunkRequest);
    chunkRequest.sendWith(dataChannel);
  }
}
```

**Απόσπασμα πηγαίου κώδικα 6: Υλοποίηση ChunkRequest.**

```
// Αρχείο chunk-request.js
import { makeArrayBufferMessage, isErrorMessage } from './utils';
export class ChunkRequest {
  constructor(url, index, chunkSize, timeoutMs) {
    this.url = url;
    this.index = index;
    this.size = chunkSize;
    this.timeoutMs = timeoutMs;
    this.data = null;
    this.onTimeout;
    this.onError;
    this.onComplete;
  }

  // Ορισμός του καναλιού δεδομένων με το οποίο θα γίνει η λήψη.
  sendWith(dataChannel) {
    const timeoutId = setTimeout(() => {
      this.onTimeout && this.onTimeout();
    }, this.timeoutMs);

    dataChannel.onerror = err => {
      console.warn('DataChannel error', err);
      clearTimeout(timeoutId);
      this.onError && this.onError(err);
    };
    // Λήψη τμήματος
    dataChannel.onmessage = ({ data }) => {
      clearTimeout(timeoutId);
      if (isErrorMessage(data)) {
        this.onError && this.onError();
      } else {
        this.data = data;
        this.onComplete && this.onComplete({ index: this.index, data });
      }
    };
    // Δημιουργία του αιτήματος.
    const requestMessage = makeArrayBufferMessage(
```

```
      { url: this.url, chunkIndex: this.index, chunkSize: this.size }
    );
    dataChannel.send(requestMessage);
  }
}

// Δημιουργία ChunkRequest για κάθε τμήμα αρχείου.
export const createChunkRequests =
  (request, fileSize, chunkSize = 65536, timeoutMs = 10000) => {
    const chunkCount = Math.ceil(fileSize / chunkSize);
    const chunkRequests = [];
    for (let i = 0; i < chunkCount; i++) {
      chunkRequests.push(
        new ChunkRequest(request.url, i, chunkSize, timeoutMs)
      );
    }
    return chunkRequests;
  };
```

## 4.4 Τρόπος χρήσης του PAWS

Η χρήση του PAWS μπορεί να γίνει με δύο τρόπους. Ο πρώτος τρόπος ελέγχει την HTML του εγγράφου προκειμένου να βρει αρχεία που απαιτούν λήψη με ομότιμο τρόπο. Τα αρχεία αυτά ορίζονται με τη χρήση της «`data-resource-url`» ιδιότητας σε μια ετικέτα `<img>`, `<link>` ή `<script>`. Η ιδιότητα αυτή πρέπει να έχει ως τιμή τη διεύθυνση URL του στοιχείου. Σε περίπτωση που ο φυλλομετρητής δεν υποστηρίζει τις απαραίτητες τεχνολογίες, τα αρχεία φορτώνονται με το παραδοσιακό μοντέλο πελάτη-εξυπηρετητή.

Ο δεύτερος τρόπος περιλαμβάνει την αξιοποίηση της διεπαφής προγραμματισμού που παρέχει στην πλευρά του φυλλομετρητή. Η απευθείας χρήση της διεπαφής προσφέρει μεγαλύτερη ευελιξία στον διαχειριστή της ιστοσελίδας. Διαθέσιμες μέθοδοι που μπορούν να αξιοποιηθούν είναι:

- `Paws.fetchFromPeers(request, fallbackToCache)`: η μέθοδος επιστρέφει ένα αντικείμενο `Response`. Η λήψη θα πραγματοποιηθεί από άλλους χρήστες, εφόσον είναι δυνατόν. Η απόκριση θα αποθηκευτεί στην Cache.
- `Paws.fetchAndCache(request)`: η μέθοδος επιτρέπει την κανονική λήψη του αρχείου από τον εξυπηρετητή και την τοποθέτησή τους στη μνήμη του PAWS, προκειμένου να μπορεί να διαμοιραστεί σε άλλους πελάτες.
- `Paws.fetch(request, useCached)`: η μέθοδος επιτρέπει την κανονική λήψη του αρχείου, εφόσον δεν υπάρχει στη μνήμη του PAWS.

Ένα παράδειγμα των τρόπων χρήσης, παρουσιάζεται στο Παράρτημα Α.

## 4.5 Σχολιασμός και αντιπαράθεση με άλλα υλοποιήσεις

Στο Κεφάλαιο 2 πραγματοποιήσαμε μία σύντομη παρουσίαση βασικών στοιχείων της θεωρίας των ομότιμων δικτύων και των δικτύων διανομής περιεχομένου και αναφέραμε άλλες προσπάθειες συνδυασμού των δύο τεχνολογιών με εφαρμογή στον παγκόσμιο ιστό. Σε αυτή την ενότητα, θα περιγράψουμε την υλοποίησή μας βάσει αυτών.

### *4.5.1 Το PAWS ως ομότιμο σύστημα*

Όπως αναφέραμε τα κατανεμημένα συστήματα, διακρίνονται σε κεντρικοποιημένα, μη κεντρικοποιημένα και υβριδικά. Αν κατατάσσαμε το PAWS, θα λέγαμε ότι είναι ένα μερικώς κατανεμημένο σύστημα που ακολουθεί την υβριδική αρχιτεκτονική και στο οποίο ο εξυπηρετητής αναλαμβάνει αυξημένες αρμοδιότητες έναντι των άλλων ομότιμων χρηστών. Μερικές από τις λειτουργίες που αναλαμβάνει είναι η λειτουργία της ένταξης νέων χρηστών στο σύστημα και της ευρετηρίασης του περιεχομένου σε αυτούς. Εφόσον το σύστημα μας είναι ομότιμο και χρησιμοποιεί κεντρικό εξυπηρετητή, μπορούμε να πούμε ότι με βάση τη δομή του δικτύου επικάλυψης και το βαθμό αποκέντρωσης, το σύστημά μας μπορεί να χαρακτηριστεί ως μη δομημένο και υβριδικά αποκεντρωμένο.

