Σελίδα τίτλου

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Η ΕΞΕΛΙΞΗ ΤΟΥ HTTP, ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΑΠΟΔΟΣΗΣ ΤΩΝ ΠΡΩΤΟΚΟΛΛΩΝ HTTP, HTTP2, HTTP3

Λύκος Παντελής

Επιβλέπουσα: Μαργαρίτη Σπυριδούλα

ΕΔΙΠ, MSc, Ph.D.

Άρτα, Μάιος, 2023

THE EVOLUTION OF HTTP, A COMPARATIVE STUDY OF HTTP, HTTP2, HTTP3 PROTOCOLS

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, Μάιος, 2023

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπουσα

   Σπυριδούλα Μαργαρίτη,

2. Μέλος επιτροπής

   Ελευθέριος Στεργίου,

3. Μέλος επιτροπής

   Δημήτριος Λιαροκάπης,

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Λύκος, Παντελής

Υπογραφή

ΠΕΡΙΛΗΨΗ

Ο Παγκόσμιος Ιστός έχει γίνει αναπόσπαστο μέρος της ζωής μας και το πρωτόκολλο HTTP είναι το θεμέλιο πάνω στο οποίο έχει χτιστεί. Από την έναρξή του, το HTTP έχει υποστεί σημαντική εξέλιξη για να καλύψει τις ανάγκες ενός διαρκώς μεταβαλλόμενου περιβάλλοντος web. Η τελευταία έκδοση του HTTP, HTTP/3, κυκλοφόρησε το 2020 και χρησιμοποιεί ένα νέο πρωτόκολλο μεταφοράς που ονομάζεται QUIC. Ενώ τα HTTP/1.1 και HTTP/2 χρησιμοποιούνται ευρέως, είναι σημαντικό να ληφθούν υπόψη τα χαρακτηριστικά και η απόδοσή τους σε σχέση με το HTTP/3. Αυτό εγείρει πολλά ερωτήματα: Ποια είναι η διαφορά μεταξύ των HTTP, HTTP/2 και HTTP/3; Πώς αποδίδουν κάτω από διαφορετικές συνθήκες; Ποια οφέλη παρέχουν σε προγραμματιστές και χρήστες ιστού; Για να απαντήσουμε στα παραπάνω ερωτήματα, στοχεύουμε να πραγματοποιήσουμε μια συγκριτική ανάλυση της εξέλιξης του HTTP. Αυτή η έρευνα εξετάζει τις διαφορές στη λειτουργικότητα, την απόδοση και την αποτελεσματικότητα των HTTP, HTTP/2 και HTTP/3. Αναλύοντας την εξέλιξη του HTTP, ελπίζουμε να συμβάλουμε στην καλύτερη κατανόηση της συνεχούς εξέλιξης των πρωτοκόλλων Διαδικτύου και του ρόλου τους στη διαμόρφωση του μέλλοντος του Διαδικτύου. Συνολικά, αυτή η μελέτη στοχεύει να παρέχει μια ολοκληρωμένη ανάλυση της εξέλιξης του HTTP και των επιπτώσεών του στον Παγκόσμιο Ιστό. Μέσω αυτής της ανάλυσης, ελπίζουμε να αποκτήσουμε μια εικόνα για τα δυνατά και αδύνατα σημεία κάθε πρωτοκόλλου, καθώς και τις δυνατότητές τους να διαμορφώσουν το μέλλον των πρωτοκόλλων Διαδικτύου.

Λέξεις-κλειδιά: HTTP, Διαδικτυακό Πρωτόκολλο, Αρχιτεκτονική Πρωτοκόλλου, Σύγκριση Πρωτοκόλλων

ABSTRACT

The World Wide Web has become an integral part of our lives, and the HTTP protocol is the foundation upon which it is built. Since its inception, HTTP has undergone significant evolution to meet the needs of an ever-changing web environment. The latest version of HTTP, HTTP/3, was released in 2020 and uses a new transport protocol called QUIC. While HTTP/1.1 and HTTP/2 are widely used, it is important to consider their features and performance relative to HTTP/3. This raises several questions: What is the difference between HTTP, HTTP/2 and HTTP/3? How do they perform under different conditions? What benefits do they provide to developers and web users? To answer the above questions, we aim to perform a comparative analysis of the evolution of HTTP. This survey examines the differences in functionality, performance, and efficiency of HTTP, HTTP/2, and HTTP/3. By analyzing the evolution of HTTP, we hope to contribute to a better understanding of the continuous evolution of Internet protocols and their role in shaping the future of the Internet. Overall, this study aims to provide a comprehensive analysis of the evolution of HTTP and its impact on the World Wide Web. Through this analysis, we hope to gain insight into each protocol's strengths and weaknesses, as well as their potential to shape the future of Internet protocols. The method used to write this thesis is the bibliographic review. The writing was carried out by studying books, scientific articles and internet search.

Keywords: HTTP, Internet Protocol, Protocol Architecture, Protocol Comparison

Περιεχόμενα

1. Εισαγωγή

1.1. Ορισμός του προβλήματος

Ο Παγκόσμιος Ιστός έχει γίνει αναπόσπαστο μέρος της ζωής μας και το πρωτόκολλο HTTP είναι το θεμέλιο πάνω στο οποίο έχει χτιστεί. Από την έναρξή του, το HTTP έχει υποστεί σημαντική εξέλιξη για να καλύψει τις ανάγκες ενός διαρκώς μεταβαλλόμενου διαδικτυακού τοπίου.

Η τελευταία έκδοση του HTTP, το HTTP/3, κυκλοφόρησε το 2020 χρησιμοποιώντας ένα νέο πρωτόκολλο μεταφοράς που ονομάζεται QUIC. Ενώ τα HTTP/1.1 και HTTP/2 χρησιμοποιούνται ευρέως, είναι σημαντικό να εξεταστούν τα χαρακτηριστικά και η απόδοσή τους σε σχέση με το HTTP/3. Αυτό εγείρει πολλά ερωτήματα: Ποιες είναι οι διαφορές μεταξύ των HTTP, HTTP/2 και HTTP/3; Πώς αποδίδουν κάτω από διάφορες συνθήκες; Ποια οφέλη προσφέρουν στους προγραμματιστές και τους χρήστες ιστού;

1.2. Κίνητρο και στόχος

Για να απαντήσουμε στα παραπάνω ερωτήματα, στοχεύουμε να πραγματοποιήσουμε μια συγκριτική ανάλυση της εξέλιξης του HTTP. Αυτή η μελέτη θα εξετάσει τις διαφορές μεταξύ των HTTP, HTTP/2 και HTTP/3 όσον αφορά τις δυνατότητες, την απόδοση και την αποτελεσματικότητα. Αναλύοντας την εξέλιξη του HTTP, ελπίζουμε να συμβάλουμε στην καλύτερη κατανόηση της συνεχούς ανάπτυξης των πρωτοκόλλων Διαδικτύου και του ρόλου τους στη διαμόρφωση του μέλλοντος του Διαδικτύου.

Συνολικά, αυτή η μελέτη επιδιώκει να παρέχει μια ολοκληρωμένη ανάλυση της εξέλιξης του HTTP και των επιπτώσεών του στον Παγκόσμιο Ιστό. Μέσω αυτής της ανάλυσης, ελπίζουμε να παρέχουμε πληροφορίες για τα δυνατά και τα αδύνατα σημεία κάθε πρωτοκόλλου και τις δυνατότητές τους να διαμορφώσουν το μέλλον των πρωτοκόλλων Διαδικτύου.

1.3. Δομή εργασίας

Στο δεύτερο κεφάλαιο θα ασχοληθούμε με την αρχή του HTTP και θα αναλύσουμε όλες τις διαδικασίες που διαμορφώνουν το πρωτόκολλο HTTP/1.1 όπως τα είδη των συνδέσεων και θα περιγράψουμε τη συνολική δομή των μηνυμάτων HTTP, όπως και το τι συμβαίνει με το caching και τα proxies.

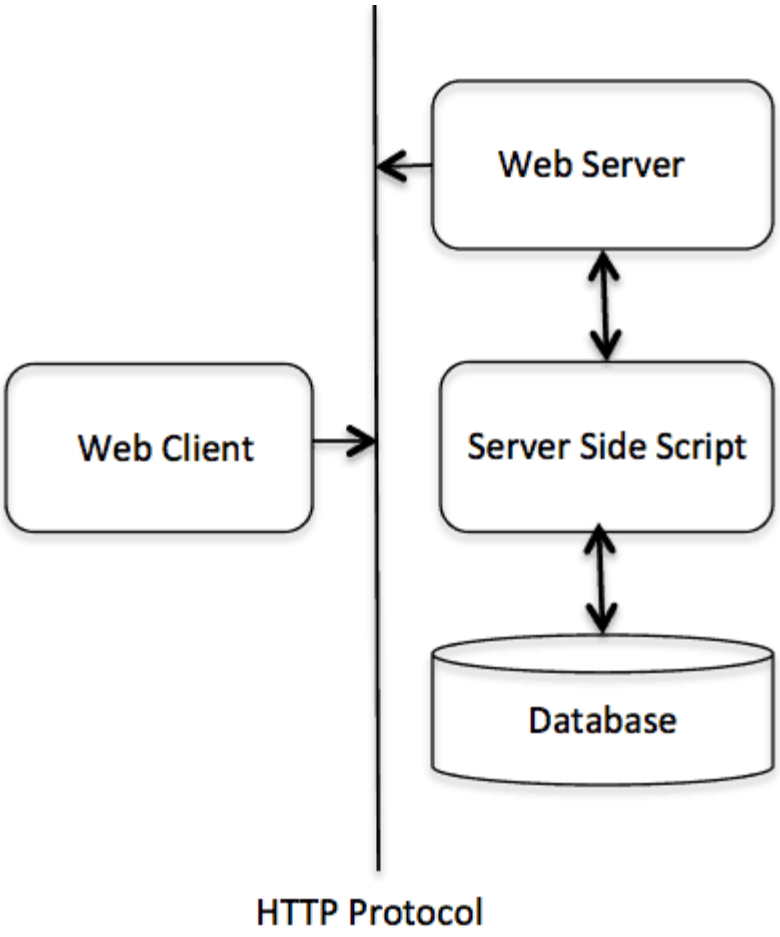
Στο τρίτο κεφάλαιο θα ασχοληθούμε με την επόμενη έκδοση του πρωτοκόλλου HTTP, το HTTP/2, και θα αναλύσουμε τη διαδικασία με την οποία δημιουργήθηκε όπως και το τι καινούριο και καινοτόμο έφερε στο πρωτόκολλο HTTP μέσω των νέων πλαισίων και διαδικασιών που εισήγαγε.

Στο τέταρτο κεφάλαιο θα ασχοληθούμε με την τρίτη και πιο πρόσφατη έκδοση του HTTP, θα περιγράψουμε πως δημιουργήθηκε, θα αναλύσουμε ποιες βελτιώσεις έφερε στο πρωτόκολλο και σε ποια προβλήματα και μικρές δυσκολίες έδωσε τις αναγκαίες λύσεις.

Στο πέμπτο και τελευταίο κεφάλαιο θα εκτελέσουμε μια σύγκριση των παραπάνω εκδόσεων σε θέματα όπως η εγκατάσταση της σύνδεσής τους και η αρχιτεκτονική τους, και θα αναλύσουμε όποια ζητήματα απόδοσης υπάρχουν μεταξύ τους με σκοπό να παρατηρήσουμε με απτά αποτελέσματα τις βελτιώσεις που έχουν γίνει στο πρωτόκολλο από την αρχή του έως και σήμερα.

2. Το πρωτόκολλο HTTP

Το πρωτόκολλο HTTP είναι ένα πρωτόκολλο αιτήματος/απόκρισης και αποτελεί τον ενδιάμεσο κρίκο στην αρχιτεκτονική client/server που επιτρέπει σε ένα πρόγραμμα πελάτη (client) να επικοινωνεί με ένα άλλο πρόγραμμα server. Τα προγράμματα που λειτουργούν σαν clients και χρησιμοποιούν το πρωτόκολλο HTTP είναι οι περιηγητές Ιστού, τα ρομπότ και οι μηχανές αναζήτησης, κ.λπ., και τα προγράμματα που εξυπηρετούν τα αιτήματα των πελατών αποτελούν τον εξυπηρετητή Ιστού ή γνωστό ως web server. Στην Εικόνα 2.1 απεικονίζεται η βασική αρχιτεκτονική μιας εφαρμογής Ιστού και η θέση που καταλαμβάνει το πρωτόκολλο HTTP [1].



Εικόνα 2.1: Βασική Αρχιτεκτονική HTTP
(Πηγή: https://www.tutorialspoint.com/http/http_overview.htm)

2.1.Ιστορικά στοιχεία

Το Hypertext Transfer Protocol (HTTP), ή αλλιώς το Πρωτόκολλο Μεταφοράς Υπερκειμένου, είναι ένα πρωτόκολλο σε επίπεδο εφαρμογής για κατανεμημένα, συνεργατικά πληροφοριακά συστήματα. Η ανάπτυξη του HTTP ξεκίνησε το 1989 από τον Tim Berners-Lee, με το HTTP να βρίσκεται σε χρήση από τον Παγκόσμιο Ιστό (World-Wide Web) από το 1990. Η πρώτη έκδοση του HTTP, αναφορικά το HTTP/0.9, ήταν ένα απλό πρωτόκολλο για μεταφορά ανεπεξέργαστων δεδομένων μέσω του διαδικτύου. Η επόμενη έκδοσή του, γνωστή ως HTTP/1.0 βελτίωσε το πρωτόκολλο με τον εξής τρόπο: Επέτρεπε στα μηνύματα να αλλάξουν διαμόρφωση και να περιέχουν πληροφορίες όσων αφορά τα δεδομένα που μεταφέρονται και τροποποιητές σχετικά με τη σημασιολογία αιτήματος/απόκρισης. Ωστόσο, το HTTP/1.0 δεν λαμβάνει υπόψιν του δεδομένα όπως τα αποτελέσματα των ιεραρχικών διακομιστών μεσολάβησης (proxy), την προσωρινή αποθήκευση (caching), την ανάγκη για μόνιμες συνδέσεις ή τους εικονικούς χρήστες (virtual hosts). Επιπλέον, ο πολλαπλασιασμός ατελώς υλοποιημένων εφαρμογών που

αυτοαποκαλούνται "HTTP/1.0" έχει φέρει την απαίτηση για μια αλλαγή της έκδοσης πρωτοκόλλου προκειμένου δυο εφαρμογές που βρίσκονται σε επικοινωνία μεταξύ τους να μπορούν να προσδιορίσουν τις πραγματικές δυνατότητες η μία της άλλης. Αυτή η προδιαγραφή ορίζει το πρωτόκολλο που αναφέρεται ως "HTTP/1.1". Αυτό το πρωτόκολλο περιλαμβάνει πιο αυστηρές απαιτήσεις από το HTTP/1.0 με σκοπό να διασφαλιστεί η αξιόπιστη εφαρμογή των δυνατοτήτων του. Το πρώτο πρότυπο που ορίζει το HTTP/1.1, την πιο κοινή έκδοση του HTTP που χρησιμοποιείται σήμερα, εμφανίστηκε το 1997. Αυτό το πρότυπο αντικαταστάθηκε σύντομα από ένα νέο, το RFC 2616, και στη συνέχεια ξανά το 2014 από μια ομάδα προτύπων, RFC 7230 έως RFC 7235. [5][6]

Τα πρακτικά πληροφοριακά συστήματα απαιτούν περισσότερη λειτουργικότητα από την απλή ανάκτηση, συμπεριλαμβανομένης της αναζήτησης, της ενημέρωσης του front-end και του σχολιασμού. Το HTTP επιτρέπει ένα ανοιχτού τύπου σύνολο μεθόδων και κεφαλίδων που υποδεικνύουν τον σκοπό ενός αιτήματος (request). Βασίζεται στην πειθαρχία αναφοράς (disciple of reference) που παρέχεται από τον Uniform Resource Identifier (URI)[1], ως τοποθεσία (URL)[2], ή ως όνομα (URN)[3], για την ένδειξη του πόρου στον οποίο πρόκειται να εφαρμοστεί μια μέθοδος.

Το HTTP χρησιμοποιείται επίσης ως γενικό πρωτόκολλο για επικοινωνία μεταξύ χρηστών και διακομιστών μεσολάβησης/πυλών σε άλλα συστήματα Διαδικτύου, συμπεριλαμβανομένων αυτών που υποστηρίζονται από τα πρωτόκολλα SMTP[4], NNTP[5],

[1] Το Uniform Resource Identifier (URI) είναι μια μοναδική ακολουθία χαρακτήρων που προσδιορίζει έναν λογικό ή φυσικό πόρο που χρησιμοποιείται από τεχνολογίες ιστού.

[2] Ο όρος URL ή Uniform Resource Locator δηλώνει μια διεύθυνση ενός πόρου του Παγκόσμιου Ιστού. Είναι παρόμοιο με το όνομα ενός αρχείου, αλλά κρατάει και επιπλέον πληροφορία σχετικά με το όνομα του εξυπηρετητή και το είδος του πρωτοκόλλου που αυτός χρησιμοποιεί. Το URL είναι υποκατηγορία του URI, αλλά συνήθως χρησιμοποιείται ως συνώνυμο αυτού.

[3] Το URN ή Uniform Resource Name είναι ένας πόρος διαδικτύου με ένα όνομα που είναι παγκόσμια μοναδικό, με σκοπό να είναι διαθέσιμα για μεγάλο χρονικό διάστημα, ακόμα και όταν ο πόρος που προσδιορίζουν παύσει να υπάρχει ή καταστεί μη διαθέσιμος, σε αντίθεση με ένα URL το οποίο μπορεί να αλλάξει. Όπως και το URL, το URN είναι υποκατηγορία του URI.

[4] Το Simple Mail Transfer Protocol (SMTP) χρησιμοποιείται για την αποστολή και λήψη email. Μερικές φορές συνδυάζεται με IMAP ή POP3 (για παράδειγμα, από μια εφαρμογή σε επίπεδο χρήστη), η οποία χειρίζεται την ανάκτηση μηνυμάτων, ενώ το SMTP στέλνει κυρίως μηνύματα σε έναν διακομιστή για προώθηση.

[5] Το Network News Transfer Protocol (NNTP) χρησιμοποιείται για τη μεταφορά ειδήσεων από το ένα δίκτυο στο άλλο. Έχει σχεδιαστεί ειδικά για τη μεταφορά ειδήσεων/άρθρων. Ένας NNTP client περιλαμβάνεται σε προγράμματα περιήγησης όπως το Netscape, η Opera και ο Internet Explorer ή μια ειδική εφαρμογή με το όνομα newsreader μπορεί να χρησιμοποιηθεί ως πελάτης NNTP.

FTP[6], Gopher[7] και WAIS[8]. Με αυτόν τον τρόπο, το HTTP επιτρέπει τη βασική πρόσβαση υπερμέσων σε πόρους που διατίθενται από διάφορες εφαρμογές.[5]

2.2. Η αρχιτεκτονική Client – Server

Το μοντέλο client - server είναι μια κατανεμημένη αρχιτεκτονική εφαρμογών που αφορά την επικοινωνία μεταξύ των:

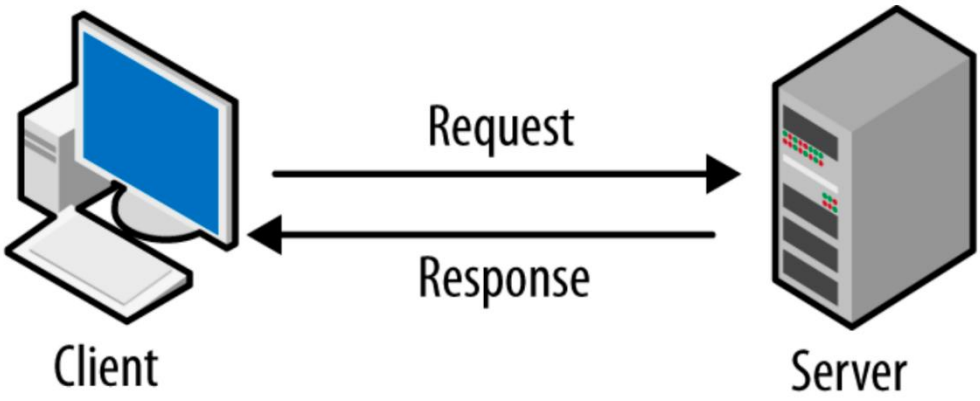
- Server: το πρόγραμμα/εφαρμογή που "εξυπηρετεί" τα αιτήματα (π.χ. σελίδες web). Ο server διαχωρίζει την εργασία ή το φόρτο εργασίας μεταξύ των παρόχων ενός πόρου ή υπηρεσίας.
- Client: το πρόγραμμα/εφαρμογή που "αιτείται" υπηρεσίες και προβάλλει τις απαντήσεις στον χρήστη.