Τέτοιου είδους συστήματα συχνά εμφανίζουν μοναδικό σημείο αποτυχίας. Στην περίπτωσή μας, η ανάγκη για κεντρικό εξυπηρετητή πηγάζει από τον τρόπο λειτουργίας του WebRTC. Το Maygh [40] προτείνει τη χρήση μίας ομάδας κεντρικών εξυπηρετητών προκειμένου να αυξήσει την αξιοπιστία του συστήματος. Όμως με αυτόν τον τρόπο αυξάνεται εκθετικά και η περιπλοκότητα του συστήματος, αφού οι εξυπηρετητές θα πρέπει να μοιράζονται κοινή βάση δεδομένων για την αποθήκευση του καταλόγου των συνδεδεμένων χρηστών. Μία λύση σε αυτή την περίπτωση θα ήταν η δημιουργία ενός επιπλέον ομότιμου δικτύου επικάλυψης, στο οποίο θα συμμετέχουν αποκλειστικά οι εξυπηρετητές – πιθανώς με την αξιοποίηση ενός DHT. Το PAWS, αν κριθεί σκόπιμο, είναι δυνατόν να εκτελεστεί σε πολλές διεργασίες. Δυστυχώς όμως, κάθε στιγμιότυπο θα

διαθέτει τους δικούς του καταλόγους χρηστών, οι οποίοι δε θα μπορούν να συνεργαστούν.

### *4.5.2 Το PAWS ως δίκτυο διανομής περιεχομένου*

Όσον αφορά τον χαρακτηρισμό του συστήματος μας ως δικτύου διανομής περιεχομένου θα λέγαμε σύμφωνα με τη θεωρία ότι, ο κεντρικός εξυπηρετητής αναλαμβάνει τη δρομολόγηση των αιτημάτων – με την έννοια ότι αυτός επιλέγει και προτείνει τους ομότιμους χρήστες από όπου θα γίνει η λήψη – ενώ οι φυλλομετρητές αναλαμβάνουν, πέρα από τον ρόλο του πελάτη, και αυτό του replica server στον οποίο τοποθετείται το περιεχόμενο. Το πρόβλημα της τοποθέτησης των replica servers δεν υφίσταται, ενώ σχετικά με το θέμα του διαμοιρασμού των αρχείων στους replica servers το τοπίο δεν είναι ξεκάθαρο. Από την μία πλευρά, μοιάζει με pull-based, αφού τα αρχεία τοποθετούνται στη μνήμη των χρηστών όταν αυτοί τα αιτούνται. Από την άλλη, μοιάζει με push-based αφού η τοποθέτηση των αντικειμένων γίνεται προκαταβολικά κι όχι όταν χρειαστούν προκειμένου να διανεμηθούν. Θα λέγαμε ότι η διανομή του περιεχομένου στους replica servers-ομότιμους χρήστες είναι απλώς ευκαιριακή.

Σχετικά με τον τύπο των δεδομένων που υποστηρίζει το σύστημά μας, σε γενικές γραμμές υποστηρίζεται κάθε είδους ψηφιακό περιεχόμενο. Αν και δεν υποστηρίζουμε εγγενώς τη λήψη και αναπαραγωγή πολυμεσικού περιεχομένου με ροή, θεωρητικά θα μπορούσε να υποστηριχθεί ένα πρωτόκολλο live streaming όπως το HLS (σε αυτό τα αρχεία πολυμέσων διασπώνται σε μικρότερα και λαμβάνονται με σειρά).

### *4.5.3 Παράλληλη λήψη*

Το σύστημά μας υποστηρίζει την παράλληλη λήψη από πολλούς χρήστες ταυτόχρονα. Η ιδέα αυτή δεν είναι πρωτότυπη, αλλά προέκυψε από τη γνωριμία μας με το πρωτόκολλο BitTorrent[1] κατά τη βιβλιογραφική μας έρευνα. Το BitTorrent επιτυγχάνει μέγιστη απόδοση στην ταχύτητα λήψης ενός αρχείου επειδή, όπως εμείς, τεμαχίζει τα δεδομένα σε μικρότερα κομμάτια, τα οποία αποστέλλονται παράλληλα. Τα συστήματα που παρουσιάσαμε στο Κεφάλαιο 2, στην πλειοψηφία τους, δεν ασχολήθηκαν με τη μεγιστοποίηση της ταχύτητας μεταφοράς. Όμως, ακόμη και σήμερα που οι ταχύτητες πρόσβασης στο διαδίκτυο είναι σαφώς γρηγορότερες από παλαιότερα έτη, το διαθέσιμο εύρος ζώνης στο uplink μιας σύνδεσης είναι εξαιρετικά περιορισμένο, κάνοντας έτσι επιτακτική την ανάγκη για δυνατότητα παράλληλης λήψης.

[1] https://www.bittorrent.com

Αυτή τη στιγμή, το PAWS διαχειρίζεται τις ομότιμες συνδέσεις σε επίπεδο αρχείου. Για παράδειγμα, η ρύθμιση `maxPeers` αναφέρεται σε ένα αρχείο και όχι συνολικά για όλα τα αρχεία που μπορεί να αποστέλλονται την ίδια στιγμή. Θα ήταν επιθυμητό στο μέλλον, να προστεθεί μια πιο ολοκληρωμένη λύση για τη διαχείριση των συνδέσεων.

### *4.5.4 Κύκλος ζωής συνδέσεων και καταλληλότερες εφαρμογές*

Στο Κεφάλαιο 3, αναφερθήκαμε στον περιορισμένο κύκλο ζωής που έχουν οι συνδέσεις WebRTC και WebSockets. Αυτό επηρεάζει σε μεγάλο βαθμό την αποδοτικότητα του PAWS, αφού αυξάνει κατακόρυφα τη φυγή των χρηστών. Ακόμη κι αν ένας χρήστης δε φύγει από έναν διαδικτυακό τόπο, αλλά απλά πλοηγηθεί σε άλλες του σελίδες, οι συνδέσεις με τους ομότιμους χρήστες θα πρέπει να κλείσουν και να ξανανοίξουν. Για το λόγο αυτό, επιλέξαμε ως κριτήριο επιλογής ομότιμων χρηστών, τον χρόνο παραμονής τους στον ιστότοπο.

Πιστεύουμε ότι αυτό θα μπορούσε να είναι ένα πρόβλημα για επίλυση σε μελλοντικές επεκτάσεις των APIs του παγκόσμιου ιστού (για παράδειγμα κάνοντας χρήση των ServiceWorkers).

Προς το παρόν, λόγω αυτού του χαρακτηριστικού, κρίνουμε ότι μεγαλύτερο όφελος από τη χρήση του συστήματός μας, θα αποκομίσουν ιστότοποι με μικρό πλήθος σελίδων. Ένα χαρακτηριστικό παράδειγμα τέτοιων σελίδων είναι οι Single Page Applications (SPAs), μια σχετικά νέα τάση στον παγκόσμιο ιστό, όπου η χρήση του ιστοτόπου δεν προκαλεί συμβάντα πλοήγησης (navigation events). Ένας χαρακτηριστικό παράδειγμα ιστοτόπου που θεωρούμε ότι θα ήταν ιδανικός για την εφαρμογή της λύσης μας είναι το giphy.com.

### *4.5.5 Λανθάνων χρόνος*

Η λήψη ενός αρχείου σε ένα ομότιμο σύστημα περιλαμβάνει πρώτα από όλα, τον εντοπισμό του στο δίκτυο. Μόνο αφού εντοπιστεί είναι δυνατόν να διανεμηθεί σε όποιον το ζήτησε. Στην πράξη αυτό σημαίνει ότι ομότιμα συστήματα διανομής περιεχομένου θα χαρακτηρίζονται πάντα από μεγαλύτερο λανθάνων χρόνο εξυπηρέτησης. Για αυτό το λόγο, θα προτείναμε την χρήση τους κυρίως σε εφαρμογές που δεν είναι ευαίσθητες στον χρόνο παράδοσης. Επίσης, τέτοιου είδους δίκτυα μπορούν να χρησιμοποιηθούν στον παγκόσμιο ιστό για την προφόρτωση δεδομένων (prefetching).

### *4.5.6 Ασφάλεια, δίκαιη χρήση, ιδιωτικότητα*

Το PAWS σχεδιάστηκε ως ένα πείραμα προκειμένου να εξερευνήσουμε τα όρια του τι είναι εφικτό στον παγκόσμιο ιστό. Για το λόγο αυτό δε δόθηκε βαρύτητα σε ζητήματα όπως η ασφάλεια, η δίκαιη χρήση και η ιδιωτικότητα.

Σχετικά με την ασφάλεια, το PAWS δε χρησιμοποιεί ελέγχους τιμών κατακερματισμού για τα αρχεία που διαθέτει. Έτσι, είναι θεωρητικά δυνατό ένας κακόβουλος χρήστης να στείλει διαφορετικά δεδομένα από αυτά που του αιτήθηκαν. Θεωρούμε ότι η ιδανικότερη λύση στο πρόβλημα της ακεραιότητας των δεδομένων, με δεδομένο ότι τα τεμαχίζουμε σε μικρότερα τμήματα, είναι η εφαρμογή ενός αλγορίθμου σαν τον «Rabin fingerprint», όπως προτείνεται στο [82].

Όσον αφορά τη δίκαιη χρήση των πόρων ενός πελάτη, πιστεύουμε ότι το προτεινόμενο σύστημα επιβαρύνει άδικα πελάτες που μένουν στην ιστοσελίδα πολλή ώρα. Θα ήταν χρήσιμη η προσθήκη ανώτατου ορίου στη χρήση δεδομένων κάθε πελάτη. Μάλιστα, το υψηλό κόστος των δεδομένων κινητής τηλεφωνίας, κάνει την ανάγκη για ύπαρξη ορίου, επιτακτική.

Σχετικά με την ιδιωτικότητα, όπως κάθε ομότιμο σύστημα, προκειμένου να είναι δυνατή η απευθείας σύνδεση των χρηστών σε επίπεδο δικτύου υποδομής, πληροφορίες όπως η IP τους είναι απαραίτητο να γνωστοποιηθούν στα ομότιμα μέλη. Μία λύση στο πρόβλημα προτείνεται στο [83], όπου γίνεται χρήση ενδιάμεσων peers προκειμένου να καταστούν δυσκολότερες οι «inference attacks». Δυστυχώς μια τέτοιου είδους λύση θα επιβάρυνε την απόδοση του συστήματος, αλλά και τα κεντρικά δίκτυα των ISP που θέλουμε να αποφορτώσουμε.

# 5 Επίλογος

## 5.1 Σύνοψη και συμπεράσματα

Κατά την εκπόνηση της έρευνάς μας, μάς δόθηκε η ευκαιρία να γνωρίσουμε σε βαθύτερο επίπεδο το πρόβλημα της διανομής περιεχομένου. Κάθε χρήση του διαδικτύου περιλαμβάνει την αποστολή και λήψη περιεχομένου. Για το λόγο αυτό, το πρόβλημα της βέλτιστης μεταφοράς του είναι καθολικό και αφορά όλα τα μέρη που συμμετέχουν στη διαδικασία: από τον δημιουργό και πάροχο του περιεχομένου, τα ενδιάμεσα μέρη – όπως οι πάροχοι υπηρεσιών διαδικτύου και CDN – έως τον τελικό καταναλωτή που έχει συνεχώς την απαίτηση για καλύτερες υπηρεσίες.