Στην αρχιτεκτονική client - server, όταν ο υπολογιστής-client στέλνει ένα αίτημα για δεδομένα στον server μέσω του Διαδικτύου, ο server αποδέχεται το αίτημα επικοινωνίας και προχωρά την απαιτούμενη διαδικασία και παραδίδει τα πακέτα δεδομένων που ζητήθηκαν πίσω στον πελάτη. Οι πελάτες δεν μοιράζονται κανέναν από τους πόρους τους. Παραδείγματα μοντέλου client - server είναι το ηλεκτρονικό ταχυδρομείο, ο Παγκόσμιος Ιστός κ.λπ. [2] όπως φαίνεται και στην .

[6] Το File Transfer Protocol (FTP) είναι ένα πρωτόκολλο δικτύου για τη μετάδοση αρχείων μεταξύ υπολογιστών μέσω συνδέσεων Transmission Control Protocol/Internet Protocol (TCP/IP). Μέσα στη σουίτα TCP/IP, το FTP θεωρείται πρωτόκολλο επιπέδου εφαρμογής.

[7] Το πρωτόκολλο Gopher είναι ένα πρωτόκολλο επικοινωνίας που έχει σχεδιαστεί για τη διανομή, την αναζήτηση και την ανάκτηση εγγράφων σε δίκτυα πρωτοκόλλου Διαδικτύου.

[8] Ο Wide Area Information Server (WAIS) είναι ένα σύστημα αναζήτησης κειμένου client-server που χρησιμοποιεί το ANSI Standard Z39.50 "Information Retrieval Service Definition and Protocol Specifications for Library Applications" (Z39.50:1988) για αναζήτηση βάσεων δεδομένων ευρετηρίου σε απομακρυσμένους υπολογιστές.



Εικόνα 2.2. Μοντέλο Client – Server

(Πηγή: https://madooei.github.io/cs421_sp20_homepage/client-server-app/)

2.3.Γενική Λειτουργία του HTTP

Το πρωτόκολλο HTTP είναι ένα πρωτόκολλο αίτησης/απόκρισης. Ένας client στέλνει ένα αίτημα στον server με τη μορφή μεθόδου αιτήματος, URI και έκδοσης πρωτοκόλλου, ακολουθούμενο από ένα μήνυμα τύπου MIME (Multipurpose Internet Mail Extensions)[9] που περιέχει τροποποιητές αιτήματος, πληροφορίες του client και πιθανό περιεχόμενο σώματος μέσω σύνδεσης με το server. Ο server απαντά με μια γραμμή κατάστασης, συμπεριλαμβανομένης της έκδοσης πρωτοκόλλου του μηνύματος και έναν κωδικό επιτυχίας ή σφάλματος, ακολουθούμενη από ένα μήνυμα τύπου MIME που περιέχει πληροφορίες server, μεταπληροφορίες οντοτήτων και πιθανό περιεχόμενο οντοτήτων.

Οι περισσότερες επικοινωνίες HTTP ξεκινούν από έναν user agent και αποτελούνται από ένα αίτημα που πρέπει να εφαρμοστεί σε έναν πόρο σε κάποιο server προέλευσης. Στην απλούστερη περίπτωση, αυτό μπορεί να επιτευχθεί μέσω μιας μοναδικής σύνδεσης μεταξύ του user agent και του server προέλευσης. Μια πιο περίπλοκη κατάσταση εμφανίζεται όταν ένας ή περισσότεροι μεσάζοντες είναι παρόντες στην αλυσίδα αιτήματος/απόκρισης. Υπάρχουν τρείς κοινές μορφές διαμεσολάβησης: proxy, gateway και tunnel.

[9] Το Multipurpose Internet Mail Extensions (MIME) είναι μια επέκταση του αρχικού πρωτοκόλλου ηλεκτρονικού ταχυδρομείου Simple Mail Transport Protocol (SMTP). Επιτρέπει στους χρήστες να ανταλλάσσουν διάφορα είδη αρχείων δεδομένων, συμπεριλαμβανομένων ήχου, βίντεο, εικόνων και προγραμμάτων εφαρμογών, μέσω email.

- Το *proxy* είναι ένας agent προώθησης, ο οποίος λαμβάνει αιτήματα για ένα URI στην απόλυτη μορφή του, ξαναγράφει ολόκληρο ή μέρος του μηνύματος και προωθεί το αναδιαμορφωμένο αίτημα προς το server που προσδιορίζεται από το URI.
- Το *gateway* είναι ένας agent λήψης, που λειτουργεί ως στρώμα πάνω από κάποιους servers και εάν είναι απαραίτητο, μεταφράζει τα αιτήματα στο πρωτόκολλο του υποκείμενου server.
- Το *tunnel* λειτουργεί ως ένα σημείο αναμετάδοσης μεταξύ δυο συνδέσεων χωρίς να αλλάζει τα μηνύματα. Τα tunnels χρησιμοποιούνται όταν η επικοινωνία πρέπει να περάσει από έναν μεσάζοντα (όπως ένα τείχος προστασίας) ακόμη και όταν ο μεσάζοντας δε μπορεί να κατανοήσει το περιεχόμενο των μηνυμάτων.

Ο στόχος του HTTP/1.1 είναι να υποστηρίξει την ευρεία ποικιλία των διαμορφώσεων που έχουν ήδη αναπτυχθεί, εισάγοντας δομές πρωτοκόλλου που καλύπτουν τις ανάγκες όσων κατασκευάζουν εφαρμογές διαδικτύου που απαιτούν υψηλή αξιοπιστία και, σε αντίθετη περίπτωση, τουλάχιστον αξιόπιστες ενδείξεις αποτυχίας.

Η επικοινωνία HTTP πραγματοποιείται συνήθως μέσω συνδέσεων TCP/IP. Η προεπιλεγμένη θύρα είναι η TCP 80, αλλά μπορούν να χρησιμοποιηθούν και άλλες θύρες. Αυτό αποκλείει την εφαρμογή του HTTP πάνω από οποιοδήποτε άλλο πρωτόκολλο στο Διαδίκτυο ή σε άλλα δίκτυα. Το HTTP προϋποθέτει μόνο μια αξιόπιστη μεταφορά, οποιοδήποτε πρωτόκολλο που παρέχει τέτοιες εγγυήσεις μπορεί να χρησιμοποιηθεί. Η αντιστοίχιση των δομών αιτήματος και απόκρισης του HTTP/1.1 στις μονάδες δεδομένων μεταφοράς του εν λόγω πρωτοκόλλου δεν εμπίπτει στο πεδίο εφαρμογής αυτής της προδιαγραφής.

Στο HTTP/1.0, οι περισσότερες υλοποιήσεις χρησιμοποιούσαν μια νέα σύνδεση για κάθε ανταλλαγή αιτήματος/απόκρισης. Στο HTTP/1.1, μια σύνδεση μπορεί να χρησιμοποιηθεί για μια ή περισσότερες ανταλλαγές αιτήματος/απόκρισης, αν και οι συνδέσεις μπορεί να κλείσουν για διάφορους λόγους. [5]

2.4.Είδη Συνδέσεων

Σε πολλές εφαρμογές Διαδικτύου, οι client και server επικοινωνούν για μεγάλο χρονικό διάστημα, με τον client να κάνει μια σειρά από requests και τον server να ανταποκρίνεται (responding) σε κάθε ένα από τα requests. Ανάλογα με την εφαρμογή και τον τρόπο με τον οποίο χρησιμοποιείται η εφαρμογή, η σειρά των request μπορεί να υποβάλλεται διαδοχικά, περιοδικά σε τακτά χρονικά διαστήματα ή και πιο αραιά. Όταν

αυτή η αλληλεπίδραση client-server λαμβάνει χώρα μέσω TCP, πρέπει να παρθεί μια πολύ σημαντική απόφαση από τον developer: αν κάθε ζεύγος request/response θα αποστέλλεται μέσω μιας ξεχωριστής σύνδεσης TCP ή εάν όλα τα requests και τα αντίστοιχα response τους θα αποστέλλονται μέσω της ίδιας σύνδεσης TCP. Στην πρώτη προσέγγιση, η εφαρμογή λέγεται ότι χρησιμοποιεί non-persistent (ή μη μόνιμες) συνδέσεις, και στην δεύτερη προσέγγιση, λέγεται ότι χρησιμοποιεί persistent (ή μόνιμες) συνδέσεις. Παρόλο που η προεπιλεγμένη λειτουργία του HTTP/1.1 είναι να χρησιμοποιεί persistent συνδέσεις, οι HTTP clients και servers μπορούν να ρυθμιστούν ώστε να χρησιμοποιούν non-persistent συνδέσεις. [6]

2.4.1. Persistent (Μόνιμες) και non-Persistent (Μη Μόνιμες) Συνδέσεις

Πριν από τις persistent συνδέσεις, στο πρωτόκολλο HTTP/1.0 έπρεπε να δημιουργείται μια ξεχωριστή σύνδεση TCP για την ανάκτηση κάθε URL, αυξάνοντας το φόρτο στους HTTP servers και προκαλώντας συμφόρηση στο Διαδίκτυο. Η χρήση ενσωματωμένων εικόνων και άλλων σχετικών δεδομένων απαιτεί συχνά από έναν client να υποβάλλει πολλαπλά αιτήματα από τον ίδιο server σε σύντομο χρονικό διάστημα. [5] [7]

Οι Persistent HTTP συνδέσεις έχουν κάποια πλεονεκτήματα:

- Ανοίγοντας και κλείνοντας λιγότερες TCP συνδέσεις, ο χρόνος της CPU εξοικονομείται σε routers και hosts (clients, servers, proxies, gateways, tunnels ή caches) και η μνήμη που χρησιμοποιείται για TCP protocol control blocks μπορεί να σωθεί σε hosts.
- Τα HTTP requests και responses μπορούν να διοχετευθούν σε μια σύνδεση. Αυτή η διοχέτευση επιτρέπει σε έναν client να κάνει πολλά requests χωρίς να περιμένει για κάθε response, επιτρέποντας τη χρήση μιας μεμονωμένης σύνδεσης TCP πολύ πιο αποτελεσματικά, σε μικρότερο χρονικό περιθώριο.
- Η συμφόρηση δικτύου ελαττώνεται μειώνοντας τον αριθμό των πακέτων που προκαλούνται από τα ανοίγματα του TCP και αφήνοντας στο TCP αρκετό χρόνο για να καθορίσει την κατάσταση συμφόρησης του δικτύου.
- Το latency σε επόμενα requests μειώνεται καθώς δεν δαπανάται χρόνος για την εναρκτήρια “χειραψία” της TCP σύνδεσης.
- Το HTTP μπορεί να εξελιχθεί με περισσότερη χάρη, καθώς τα σφάλματα μπορούν να αναφερθούν χωρίς την ποινή του κλεισίματος της TCP σύνδεσης. Οι clients που χρησιμοποιούν μελλοντικές εκδόσεις ενδέχεται να δοκιμάσουν αισιόδοξα μια νέα καινοτομία, αλλά εάν επικοινωνούν με έναν παλαιότερο server, να ξαναδοκιμάσουν με την παλιά σημασιολογία εφόσον αναφερθεί κάποιο σφάλμα.

2.4.2. Συνολική Λειτουργία

Μια σημαντική διαφορά μεταξύ του HTTP/1.1 και των προηγούμενων εκδόσεων του HTTP είναι ότι οι persistent συνδέσεις είναι η προεπιλεγμένη συμπεριφορά οποιασδήποτε σύνδεσης HTTP. Δηλαδή, εκτός αν υποδεικνύεται διαφορετικά, ο client πρέπει να υποθέσει ότι ο server θα διατηρήσει μια persistent σύνδεση, ακόμη και μετά από response σφάλματος από τον server. Οι persistent συνδέσεις παρέχουν έναν μηχανισμό μέσω του οποίου ένας client και ένας server μπορούν να σηματοδοτήσουν το κλείσιμο μιας TCP σύνδεσης. Αυτή η σηματοδότηση πραγματοποιείται χρησιμοποιώντας το Header field "Connection". Μόλις σηματοδοτηθεί το κλείσιμο, ο client δεν πρέπει να στείλει άλλα αιτήματα σε αυτή τη σύνδεση. [5]

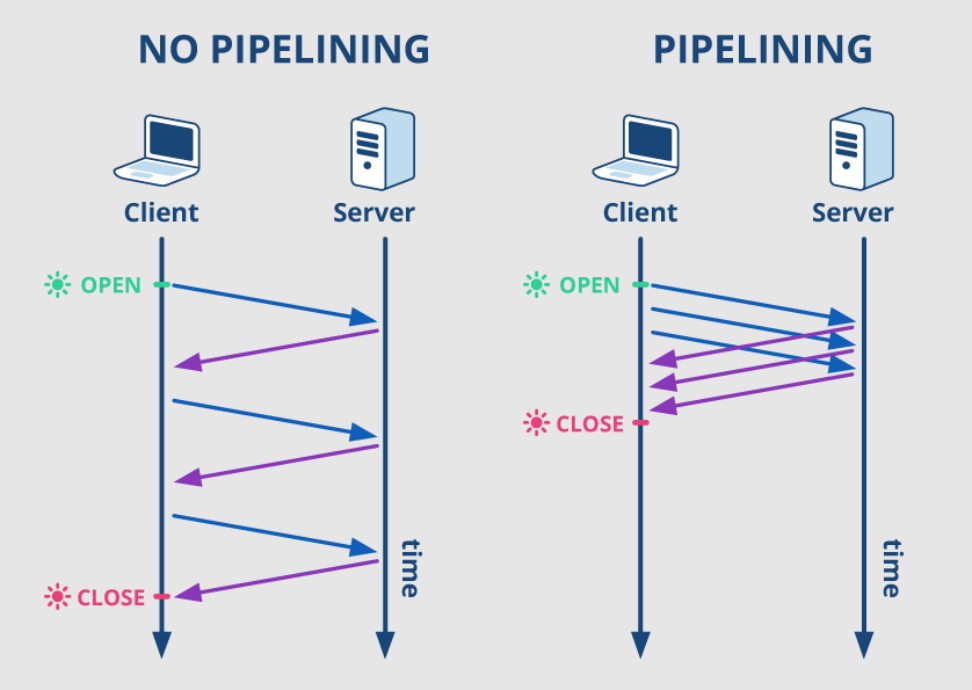
2.4.3. Διαπραγμάτευση

Ένας HTTP/1.1 server μπορεί να υποθέσει ότι ένας HTTP/1.1 client σκοπεύει να διατηρήσει μια μόνιμη σύνδεση, εκτός αν μια "Connection" κεφαλίδα η οποία περιλαμβάνει το διακριτικό σύνδεσης "close" έχει σταλεί στο request. Εάν ο server επιλέξει να κλείσει τη σύνδεση αμέσως μετά την αποστολή του response, είναι αναγκαίο να στείλει μια "Connection" κεφαλίδα που θα περιλαμβάνει το διακριτικό σύνδεσης "close". Από την άλλη, ένας HTTP/1.1 client μπορεί να αναμένει ότι μια σύνδεση θα παραμένει ανοιχτή, αλλά θα αποφασίσει να τη διατηρήσει ανοιχτή με βάση το εάν η απάντηση από έναν server περιέχει μια "Connection" κεφαλίδα με το διακριτικό σύνδεσης "close". Σε περίπτωση που ο client δε θέλει να διατηρήσει μια σύνδεση για περισσότερο από αυτό το request, θα πρέπει να στείλει μια "Connection" κεφαλίδα που να περιλαμβάνει το διακριτικό σύνδεσης "close". Εάν είτε ο client είτε ο server στείλει το διακριτικό "close" στην "Connection" κεφαλίδα, αυτό το request γίνεται το τελευταίο για τη σύνδεση.

Οι clients και οι servers δε πρέπει να υποθέτουν ότι μια μόνιμη σύνδεση διατηρείται για εκδόσεις HTTP μικρότερες από την 1.1, εκτός εάν έχει σηματοδοτηθεί ρητά. Για να παραμείνουν persistent, όλα τα μηνύματα στη σύνδεση πρέπει να έχουν αυτοκαθορισμένο μήκος μηνύματος, δηλαδή ένα που δεν ορίζεται από το κλείσιμο της σύνδεσης, όπως θα δούμε σε παρακάτω κεφάλαιο. [5]

2.4.4. Pipelining (Σωλήνωση)

Ένας client που υποστηρίζει persistent συνδέσεις μπορεί να κάνει “pipeline” τα requests του, δηλαδή να στέλνει πολλαπλά requests χωρίς να περιμένει response για το καθένα ξεχωριστά. Ένας server πρέπει να στείλει τα responses του σε αυτά τα requests με την ίδια σειρά που ελήφθησαν τα εν λόγω requests.



Εικόνα 2.3. Παράδειγμα Pipelining

(Πηγή: https://www.haproxy.com/blog/http-keep-alive-pipelining-multiplexing-and-connection-pooling/)

Οι clients που αναλαμβάνουν persistent συνδέσεις και κάνουν «σωλήνωση» αμέσως μετά την επίτευξη της σύνδεσης είναι αναγκαίο να είναι προετοιμασμένοι να ξαναδοκιμάσουν την σύνδεσή τους εάν η πρώτη προσπάθεια «σωλήνωσης» αποτύχει. Εάν ένας client κάνει μία τέτοια επανάληψη, δεν πρέπει να κάνει προσπάθεια για «σωλήνωση» προτού καταλάβει ότι η σύνδεση είναι persistent. Οι clients πρέπει επίσης να είναι έτοιμοι να στείλουν ξανά τα requests τους εάν ο server κλείσει τη σύνδεση πριν στείλει όλα τα αντίστοιχα responses. [5]

2.4.5. Πρακτικές εκτιμήσεις

Οι servers συνήθως θα έχουν κάποιο time-out value πέραν του οποίου δε θα διατηρούν πλέον μια ανενεργή σύνδεση. Οι proxy servers ενδέχεται να κάνουν αυτό το value υψηλότερο, καθώς είναι πιθανό ο client να πραγματοποιεί περισσότερες συνδέσεις μέσω του ίδιου server. Η χρήση persistent συνδέσεων δεν θέτει απαιτήσεις σχετικά με τη διάρκεια (ή την ύπαρξη) αυτού του time-out είτε για τον client είτε για τον server.

Όταν ένας client ή server επιθυμεί να κάνει time-out, πρέπει να κάνει κλείσει την σύνδεση. Οι clients και οι servers πρέπει να παρακολουθούν συνεχώς για το κλείσιμο της άλλης πλευράς και να αποκρίνονται όπως αρμόζει. Εάν ένας client ή server δεν εντοπίσει το κλείσιμο της άλλης πλευράς αμέσως, πιθανότατα θα προκληθεί περιττή διαρροή πόρων στο δίκτυο.

Ένας client, server, ή proxy μπορεί να κλείσει τη σύνδεση μεταφοράς ανά πάσα στιγμή. Για παράδειγμα, ένας client μπορεί να αρχίσει να στέλνει ένα νέο request την ίδια στιγμή που ο server αποφάσισε να κλείσει την αδρανή σύνδεση. Από τη πλευρά του server, η σύνδεση κλείνει επειδή ήταν ανενεργή, αλλά από την πλευρά του client, ένα request βρίσκεται σε εξέλιξη.

Αυτό σημαίνει ότι οι clients, οι servers και οι proxies πρέπει αναγκαστικά να μπορούν να επανέλθουν μετά από ασύγχρονα συμβάντα κλεισίματος. Το λογισμικό του client πρέπει να ξανά ανοίξει τη σύνδεση και να μεταδώσει εκ νέου τη ματαιωμένη ακολουθία request χωρίς αλληλεπίδραση με τον χρήστη, εφόσον η ακολουθία αιτημάτων είναι idempotent (ή ταυτοδύναμη, δηλαδή να μπορεί να εφαρμοστεί πολλές φορές χωρίς να αλλάξει το αποτέλεσμα πέρα από την αρχική εφαρμογή). Οι μη ταυτοδύναμες μέθοδοι ή οι αλληλουχίες απαγορεύεται να ξανά στέλνονται αυτόματα, αν και οι user agents μπορούν να προσφέρουν σε έναν χρήστη την επιλογή να ξανά στείλει το (ή τα) request. Η επιβεβαίωση από ένα user-agent λογισμικό με σημασιολογική κατανόηση της εφαρμογής μπορεί να υποκαταστήσει την επιβεβαίωση του χρήστη. Η αυτόματη επαναποστολή δεν πρέπει να επαναληφθεί εάν η δεύτερη σειρά request αποτύχει.

Οι servers πρέπει πάντα να ανταποκρίνονται σε τουλάχιστον ένα request ανά σύνδεση, εάν αυτό είναι δυνατόν. Οι servers δε πρέπει να κλείνουν μια σύνδεση στη μέση της μετάδοσης ενός response εκτός εάν υπάρχει υποψία αποτυχίας δικτύου ή client. Οι clients που χρησιμοποιούν persistent συνδέσεις πρέπει να περιορίζουν τον αριθμό των ταυτόχρονων συνδέσεων που διατηρούν σε έναν δεδομένο server. Ένας client μοναδικού

χρήστη δεν πρέπει να διατηρεί περισσότερες από 2 συνδέσεις με οποιονδήποτε server ή proxy. Ένας proxy πρέπει να χρησιμοποιεί έως και 2*N συνδέσεις σε άλλο server ή proxy, όπου N είναι ο αριθμός των ταυτόχρονα ενεργών χρηστών. Αυτές οι οδηγίες αποσκοπούν στη βελτίωση των χρόνων απόκρισης του HTTP και την αποφυγή συμφόρησης. [5]

2.5.Μηνύματα HTTP

2.5.1. Δομή μηνύματος

Τα μηνύματα HTTP αποτελούνται από requests (αιτήματα) από τον client προς τον server και από responses (αποκρίσεις) από τον server προς τον client και έχουν την ακόλουθη μορφή:

HTTP-message = request | response

Τα μηνύματα request και response χρησιμοποιούν τη γενική μορφή μηνύματος RFC 822 για τη μεταφορά οντοτήτων (δηλαδή το ωφέλιμο φορτίο του μηνύματος). Και οι δύο τύποι μηνυμάτων αποτελούνται από μια γραμμή έναρξης, κανένα ή περισσότερα πεδία κεφαλίδας (γνωστά ως header fields ή "headers"), μια κενή γραμμή (δηλαδή μια γραμμή χωρίς τίποτα πριν το CRLF[10]) που υποδεικνύει το τέλος των πεδίων κεφαλίδας και πιθανώς ένα σώμα μηνύματος.

generic-message = start-line * (message-header CRLF)

CRLF

[Message-body]

Start-line = Request-Line | Status-Line

Για λόγους ευρωστίας, οι servers θα πρέπει να αγνοούν τυχόν κενές γραμμές που ελήφθησαν όπου αναμένεται μια Request-Line. Με άλλα λόγια, εάν ο server διαβάζει τη

[10] Το "CRLF" αναφέρεται στα στοιχεία ειδικών χαρακτήρων "Carriage Return" και "Line Feed". Αυτά τα στοιχεία είναι ενσωματωμένα σε κεφαλίδες HTTP και γενικά σε άλλο κώδικα λογισμικού για να δηλώνουν έναν δείκτη τέλους γραμμής ή End of Line (EOL).

ροή πρωτοκόλλου στην αρχή ενός μηνύματος και λάβει πρώτα ένα CRLF, θα πρέπει να το αγνοήσει.

Ορισμένες "buggy" υλοποιήσεις client στο HTTP/1.0 δημιουργούν επιπλέον CRLF μετά από ένα POST request. Για να επαναδιατυπωθεί αυτό που απαγορεύεται ρητά από τη BNF[11], ένας HTTP/1.1 client αναγκαστικά δεν πρέπει να προλογίζει η να ακολουθεί ένα νέο αίτημα με επιπλέον CRLF. [5]

2.5.2. Headers ή Κεφαλίδες μηνυμάτων

Τα πεδία κεφαλίδας HTTP, τα οποία περιλαμβάνουν πεδία γενικής κεφαλίδας (general-header), κεφαλίδας αιτήματος (request-header), κεφαλίδας απόκρισης (response-header), και κεφαλίδας οντότητας (entity-header), ακολουθούν την ίδια γενική μορφή με αυτήν που δίνεται στο RFC 822. Κάθε πεδίο κεφαλίδας αποτελείται από ένα όνομα ακολουθούμενο από άνω κάτω τελεία (":") και την τιμή του πεδίου. Τα ονόματα πεδίων δεν έχουν διάκριση πεζών-κεφαλαίων. Η τιμή του πεδίου μπορεί να προηγείται από οποιοδήποτε ποσό LWS (Linear White Space) αν και προτιμάται ένα μόνο SP (space). Τα πεδία κεφαλίδας μπορούν να επεκταθούν σε πολλές γραμμές εάν προηγούνται από κάθε επιπλέον γραμμή τουλάχιστον ένα SP ή ένα HT (Horizontal Tab). Οι εφαρμογές θα πρέπει να ακολουθούν την «κοινή μορφή», όπου είναι γνωστό ή υποδεικνύεται, κατά τη δημιουργία δομών HTTP, καθώς ενδέχεται να υπάρχουν κάποιες που δεν αποδέχονται τίποτα πέρα από τις κοινές φόρμες.

Message-header = field name ":" [field-value]

Field-name = token

Field-value = *(field-content | LWS)

Field-content = < the OCTETs making up the field value and consisting of either *TEXT or combinations of token, separators, and quoted-string>

[11] Στη θεωρητική πληροφορική, BNF ή Backus-Naur Form ονομάζεται μια τεχνική συμβολισμού για γραμματικές χωρίς συμφραζόμενα, που συχνά χρησιμοποιείται για να περιγράψει τη σύνταξη μιας γλώσσας της πληροφορικής, όπως οι γλώσσες προγραμματισμού, οι τύποι εγγράφων, τα σύνολα εντολών και τα πρωτόκολλα επικοινωνιών.

Το περιεχόμενο πεδίου δεν περιλαμβάνει κανένα LWS στην αρχή ή στο τέλος, δηλαδή ένα LWS το οποίο εμφανίζεται πριν από τον πρώτο κανονικό χαρακτήρα της τιμής πεδίου η μετά τον τελευταίο χαρακτήρα της τιμής πεδίου. Τέτοια προπορευόμενα ή υστερούντα LWS μπορούν να αφαιρεθούν χωρίς αλλαγή της σημασιολογίας της τιμής του πεδίου. Οποιοδήποτε LWS εμφανίζεται μεταξύ ανάμεσα στο περιεχόμενο πεδίου μπορεί να αντικατασταθεί με ένα μόνο SP πριν από την ερμηνεία της τιμής του πεδίου ή την προώθηση του μηνύματος downstream.

Η σειρά με την οποία λαμβάνονται τα πεδία κεφαλίδας με διαφορετικά ονόματα πεδίων δεν είναι σημαντική. Ωστόσο, είναι "καλή πρακτική" να στέλνονται πρώτα τα πεδία γενικής κεφαλίδας, ακολουθούμενα από τα πεδία κεφαλίδας αιτήματος ή κεφαλίδας απόκρισης και να τέλος τα πεδία κεφαλίδας οντότητας.

Μπορούν να υπάρχουν πολλαπλά πεδία κεφαλίδας μηνύματος με το ίδιο όνομα πεδίου σε ένα μήνυμα εάν και μόνο εάν ολόκληρη η τιμή πεδίου ορίζεται ως λίστα διαχωρισμένη με κόμμα [δηλαδή #(values)]. Επιβάλλεται να είναι δυνατός ο συνδυασμός των πολλαπλών πεδίων κεφαλίδας σε ένα ζεύγος "field-name : field-value", χωρίς αλλαγή της σημασιολογίας του μηνύματος, προσθέτοντας κάθε επόμενο πεδίο-τιμή στο πρώτο, το καθένα χωρισμένο με κόμμα. Η σειρά με την οποία λαμβάνονται τα πεδία κεφαλίδας με το ίδιο όνομα πεδίου είναι επομένως σημαντική για την ερμηνεία της τιμής του συνδυασμένου πεδίου και επομένως ένας proxy απαγορεύεται να αλλάξει τη σειρά αυτών των τιμών πεδίων όταν προωθείται ένα μήνυμα. [5]

2.5.3. Body ή σώμα μηνύματος

Message-body = entity-body | <entity-body encoded as per Transfer-Encoding>

Το σώμα του μηνύματος (αν υπάρχει) ενός HTTP μηνύματος χρησιμοποιείται για τη μεταφορά του σώματος οντότητας που σχετίζεται με το request ή το response. Το σώμα του μηνύματος διαφέρει από το σώμα οντότητας μόνο όταν έχει εφαρμοστεί μια

κωδικοποίηση μεταφοράς (transfer-coding). Η κεφαλίδα Transfer-Encoding[12] επιβάλλεται να χρησιμοποιηθεί για να υποδείξει τυχών κωδικοποιήσεις μεταφοράς που εφαρμόζονται από μια εφαρμογή για να διασφαλιστεί η ασφαλής και σωστή μεταφορά του μηνύματος. Η Transfer-Encoding είναι ιδιότητα του μηνύματος, όχι της οντότητας, και επομένως μπορεί να προστεθεί ή αφαιρεθεί από οποιαδήποτε εφαρμογή κατά μήκος της αλυσίδας request/response. Προφανώς όμως, υπάρχουν και περιορισμοί.

Οι κανόνες για το πότε ένα σώμα μηνύματος επιτρέπεται σε ένα μήνυμα διαφέρουν για requests και για responses. Η παρουσία ενός σώματος μηνύματος σε ένα request σηματοδοτείται με τη συμπερίληψη ενός πεδίου κεφαλίδας Content-Length[13] ή Transfer-Encoding στις κεφαλίδες μηνυμάτων του request. Ένα σώμα μηνύματος απαγορεύεται να περιλαμβάνεται σε ένα request εάν η προδιαγραφή της μεθόδου request δεν επιτρέπει την αποστολή ενός σώματος οντότητας σε requests. Ένας server πρέπει να διαβάζει και να προωθεί ένα σώμα μηνύματος σε οποιοδήποτε request. Εάν η μέθοδος request δεν περιλαμβάνει καθορισμένη σημασιολογία για ένα σώμα οντότητας, τότε το σώμα μηνύματος πρέπει να αγνοηθεί κατά το χειρισμό του request.

Όσων αφορά τα responses, το εάν ένα σώμα μηνύματος περιλαμβάνεται η όχι στο μήνυμα εξαρτάται τόσο από τη μέθοδο request όσο και από τον κωδικό κατάστασης του response. Όλα τα responses στη μέθοδο request HEAD απαγορεύεται να περιλαμβάνουν σώμα μηνύματος, παρόλο που η παρουσία entity-header πεδίων μπορεί να μας κάνει να πιστεύουμε ότι μπορούν. Όλα τα responses με κωδικό 1xx (ενημερωτικά), 204 (χωρίς περιεχόμενο) και 304 (χωρίς τροποποίηση) δεν πρέπει να περιλαμβάνουν σώμα μηνύματος. Όλα τα άλλα responses περιλαμβάνουν σώμα μηνύματος, αν και μπορεί να είναι μηδενικού μήκους. [5]

[12] Το γενικό πεδίο κεφαλίδας Transfer-Encoding υποδεικνύει ποιος (αν υπάρχει) τύπος μετασχηματισμού έχει εφαρμοστεί στο σώμα του μηνύματος προκειμένου να μεταφερθεί με ασφάλεια μεταξύ του αποστολέα και του παραλήπτη. Αυτό διαφέρει από την κωδικοποίηση περιεχομένου στο ότι η κωδικοποίηση μεταφοράς είναι ιδιότητα του μηνύματος και όχι της οντότητας. Εάν έχουν εφαρμοστεί πολλαπλές κωδικοποιήσεις σε μια οντότητα, οι κωδικοποιήσεις μεταφοράς πρέπει να παρατίθενται με τη σειρά με την οποία εφαρμόστηκαν. Πρόσθετες πληροφορίες σχετικά με τις παραμέτρους κωδικοποίησης μπορεί να παρέχονται από άλλα πεδία κεφαλίδας οντοτήτων που δεν ορίζονται από αυτήν την προδιαγραφή. Πολλές παλαιότερες εφαρμογές HTTP/1.0 δεν κατανοούν την κεφαλίδα Transfer-Encoding.

[13] Το πεδίο κεφαλίδας-οντότητας Content-Length δείχνει το μέγεθος του σώματος οντότητας, σε δεκαδικό αριθμό OCTET, που αποστέλλεται στον παραλήπτη ή, στην περίπτωση της μεθόδου HEAD, το μέγεθος του σώματος οντότητας που θα είχε σταλεί, εάν το αίτημα ήταν GET.

2.5.4. Μήκος μηνύματος

Το μήκος μεταφοράς (transfer-length) ενός μηνύματος είναι το μήκος του σώματος μηνύματος όπως εμφανίζεται στο μήνυμα, δηλαδή αφού έχουν εφαρμοστεί τυχόν μεταφορές ή κωδικοποιήσεις. Όταν ένα σώμα μηνύματος περιλαμβάνεται σε ένα μήνυμα, το μήκος μεταφοράς αυτού του σώματος καθορίζεται από ένα από τα ακόλουθα (κατά σειρά προτεραιότητας):

1. Οποιοδήποτε response που "απαγορεύεται" να περιλαμβάνει ένα σώμα (όπως τα responses 1xx, 204 και 304 και οποιοδήποτε response σε HEAD request) τερματίζεται πάντα από την πρώτη κενή γραμμή μετά τα πεδία κεφαλίδας, ανεξάρτητα από τα πεδία οντότητας – κεφαλίδας που υπάρχουν στο μήνυμα.
2. Εάν υπάρχει ένα Transfer-Encoding πεδίο κεφαλίδας και έχει οποιαδήποτε άλλη τιμή εκτός από την "identity", τότε το μήκος μεταφοράς ορίζεται με τη χρήση της "chunked"[14], εκτός εάν το μήνυμα τερματίζεται με το κλείσιμο της σύνδεσης.
3. Εάν υπάρχει ένα Content-Length πεδίο κεφαλίδας, η δεκαδική του τιμή σε OCTET[15] αντιπροσωπεύει τόσο το μήκος οντότητας όσο και το μήκος μεταφοράς. Το Content-Length απαγορεύεται να αποσταλεί εάν αυτά τα δυο μήκη είναι διαφορετικά (δηλαδή αν υπάρχει κεφαλίδα Transfer-Encoding). Εάν ληφθεί ένα μήνυμα με πεδίο κεφαλίδας Transfer-Encoding και πεδίο κεφαλίδας Content-Length, τότε το Content-Length πρέπει να αγνοηθεί.
4. Εάν το μήνυμα χρησιμοποιεί τον τύπο μέσου "multipart/byteranges" και το μήκος μεταφοράς δεν καθορίζεται διαφορετικά, τότε αυτός ο αυτο-οριοθετούμενος τύπος μέσου δεν πρέπει να χρησιμοποιείται εκτός εάν ο αποστολέας γνωρίζει ότι ο παραλήπτης μπορεί να το αναλύσει. Η παρουσία σε ένα request μιας Range κεφαλίδας με πολλούς προσδιοριστές byte-range από έναν HTTP/1.1 client υποδηλώνει ότι ο client μπορεί να αναλύσει multipart/byteranges responses.
5. Από τον server που κλείνει τη σύνδεση. (Το κλείσιμο της σύνδεσης δεν μπορεί να χρησιμοποιηθεί για να υποδείξει το τέλος ενός σώματος request, καθώς αυτό δε θα άφηνε τη δυνατότητα στο server να στείλει πίσω ένα response.)

Μια Range κεφαλίδαμπορεί να προωθηθεί από έναν proxy 1.0 που δεν κατανοεί multipart/byteranges. Σε αυτήν την περίπτωση ο server επιβάλλεται να οριοθετήσει το μήνυμα χρησιμοποιώντας τις μεθόδους που ορίζονται στα παραπάνω σημεία 1, 3 ή 5 αυτής της ενότητας.

[14] Κάθε φορά που εφαρμόζεται μια κωδικοποίηση μεταφοράς σε ένα σώμα μηνύματος, το σύνολο των κωδικοποιήσεων μεταφοράς πρέπει να περιλαμβάνει το "chunked", εκτός εάν το μήνυμα τερματίζεται με το κλείσιμο της σύνδεσης. Όταν χρησιμοποιείται η "chunked" κωδικοποίηση μεταφοράς, επιβάλλεται να είναι η τελευταία κωδικοποίηση μεταφοράς που εφαρμόζεται στο σώμα του μηνύματος. Η "chunked" κωδικοποίηση μεταφοράς απαγορεύεται να εφαρμόζεται περισσότερες από μία φορές σε ένα σώμα μηνύματος. Αυτοί οι κανόνες επιτρέπουν στον παραλήπτη να καθορίσει το μήκος μεταφοράς του μηνύματος.

[15] OCTET ή οκτάδα είναι μια μονάδα ψηφιακών πληροφοριών στους υπολογιστές και τις τηλεπικοινωνίες που αποτελείται από οκτώ bit. Ο όρος χρησιμοποιείται συχνά όταν ο όρος byte μπορεί να είναι διφορούμενος, καθώς το byte έχει χρησιμοποιηθεί ιστορικά για μονάδες αποθήκευσης διαφόρων μεγεθών.

Για συμβατότητα με εφαρμογές HTTP/1.0, τα HTTP/1.1 requests που περιέχουν σώμα μηνύματος πρέπει να περιλαμβάνουν ένα έγκυρο Content-Length [19] πεδίο κεφαλίδας, εκτός εάν είναι γνωστό ότι ο server είναι συμβατός με το HTTP/1.1. Εάν ένα request περιέχει σώμα μηνύματος και δε δίνεται Content-Length, ο server πρέπει να στείλει response 400 (bad request) αν δε μπορεί να καθορίσει τη διάρκεια του μηνύματος ή να στείλει response 411 (length required) αν επιθυμεί να επιμείνει λήψη έγκυρου Content-Length.

Όλες οι HTTP/1.1 εφαρμογές που λαμβάνουν οντότητες επιβάλλεται να αποδέχονται την "chunked" κωδικοποίηση μεταφοράς, επιτρέποντας έτσι τη χρήση αυτού του μηχανισμού για μηνύματα όταν το μήκος του μηνύματος δεν μπορεί να καθοριστεί εκ των προτέρων.

Τα μηνύματα απαγορεύεται να περιλαμβάνουν και Content-Length κεφαλίδα και non-identity κωδικοποίηση μεταφοράς, η Content-Length πρέπει να αγνοηθεί. Όταν δίνεται μια Content-Length κεφαλίδα σε ένα μήνυμα που επιτρέπεται ένα σώμα μηνύματος, η τιμή πεδίου επιβάλλεται να αντιστοιχεί ακριβώς στον αριθμό των OCTET στο σώμα μηνύματος. Οι HTTP/1.1 user agents πρέπει να ειδοποιούν τον χρήστη όταν λαμβάνεται και ανιχνεύεται μη έγκυρο μήκος. [5]

2.5.5. Γενικά πεδία κεφαλίδας

Υπάρχουν μερικά πεδία κεφαλίδας που έχουν γενική εφαρμογή τόσο για τα request όσο και για τα response, αλλά δεν ισχύουν για την οντότητα που μεταφέρεται. Αυτά τα πεδία κεφαλίδας ισχύουν μόνο για το μήνυμα που μεταδίδεται. Ονομαστικά αυτά είναι:

General-header:

Cache-Control

Connection

Date

Pragma

Trailer

Transfer-Encoding

Upgrade

Via

Warning

Αυτά τα ονόματα μπορούν να επεκταθούν αξιόπιστα μόνο σε συνδυασμό με μία αλλαγή στην έκδοση πρωτοκόλλου. Ωστόσο, σε νέα ή πειραματικά πεδία κεφαλίδας μπορεί να δοθεί η σημασιολογία των γενικών πεδίων κεφαλίδας εάν όλα τα μέρη στην επικοινωνία τα αναγνωρίσουν ως γενικά πεδία κεφαλίδας. Τα μη αναγνωρισμένα πεδία κεφαλίδας αντιμετωπίζονται ως πεδία κεφαλίδας οντότητας (entity-header fields). [5]

2.5.6. Μηνύματα Αίτησης (Request)

Ένα μήνυμα request από έναν client σε έναν server περιλαμβάνει στην πρώτη γραμμή του τη μέθοδο που θα εφαρμοστεί στον πόρο, το αναγνωριστικό του πόρου και την έκδοση πρωτοκόλλου που χρησιμοποιείται.