Αρχικά μελετήσαμε την ομότιμη αρχιτεκτονική συστημάτων, προκειμένου να αποκτήσουμε τις απαραίτητες θεωρητικές βάσεις που θα μας βοηθούσαν στη σχεδίαση του διακομιστή μας. Αντίστοιχη έρευνα έγινε και για τα δίκτυα διανομής περιεχομένου. Έχοντας κατανοήσει σε σημαντικό βαθμό τη θεωρία, συνεχίσαμε ερευνώντας τρόπους που έχουν προταθεί προκειμένου να συνδυαστούν οι δύο τεχνολογίες. Κύρια κίνητρα γύρω από αυτή την ιδέα, είναι η σχεδίαση υβριδικών δικτύων διανομής περιεχομένου που θα χαρακτηρίζονται τόσο από αξιοπιστία και υψηλές επιδόσεις – χαρακτηριστικά των CDN, όσο και από χαμηλό κόστος λειτουργίας και υψηλή ικανότητα κλιμάκωσης της απόδοσής τους – χαρακτηριστικά των ομότιμων συστημάτων.

Μετά από την έρευνά μας, συνεχίζουμε να πιστεύουμε ότι η αξιοποίηση των πόρων των πελατών για την υποστήριξη της διανομής περιεχομένου είναι μία πολλά υποσχόμενη ιδέα. Μάλιστα προτείνουμε τη χρήση της με σκοπό την μείωση των διαφημίσεων που κατακλύζουν κάθε γωνιά του παγκόσμιου ιστού. Ιδανικά, θα θέλαμε να δοθεί η επιλογή σε πελάτες που δεν επιθυμούν να βλέπουν διαφημίσεις – πελάτες που πιθανώς χρησιμοποιούν λογισμικό απόκρυψης διαφημίσεων και στερούν έσοδα από αγαπημένες ιστοσελίδες τους – να συνδράμουν μέσω της απορρόφησης ενός μέρους του κόστους εξυπηρέτησης.

Στη συνέχεια της διπλωματικής μας, ασχοληθήκαμε με τις υπάρχουσες τεχνολογίες ιστού που κάνουν δυνατή την ανάπτυξη, διάφανων στο χρήστη, ομότιμων δικτύων διανομής περιεχομένου. Τεχνολογίες όπως το WebRTC και η Cache, κάνουν εφικτή τη δημιουργία εφαρμογών που πριν λίγα χρόνια θα ήταν ιδιαιτέρως δύσκολες στην υλοποίησή τους, θα απαιτούσαν εγκατάσταση τρίτου λογισμικού ή θα ήταν απλά

αδύνατες. Παίρνοντας έμπνευση από το τι είναι δυνατό να κατασκευαστεί στον παγκόσμιο ιστό σήμερα, κάνουμε απόπειρα να σχεδιάσουμε και να αναπτύξουμε τη δική μας πρόταση για την ομότιμη παροχή περιεχομένου σε αυτόν. Έτσι προέκυψε το PAWS, ένας εξυπηρετητής ιστού υποβοηθούμενος από ομότιμους χρήστες. Το σύστημά μας φιλοδοξεί να εξοικονομήσει το εύρος ζώνης του συστήματος στο οποίο τρέχει, μειώνοντας έτσι το κόστος λειτουργίας του.

## 5.2 Όρια και περιορισμοί της έρευνας

Προκειμένου να εστιάσουμε στα βασικά επιθυμητά χαρακτηριστικά του PAWS, παραλείψαμε να ασχοληθούμε με άλλες πλευρές του προβλήματος διανομής περιεχομένου, όπως είναι η ασφάλεια και η ιδιωτικότητα. Στα δύο αυτά θέματα που είναι ιδιαιτέρως σημαντικά, θα πρέπει να δώσουμε ικανοποιητικές απαντήσεις εφόσον επιθυμούμε το PAWS, ή ένα άλλο παρόμοιό του σύστημα, να δοκιμαστεί στον πραγματικό κόσμο.

## 5.3 Μελλοντικές Επεκτάσεις

Αναφορικά με το PAWS, θα ήταν χρήσιμο να πραγματοποιηθεί μια ανάλυση της λειτουργίας του σε πραγματικές συνθήκες,. Έτσι θα διαπιστωθεί αν παρουσιάζει όντως τα επιθυμητά επίπεδα απόδοσης και αν συντελεί στην αποσυμφόρηση των πόρων του συστήματος όπου και τρέχει.

Επίσης εξαιρετικό ενδιαφέρον θα έχει να διαπιστώσουμε την αποδοχή του από το κοινό. Η χρήση άλλων ομότιμων συστημάτων, όπως είναι εφαρμογές ανταλλαγής αρχείων, παρουσιάζουν θετική αποδοχή από τους χρήστες τους. Όμως, αντίθετα με το PAWS, τα κίνητρα χρήσης των συστημάτων αυτών δεν είναι πάντα ανιδιοτελή.

# Βιβλιογραφία


[1] B. Chappell, ‘Stephen Hawking’s Ph.D. Thesis Crashes Cambridge Site After It’s Posted Online : The Two-Way : NPR’, 2017. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.npr.org/sections/thetwo-way/2017/10/23/559582380/stephen-hawkings-ph-d-thesis-crashes-cambridge-site-after-it-s-posted-online?t=1570037274143. [Ημερομηνία πρόσβασης: 02-Αυγούστου-2019].

[2] A. Pam και A. Xanadu, ‘Where world wide web went wrong’, στο *Proceedings of AUUG*, 1995, τ. 95.

[3] S. Adler, ‘The Slashdot effect: an analysis of three Internet publications’, *Linux Gaz.*, τ. 38, τχ. 2, 1999.

[4] A. Tsou, ‘How does the front page of the Internet behave? Readability, emoticon use, and links on Reddit’, *First Monday*, τ. 21, τχ. 11, Οκτωβρίου 2016.

[5] Ι. Κάβουρας, Ι. Μήλης, Γ. Ξυλωμένος, και Α. Ρουκουνάκη, *Συστήματα Υπολογιστών - Κατανεμημένα Συστήματα με Java*, 3ο έκδ. Αθήνα: Εκδόσεις Κλειδάριθμος, 2011.

[6] S. M. Thampi, *Introduction to Distributed Systems*. 2009.

[7] M. van Steen και A. Tanenbaum, *Distributed Systems*, 3.02. Maarten van Steen, 2018.

[8] J. F. Buford, H. Yu, και E. K. Lua, *P2P Networking and Applications*. Morgan Kaufmann, 2009.

[9] IAB και G. Camarillo, ‘Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability’, τχ. 5694. RFC Editor, σσ 1–26, 2009.

[10] S. Androutsellis-Theotokis και D. Spinellis, ‘A survey of peer-to-peer content distribution technologies’, *ACM Comput. Surv.*, τ. 36, τχ. 4, σσ 335–371, 2004.

[11] S. M. Thampi και C. S. K, ‘Survey of Search and Replication Schemes in Unstructured P2P Networks’, Αυγούστου 2010.

[12] A. Passarella, ‘A survey on content-centric technologies for the current Internet: CDN and P2P solutions’, *Comput. Commun.*, τ. 35, τχ. 1, σσ 1–32, Ιανουαρίου 2012.

[13] P. M. D. Gray *κ.ά.*, ‘Peer to Peer Overlay Networks: Structure, Routing and Maintenance’, στο *Encyclopedia of Database Systems*, Boston, MA: Springer US, 2009, σσ 2056–2061.

[14] R. Rodrigues και P. Druschel, ‘Peer-to-peer systems’, *Commun. ACM*, τ. 53, τχ. 10, σ 72, 2010.

[15] Cisco Systems, ‘Cisco Visual Networking Index: Forecast and Trends, 2017–

2022’, 2018.

[16] Zion Market Research, ‘Content Delivery Network (CDN) Market by Product (Standard/Non-Video CDN and Video CDN), by Content (Static and Dynamic), by Component (Solutions and Services), by Provider (P2P CDN, Traditional CDN, Cloud CDN, Telco CDN, and Others), and by Application A’, 2019.

[17] M. Satyanarayanan, ‘The emergence of edge computing’, *Computer (Long. Beach. Calif).*, τ. 50, τχ. 1, σσ 30–39, Ιανουαρίου 2017.

[18] G. Peng, ‘CDN: Content Distribution Network’, σσ 1–26, Νοεμβρίου 2004.

[19] M. H. Al-Shayeji, S. Rajesh, M. Alsarraf, και R. Alsuwaid, ‘A comparative study on replica placement algorithms for content delivery networks’, *Proc. - 2010 2nd Int. Conf. Adv. Comput. Control Telecommun. Technol. ACT 2010*, σσ 140–142, 2010.

[20] A. K. Pathan και R. Buyya, ‘A Taxonomy and Survey of Content Delivery Networks’, *Grid Comput. Distrib. Syst. GRIDS Lab. Univ. Melb. Park. Aust.*, τ. 148, σσ 1–44, 2006.

[21] V. N. Padmanabhan και K. Sripanidkulchai, ‘The case for cooperative networking’, στο *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002, τ. 2429, σσ 178–190.

[22] T. Stading, P. Maniatis, και M. Baker, ‘Peer-to-Peer Caching Schemes to Address Flash Crowds’, στο *Peer-to-Peer Systems*, τ. 2429, τχ. March, 2002, σσ 203–213.

[23] X. Y. Wang, W. S. Ng, B. C. Ooi, K. L. Tan, και A. Y. Zhou, ‘BuddyWeb: A P2P-based collaborative web caching system’, στο *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002, τ. 2376 LNCS, σσ 247–251.

[24] S. Iyer, A. Rowstron, και P. Druschel, ‘Squirrel: A decentralized peer-to-peer web cache’, PODC, 2002.

[25] G. Linden, ‘Geeking with Greg: Marissa Mayer at Web 2.0’, 2006. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html. [Ημερομηνία πρόσβασης: 24-Σεπτεμβρίου-2019].

[26] M. J. Freedman, E. Freudenthal, και D. Mazières, ‘Democratizing content publication with coral’, στο *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, 2004, σσ 239–252.

[27] M. Freedman και R. H. Arpaci-Dusseau, ‘Experiences with CoralCDN: A Five-Year Operational View’, στο *ACM Transactions on Computer Systems*, 2010, τ. 21, τχ. 1, σσ 95–110.

[28] L. Wang, K. Park, R. Pang, V. Pai, και L. Peterson, ‘Reliability and Security in the {CoDeeN} Content Distribution Network’, *Proc. 2004 USENIX Annu. Tech.*

*Conf.*, 2004.

[29] ‘So, no more nyud.net?’ [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://coral-users.cs.nyu.narkive.com/wnbTQFSt/so-no-more-nyud-net. [Ημερομηνία πρόσβασης: 28-Σεπτεμβρίου-2019].

[30] H. B. Ribeiro, L. C. Lung, A. O. Santin, και N. L. Brisola, ‘Web2Peer: A Peer-to-Peer Infrastructure for Publishing/Locating/Replicating Web Pages on Internet’, στο *Eighth International Symposium on Autonomous Decentralized Systems (ISADS’07)*, 2007, σσ 421–428.