Request = Request Line * ((general-header | request-header | entity-header) CRLF)

CRLF

[message-body]

Η γραμμή Request-Line ξεκινά με ένα διακριτικό μεθόδου, ακολουθούμενο από το Request-URI και την έκδοση πρωτοκόλλου και τελειώνει με το CRLF. Τα στοιχεία

διαχωρίζονται με SP (space) χαρακτήρες. Δεν επιτρέπεται η χρήση CR ή LF εκτός από την τελική ακολουθία CRLF.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Το διακριτικό Method υποδεικνύει τη μέθοδο που πρέπει να εκτελεστεί στον πόρο που προσδιορίζεται από το Request-URI. Η μέθοδος έχει διάκριση πεζών-κεφαλαίων. Παρακάτω παρατίθεται η λίστα με τις μεθόδους:

Method:

"OPTIONS"

"GET"

"HEAD"

"POST"

"PUT"

"DELETE"

"TRACE"

"CONNECT"

Η λίστα μεθόδων που επιτρέπονται από έναν πόρο μπορεί να καθοριστεί από ένα πεδίο κεφαλίδας Allow. Ο κωδικός επιστροφής του response ειδοποιεί πάντα τον client εάν μια μέθοδος επιτρέπεται την εκάστοτε στιγμή σε έναν πόρο, καθώς το σύνολο των επιτρεπόμενων μεθόδων μπορεί να αλλάξει δυναμικά. Ένας server προέλευσης πρέπει να επιστρέψει τον κωδικό κατάστασης 405 (Method Not Allowed) εάν η μέθοδος είναι γνωστή από τον server προέλευσης αλλά δεν επιτρέπεται για τον ζητούμενο πόρο, και 501 (Not Implemented) αν η μέθοδος δεν αναγνωρίζεται ή δεν εφαρμόζεται από τον server

προέλευσης. Οι μέθοδοι GET και HEAD επιβάλλεται να υποστηρίζονται από όλους τους server γενικής χρήσης. Όλες οι άλλες μέθοδοι είναι προαιρετικές. Ωστόσο, αν εφαρμοστούν οι παραπάνω μέθοδοι, πρέπει να υλοποιηθούν με την ίδια σημασιολογία με τις υπόλοιπες καθορισμένες μεθόδους. [5]

2.5.6.1. Πεδία κεφαλίδας Request (Request Header Fields)

Τα πεδία της κεφαλίδας request επιτρέπουν στον client να διαβιβάσει πρόσθετες πληροφορίες σχετικά με το request και για τον ίδιο τον client στον server. Αυτά τα πεδία λειτουργούν ως τροποποιητές request, με σημασιολογία ισοδύναμη με τις παραμέτρους σε μια επίκληση μεθόδου γλώσσας προγραμματισμού.

Request-header:

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Authorization

Expect

From

Host

If-Match

If-Modified-Since

If-None-Match

If-Range

If-Unmodified-Since

Max-Forwards

Proxy-Authorization

Range

Referer

TE

User-Agent

Τα ονόματα των πεδίων κεφαλίδας request μπορούν να επεκταθούν αξιόπιστα μόνο σε συνδυασμό με μια αλλαγή στην έκδοση πρωτοκόλλου. Ωστόσο, στα νέα ή πειραματικά πεδία κεφαλίδας μπορεί να δοθεί η σημασιολογία των πεδίων κεφαλίδας request, εάν όλα τα μέρη στην επικοινωνία τα αναγνωρίσουν ως πεδία κεφαλίδας request. Τα μη αναγνωρισμένα πεδία κεφαλίδας αντιμετωπίζονται ως πεδία κεφαλίδας οντότητας. [5]

2.5.7. Μηνύματα απόκρισης (Response)

Μετά τη λήψη και την ερμηνεία ενός μηνύματος request, ένας διακομιστής απαντά με ένα HTTP μήνυμα response.

Response = Status-Line * ((general-header | response-header | entity-header) CRLF)

CRLF

[message-body]

Η πρώτη γραμμή ενός response είναι η Status-Line, που αποτελείται από την έκδοση του πρωτοκόλλου ακολουθούμενη από έναν αριθμητικό κωδικό κατάστασης και τη

σχετική φράση κειμένου, με κάθε στοιχείο διαχωρισμένο με χαρακτήρες SP. Δεν επιτρέπεται CR ή LF εκτός από την τελική ακολουθία CRLF. [5]

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

2.5.7.1. Πεδία κεφαλίδας Απόκρισης (Response Header Fields)

Τα πεδία κεφαλίδας απόκρισης επιτρέπουν στον διακομιστή να μεταβιβάζει πρόσθετες πληροφορίες σχετικά με το response που δεν μπορούν να τοποθετηθούν στο Status-Line. Αυτά τα πεδία κεφαλίδας παρέχουν πληροφορίες για τον server και για περαιτέρω πρόσβαση στον πόρο που προσδιορίζεται από το Request-URI.

Response-header:

Accept-Ranges

Age

ETag

Location

Proxy-Authenticate

Retry-After

Server

Vary

WWW-Authenticate

Όπως και με τα ονόματα των πεδίων κεφαλίδας request, έτσι και τα ονόματα των πεδίων κεφαλίδας response μπορούν να επεκταθούν αξιόπιστα μόνο σε συνδυασμό με μια

αλλαγή στην έκδοση πρωτοκόλλου. Ωστόσο, στα νέα ή πειραματικά πεδία κεφαλίδας μπορεί να δοθεί η σημασιολογία των πεδίων κεφαλίδας response, εάν όλα τα μέρη στην επικοινωνία τα αναγνωρίσουν ως πεδία κεφαλίδας response. Τα μη αναγνωρισμένα πεδία κεφαλίδας αντιμετωπίζονται ως πεδία κεφαλίδας οντότητας. [5]

2.5.8. Ο κώδικας κατάστασης (Status Code) και η φράση αιτιολογίας (Reason Phrase)

Το στοιχείο Status-Code είναι ένας 3ψήφιος ακέραιος κωδικός αποτελέσματος της προσπάθειας κατανόησης και ικανοποίησης του αιτήματος. Η Reason-Phrase προορίζεται να δώσει μια σύντομη κειμενική περιγραφή του Status-Code. Ο Status-Code προορίζεται για αυτόματη χρήση και η Reason-Phrase προορίζεται για τον άνθρωπο χρήστη. Ο client δεν απαιτείται να εξετάσει ή να εμφανίσει την Reason-Phrase.

Το πρώτο ψηφίο του Status-Code ορίζει την κλάση response. Τα δύο τελευταία ψηφία δεν έχουν κανένα ρόλο κατηγοριοποίησης. Υπάρχουν 5 τιμές για το πρώτο ψηφίο:

- 1xx: Informational – Έγινε λήψη request, συνεχιζόμενη διαδικασία
- 2xx: Success – Η ενέργεια ελήφθη με επιτυχία, κατανοήθηκε και έγινε αποδεκτή
- 3xx: Redirection – Πρέπει να γίνουν περαιτέρω ενέργειες για να ολοκληρωθεί το request.
- 4xx: Client Error – Το request περιέχει κακή σύνταξη ή δεν μπορεί να εκπληρωθεί.
- 5xx: Server Error – Ο server επέτυχε να εκπληρώσει ένα φαινομενικά έγκυρο request.

Οι επιμέρους τιμές των αριθμητικών κωδικών κατάστασης που ορίζονται για το HTTP/1.1 παρουσιάζονται παρακάτω. Οι Reason-Phrases που παρατίθενται εδώ είναι οι προτεινόμενες, δηλαδή μπορούν να αντικατασταθούν από τοπικά ισοδύναμες χωρίς να επηρεαστεί το πρωτόκολλο.

"Status Code": Reason Phrase

"100": Continue

"101": Switching Protocols

"200": OK

"201": Created

"202": Accepted

"203": Non-Authoritative Information

"204": No Content

"205": Reset Content

"206": Partial Content

"300": Multiple Choices

"301": Moved Permanently

"302": Found

"303": See Other

"304": Not Modified

"305": Use Proxy

"307": Temporary Redirect

"400": Bad Request

"401": Unauthorized

"402": Payment Required

"403": Forbidden

"404": Not Found

"405": Method Not Allowed

"406": Not Acceptable

"407": Proxy Authentication Required

"408": Request Time-out

"409": Conflict

"410": Gone

"411": Length Required

"412": Precondition Failed

"413": Request Entity Too Large

"414": Request-URI Too Large

"415": Unsupported Media Type

"416": Requested range not satisfiable

"417": Expectation Failed

"500": Internal Server Error

"501": Not Implemented

"502": Bad Gateway

"503": Service Unavailable

"504": Gateway Time-out

"505": HTTP Version not supported

Οι κωδικοί κατάστασης HTTP είναι επεκτάσιμοι. Οι εφαρμογές HTTP δεν απαιτείται να κατανοούν την έννοια όλων των καταχωρημένων κωδικών κατάστασης, αν και αυτή η κατανόηση είναι προφανώς επιθυμητή. Ωστόσο, οι εφαρμογές επιβάλλεται να κατανοούν την κλάση οποιουδήποτε κωδικού κατάστασης, όπως υποδεικνύεται από το πρώτο ψηφίο,

και να αντιμετωπίζουν οποιαδήποτε μη αναγνωρισμένη απόκριση ως ισοδύναμη με τον κωδικό κατάστασης x00 αυτής της κατηγορίας, με την εξαίρεση ότι μια μη αναγνωρισμένη απάντηση απαγορεύεται να αποθηκευτεί προσωρινά. Για παράδειγμα, εάν ληφθεί ένας μη αναγνωρισμένος κωδικός κατάστασης 431 από τον client, μπορεί με ασφάλεια να υποθέσει ότι κάτι δεν πήγαινε καλά με το request του και να αντιμετωπίσει το response σαν να είχε λάβει κωδικό κατάστασης 400. Σε τέτοιες περιπτώσεις, οι user agents πρέπει να παρουσιάζουν στον χρήστη την οντότητα που επέστρεψε με την απάντηση, καθώς αυτή η οντότητα είναι πιθανό να περιλαμβάνει πληροφορίες αναγνώσιμες από τον χρήστη που θα εξηγούν την ασυνήθιστη κατάσταση. [5]

2.6. Caching / Proxies

2.6.1. Caching

Η προσωρινή αποθήκευση ή caching είναι μια τεχνική που χρησιμοποιείται για την αποθήκευση ενός αντιγράφου ενός δεδομένου πόρου, ώστε να μπορεί να επιστρέφεται όταν ζητηθεί ξανά. Υπάρχουν κυρίως δύο λόγοι για το caching δεδομένων: η βελτίωση της απόδοσης και η βελτίωση της αξιοπιστίας. Η αναπαραγωγή δεδομένων δημιουργεί πρόβλημα συνέπειας κάθε φορά που ενημερώνεται ένα αντίγραφο.

Ένα cache Ιστού μπορεί να είναι είτε ιδιωτικό είτε κοινόχρηστο. Τα ιδιωτικά cache προορίζονται για την εξοικονόμηση πόρων για έναν μόνο χρήστη, όπως το cache ενός προγράμματος περιήγησης. Το κοινόχρηστο cache αποθηκεύει responses για πολλούς χρήστες. Η συνέπεια σε ένα cache Ιστού είναι εύκολο να εφαρμοστεί καθώς κανένα αντίγραφο δεν αλλάζει τα δεδομένα. Μόνο τα αρχικά δεδομένα που βρίσκονται στον server Ιστού αλλάζουν.

Δεν είναι δυνατό για ένα cache Ιστού να έχει κάνει cache κάθε ιστοσελίδα, επειδή η μνήμη είναι περιορισμένη. Για αυτόν τον λόγο, κάνει cache δεδομένα μόνο για τους server ιστού που ζητούνται ανά πάσα στιγμή. Όταν ζητούνται δεδομένα που δεν έχουν γίνει cache, γίνεται το λεγόμενο cache miss, ή αλλιώς απώλεια προσωρινής αποθήκευσης. Cache miss εμφανίζεται επίσης όταν τα δεδομένα γίνονται cache, αλλά πρέπει να ανανεωθούν. Σε ένα cache miss τα δεδομένα πρέπει να επιστραφούν από το server. Όταν παρουσιαστεί cache miss και το cache είναι πλήρες, το response πρέπει να αντικατασταθεί. [3]

2.6.2. Proxy Caching

Μία δημοφιλής τεχνική που αποσκοπεί στη βελτίωση της απόδοσης του Παγκόσμιου Ιστού είναι η proxy caching, στην οποία ένας ή περισσότεροι υπολογιστές λειτουργούν ως cache εγγράφων για ένα σύνολο client του Παγκόσμιου Ιστού. Αυτοί οι clients έχουν ρυθμιστεί ώστε να στέλνουν HTTP requests στον proxy. Εάν είναι δυνατό, ο proxy εξυπηρετεί τα requests από το δικό του cache. Εάν όχι, ο proxy προωθεί το request στον αντίστοιχο πάροχο του περιεχομένου, δηλαδή στον server που περιέχει το πηγαίο αντίγραφο των ζητούμενων δεδομένων.

Το proxy caching επιχειρεί να βελτιώσει την απόδοση του Παγκόσμιου Ιστού με τρείς τρόπους:[4]

1. Προσπαθεί να μειώσει το latency που αντιλαμβάνεται ο χρήστης, το οποίο σχετίζεται με τη λήψη εγγράφων Ιστού. Αυτό επιτυγχάνεται επειδή ο proxy συνήθως βρίσκεται πιο κοντά στον client από ότι ο κύριος πάροχος του περιεχομένου.
2. Επιχειρεί να μειώσει την κυκλοφορία δικτύου από τους servers Ιστού. Ο φόρτος δικτύου μπορεί να μειωθεί επειδή τα έγγραφα που εξυπηρετούνται από τη μνήμη cache συνήθως διασχίζουν λιγότερο από το δίκτυο από ό,τι όταν εξυπηρετούνται από τον κύριο πάροχο περιεχομένου.
3. Μπορεί να μειώσει τις απαιτήσεις υπηρεσιών στους παρόχους περιεχομένου, καθώς οι επισκέψεις στην κρυφή μνήμη δεν χρειάζεται να τον περιλαμβάνουν. Μπορεί επίσης να μειώσει το κόστος μεταφοράς για τους παρόχους πρόσβασης.

3. Το HTTP/2

3.1. Ιστορία του SPDY και του HTTP/2

Το HTTP είναι μια επιτυχημένη τεχνολογία Διαδικτύου πάνω στην οποία βασίζεται ένα μεγάλο μέρος του Παγκόσμιου Ιστού. Ωστόσο, οι περιορισμοί με τις τρέχουσες προδιαγραφές του έχουν ενθαρρύνει ορισμένους να αναζητήσουν την επόμενη γενιά HTTP. Επίσης το αυξανόμενο μέγεθος και η περιπλοκότητα των ιστοσελίδων έκαναν την πλοήγηση στο Διαδίκτυο όλο και πιο απαιτητική. Σύμφωνα με το HTTP Archive, ο μέσος όρος μεγέθους μίας ιστοσελίδας ξεπέρασε το 1MB τον Απρίλιο του 2012, και τον Ιανουάριο του 2014 μία επίσκεψη σε μία από τις χίλιες κορυφαίες ιστοσελίδες ξεπέρασε κατά μέσο όρο το 1.5MB. Αυτή η αύξηση στο μέγεθος και την πολυπλοκότητα των ιστοσελίδων μπορεί να μειώσει δραματικά την ανάκτηση μίας σελίδας, με μεγαλύτερους

ηττημένους τις εμπορικές ιστοσελίδες, οι οποίες παρατήρησαν πτώση στις πωλήσεις τους μέσω Διαδικτύου λόγω της μεγάλης καθυστέρησης στο φόρτωμα της ιστοσελίδας.

Για να μειωθεί αυτός ο χρόνος αναμονής, προτάθηκαν διάφορες επεκτάσεις για το HTTP. Στην πράξη όμως φάνηκε ότι δεν είχε γίνει αρκετή πρόοδος, με πολλούς διακομιστές, μεσολαβητές και περιηγητές να καθυστερούν να υιοθετήσουν αυτές τις νέες αλλαγές. Τη λύση στο πρόβλημα έρχεται να δώσει η πασίγνωστη πλέον εταιρία Google, με την νέα της πρόταση και στη συνέχεια υλοποίηση ενός δίαυλου για τη μεταφορά μηνυμάτων HTTP, επονομαζόμενο ως SPDY, το οποίο αρχικά έκανε την εμφάνισή του το 2009. Με τη μεγάλη απήχηση που είχε στο κοινό, και αφού υιοθετήθηκε από την IETF (Internet Engineering Task Force), το SPDY αποτέλεσε τη βάση για την επόμενη έκδοση του πρωτοκόλλου HTTP, το HTTP/2.0.

Το SPDY είναι ένα πειραματικό πρωτόκολλο σε επίπεδο εφαρμογής το οποίο χρησιμοποιεί πλήρως τη σημασιολογία του HTTP. Ως εκ τούτου, διατηρεί όλα τα χαρακτηριστικά, συμπεριλαμβανομένων των cookies, των ETag και των διαπραγματεύσεων για την κωδικοποίηση περιεχομένου. Οι κύριοι στόχοι που ήλπιζε να πετύχει το SPDY είναι οι ακόλουθοι:

- 50% μείωση στο χρόνο φόρτωσης σελίδας.
- Αποφυγή της ανάγκης για οποιεσδήποτε αλλαγές στο περιεχόμενο από τους συντάκτες της ιστοσελίδας.
- Ελαχιστοποίηση της πολυπλοκότητας ανάπτυξης και αποφυγή αλλαγών στην υποδομή δικτύου.
- Ανάπτυξη αυτού του πρωτοκόλλου σε συνεργασία με την κοινότητα ανοιχτού κώδικα (open-source code).
- Συγκέντρωση πραγματικών δεδομένων απόδοσης με σκοπό την επικύρωση ή όχι αυτού του πειραματικού πρωτοκόλλου.

Τα αρχικά αποτελέσματα ήταν αρκετά ενθαρρυντικά σύμφωνα με τους μηχανικούς πληροφορικής της Google. Σε περιβάλλον εργαστηρίου, όταν έγινε απόπειρα λήψης των κορυφαίων 25 ιστοσελίδων μέσω ενός προσομοιωμένου οικιακού δικτύου, παρατηρήθηκε σημαντική βελτίωση στην απόδοση, με σελίδες να φορτώνουν έως και 55% πιο γρήγορα.

Το SPDY θέλει να αλλάξει τον τρόπο με τον οποίο τα δεδομένα γράφονται στο δίκτυο, με σκοπό να μειώσει το χρόνο απόκρισης, και το πετυχαίνει εισάγοντας τους παρακάτω μηχανισμούς:

- *Multiplexing*: Ένα επίπεδο πλαισίωσης το οποίο πολυπλέκει τις ροές σε μια μόνο σύνδεση, καταργώντας την ανάγκη δημιουργίας ξεχωριστών συνδέσεων TCP για τη μεταφορά διαφορετικών διαδικτυακών πόρων.
- *Compression*: Όλα τα δεδομένα κεφαλίδας συμπιέζονται για να μειωθούν τα γενικά έξοδα πολλών σχετικών αιτημάτων.
- *Universal encryption*: Το SPDY διαπραγματεύεται μέσω SSL/TLS και συνεπώς λειτουργεί αποκλειστικά μέσω ενός ασφαλούς καναλιού προκειμένου να αντιμετωπίσει την αυξανόμενη κίνηση δεδομένων που αποστέλλονται μέσω μη ασφαλών διαδρομών, όπως π.χ. το δημόσιο Wi-Fi.
- *Server Push/Hint*: Οι διακομιστές μπορούν να προωθήσουν προληπτικά πόρους στους clients (π.χ. εικόνες ή scripts που θα απαιτηθούν). Εναλλακτικά, το SPDY μπορεί να στείλει υποδείξεις, συμβουλεύοντας τους clients να λάβουν εκ των προτέρων τυχών απαιτούμενο περιεχόμενο.
- *Content prioritisation*: Ένας client μπορεί να καθορίσει την προτιμώμενη σειρά με την οποία θα πρέπει να μεταφερθούν οι πόροι.

Το SPDY αποτελείται από δύο μέρη. Το πρώτο μέρος παρέχει πλαισίωση των δεδομένων, επιτρέποντας έτσι κινήσεις όπως η συμπίεση (compression) και πολυπλεξία (multiplexing). Το επίπεδο πλαισίωσης λειτουργεί πάνω από ασφαλείς (SSL/TLS) μόνιμες συνδέσεις TCP, οι οποίες διατηρούνται ζωντανές όσο βρίσκονται ανοιχτές οι αντίστοιχες ιστοσελίδες. Clients και servers ανταλλάσσουν πλαίσια ελέγχου και δεδομένων, τα οποία περιέχουν μια κεφαλίδα 8 byte. Πλαίσια ελέγχου χρησιμοποιούνται για τη μεταφορά σημάτων σχετικά με τη διαχείριση σύνδεσης και για επιλογές διαμόρφωσης, ενώ τα πλαίσια δεδομένων μεταφέρουν αιτήματα και αποκρίσεις HTTP. Το δεύτερο μέρος αντιστοιχίζει την επικοινωνία HTTP σε πλαίσια δεδομένων SPDY. Πολλαπλές λογικές ροές HTTP μπορούν να πολυπλέκονται χρησιμοποιώντας παρεμβαλλόμενα πλαίσια δεδομένων σε μία μόνο σύνδεση TCP. [8]

3.1.1. Από το SPDY στο HTTP/2

Προχωρώντας λοιπόν στο 2012, αυτό το νέο πειραματικό πρωτόκολλο υποστηρίζεται σε κάποιους από τους μεγαλύτερους browsers όπως οι Chrome, Firefox και Opera, και ένας ταχέως αυξανόμενος αριθμό ιστοσελίδων τόσο μεγάλων (π.χ. Google, Twitter, Facebook) όσο και μικρότερων, αναπτύσσουν το SPDY στην υποδομή τους. Στην πραγματικότητα, το SPDY βρισκόταν στην πορεία να γίνει το πρότυπο μέσω της αυξανόμενης υιοθέτησης του από τον κλάδο της πληροφορικής.

Παρατηρώντας την παραπάνω τάση, η Ομάδα Εργασίας HTTP (HTTP-WG) ξεκίνησε μια νέα προσπάθεια για να πάρει τα μαθήματα που μας δίδαξε το SPDY, να τα αναπτύξει και να τα βελτιώσει περεταίρω και να παραδώσει ένα επίσημο πρότυπο “HTTP/2.0”:

Χαρτογραφήθηκε μια νέα πορεία, πραγματοποιήθηκε μια ανοιχτή πρόσκληση για προτάσεις πάνω στο HTTP/2.0 και μετά από πολλή συζήτηση στην ομάδα εργασίας, οι προδιαγραφές του SPDY υιοθετήθηκαν ως σημείο εκκίνησης για το νέο πρωτόκολλο HTTP/2.

Στα χρόνια που ακολούθησαν το SPDY και το HTTP/2 θα συνέχιζαν να εξελίσσονται παράλληλα, με το SPDY να λειτουργεί ως πειραματικό μέσο που χρησιμοποιήθηκε για δοκιμή νέων χαρακτηριστικών και προτάσεων για το HTTP/2: αυτό που στη θεωρία φαίνεται καλό μπορεί να μη λειτουργεί στην πράξη και αντίστροφα. Με αυτό τον τρόπο το SPDY αποτέλεσε τον τρόπο δοκιμής και αξιολόγησης της κάθε πρότασης πριν την συμπερίληψή της στο πρότυπο HTTP/2. Τελικά, αυτή η διαδικασία κράτησε τρία χρόνια και κατέληξε σε πάνω από δώδεκα ενδιάμεσα σχέδια:

- Μάρτιος 2012: Πρόσκληση για προτάσεις πάνω στο HTTP/2
- Νοέμβριος 2012: Πρώτο σχέδιο του HTTP/2 (βασισμένο στο SPDY)
- Αύγουστος 2014: Εκδίδονται τα HTTP/2 σχέδιο-17 και HPACK σχέδιο-12
- Αύγουστος 2014: Τελευταία πρόσκληση της Ομάδας Εργασίας για το HTTP/2
- Φεβρουάριος 2015: Το IESG (Internet Engineering Steering Group) ενέκρινε τα τελικά σχέδια των HTTP/2 και HPACK.
- Μάιος 2015: Εκδίδονται τα RFC 7540 (HTTP/2) και RFC 7541 (HPACK)

Η ταυτόχρονη εξέλιξη του SPDY και του HTTP/2 επέτρεψε στους προγραμματιστές διακομιστών, προγραμμάτων περιήγησης και ιστοτόπων να αποκτήσουν πραγματική εμπειρία με το νέο πρωτόκολλο κατά τη διάρκεια της ανάπτυξής του. Αυτό είχε ως αποτέλεσμα το πρότυπο HTTP/2 να είναι ένα από τα καλύτερα και πιο εκτενώς ελεγμένα πρότυπα από το ξεκίνημά του. Μέχρι τη στιγμή που το HTTP/2 εγκρίθηκε από το IESG, υπήρχαν ήδη δεκάδες πλήρως ελεγμένες και έτοιμες για παραγωγή εφαρμογές client και server. Στην πραγματικότητα, μόλις εβδομάδες μετά την έγκριση του τελικού πρωτοκόλλου, πολλοί χρήστες απολάμβαναν ήδη τα πλεονεκτήματά του καθώς ήδη πολλά προγράμματα περιήγησης και πολλοί ιστότοποι ανέπτυξαν πλήρη υποστήριξη για το HTTP/2. [8]

3.1.2. Η αναβάθμιση σε HTTP/2

Η αλλαγή στο HTTP/2 δε γίνεται να συμβεί σε σύντομο χρονικό διάστημα. Εκατομμύρια διακομιστές πρέπει να αναβαθμιστούν για να χρησιμοποιήσουν το νέο δυαδικό πλαισίωμα, και δισεκατομμύρια clients πρέπει με παρόμοιο τρόπο να

ενημερώσουν τις βιβλιοθήκες δικτύωσης, τα προγράμματα περιήγησης και άλλες εφαρμογές τους.

Τα καλά νέα είναι ότι όλα τα σύγχρονα προγράμματα περιήγησης έχουν δεσμευτεί να υποστηρίξουν το HTTP/2, και τα περισσότερα από αυτά χρησιμοποιούν αποτελεσματικούς παρασκηνιακούς μηχανισμούς ενημέρωσης , οι οποίοι έχουν ήδη ενεργοποιήσει την υποστήριξη του HTTP/2 με ελάχιστη παρέμβαση για μεγάλο ποσοστό υπαρχόντων χρηστών. Παρ όλα αυτά, ορισμένοι χρήστες θα κολλήσουν σε περιηγητές παλαιού τύπου και οι διακομιστές και οι μεσάζοντες θα πρέπει επίσης να ενημερωθούν για να υποστηρίζουν το HTTP/2, η οποία είναι μια πολύ μεγαλύτερη (και έντασης εργασίας και κεφαλαίου) διαδικασία.

Το HTTP/1.x θα είναι σε χρήση για τουλάχιστον άλλη μία δεκαετία και οι περισσότεροι servers και clients θα πρέπει να υποστηρίζουν τόσο το πρότυπα HTTP/1.x όσο και το HTTP/2. Ως αποτέλεσμα, ένας HTTP/2 client και server θα πρέπει να μπορεί να ανακαλύψει και να ξεχωρίσει ποιο πρωτόκολλο θα χρησιμοποιηθεί πριν από την ανταλλαγή δεδομένων. Για να αντιμετωπιστεί αυτό, το πρωτόκολλο HTTP/2 ορίζει τους ακόλουθους μηχανισμούς:

1. Διαπραγμάτευση HTTP/2 μέσω μιας ασφαλούς σύνδεσης με TLS και ALPN
2. Αναβάθμιση μιας σύνδεσης απλού κειμένου σε HTTP/2 χωρίς προηγούμενη γνώση
3. Εκκίνηση μιας σύνδεσης HTTP/2 απλού κειμένου με προηγούμενη γνώση.

Το πρότυπο HTTP/2 δεν απαιτεί τη χρήση TLS, αλλά πρακτικά είναι ο πιο αξιόπιστος τρόπος για την ανάπτυξη ενός νέου πρωτοκόλλου στην παρουσία ενός μεγάλου αριθμού ήδη υπαρχόντων διαμεσολαβητών. Ως αποτέλεσμα, η χρήση του TLS και του ALPN είναι ο προτεινόμενος μηχανισμός για τη χρήση και τη διαπραγμάτευση του HTTP/2: ο client και ο server διαπραγματεύονται το επιθυμητό πρωτόκολλο ως μέρος της χειραψίας TLS χωρίς να προσθέτουν επιπλέον latency ή διαδρομές μετ' επιστροφής. Επιπλέον, ως πρόσθετος περιορισμός, ενώ όλα τα προγράμματα περιήγησης έχουν δεσμευτεί να υποστηρίζουν το HTTP/2 μέσω TLS, ορισμένα έχουν επίσης υποδείξει ότι θα ενεργοποιήσουν το HTTP/2 αποκλειστικά μέσω TLS-π.χ. τα Firefox και Chrome. Αυτό έχει ως αποτέλεσμα το TLS με διαπραγμάτευση ALPN να είναι ουσιαστικά μία απαίτηση για την ενεργοποίηση του HTTP/2 στο πρόγραμμα περιήγησης. [8]

3.1.3. Σχεδιασμός και τεχνικοί στόχοι

Οι πρώτες εκδόσεις του πρωτοκόλλου HTTP σχεδιάστηκαν με σκοπό την απλότητα υλοποίησης: Το HTTP/0.9 ήταν ένα one-line πρωτόκολλο για την εκκίνηση του Παγκόσμιου Ιστού. Το HTTP/1.0 τεκμηρίωσε τις δημοφιλείς επεκτάσεις του HTTP/0.9 σε ένα ενημερωτικό πρότυπο. Το HTTP/1.1 εισήγαγε ένα επίσημο IETF πρότυπο. Ως εκ τούτου, το HTTP/0.9-1.x παρέδωσε ακριβώς αυτό που είχε σκοπό να κάνει: Το HTTP είναι πλέον ένα από τα πιο πανταχού παρόντα και ευρέως διαδεδομένα πρωτόκολλα εφαρμογών στο Διαδίκτυο.

Δυστυχώς, η απλότητα υλοποίησης ήρθε με κόστος την απόδοση της εφαρμογής, όπως:

- Οι HTTP/1.x clients πρέπει να χρησιμοποιούν πολλαπλές συνδέσεις για να επιτύχουν ταυτοχρονισμό και να μειώσουν τον λανθάνοντα χρόνο.
- Το HTTP/1.x δεν συμπιέζει τις κεφαλίδες αιτημάτων και απαντήσεων, προκαλώντας έτσι περιττή κίνηση δικτύου.
- Το HTTP/1.x δεν επιτρέπει την αποτελεσματική ιεράρχηση πόρων, με αποτέλεσμα την κακή χρήση της υποκείμενης σύνδεσης TCP.

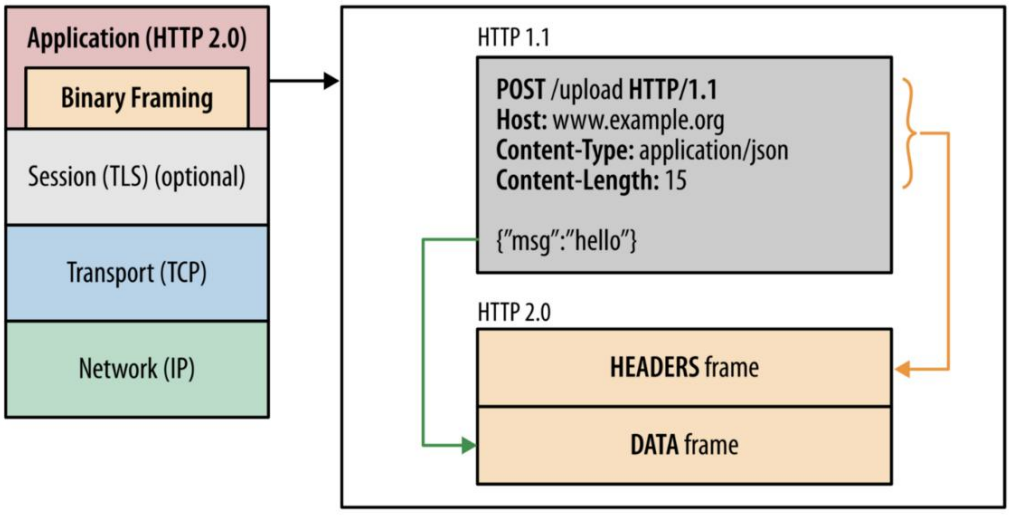
Και ούτω καθεξής.

Αυτοί οι περιορισμοί δεν ήταν μοιραίοι, αλλά καθώς οι εφαρμογές ιστού συνέχισαν να αυξάνονται ως προς το εύρος, την πολυπλοκότητα και τη σημασία τους στην καθημερινότητά μας, επέβαλαν ένα αυξανόμενο βάρος τόσο στους προγραμματιστές όσο και στους χρήστες του ιστού, το οποίο είναι και το ακριβές κενό που το HTTP/2 ήταν σχεδιασμένο να απευθύνεται.

Είναι σημαντικό να σημειωθεί ότι το HTTP/2 επεκτείνει, δεν αντικαθιστά τα προηγούμενα πρότυπα HTTP. Η σημασιολογία της εφαρμογής του HTTP είναι η ίδια και δεν έγιναν αλλαγές στην προσφερόμενη λειτουργικότητα ή στις βασικές έννοιες, όπως τις HTTP μεθόδους, τους κωδικούς κατάστασης, τα URI και τα πεδία κεφαλίδων. Οι αλλαγές αυτές ήταν ρητά εκτός πεδίου εφαρμογής για την όλη HTTP/2 διαδικασία. Παρόλα αυτά, ενώ το υψηλού επιπέδου API(Application Programming Interface) παραμένει το ίδιο, είναι σημαντικό να κατανοήσουμε πώς οι χαμηλού επιπέδου αλλαγές αντιμετωπίζουν τους περιορισμούς απόδοσης των προηγούμενων πρωτοκόλλων. Ας κάνουμε μια σύντομη περιήγηση στο επίπεδο δυαδικού πλαισίου και στα χαρακτηριστικά του. [8]

3.2. Επίπεδο δυαδικού πλαισίου (Binary Framing Layer)

Στον πυρήνα όλων των βελτιώσεων απόδοσης του HTTP/2 βρίσκεται το νέο επίπεδο δυαδικού πλαισίου που υπαγορεύει τον τρόπο ενθυλάκωσης και μεταφοράς των μηνυμάτων HTTP μεταξύ του client και του server.



Εικόνα 3.1. Binary Framing Layer

(Πηγή: https://hpbn.co/http2/)

Το "επίπεδο" αναφέρεται σε μια σχεδιαστική επιλογή για την εισαγωγή ενός βελτιστοποιημένου μηχανισμού κωδικοποίησης μεταξύ της διεπαφής υποδοχής και του υψηλότερου HTTP API που εκτίθεται στις εφαρμογές μας: η σημασιολογία του HTTP, όπως τα ρήματα, οι μέθοδοι και οι κεφαλίδες δεν επηρεάζονται, αλλά ο τρόπος με τον οποίο κωδικοποιούνται κατά τη διάρκεια της μετάβασης είναι αυτό που διαφέρει. Σε αντίθεση με το οριοθετημένο πρωτόκολλο απλού κειμένου HTTP/1.x, όλη η επικοινωνία μέσω HTTP/2 χωρίζεται σε μικρότερα μηνύματα και πλαίσια, καθένα από τα οποία κωδικοποιείται σε δυαδική μορφή.

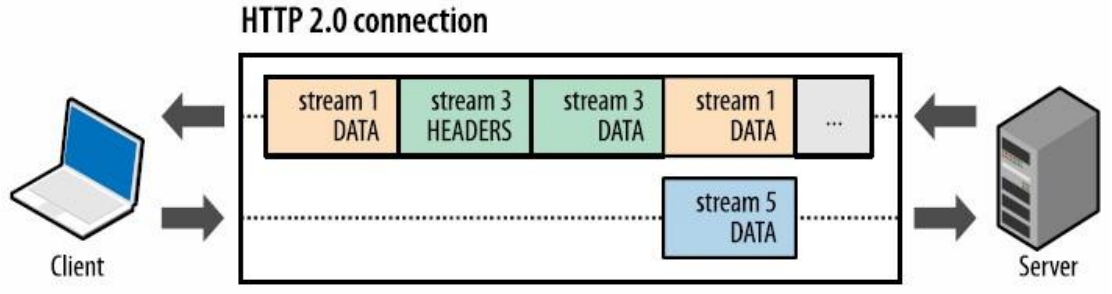
Ως αποτέλεσμα, τόσο ο client όσο και ο server πρέπει να χρησιμοποιούν τον νέο μηχανισμό δυαδικής κωδικοποίησης για να κατανοούν ο ένας τον άλλο: ένας HTTP/1.x client δε θα κατανοεί έναν αποκλειστικά HTTP/2 server, και το αντίστροφο. Ευτυχώς, οι

εφαρμογές μας παραμένουν αγνώμονες σε όλες αυτές τις αλλαγές, καθώς οι client και server εκτελούν όλες αυτές τις απαραίτητες εργασίες πλαισίωσης για λογαριασμό μας. [8]

3.3. Πολυπλεξία αιτήματος και απόκρισης (Request και Response Multiplexing)

Με το HTTP/1.x, εάν ο client θέλει να κάνει πολλαπλά παράλληλα αιτήματα για να βελτιώσει την απόδοση, τότε πρέπει να χρησιμοποιηθούν πολλαπλές συνδέσεις TCP. Αυτή η συμπεριφορά είναι άμεση συνέπεια του μοντέλου παράδοσης HTTP/1.x, το οποίο διασφαλίζει ότι μπορεί να παραδοθεί μόνο μια απόκριση κάθε φορά (response queuing) ανά σύνδεση. Ακόμη χειρότερα, αυτό έχει ως αποτέλεσμα τον αποκλεισμό της γραμμής κεφαλής (head-of-line blocking ή HOL blocking) και την αναποτελεσματική χρήση της υποκείμενης σύνδεσης TCP.

Το νέο επίπεδο δυαδικού πλαισίου στο HTTP/2 καταργεί αυτούς τους περιορισμούς και επιτρέπει την πλήρη πολυπλεξία αιτημάτων και απόκρισης, επιτρέποντας στον client και τον server να αναλύσουν ένα μήνυμα HTTP σε ανεξάρτητα πλαίσια, να τα παρεμβάλουν και στη συνέχεια να τα συναρμολογήσουν ξανά στο άλλο άκρο.



Εικόνα 3.2. Request and Response Multiplexing

(Πηγή: https://hpbn.co/http2/)

Το παραπάνω στιγμιότυπο καταγράφει πολλαπλές ροές κατά την πτήση εντός της ίδιας σύνδεσης: ο client μεταδίδει ένα πλαίσιο DATA (stream 5) στον server, ενώ ο server μεταδίδει μια αλληλοδιαπλεκόμενη ακολουθία πλαισίων στον client για τα stream 1 και 3. Ως αποτέλεσμα υπάρχουν τρία παράλληλα streams σε πτήση.