[31] M. Sirivianos, J. H. Park, X. Yang, και S. Jarecki, ‘Dandelion: Cooperative Content Distribution with Robust Incentives’, στο *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, 2007, σσ 12:1--12:14.

[32] J. P. Mulerikkal και I. Khalil, ‘An Architecture for Distributed Content Delivery Network’, *2007 15th IEEE Int. Conf. Networks*, σσ 359–364, 2007.

[33] D. A. G. Manzato και N. L. S. Da Fonseca, ‘Incentive mechanisms for cooperation in peer-to-peer networks’, στο *Handbook of Peer-to-Peer Networking*, Springer US, 2010, σσ 631–660.

[34] ‘NetSession Interface Frequently Asked Questions | Akamai’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.akamai.com/uk/en/products/media-delivery/netsession-interface-faq.jsp. [Ημερομηνία πρόσβασης: 01-Οκτωβρίου-2019].

[35] J. Terrace, H. Laidlaw, H. E. Liu, S. Stern, και M. J. Freedman, ‘Bringing P2P to the Web: Security and Privacy in the Firecoral Network’, *IPTPS Proc. 8th Int. Conf. Peer-to-Peer Syst.*, σσ 1–6, 2009.

[36] ‘XPCOM - Mozilla | MDN’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM. [Ημερομηνία πρόσβασης: 23-Αυγούστου-2019].

[37] M. El Dick, E. Pacitti, και B. Kemme, ‘Flower-CDN’, *Proc. 12th Int. Conf. Extending Database Technol. Adv. Database Technol. - EDBT ’09*, σ 427, 2009.

[38] M. El Dick, E. Pacitti, R. Akbarinia, και B. Kemme, ‘Building a peer-to-peer content distribution network with high performance, scalability and robustness’, *Inf. Syst.*, τ. 36, τχ. 2, σσ 222–247, Απριλίου 2011.

[39] F. Zhou, L. Zhang, E. Franco, A. Mislove, R. Revis, και R. Sundaram, ‘WebCloud: Recruiting Social Network Users to Assist in Content Distribution’, στο *2012 IEEE 11th International Symposium on Network Computing and Applications*, 2012, σσ 10–19.

[40] L. Zhang, F. Zhou, A. Mislove, και R. Sundaram, ‘Maygh: building a CDN from client web browsers’, στο *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys ’13*, 2013, σ 281.

[41] F. Ribeiro, N. Barbosa, L. Fernando, και G. Soares, ‘Towards the Application of WebRTC Peer-to-Peer to Scale Live Video Streaming over the Internet’.

[42] MarketsandMarkets Analysis, ‘Commercial P2P CDN Market by Content Type (Video and Non-video), Solution (Web Performance Optimization, Media Delivery, and Cloud Security), Service, End-User Segment (Consumer and Business), Vertical, and Region – Global Forecast to 2023’, 2018.

[43] ‘Introducing WebRTC -- An Open Realtime Communications Project | WebRTC’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://webrtc.org/blog/2011/05/03/introducing-webrtc-an-open-realtime-communications-project.html. [Ημερομηνία πρόσβασης: 14-Σεπτεμβρίου-2019].

[44] A. Deveria, ‘Can I use... Support tables for HTML5, CSS3, etc’, *Fyrd*, 2013. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://caniuse.com/. [Ημερομηνία πρόσβασης: 14-Σεπτεμβρίου-2019].

[45] J.-I. Bruaroey, ‘Perfect negotiation in WebRTC - Advancing WebRTC’, 2019. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://blog.mozilla.org/webrtc/perfect-negotiation-in-webrtc/. [Ημερομηνία πρόσβασης: 03-Οκτωβρίου-2019].

[46] P. Thatcher, ‘What would you like to see in WebRTC next? A low-level API?’ W3C public-webrtc Mailing List, 2018.

[47] J. Rosenberg, ‘RFC 5245 - Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/rfc5245. [Ημερομηνία πρόσβασης: 20-Ιουλίου-2019].

[48] J. Uberti, C. Jennings, και E. Rescorla, ‘draft-ietf-rtcweb-jsep-26 - JavaScript Session Establishment Protocol’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-26. [Ημερομηνία πρόσβασης: 02-Οκτωβρίου-2019].

[49] S. Nandakumar και C. Jennings, ‘draft-ietf-rtcweb-sdp-11 - Annotated Example SDP for WebRTC’, 2018. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/draft-ietf-rtcweb-sdp-11. [Ημερομηνία πρόσβασης: 06-Ιουνίου-2019].

[50] J. Rosenberg, R. Mahy, P. Matthews, και D. Wing, ‘RFC 5389 - Session Traversal Utilities for NAT (STUN)’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/rfc5389. [Ημερομηνία πρόσβασης: 20-Ιουλίου-2019].

[51] R. Mahy, P. Matthews, και J. Rosenberg, ‘RFC 5766 - Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/rfc5766. [Ημερομηνία πρόσβασης: 21-Ιουλίου-2019].

[52] E. Ivov, E. Rescorla, J. Uberti, και P. Saint-Andre, ‘draft-ietf-ice-trickle-21 - Trickle ICE: Incremental Provisioning of Candidates for the Interactive

Connectivity Establishment (ICE) Protocol’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/draft-ietf-ice-trickle-21. [Ημερομηνία πρόσβασης: 20-Ιουλίου-2019].

[53] I. Grigorik, *High Performance Browser Networking: What every web developer should know about networking and web performance*. ‘ O’Reilly Media, Inc.’, 2013.

[54] M. Baugher, D. McGrew, M. Naslund, E. Carrara, και K. Norrman, ‘RFC 3711 - The Secure Real-time Transport Protocol (SRTP)’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/rfc3711. [Ημερομηνία πρόσβασης: 01-Οκτωβρίου-2019].