Η δυνατότητα διάσπασης ενός μηνύματος HTTP σε ανεξάρτητα πλαίσια, της παρεμβολής αυτών και στη συνέχεια επανασυναρμολόγησής τους στο άλλο άκρο είναι η μοναδική πιο σημαντική βελτίωση του HTTP/2. Στην πραγματικότητα, εισάγει ένα κυματιστικό αποτέλεσμα πολυάριθμων πλεονεκτημάτων απόδοσης σε ολόκληρη τη στοίβα όλων των τεχνολογιών ιστού, τα οποία μας δίνουν τη δυνατότητα να:

- Παρεμβάλλουμε πολλαπλά requests παράλληλα χωρίς να "μπλοκάρουμε" σε κανένα
- Παρεμβάλουμε πολλαπλά responses παράλληλα χωρίς να "μπλοκάρουμε" σε κανένα
- Χρησιμοποιούμε μία μόνο σύνδεση για να παραδίδουμε πολλαπλά requests και responses παράλληλα
- Καταργήσουμε περιττές HTTP/1.x λύσεις όπως συνενωμένα αρχεία, sprites εικόνας και κοινή χρήση τομέα (domain sharding)
- Παρέχουμε χαμηλότερους χρόνους φόρτωσης σελίδας (page load time) εξαλείφοντας τον περιττό λανθάνοντα χρόνο (latency) και βελτιώνοντας τη χρήση της διαθέσιμης χωρητικότητας δικτύου

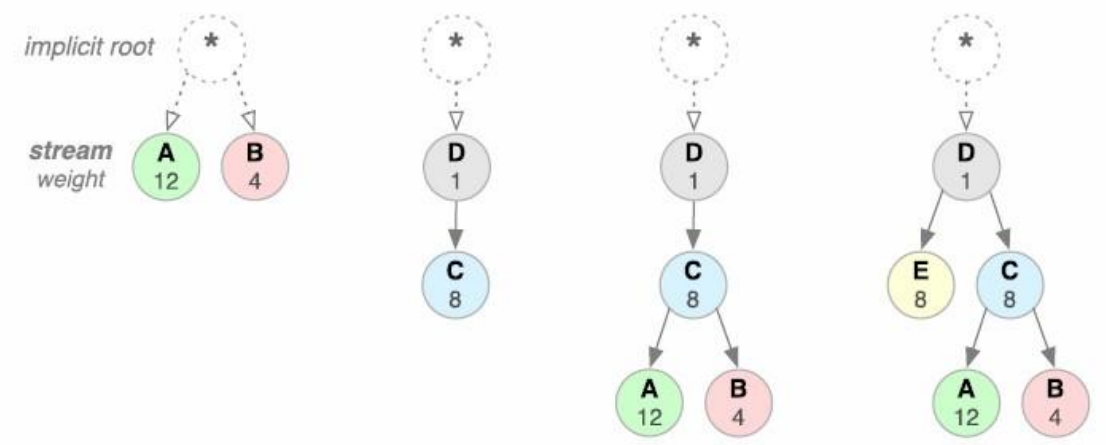
Το νέο επίπεδο δυαδικού πλαισίου στο HTTP/2 επιλύει το πρόβλημα head-of line αποκλεισμού που εντοπίζεται στο HTTP/1.x και εξαλείφει την ανάγκη για πολλαπλές συνδέσεις για να καταστεί δυνατή η παράλληλη επεξεργασία και παράδοση request και response. Ως αποτέλεσμα, αυτό κάνει τις εφαρμογές μας πιο γρήγορες, φθηνότερες στην ανάπτυξη και απλούστερες στη χρήση. [8]

3.4.Προτεραιότητα ροής (Stream Prioritization)

Μόλις ένα μήνυμα HTTP μπορεί να χωριστεί σε πολλά μεμονωμένα πλαίσια και επιτρέψουμε την πολυπλεξία πλαισίων από πολλαπλές ροές, η σειρά με την οποία τα πλαίσια παρεμβάλλονται και παραδίδονται τόσο από τον client όσο και από τον server γίνεται κρίσιμος παράγοντας απόδοσης. Για τη διευκόλυνσή μας, το πρότυπο HTTP/2 επιτρέπει σε κάθε ροή να έχει σχετικό βάρος και εξάρτηση:

- Σε κάθε ροή μπορεί να καταχωρηθεί ένα βάρος ακεραίου αριθμού μεταξύ 1 και 256
- Σε κάθε ροή μπορεί να δοθεί μια ρητή εξάρτηση από μια άλλη ροή

Ο συνδυασμός εξαρτήσεων και βαρών ροής επιτρέπει στον client να κατασκευάσει και να επικοινωνήσει ένα "δέντρο ιεραρχίας" που εκφράζει πώς θα προτιμούσε να λάβει τις απαντήσεις. Με τη σειρά του, ο server μπορεί να χρησιμοποιήσει αυτές τις πληροφορίες για να δώσει προτεραιότητα στην επεξεργασία ροής ελέγχοντας την κατανομή της CPU, της μνήμης και άλλων πόρων, και μόλις είναι διαθέσιμα τα δεδομένα απόκρισης, ελέγχει την κατανομή εύρους ζώνης για να εξασφαλιστεί η βέλτιστη παράδοση των αποκρίσεων υψηλής προτεραιότητας στον client.



Εικόνα 3.3. HTTP/2 Stream Dependencies and Weights

(Πηγή: https://hpbn.co/http2/)

Η εξάρτηση ροής εντός του HTTP/2 δηλώνεται με αναφορά στο μοναδικό αναγνωριστικό μιας άλλης ροής ως γονέα. Αν παραλειφθεί η ροή λέγεται ότι εξαρτάται από την "πηγαία ροή" ("root stream"). Η δήλωση εξάρτησης ροής υποδηλώνει ότι, εάν είναι δυνατό, στην γονική ροή θα πρέπει να κατανεμηθούν πόροι πριν από τις εξαρτήσεις της. Π.χ., η επεξεργασία και η παράδοση του D response να γίνει πριν του D response.

Στις ροές που μοιράζονται τον ίδιο γονέα (δηλαδή αδελφικές ροές) θα πρέπει να κατανέμονται πόροι ανάλογα με το βάρος τους. Για παράδειγμα, εάν η ροή A έχει βάρος 12 και η αδερφική ροή της, η B, έχει βάρος 4, τότε για να προσδιορίσουμε την αναλογία των πόρων θα πρέπει κάθε μια από αυτές τις ροές να λάβει:

1. Άθροισμα όλων των βαρών: 4 + 12 = 16

2. Διαίρεση κάθε βάρους ροής με το συνολικό βάρος: A =12/16, B = 4/16

Οπότε, η ροή A θα πρέπει να λάβει τα τρία τέταρτα και η ροή B θα πρέπει να λάβει το ένα τέταρτο των διαθέσιμων πόρων. Η ροή B θα πρέπει να λάβει το ένα τρίτο των πόρων που διατίθενται στη ροή A. Ας δούμε μερικά ακόμη πρακτικά παραδείγματα στο προηγούμενο σχήμα. Από τα αριστερά στα δεξιά:

1. Ούτε η ροή A ούτε η B καθορίζουν μια γονική εξάρτηση και λέγεται ότι εξαρτώνται από την "πηγαία ροή". Το A έχει βάρος 12 και το B έχει βάρος 4. Έτσι, βάσει αναλογικών βαρών: η ροή B πρέπει να λαμβάνει το ένα τρίτο των πόρων που διατίθενται για την ροή A.
2. Η ροή D εξαρτάται από την "πηγαία ροή". Η ροή C εξαρτάται από την ροή D. Επομένως, η ροή D θα πρέπει να λάβει πλήρη κατανομή πόρων πριν από την ροή C. Τα βάρη είναι ασήμαντα επειδή η εξάρτηση της ροής C μεταδίδει μια ισχυρότερη προτίμηση.
3. Η ροή D θα πρέπει να λάβει πλήρη κατανομή των πόρων πριν από την ροή C. Η ροή C θα πρέπει να λάβει πλήρη κατανομή των πόρων πριν από την ροή A και την ροή B. Η ροή B θα πρέπει να λάβει το ένα τρίτο των πόρων που διατίθενται στη ροή A.
4. Η ροή D θα πρέπει να λάβει πλήρη κατανομή των πόρων πριν από τις ροές E και C. Οι E και C ροές θα πρέπει να λάβουν ίση κατανομή πριν από τις ροές A και B. οι ροές A και B θα πρέπει να λάβουν αναλογική κατανομή με βάση το βάρος τους.

Όπως μας δείχνουν τα παραπάνω παραδείγματα, ο συνδυασμός εξαρτήσεων ροής και βαρών παρέχει μια εκφραστική γλώσσα για την ιεράρχηση πόρων, η οποία είναι ένα κρίσιμο χαρακτηριστικό για τη βελτίωση της απόδοσης περιήγησης όπου έχουμε πολλούς τύπους πόρων με διαφορετικές εξαρτήσεις και βάρη. Ακόμα καλύτερα, το πρωτόκολλο HTTP/2 επιτρέπει επίσης στον client να ενημερώνει αυτές τις προτιμήσεις ανά πάσα στιγμή, κάτι που επιτρέπει περαιτέρω βελτιστοποιήσεις στο πρόγραμμα περιήγησης. Π.χ. μπορούμε να αλλάξουμε εξαρτήσεις και να ανακατανείμουμε τα βάρη ως απόκριση στην αλληλεπίδραση του χρήστη και σε άλλα σήματα. [8]

3.5.Μια σύνδεση ανά προέλευση

Με τον νέο μηχανισμό δυαδικής πλαισίωσης σε εφαρμογή, το HTTP/2 δεν χρειάζεται πλέον πολλαπλές συνδέσεις TCP σε παράλληλες ροές πολυπλεξίας. Κάθε ροή χωρίζεται σε πολλά πλαίσια, τα οποία μπορούν να παρεμβληθούν και να ιεραρχηθούν. Ως αποτέλεσμα, όλες οι συνδέσεις HTTP/2 είναι μόνιμες και απαιτείται μόνο μια σύνδεση ανά προέλευση, γεγονός που προσφέρει πολλά πλεονεκτήματα απόδοσης.

Οι περισσότερες μεταφορές HTTP είναι σύντομες και εκρηκτικές, ενώ το TCP είναι βελτιστοποιημένο για μακροχρόνιες, μαζικές μεταφορές δεδομένων. Με την επαναχρησιμοποίηση της ίδιας σύνδεσης, το HTTP/2 μπορεί να κάνει πιο αποτελεσματική χρήση κάθε σύνδεσης TCP και επίσης να μειώσει σημαντικά τη συνολική επιβάρυνση του πρωτοκόλλου. Επιπλέον, η χρήση λιγότερων συνδέσεων μειώνει τη μνήμη και το αποτύπωμα επεξεργασίας κατά μήκος της πλήρους διαδρομής σύνδεσης (δηλαδή client, server προέλευσης, και όλους τους μεσάζοντες τους), γεγονός που μειώνει το συνολικό λειτουργικό κόστος και βελτιώνει τη χρήση και τη χωρητικότητα του δικτύου. Ως αποτέλεσμα, η μετάβαση στο HTTP/2 όχι μόνο θα πρέπει να βελτιώσει την καθυστέρηση του δικτύου, αλλά και να συμβάλει στη βελτίωση της απόδοσης και στη μείωση του λειτουργικού κόστους. [8]

3.6.Έλεγχος ροής (Flow control)

Ο έλεγχος ροής είναι ένας μηχανισμός που εμποδίζει τον αποστολέα να κατακλύσει Να αλλαχθεί σε "ροής"

τον παραλήπτη με δεδομένα που μπορεί να μην θέλει ή να μην είναι σε θέση να επεξεργαστεί: ο δέκτης μπορεί να είναι απασχολημένος, υπό μεγάλο φόρτο ή μπορεί να είναι πρόθυμος να διαθέσει μόνο ένα σταθερό ποσό πόρων για μια συγκεκριμένη ροή. Για παράδειγμα, ο client μπορεί να έχει ζητήσει μια μεγάλη ροή βίντεο με υψηλή προτεραιότητα, αλλά ο χρήστης έχει κάνει παύση το βίντεο και ο client θέλει τώρα να διακόψει ή να επιταχύνει την παράδοσή του από τον server για να αποφύγει την ανάκτηση και αποθήκευση περιττών δεδομένων στην προσωρινή μνήμη. Εναλλακτικά, ένας proxy server μπορεί να έχει γρήγορες downstream και αργές upstream συνδέσεις και παρόμοια θέλει να ρυθμίσει πόσο γρήγορα η downstream παραδίδει δεδομένα ώστε να ταιριάζει την ταχύτητα της upstream για να ελέγχει τη χρήση των πόρων της, και ούτω καθεξής.

Οι παραπάνω απαιτήσεις θυμίζουν τον έλεγχο ροής TCP, καθώς το πρόβλημα είναι ουσιαστικά πανομοιότυπο. Ωστόσο, επειδή οι ροές HTTP/2 πολυπλέκονται σε μια μόνο σύνδεση TCP, ο έλεγχος ροής TCP δεν είναι αρκετά λεπτομερής και δεν παρέχει τα απαραίτητα API σε επίπεδο εφαρμογής για τη ρύθμιση παράδοσης μεμονωμένων ροών. Για να το αντιμετωπίσει αυτό, το HTTP/2 παρέχει ένα σύνολο απλών δομικών στοιχείων που επιτρέπουν στον client και τον server να εφαρμόσουν τον δικό τους έλεγχο ροής, τόσο σε επίπεδο ροής όσο και σε επίπεδο σύνδεσης: [8]

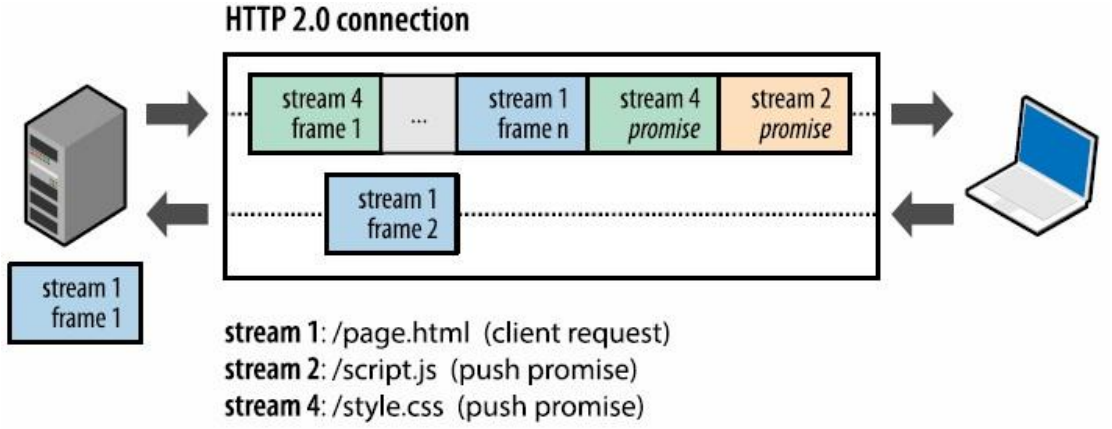
- Ο έλεγχος ροής είναι κατευθυντικός. Κάθε δέκτης μπορεί να ορίσει οποιοδήποτε μέγεθος παραθύρου επιθυμεί για κάθε ροή και για ολόκληρη την σύνδεση.
- Ο έλεγχος ροής βασίζεται σε πιστώσεις. Κάθε δέκτης διαφημίζει την αρχική του σύνδεση και το παράθυρο ελέγχου ροής (σε byte), το οποίο μειώνεται κάθε φορά που ο αποστολέας εκπέμπει ένα πλαίσιο DATA και αυξάνεται μέσω ενός πλαισίου WINDOW_UPDATE που αποστέλλεται από το δέκτη.
- Ο έλεγχος ροής δεν μπορεί να απενεργοποιηθεί. Όταν δημιουργηθεί η σύνδεση HTTP/2 ο client και ο server ανταλλάσσουν πλαίσια SETTINGS, τα οποία ορίζουν τα μεγέθη των παραθύρων ελέγχου ροής και προς τις δυο κατευθύνσεις. Η προεπιλεγμένη τιμή του παραθύρου ελέγχου ροής έχει οριστεί στα 65.535 byte, αλλά ο δέκτης μπορεί να ορίσει ένα μεγάλο μέγιστο μέγεθος παραθύρου (2^31 -1 bytes) και να το διατηρήσει στέλνοντας ένα πλαίσιο WINDOW_UPDATE κάθε φορά που λαμβάνονται δεδομένα.
- Ο έλεγχος ροής γίνεται από άλμα σε άλμα (hop-by-hop), όχι από άκρο σε άκρο (end-to-end). Δηλαδή, ένας ενδιάμεσος μπορεί να τον χρησιμοποιήσει για να ελέγξει τη χρήση πόρων και να εφαρμόσει μηχανισμούς κατανομής πόρων με βάση τα δικά του κριτήρια και ευρετικές μεθόδους.

Το HTTP/2 δεν καθορίζει κάποιον συγκεκριμένο αλγόριθμο για την υλοποίηση του ελέγχου ροής. Αντίθετα, παρέχει τα απλά δομικά στοιχεία και αναβάλλει την υλοποίηση στον client και τον server, οι οποίοι μπορούν να τον χρησιμοποιήσουν για να εφαρμόσουν προσαρμοσμένες στρατηγικές για τη ρύθμιση της χρήσης και κατανομής πόρων, καθώς και να εφαρμόσουν νέες δυνατότητες παράδοσης που μπορεί να βοηθήσουν στη βελτίωση τόσο της πραγματικής όσο και της αντιληπτής απόδοσης των διαδικτυακών μας εφαρμογών.

Για παράδειγμα, ο έλεγχος ροής σε επίπεδο εφαρμογής επιτρέπει στο πρόγραμμα περιήγησης να ανακτήσει μόνο ένα μέρος ενός συγκεκριμένου πόρου, να θέσει την ανάκτηση σε αναμονή μειώνοντας το παράθυρο ελέγχου ροής στο μηδέν και να το συνεχίσει αργότερα. Π.χ., να ανακτήσει μια προεπισκόπηση ή μια πρώτη σάρωση μιας εικόνας, να την εμφανίσει και να επιτρέψει σε άλλες ανακτήσεις υψηλής προτεραιότητας να προχωρήσουν και να συνεχίσει την ανάκτηση μόλις ολοκληρωθεί η φόρτωση άλλων σημαντικών πόρων. [8]

3.7.Προώθηση Server (Server Push)

Ένα άλλο ισχυρό χαρακτηριστικό του HTTP/2 είναι η δυνατότητά του server να στέλνει πολλαπλές απαντήσεις για ένα μόνο αίτημα πελάτη. Αυτό σημαίνει ότι, εκτός από την απάντηση στο αρχικό αίτημα, ο server μπορεί να προωθήσει επιπλέον πόρους στον client χωρίς αυτός να χρειάζεται να ζητήσει τον καθένα ξεχωριστά.



Εικόνα 3.4. Server initiates new streams (promises) for push resources

(Πηγή: https://hpbn.co/http2/)

Αυτός ο μηχανισμός χρειάζεται σε ένα πρόγραμμα περιήγησης επειδή μια τυπική διαδικτυακή εφαρμογή αποτελείται από δεκάδες πόρους, οι οποίοι ανακαλύπτονται όλοι από τον client εξετάζοντας το έγγραφο που παρέχεται από τον server. Οπότε, για να εξαλειφθεί το επιπλέον latency ο server προωθεί τους σχετικούς πόρους εκ των προτέρων, επειδή ήδη γνωρίζει ποιους πόρους θα χρειαστεί ο client. Αυτό είναι το Server Push. Με αυτή τη διαδικασία έρχονται και κάποια επιπρόσθετα οφέλη απόδοσης:

- Οι προωθημένοι πόροι μπορούν να αποθηκευτούν προσωρινά (cache) από τον client
- Οι προωθημένοι πόροι μπορούν να επαναχρησιμοποιηθούν σε διαφορετικές σελίδες
- Οι προωθημένοι πόροι μπορούν να πολυπλεκτούν μαζί με άλλους πόρους
- Οι προωθημένοι πόροι μπορούν να τεθούν σε προτεραιότητα από τον server
- Οι προωθημένοι πόροι μπορούν να απορριφθούν από τον client

Κάθε προωθημένος πόρος είναι μια ροή που, σε αντίθεση με έναν ενσωματωμένο πόρο, του επιτρέπει την ατομική πολυπλεξία, την ιεράρχηση και την επεξεργασία του από τον client. Ο μόνος περιορισμός ασφαλείας, όπως επιβάλλεται από το πρόγραμμα περιήγησης, είναι ότι οι προωθημένοι πόροι πρέπει να είναι υπάκουοι στην πολιτική της ίδιας προέλευσης, δηλαδή ο server πρέπει να είναι έγκυρος για το περιεχόμενο που παρέχεται. [8]

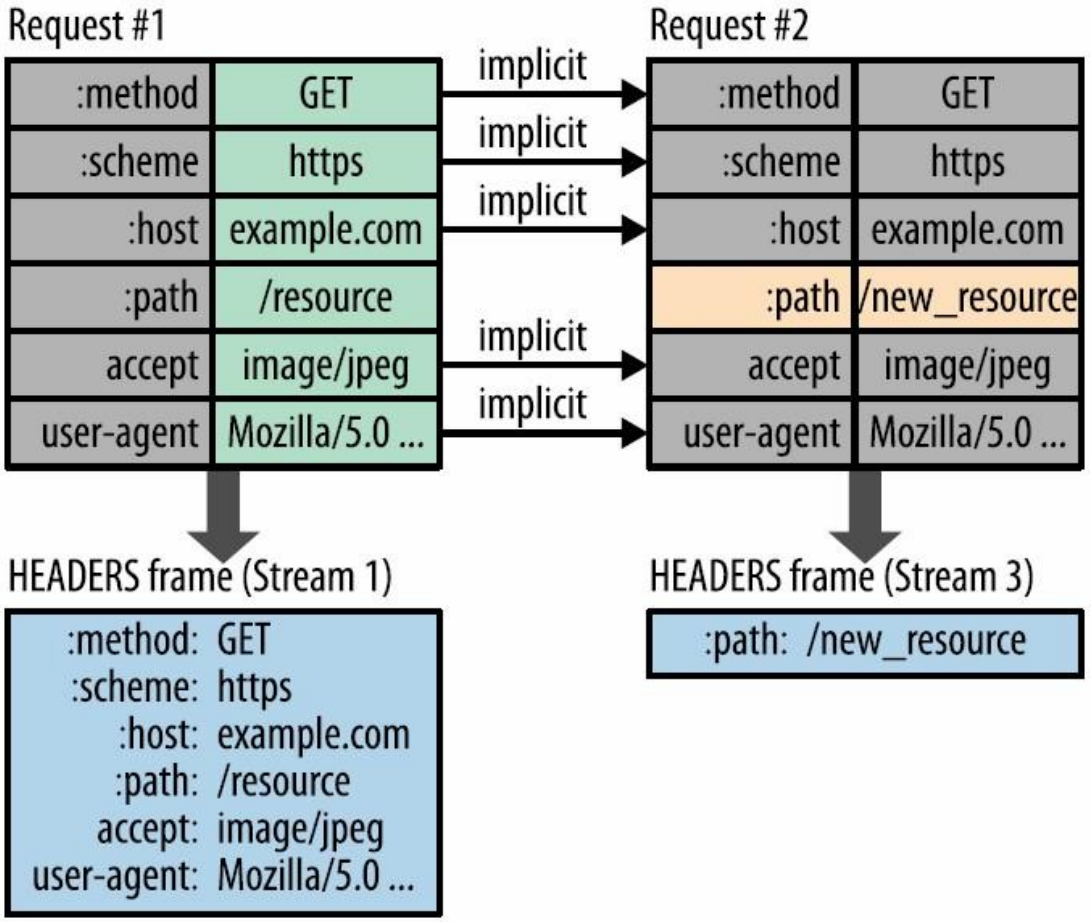
3.8.Συμπίεση κεφαλίδας (Header Compression)

Κάθε μεταφορά HTTP φέρει ένα σύνολο κεφαλίδων που περιγράφουν τον μεταφερόμενο πόρο και τις ιδιότητές του. Στο HTTP/1.x, αυτά τα metadata αποστέλλονται πάντα ως απλό κείμενο και προσθέτουν οπουδήποτε από 500 με 800 byte επιβάρυνσης ανά μεταφορά και μερικές φορές φτάνουν έως και κάποια kilobyte εάν χρησιμοποιούνται HTTP cookies. Για να μειώσει αυτή την επιβάρυνση και να βελτιώσει την απόδοση, το HTTP/2 συμπιέζει τα metadata της κεφαλίδας αιτημάτων και αποκρίσεων χρησιμοποιώντας τη μορφή συμπίεσης HPACK που χρησιμοποιεί δυο απλές αλλά ισχυρές τεχνικές:

1. Επιτρέπει την κωδικοποίηση των μεταδιδόμενων πεδίων κεφαλίδας μέσω ενός στατικού κώδικα Huffman[16], ο οποίος μειώνει το ατομικό τους μέγεθος μεταφοράς.
2. Απαιτεί τόσο ο client όσο και ο server να διατηρούν και να ενημερώνουν μια ευρετηριασμένη λίστα ήδη γνωστών πεδίων κεφαλίδας (δηλαδή δημιουργεί ένα κοινό πλαίσιο συμπίεσης), η οποία στη συνέχεια χρησιμοποιείται ως αναφορά για την αποτελεσματική κωδικοποίηση τιμών που είχαν μεταδοθεί προηγουμένως.

Η κωδικοποίηση Huffman επιτρέπει τη συμπίεση των μεμονωμένων τιμών κατά τη μεταφορά και η ευρετηριασμένη λίστα των τιμών που είχαν μεταφερθεί προηγουμένως μας επιτρέπει να κωδικοποιήσουμε διπλότυπες τιμές (παρακάτω εικόνα) μεταφέροντας τιμές του ευρετηρίου που μπορούν να χρησιμοποιηθούν για την αποτελεσματική αναζήτηση και ανακατασκευή των πλήρων κεφαλίδων και τιμών.

[16] Ο κώδικας Huffman είναι ένας τρόπος για την κωδικοποίηση πληροφοριών χρησιμοποιώντας συμβολοσειρές μεταβλητού μήκους για την αναπαράσταση συμβόλων ανάλογα με το πόσο συχνά εμφανίζονται. Η ιδέα είναι ότι τα σύμβολα που χρησιμοποιούνται πιο συχνά θα πρέπει να είναι μικρότερα, ενώ τα σύμβολα που εμφανίζονται πιο σπάνια μπορεί να είναι μεγαλύτερα.



Εικόνα 3.5. HTTP/2 Server Compression

(Πηγή: https://hpbn.co/http2/)

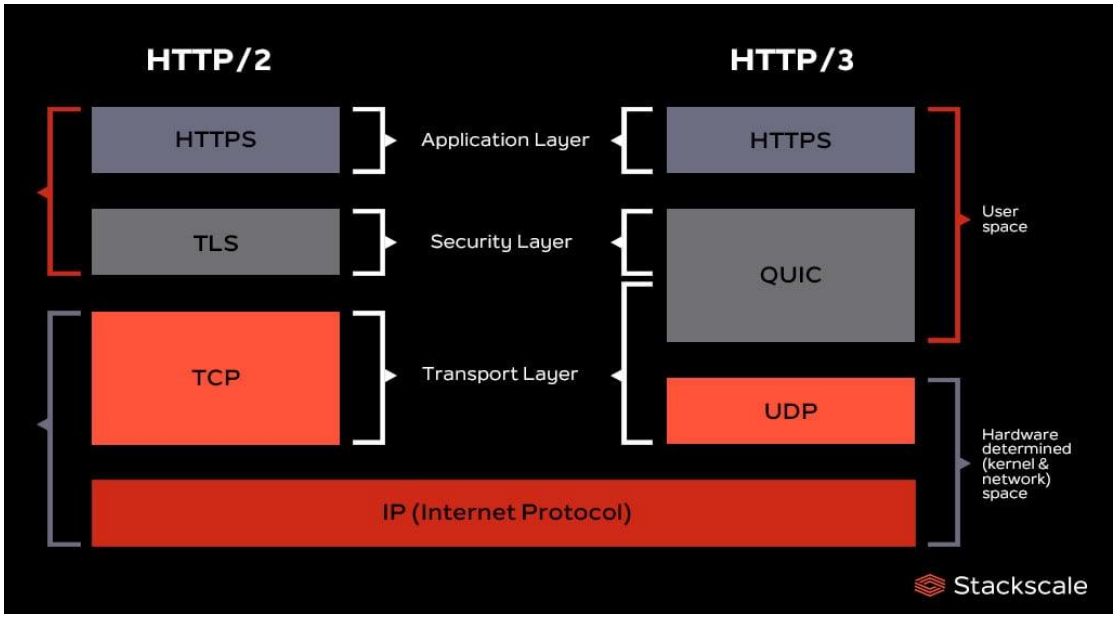
Ως μια περεταίρω βελτιστοποίηση, το HPACK πλαίσιο συμπίεσης αποτελείται από στατικούς και δυναμικούς πίνακες: ο στατικός πίνακας ορίζεται στην προδιαγραφή και παρέχει μια λίστα με κοινά πεδία κεφαλίδων HTTP που είναι πιθανό να χρησιμοποιούν όλες οι συνδέσεις (π.χ. έγκυρα ονόματα κεφαλίδων). Ο δυναμικός πίνακας είναι αρχικά κενός και ενημερώνεται με βάση τις τιμές που ανταλλάσσονται σε μια συγκεκριμένη σύνδεση. Ως αποτέλεσμα, το μέγεθος κάθε αιτήματος μειώνεται με τη χρήση της στατικής κωδικοποίησης Huffman για τιμές που δεν έχουν εμφανιστεί πριν, και την αντικατάσταση των ευρετηρίων για τιμές που υπάρχουν ήδη στους στατικούς ή δυναμικούς πίνακες σε κάθε πλευρά. [8]

4. Το HTTP/3

4.1.Ιστορία του QUIC

Το QUIC (Quick UDP Internet Connections) αναπτύχθηκε για πρώτη φορά από την Google το 2012. Το QUIC επαναπροσδιορίζει τα όρια των επιπέδων δικτύου, βασιζόμενο

σε πρωτόκολλο UDP χαμηλότερου επιπέδου, επαναπροσδιορίζοντας τις "χειραψίες", τις λειτουργίες αξιοπιστίας και τα χαρακτηριστικά ασφαλείας στο "χώρο χρήστη" (ή "user-space"), αποφεύγοντας την ανάγκη αναβάθμισης kernel σε όλο το διαδίκτυο. Ακριβώς όπως με το HTTP/2, μια πρόοδο που πρωτοστάτησε από το SPDY της Google, το HTTP/3 θα βασιστεί ξανά σε αυτά τα επιτεύγματα.



Εικόνα 4.1. HTTP/2 Stack vs HTTP/3 Stack

(Πηγή: https://www.stackscale.com/blog/http3/)

Ενώ το HTTP/2 μας έδωσε το multiplexing (ή πολυπλεξία) και μετρίασε το head-of-line-blocking, περιορίζεται από το TCP. Όπως αναφέρθηκε και σε προηγούμενα κεφάλαια, μια ενιαία σύνδεση TCP μπορεί να χρησιμοποιηθεί για πολλαπλές ροές που πολυπλέκονται μαζί για τη μεταφορά δεδομένων, αλλά όταν μία από αυτές τις ροές υφίσταται απώλεια πακέτου, ολόκληρη η σύνδεση (και όλα τα streams της) παραμένουν όμηροι, ας πούμε, έως ότου το TCP κάνει τη δουλειά του, δηλαδή αναμεταδώσει το χαμένο πακέτο. Αυτό σημαίνει ότι όλα τα πακέτα, ακόμα κι αν έχουν ήδη μεταδοθεί και περιμένουν στην προσωρινή μνήμη του κόμβου προορισμού, μπλοκάρονται μέχρι να επαναμεταδοθεί το χαμένο πακέτο.

Το QUIC δεν περιορίζεται από αυτό. Όπως ξέρουμε, κάθε συσκευή στο δίκτυο μπορεί να επιθεωρήσει (και το κάνει) τα πακέτα TCP καθώς αυτά ρέουν. Ως αποτέλεσμα, είναι

δύσκολο να αναπτυχθούν με επιτυχία αλλαγές στο TCP με αποτέλεσμα πολλές καινοτομίες για το TCP να έχουν ελάχιστη χρήση στο ζωντανό διαδίκτυο. Με το QUIC όμως να βασίζεται στο connectionless πρωτόκολλο UDP, η έννοια της σύνδεσης δεν φέρει τους περιορισμούς του TCP και οι αποτυχίες μιας ροής δεν χρειάζεται να επηρεάσουν τις υπόλοιπες. [9][10]

Το SCTP (Stream Control Transmission Protocol) ήταν μια προηγούμενη προσπάθεια αντικατάστασης και βελτίωσης του TCP. Αντίθετα, έδειξε ότι είναι πολύ δύσκολο να αναπτυχθεί ένα πρωτόκολλο διαφορετικό από το TCP ή το UDP στο Διαδίκτυο. Το QUIC λύνει αυτό το πρόβλημα με την εκ νέου εφαρμογή βασικών υπηρεσιών μεταφοράς μέσα σε ένα κρυπτογραφημένο φάκελο, χρησιμοποιώντας το UDP για τη μεταφορά του μέσω διαδικτύου. Η Google ανακοίνωσε για πρώτη φορά τη διαθεσιμότητα του Google QUIC το 2013, χρησιμοποιώντας το μεταξύ του Google Chrome και των υπηρεσιών Google. Αυτό τους επέτρεψε να κάνουν βελτιώσεις ανεξάρτητα από το λειτουργικό σύστημα ή το πρόγραμμα ενημέρωσης του λειτουργικού συστήματος.[10]

Με εστίαση στις ροές UDP, το QUIC επιτυγχάνει πολυπλεξία χωρίς να χρειάζεται να περιοριστεί σε μία σύνδεση TCP. Το QUIC χτίζει τη σύνδεσή του σε υψηλότερο επίπεδο από το TCP. Οι νέες ροές εντός των συνδέσεων QUIC δεν αναγκάζονται να περιμένουν να τελειώσουν οι υπόλοιπες. Οι συνδέσεις QUIC επωφελούνται επίσης από την κατάργηση της επιβάρυνσης της χειραψίας TCP, η οποία μειώνει το latency. [9]

Η Google έφερε την QUIC στο IETF το 2015 για να ξεκινήσει τη διαδικασία προτύπων. Οι συνεισφέροντες στο IETF εξέφρασαν ενδιαφέρον για την ανάπτυξη μιας νέας μεταφοράς ξεκινώντας από το πειραματικό τους πρωτόκολλο. Αυτή η ιδιόκτητη έκδοση ονομάζεται συχνά "Google QUIC" ή "gQUIC". Μεταξύ του Google Chrome, του YouTube, του Gmail, της αναζήτησης της Google και άλλων υπηρεσιών της, η Google μπόρεσε να αναπτύξει το QUIC σε ένα μεγάλο κομμάτι του Διαδικτύου, χωρίς να περιμένει το IETF. Οι μηχανικοί της Google ισχυρίζονται ότι το 2017, το 7% της κίνησης στο Διαδίκτυο διεξαγόταν ήδη μέσω QUIC.

Το IETF σχημάτισε μια ομάδα εργασίας QUIC το 2016. Ο καταστατικός τους χάρτης ήταν να λάβουν την εφαρμογή της Google ειδικά για τον ιστό και να την προσαρμόσουν ώστε να είναι ένα πρωτόκολλο μεταφοράς γενικής χρήσης. Η έκδοση QUIC της Google επικεντρώθηκε μόνο στη μεταφορά HTTP, χρησιμοποιώντας σύνταξη HTTP/2. Τα άτομα

από το IETF που είναι υπεύθυνοι για την τυποποίηση του QUIC όμως αποφάσισαν ότι η έκδοση IETF του QUIC θα πρέπει να μπορεί να μεταφέρει περισσότερα από απλά HTTP. Προς το παρόν, ωστόσο, οποιαδήποτε εργασία σε πρωτόκολλα που δεν είναι HTTP μέσω QUIC είναι σε αναμονή. [9][10]

Το Google QUIC εφαρμόστηκε ως μια ενιαία βάση κωδικών αναφοράς που εξέθεσε ένα API τύπου HTTP, χρησιμοποιούσε προσαρμοσμένη κρυπτογράφηση και έβγαζε πακέτα UDP στο κάτω μέρος. Η διαδικασία IETF εξέλιξε αυτό το πρωτόκολλο με πολλούς τρόπους, όπως:

- Διαχωρισμός του πρωτοκόλλου επιπέδου εφαρμογής από το επίπεδο μεταφοράς. Το IETF QUIC είναι μια μεταφορά γενικής χρήσης, συγκρίσιμη με το TCP. Το HTTP/3 είναι η αντιστοίχιση του HTTP που χρησιμοποιεί το IETF QUIC ως μεταφορά.
- Δημιουργία ενός ανεξάρτητου από την εφαρμογή συνόλου πρωτόγονων μεταφορών που μπορεί να χρησιμοποιήσει οποιαδήποτε εφαρμογή. Οι ομάδες εργασίας του IETF διερευνούν ήδη το QUIC ως μέσο μεταφοράς για DNS, SSH, BGP, RTP και πολλά άλλα.
- Αντικατάσταση της προσαρμοσμένης κρυπτογραφίας με TLS 1.3. Η QUIC Crypto ενημέρωσε τη σχεδίαση του TLS 1.3, το οποίο έγινε Προτεινόμενο Πρότυπο το 2018. Μερικές από τις ιδέες που εξερευνήθηκαν για πρώτη φορά από την QUIC Crypto είναι τώρα προτεινόμενες επεκτάσεις TLS. Η μετάβαση από την προσαρμοσμένη κρυπτογραφία στο τυποποιημένο TLS 1.3 βοηθά στην άρση της ασάφειας σχετικά με το εάν η χρήση του gQUIC αποτελεί πρόβλημα για site με ρυθμιστικές απαιτήσεις ή/και απαιτήσεις συμμόρφωσης.
- Άλλες ουσιαστικές αλλαγές στο πρωτόκολλο. Χιλιάδες τεύχη και αλλαγές προδιαγραφών έχουν βελτιώσει την ποιότητα και τη σαφήνεια των προδιαγραφών.

Σε αυτό το σημείο υπάρχουν διάφορες υλοποιήσεις IETF QUIC ανοιχτού κώδικα, με πολλούς από τους συγγραφείς τους να έχουν συμμετάσχει σε IETF Hackathons. Αυτά τα events ανακάλυψαν ζητήματα στις προδιαγραφές και βοήθησαν να διευκρινιστούν. Όταν δύο προγραμματιστές παράγουν ασύμβατο κώδικα από την ίδια προδιαγραφή, αυτό σημαίνει ότι η προδιαγραφή έχει πρόβλημα.

Οι παρακάτω προδιαγραφές είναι η καρδιά του πρωτοκόλλου μεταφοράς IETF QUIC:

- **RFC 8999:** (Version-Independent Properties of QUIC) [11] περιγράφει το μικρό μέρος του QUIC που μπορεί το δίκτυο να παρατηρήσει. Αυτό δεν μπορεί να αλλάξει μεταξύ των εκδόσεων IETF QUIC.
- **RFC 9000:** (QUIC: A UDP-Based Multiplexed and Secure Transport) [12] είναι η έκδοση 1 του IETF QUIC, του βασικού πρωτοκόλλου μεταφοράς. Περιγράφει πώς οι πελάτες και οι διακομιστές επικοινωνούν μεταξύ τους σε αυτήν την έκδοση.

- **RFC 9001:** (Using TLS to Secure QUIC) [13] περιγράφει την ενοποίηση του TLS 1.3 και της έκδοσης QUIC 1. Το TLS δεν βρίσκεται ούτε πάνω ούτε κάτω από το QUIC. η σχέση είναι πιο περίπλοκη. Το QUIC παρέχει στο TLS μια αξιόπιστη ροή κατά παραγγελία, ενώ το TLS παρέχει κλειδιά κρυπτογράφησης QUIC και επικύρωση χειραψίας.
- **RFC 9002:** (QUIC Loss Detection and Congestion Control) [14] δίνει ένα παράδειγμα αλγορίθμων ανίχνευσης απώλειας και ελέγχου συμφόρησης για το IETF QUIC. Όπως και με το TCP, υπάρχουν πολλές στρατηγικές που μπορεί να χρησιμοποιήσει ένα τελικό σημείο τόσο για τον εντοπισμό απωλειών όσο και για τον έλεγχο συμφόρησης, και αναμένεται ότι αυτός θα συνεχίσει να είναι ένας τομέας ενεργού πειραματισμού και καινοτομίας. Αυτό το προσχέδιο παρέχει ένα σημείο εκκίνησης για όποιον προσπαθεί να απογειώσει το QUIC.

Με την ολοκλήρωση των προδιαγραφών συνεπάγεται ότι η έκδοση IETF της μεταφοράς είναι σταθερή και έτοιμη για χρήση στην παραγωγή. Η Google έχει ήδη αποσύρει όλες τις εκδόσεις του Google QUIC που δεν συμμορφώνονται με τις αμετάβλητες στο RFC 8999 και σύντομα θα αποσύρει όλες τις υπόλοιπες εκδόσεις του Google QUIC. [10]

4.2.Από το QUIC στο HTTP/3

Υπάρχουν όμως ορισμένα πράγματα που λείπουν από την προηγούμενη λίστα, επειδή οι εργασίες συνεχίζονται σε διάφορους τομείς, τόσο στο QUIC Working Group όσο και στο IETF. Το παραπάνω σύνολο RFC δεν περιλαμβάνει το HTTP/3 (ή το συνοδευτικό του έγγραφο, QPACK), το οποίο μπορεί να αποτελεί έκπληξη δεδομένου ότι η προδιαγραφή γράφτηκε σε κώδικα και αναπτύχθηκε από το QUIC Working Group και υποβλήθηκε για δημοσίευση ταυτόχρονα. [9][10]

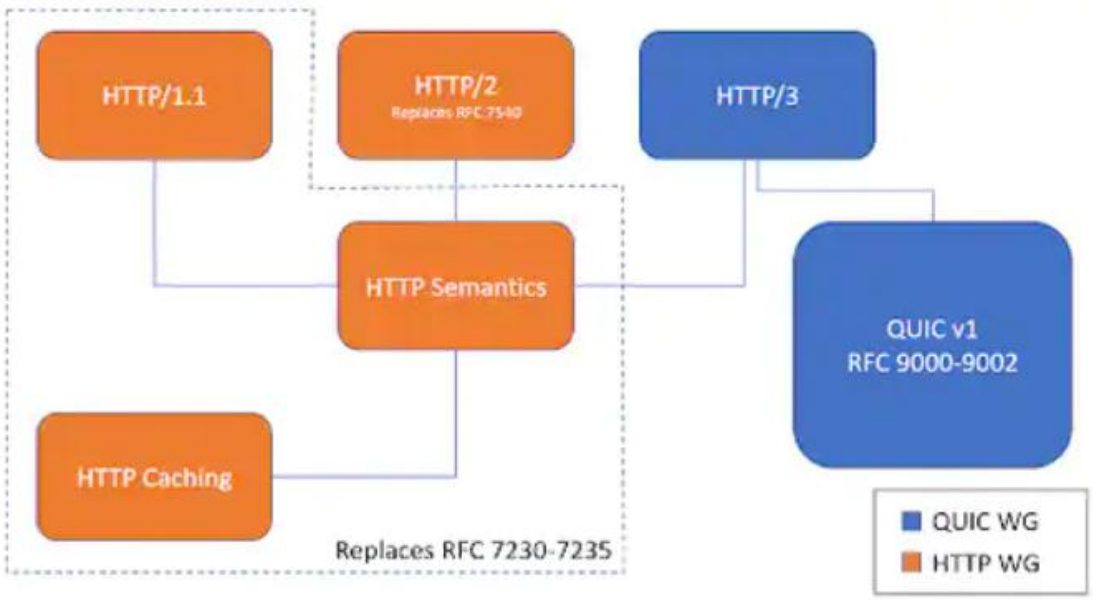
Η κεντρική σημασιολογία του HTTP δεν αλλάζει μεταξύ των εκδόσεων -- το πρωτόκολλο HTTP έχει ουσιαστικά την ίδια λειτουργικότητα, είτε είναι HTTP/1.1, HTTP/2 ή HTTP/3. Ωστόσο, η προδιαγραφή της σημασιολογίας HTTP είναι επί του παρόντος συνυφασμένη με τον ορισμό του HTTP/1.1. Αυτό προς το παρόν βρίσκεται στο RFC 7230 και τα συνοδευτικά έγγραφα. Το IETF HTTP Working Group εξάγει αυτές τις ανεξάρτητες από την έκδοση σημασιολογίες σε δύο νέα έγγραφα: [10]

- **RFC 9110:** (HTTP Semantics) [15] περιγράφει τη λειτουργία του HTTP σε οποιαδήποτε μεταφορά ή χαρτογράφηση.