[55] R. Stewart, ‘RFC 4960 - Stream Control Transmission Protocol’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://tools.ietf.org/html/rfc4960. [Ημερομηνία πρόσβασης: 01-Οκτωβρίου-2019].

[56] Nils Ohlmeier, ‘Large Data Channel Messages - Advancing WebRTC’, 2017. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://blog.mozilla.org/webrtc/large-data-channel-messages/. [Ημερομηνία πρόσβασης: 01-Μαΐου-2019].

[57] ‘Add support for WebRTC Data Channel in Workers · Issue #230 · w3c/webrtc-pc · GitHub’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/w3c/webrtc-pc/issues/230. [Ημερομηνία πρόσβασης: 12-Οκτωβρίου-2019].

[58] R. Jesup, S. Loreto, και M. Tuexen, ‘draft-ietf-rtcweb-data-channel-13 - WebRTC Data Channels’, 2015. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://datatracker.ietf.org/doc/draft-ietf-rtcweb-data-channel/. [Ημερομηνία πρόσβασης: 09-Σεπτεμβρίου-2019].

[59] A. Russell, ‘Comet: Low Latency Data for the Browser – Infrequently Noted’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/. [Ημερομηνία πρόσβασης: 11-Μαΐου-2019].

[60] S. Loreto, P. Saint-Andre, S. Salsano, και G. Wilkins, ‘Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP’, Απριλίου 2011.

[61] ‘whatwg@whatwg.org from June 2008: by thread’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://lists.w3.org/Archives/Public/public-whatwg-archive/2008Jun/thread.html. [Ημερομηνία πρόσβασης: 09-Ιουλίου-2019].

[62] I. Fette και A. Melnikov, ‘The WebSocket Protocol’, Δεκεμβρίου 2011.

[63] ‘HTML Standard’, *WHATWG*. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://html.spec.whatwg.org/multipage/web-sockets.html#network. [Ημερομηνία πρόσβασης: 15-Ιουλίου-2019].

[64] P. McManus, ‘Bootstrapping WebSockets with HTTP/2’, Σεπτεμβρίου 2018.

[65] A. Russel, J. Song, J. Archibald, και M. Kruisselbrink, ‘Service Workers’, *W3C*, 2019. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.w3.org/TR/service-workers/. [Ημερομηνία πρόσβασης: 13-Αυγούστου-2019].

[66] J. Archibald, ‘Application Cache is a D******bag - A List Apart’, 2012. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://alistapart.com/article/application-cache-is-a-douchebag/. [Ημερομηνία πρόσβασης: 01-Αυγούστου-2019].

[67] ‘Browser storage limits and eviction criteria - Web APIs | MDN’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_limits_and_eviction_criteria. [Ημερομηνία πρόσβασης: 20-Σεπτεμβρίου-2019].

[68] ‘storage/browser/quota/quota_settings.cc - chromium/src - Git at Google’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://chromium.googlesource.com/chromium/src/+/refs/heads/master/storage/browser/quota/quota_settings.cc. [Ημερομηνία πρόσβασης: 20-Σεπτεμβρίου-2019].

[69] A. Alabbas, ‘Service Worker: Going beyond the page | Microsoft Edge Web Summit 2017 | Channel 9’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://channel9.msdn.com/Events/WebPlatformSummit/Microsoft-Edge-Web-Summit-2017/ES05. [Ημερομηνία πρόσβασης: 21-Σεπτεμβρίου-2019].

[70] B. Eich, ‘Popularity – Brendan Eich’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://brendaneich.com/2008/04/popularity/. [Ημερομηνία πρόσβασης: 31-Αυγούστου-2019].

[71] H. Demirkan και J. Spohrer, ‘T-Shaped Innovators: Identifying the Right Talent to Support Service Innovation’, *Res. Manag.*, τ. 58, τχ. 5, σσ 12–15, Σεπτεμβρίου 2015.

[72] A. Pechkurov, ‘An Intro to Node.js That You May Have Missed - ITNEXT’, 2018. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://itnext.io/an-intro-to-node-js-that-you-may-have-missed-b175ef4277f7. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[73] ‘rollup.js’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://rollupjs.org/guide/en/. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[74] ‘Express - Node.js web application framework’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://expressjs.com/. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[75] ‘GitHub - websockets/ws: Simple to use, blazing fast and thoroughly tested WebSocket client and server for Node.js’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/websockets/ws. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[76] ‘GitHub - broofa/node-mime: Mime types for JavaScript’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/broofa/node-mime. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[77] ‘GitHub - kelektiv/node-uuid: Generate RFC-compliant UUIDs in JavaScript’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/kelektiv/node-uuid. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[78] ‘Yargs be a node.js library fer hearties tryin’ ter parse optstrings.’ [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://yargs.js.org/. [Ημερομηνία πρόσβασης: 07-Ιουλίου-2019].

[79] J. Musacchio, J. Musacchio, G. Schwartz, και J. Walrand, ‘Network Economics: Neutrality, Competition, and Service Differentiation’.

[80] T. Karagiannis, P. Rodriguez, και K. Papagiannaki, ‘Should internet service providers fear peer-assisted content distribution?’, στο *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2005, σσ 63–76.

[81] R. Bindal *κ.ά.*, ‘Improving Traffic Locality in BitTorrent via Biased Neighbor Selection’, στο *26th IEEE International Conference on Distributed Computing Systems (ICDCS’06)*, 2006, τ. 2006, σσ 66–66.

[82] S. Ihm και V. S. Pai, ‘Towards understanding modern web traffic’, *ACM SIGMETRICS Perform. Eval. Rev.*, τ. 39, τχ. 1, σ 335, 2011.

[83] Y. Jia, G. Bai, P. Saxena, και Z. Liang, ‘Anonymity in Peer-assisted CDNs: Inference Attacks and Mitigation’, *Proc. Priv. Enhancing Technol.*, τ. 2016, τχ. 4, σσ 294–314.

# Παράρτημα Α - Παράδειγμα χρήσης

Στο παρακάτω έγγραφο HTML μπορούμε να διαπιστώσουμε με ποιον τρόπο μπορούμε να χρησιμοποιήσουμε το PAWS σε μία ιστοσελίδα. Παρουσιάζεται τόσο ο τρόπος χρήσης με `data-resource-url` (σειρές 47, 56-67, 71) όσο και με την παρεχόμενη διεπαφή προγραμματισμού (σειρές 91-103).

```
1.  <!DOCTYPE html>
2.  <html lang="en">
3.    <head>
4.      <meta charset="UTF-8" />
5.      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6.      <meta http-equiv="X-UA-Compatible" content="ie=edge" />
7.      <title>Image Gallery</title>
8.      <style>
9.        body {
10.         padding: 20px;
11.       }
12.       .container {
13.         display: flex;
14.         flex-wrap: wrap;
15.       }
16.       img {
17.         margin: 10px 14px;
18.       }
19.       .fullview {
20.         position: fixed;
21.         top: 0;
22.         left: 0;
23.         width: 100%;
24.         height: 100%;
25.         background: rgba(0, 0, 0, 0.7);
26.         display: flex;
27.         justify-content: center;
28.         align-items: center;
29.         display: none;
30.       }
31.       #fullview_image {
32.         max-width: 85vw;
33.         max-height: 85vh;
34.       }
35.       #close_button {
36.         position: absolute;
37.         top: 20px;
38.         right: 20px;
39.         background: none;
40.         border: none;
41.         color: white;
42.         text-decoration: underline;
43.         font-size: 1.3rem;
44.         cursor: pointer;
45.       }
46.     </style>
47.     <script data-resource-
    url="/scripts/test.js" type="text/javascript"></script>
48.   </head>
49.   <body>
```

```
    <h1>Gallery</h1>
    <p>
      This gallery uses PAWS to load images. The thumbnails are loaded using th
e "scanElements" functionality
      of PAWS while the fullsize versions are fetched programmatically by utili
sing the exposed API of the PAWS library.
    </p>
    <div class="container">
      <img data-resource-url="/images/thumb/Ashim DSilva.jpg" />
      <img data-resource-url="/images/thumb/Carmine De Fazio.jpg" />
      <img data-resource-url="/images/thumb/Jonas Nilsson Lee.jpg" />
      <img data-resource-url="/images/thumb/leigh-kendell-581.jpg" />
      <img data-resource-url="/images/thumb/Luca Bravo.jpg" />
      <img data-resource-url="/images/thumb/Morskie Oko.jpg" />
      <img data-resource-url="/images/thumb/Mr. Lee.jpg" />
      <img data-resource-url="/images/thumb/Pablo Garcia Saldana.jpg" />
      <img data-resource-url="/images/thumb/Photo by SpaceX.jpg" />
      <img data-resource-url="/images/thumb/Rob Bye.jpg" />
      <img data-resource-url="/images/thumb/Ryan Schroeder.jpg" />
      <img data-resource-url="/images/thumb/Sunset by the Pier.jpg" />
    </div>
    <div class="fullview">
      <button id="close_button">Close</button>
      <img id="fullview_image" data-resource-
url="/images/full/Rob Bye.jpg" />
    </div>
  </body>

  <script src="/paws.js"></script>
  <script>
    const container = document.getElementsByClassName('container')[0];
    const fullview = document.getElementsByClassName('fullview')[0];
    const fullviewImage = document.getElementById('fullview_image');
    const closeButton = document.getElementById('close_button');

    const closeFullView = () => (fullview.style.display = 'none');

    closeButton.addEventListener('click', closeFullView);
    fullview.addEventListener('click', e => e.target.tagName !== 'IMG' && close
FullView());

    paws.addEventListener('open', () => {
      container.addEventListener('click', e => {
        if (e.target.tagName === 'IMG') {
          const fullImageSrc = e.target.dataset.resourceUrl.replace('thumb', 'f
ull');
          if (paws.isClientSupported) {
            paws
              .fetchFromPeers(fullImageSrc)
              .catch(err => {
                console.warn(`${fullImageSrc} could not be fetched from peers.
Fetching normally...`, err);
                paws.fetch(fullImageSrc);
              })
              .then(res => res.blob())
              .then(blob => {
                fullviewImage.src = URL.createObjectURL(blob);
                fullview.style.display = 'flex';
              });
          } else {
            console.warn('This browser is not supported by PAWS. Fetching norma
lly...');
            fullviewImage.src = fullImageSrc;
            fullview.style.display = 'flex';
          }
```

```
        }
      });
      });
      </script>
      </html>
```

Το αρχείο ρυθμίσεων του διακομιστή PAWS, που υποστηρίζει αυτή την ιστοσελίδα παρατίθεται παρακάτω:

```
{
  "httpPort": 8080,
  "httpsPort": 8443,
  "redirect": true,
  "ssl": {
    "cert": "./ssl/test-cert.pem",
    "key": "./ssl/test-key.pem"
  },
  "webSocketPath": "/ws-paws",
  "mountPoint": { "urlPath": "/", "localPath": "./examples/image-gallery/" },
  "index": "index.html",
  "trustProxy": "loopback",
  "scanElements": true,
  "exportName": "paws",
  "timeoutTimeMs": 5000,
  "maxPeers": 5,
  "minPeers": 1,
  "iceServers": [{ "urls": [
    "stun:stun1.l.google.com:19302", "stun:stun2.l.google.com:19302"
  ] }]
}
```