- **RFC 9111:** (HTTP Caching) [16] περιγράφει πώς τα τελικά σημεία μπορούν να κάνουν cache και να επαναχρησιμοποιήσουν responses σε οποιαδήποτε μεταφορά ή χαρτογράφηση.

Μια νέα τριάδα προδιαγραφών που καθορίζουν τις διαφορετικές αντιστοιχίσεις της σημασιολογίας HTTP σε διάφορα πρωτόκολλα μεταφοράς βασίζονται σε αυτά τα δύο έγγραφα: [25]

- **RFC 9112:** (HTTP/1.1) [17] χρησιμοποιείται συνήθως μέσω TCP και συχνά χρησιμοποιεί TLS. Αυτό είναι το υπόλοιπο από αυτό που υπάρχει αυτή τη στιγμή στο RFC 7230.
- **RFC 9113:** (HTTP/2) [18] χρησιμοποιεί και TLS 1.2+ και TCP.
- **RFC 9114:** (HTTP/3) [19] χρησιμοποιεί IETF QUIC έκδοση 1, η οποία ενσωματώνει TLS 1.3.



Εικόνα 4.2. RFC Changes

(Πηγή: https://www.akamai.com/blog/performance/http3-and-quic-past-present-and-future)

Είναι σημαντικό να σημειωθεί ότι καμία από αυτές τις εκδόσεις του HTTP δεν καταργεί τις άλλες, επειδή κάθε μία είναι κατάλληλη για διαφορετικές περιπτώσεις χρήσης. Υπάρχουν δίκτυα που μπλοκάρουν το UDP, που σημαίνει ότι το HTTP/3 δεν μπορεί να χρησιμοποιηθεί. Γι' αυτό προδιαγραφή συνιστά τη χρήση μιας έκδοσης HTTP που βασίζεται σε TCP σε αυτά τα δίκτυα. Τα HTTP/1.1 και HTTP/2 εξακολουθούν να λαμβάνουν την προσοχή του IETF. Επειδή όλα αυτά τα νέα έγγραφα αντιστοίχισης βασίζονται στο έγγραφο Semantics, κανένα από αυτά δεν μπορεί να δημοσιευτεί έως ότου ολοκληρωθεί το έγγραφο Semantics, το οποίο κατά τη συγγραφή αυτής της εργασίας δεν

έχει ολοκληρωθεί ακόμα. Το RFC Editor θα δημοσιεύσει την ενημερωμένη προδιαγραφή HTTP/1.1 και την προδιαγραφή HTTP/3 σχεδόν ταυτόχρονα, καθώς έχουν αναπτυχθεί από κοινού με το HTTP Semantics και το QUIC αντίστοιχα. Το HTTP Working Group έχει υιοθετήσει την αντίστοιχη ενημερωμένη προδιαγραφή HTTP/2, η οποία (κατά τη συγγραφή αυτής της εργασίας) έχει σχεδόν ολοκληρωθεί. [10]

4.3.Διαπραγμάτευση έκδοσης

Το Google QUIC είχε έναν μηχανισμό διαπραγμάτευσης μεταξύ πολλαπλών εκδόσεων του πρωτοκόλλου. Κατά τη διάρκεια της διαδικασίας τυποποίησης, ο καθορισμός των ακριβών στόχων ασφαλείας αυτού του τμήματος του πρωτοκόλλου έγινε αντικείμενο διαμάχης.

Για να ξεμπλοκαριστεί η δημοσίευση της έκδοσης 1 του IETF QUIC, συμφωνήθηκε ότι η v1 προδιαγραφή θα καθόριζε μόνο τον τρόπο με τον οποίο θα υποδεικνύεται ότι η έκδοση 1 δεν υποστηρίζεται. Οι clients που υποστηρίζουν μόνο την έκδοση 1 δεν θα μπορούσαν να συνδεθούν σε server που δεν υποστηρίζει αυτήν την έκδοση. Αυτό είναι αρκετό για το HTTP/3, επειδή οι μέθοδοι εύρεσης υποστήριξης QUIC σε ένα origin μπορούν επίσης να μας ενημερώσουν ποια έκδοση του QUIC υποστηρίζει ο server.

Το QUIC WG αναπτύσσει ενεργά τον μηχανισμό διαπραγμάτευσης πλήρους έκδοσης (Compatible Version Negotiation for QUIC) [20], ο οποίος θα καθορίσει πώς οι υλοποιήσεις μπορούν να υποστηρίξουν πολλαπλές εκδόσεις του IETF QUIC και πως να συμφωνήσουν σε μια έκδοση χωρίς προηγούμενη γνώση. [10]

4.4.Εφαρμογή και δυνατότητα διαχείρισης

Ένα κρυπτογραφημένο πρωτόκολλο μεταφοράς έχει επιπτώσεις τόσο για τους μηχανικούς που επιλέγουν μια μεταφορά για τα πρωτόκολλά τους όσο και για τους χειριστές δικτύου που προσπαθούν να παρακολουθήσουν τη δραστηριότητα.

Το **RFC 9308:** (Applicability of the QUIC Transport Protocol) [21] περιγράφει ζητήματα για πρωτόκολλα εφαρμογών που μπορεί να θέλουν να χρησιμοποιήσουν το QUIC ως επίπεδο μεταφοράς. Οι ιδιότητες του QUIC είναι διαφορετικές από εκείνες του

TCP με μερικούς ενδιαφέροντες και διακριτικούς τρόπους, όπως η διαθεσιμότητα δεδομένων πριν ολοκληρωθεί η "χειραψία" ή η διαθεσιμότητα πολλαπλών ταυτόχρονων streams. Αυτό το έγγραφο εξετάζει τις σχεδιαστικές επιλογές που μπορεί να χρειαστεί να εξετάσει κάποιος που εφαρμόζει ένα πρωτόκολλο όταν σχεδιάζει μια έκδοση που χρησιμοποιεί το IETF QUIC για μεταφορά.

Το **RFC 9312:** (Manageability of the QUIC Transport Protocol) [22] περιγράφει τον αντίκτυπο ενός κρυπτογραφημένου επιπέδου μεταφοράς στην παρακολούθηση και διαχείριση του δικτύου. Πολλά δίκτυα αναλαμβάνουν τη δυνατότητα επιθεώρησης της κατάστασης του TCP για ενδείξεις προβλημάτων δικτύου (κακή απόδοση, υψηλή απώλεια πακέτων) ή καταχρηστική κίνηση (data exfiltration, επιθέσεις). Εφόσον το QUIC κρύβει τα περιεχόμενα των πακέτων, αυτό το έγγραφο εξετάζει τους περιορισμούς και τις εναλλακτικές στρατηγικές για την επίτευξη αυτών των στόχων. Ως άλλο παράδειγμα, αυτό το έγγραφο συνιστά ότι τα δίκτυα θα πρέπει είτε να επιτρέπουν πλήρως είτε να αποκλείουν τις συνδέσεις QUIC, καθώς η τυχαία απόρριψη πακέτων QUIC θα βλάψει την απόδοση του χρήστη. [10]

4.5. Άλλα πρωτόκολλα εφαρμογών μέσω QUIC

Το QUIC ξεκίνησε ως ένα έργο για τη βελτίωση της απόδοσης του προγράμματος περιήγησης ιστού και το QUIC Working Group έχει στόχο να κάνει την έκδοση 1 να περιέχει τις απαραίτητες δυνατότητες για την ενεργοποίηση μιας χαρτογράφησης αντίστοιχης με το HTTP/2. Αλλά το IETF QUIC δεν είναι μόνο για προγράμματα περιήγησης ιστού. Όπως σημειώθηκε παραπάνω, το IETF εργάζεται για την ενημέρωση πολλών άλλων πρωτοκόλλων με σκοπό τη χρήση του QUIC για μεταφορά. Για παράδειγμα, το DNS-over-QUIC μπορεί να είναι ένας φυσικός τρόπος για να ασφαλιστεί DNS αναδρομικά σε έγκυρες επικοινωνίες. Υπάρχουν επίσης εξωτερικοί φορείς εφαρμογών και προτύπων που έχουν ήδη αρχίσει να χτίζουν πάνω από το IETF QUIC για τα δικά τους εσωτερικά πρωτόκολλα. [10]

4.6. Ενσωμάτωση εξισορροπιτή φορτίου

Με το TCP, ήταν σύνηθες για τους εξισορροπητές φορτίου να τερματίζουν το TCP (και ίσως το TLS) πριν περάσουν τη σύνδεση επιπέδου εφαρμογής σε έναν από τους πολλούς servers πίσω από αυτό. Αυτό είναι πιο δύσκολο με το QUIC, αλλά το QUIC παρέχει επίσης δυνατότητες δρομολόγησης όπως ένα αναγνωριστικό σύνδεσης ορατό σε κάθε πακέτο. Αυτό το προσχέδιο [23] περιγράφει τρόπους συνεργασίας των server με τους εξισορροπητές φορτίου στη δημιουργία αναγνωριστικών σύνδεσης, ώστε να μπορούν να χρησιμοποιηθούν για δρομολόγηση από το πρόγραμμα εξισορρόπησης φορτίου χωρίς να διακυβεύονται οι εγγυήσεις απορρήτου και ασφάλειας της QUIC. [10]

4.7.Datagrams

Ενώ το QUIC εκτελείται μέσω UDP, η διεπαφή εφαρμογών του μοιάζει πολύ περισσότερο με TCP ή SCTP: μια σειρά αξιόπιστων, κατά σειρά byte stream. Για πολλές εφαρμογές, αυτό είναι τέλειο -- η αξιόπιστη παράδοση είναι απαραίτητη σε πολλά πρωτόκολλα. Ωστόσο άλλα πρωτόκολλα είναι πιο ευαίσθητα στην καθυστέρηση παρά στην απώλεια, και είναι καλύτερο να παραλειφτούν τα χαμένα δεδομένα αντί να επαναμεταδοθούν. Το **RFC 9221:** (An Unreliable Datagram Extension to QUIC) [24], θα προσθέσει τη δυνατότητα αποστολής αναξιόπιστων datagrams μέσα σε μια σύνδεση QUIC -- η εφαρμογή μαθαίνει εάν το datagram ελήφθη ή όχι και μπορεί να αποφασίσει ποια χαμένα δεδομένα πρέπει να επαναμεταδοθούν. [10]

5. Σύγκριση εκδόσεων

5.1. Εγκατάσταση σύνδεσης

Στα πρωτόκολλα client-server, όπως το HTTP, οι συνεδρίες αποτελούνται από τρεις φάσεις:

1. **Ο client δημιουργεί μια σύνδεση TCP** (ή την κατάλληλη σύνδεση εάν το επίπεδο μεταφοράς δεν είναι TCP). Το άνοιγμα μιας σύνδεσης σε HTTP σημαίνει έναρξη μιας σύνδεσης στο υποκείμενο επίπεδο μεταφοράς (transport layer), το οποίο συνήθως είναι το TCP. Με το TCP, η προεπιλεγμένη θύρα από έναν HTTP server σε έναν υπολογιστή, είναι η θύρα 80. Μπορούν επίσης να χρησιμοποιηθούν και άλλες θύρες, όπως 8000 ή 8080. Η διεύθυνση URL μιας σελίδας προς ανάκτηση περιέχει τόσο το όνομα τομέα (domain name) όσο και τον αριθμό θύρας, αν και αυτός μπορεί να

παραλειφθεί εάν είναι η θύρα 80. Αυτή η φάση που περιγράφεται εδώ ονομάζεται 3-Way Handshake.

2. **Ο client στέλνει το request του και περιμένει το response.** Μόλις δημιουργηθεί η σύνδεση, ο user-agent μπορεί να στείλει το request. Όπως έχει αναφερθεί σε προηγούμενη ενότητα, ένα client request αποτελείται από οδηγίες κειμένου, χωρισμένες με CRLF, χωρισμένες σε τρία μπλοκ:

    - Η πρώτη γραμμή περιέχει μια μέθοδο αιτήματος ακολουθούμενη από τις παραμέτρους της: τη διαδρομή του εγγράφου, ως απόλυτη διεύθυνση URL χωρίς το πρωτόκολλο ή το όνομα τομέα, και την έκδοση πρωτοκόλλου HTTP.
    - Οι επόμενες γραμμές αντιπροσωπεύουν μια κεφαλίδα HTTP, δίνοντας στον server πληροφορίες σχετικά με τον τύπο δεδομένων που είναι κατάλληλος (για παράδειγμα, ποια γλώσσα, ποιοι τύποι MIME) ή άλλα δεδομένα που αλλάζουν τη συμπεριφορά του (για παράδειγμα, μη αποστολή response εάν είναι ήδη αποθηκευμένη στο cache). Αυτές οι κεφαλίδες HTTP σχηματίζουν ένα μπλοκ που τελειώνει με μια κενή γραμμή.
    - Το τελικό μπλοκ είναι ένα προαιρετικό μπλοκ δεδομένων, το οποίο μπορεί να περιέχει περαιτέρω δεδομένα που χρησιμοποιούνται κυρίως από τη μέθοδο POST.

3. **Ο server επεξεργάζεται το request, στέλνοντας πίσω το response του, παρέχοντας έναν κωδικό κατάστασης και τα κατάλληλα δεδομένα.** Αφού ο συνδεδεμένος agent στείλει το request του, ο web server το επεξεργάζεται και τελικά επιστρέφει ένα response. Όπως έχει αναφερθεί σε προηγούμενη ενότητα και παρόμοια με ένα client request, ένα server response αποτελείται από οδηγίες κειμένου, που χωρίζονται με CRLF, αν και χωρίζονται σε τρία μπλοκ:

    - Η πρώτη γραμμή, η γραμμή κατάστασης, αποτελείται από μια επιβεβαίωση της χρησιμοποιούμενης έκδοσης HTTP, ακολουθούμενη από έναν κώδικα κατάστασης (Status Code) και τη φράση αιτιολογίας του (Reason Phrase), δηλαδή τη σύντομη σημασία του σε αναγνώσιμο κείμενο.
    - Οι επόμενες γραμμές αντιπροσωπεύουν συγκεκριμένες κεφαλίδες HTTP, δίνοντας στον client πληροφορίες σχετικά με τα δεδομένα που αποστέλλονται (για παράδειγμα, τύπος, μέγεθος δεδομένων, χρησιμοποιούμενος αλγόριθμος συμπίεσης, υποδείξεις σχετικά με το caching). Παρόμοια με το μπλοκ των κεφαλίδων HTTP για ένα client request, αυτές οι κεφαλίδες HTTP σχηματίζουν ένα μπλοκ που τελειώνει με μια κενή γραμμή.
    - Το τελικό μπλοκ είναι ένα data block, το οποίο περιέχει τα προαιρετικά δεδομένα.

Για να εγκατασταθεί μια σύνδεση HTTP γίνεται η έναρξη της σύνδεσης TCP που δαπανά ένα RTT[17]. Για να σταλεί το HTTP request και να ληφθεί το response χρειάζεται επίσης ένα RTT. Επομένως, ο συνολικός χρόνος για τη μετάδοση είναι:

$$T_{total} = 2RTT+transmit\ time$$

Η κρυπτογράφηση TLS στο HTTP/1.1 είναι προαιρετική. Στην περίπτωση που συμπεριλαμβάνεται χρειάζεται ένα ακόμα RTT για την εφαρμογή της.

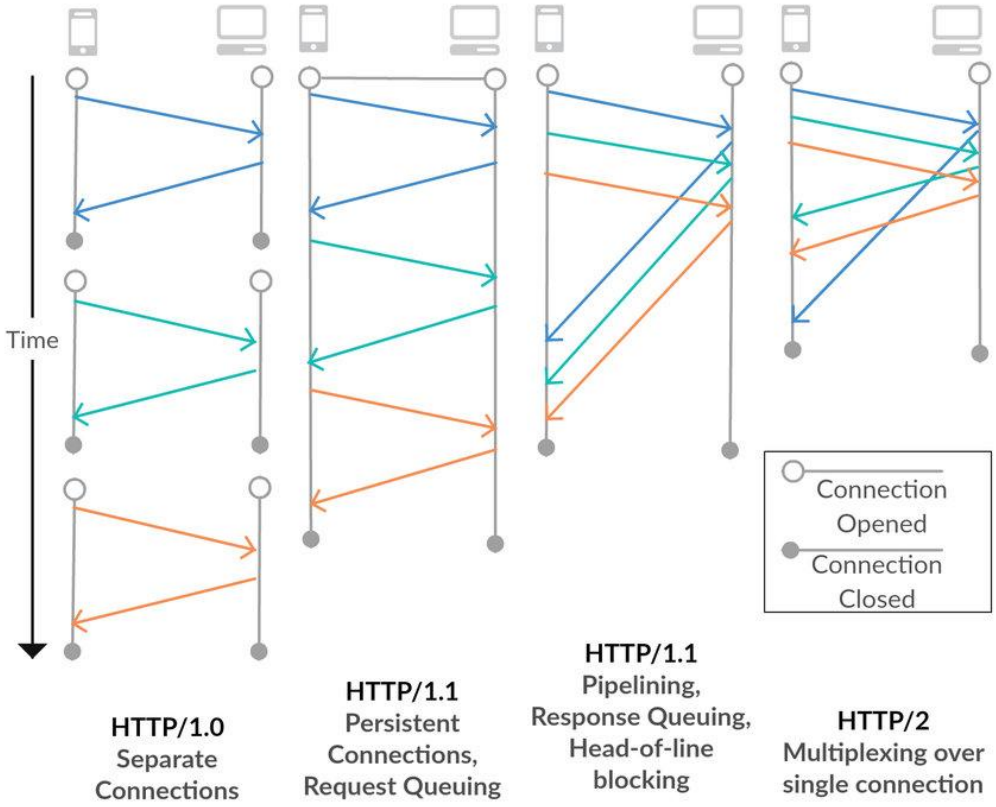
Οι συνδέσεις HTTP μπορεί να είναι:

- μόνιμες (Persistent): η σύνδεση παραμένει ανοικτή
- μη μόνιμες (Non-Persistent): η σύνδεση κλείνει με την ολοκλήρωση της μεταφοράς του αντικειμένου

Οι non-persistent συνδέσεις μπορεί να είναι απλές ή παράλληλες. Στην πρώτη περίπτωση κάθε αντικείμενο χρειάζεται 2 RTT, δηλαδή ένα για την σύνδεση TCP και ένα για το request / response του HTTP. Στη δεύτερη περίπτωση τα αντικείμενα μπορεί να μεταφέρονται παράλληλα και η διαδικασία γίνεται ταχύτερη, όπως αποτυπώνεται και στην Εικόνα 5.1.

Οι persistent συνδέσεις HTTP διακρίνονται σε Non-pipelined και Pipelined. Στις Non-pipelined ο client στέλνει ένα request μόνο εφόσον έχει απαντηθεί το προηγούμενο. Στις Pipelined ο client μπορεί να στείλει ένα request στον server ακόμα και αν το προηγούμενο request δεν έχει λάβει response από τον server.

[17] Round-Trip Time ή RTT ορίζεται ως ο χρόνος που χρειάζεται ένα πακέτο πληροφοριών για να ταξιδέψει από έναν client σε έναν server και στη συνέχεια να επιστρέψει πίσω στον client.



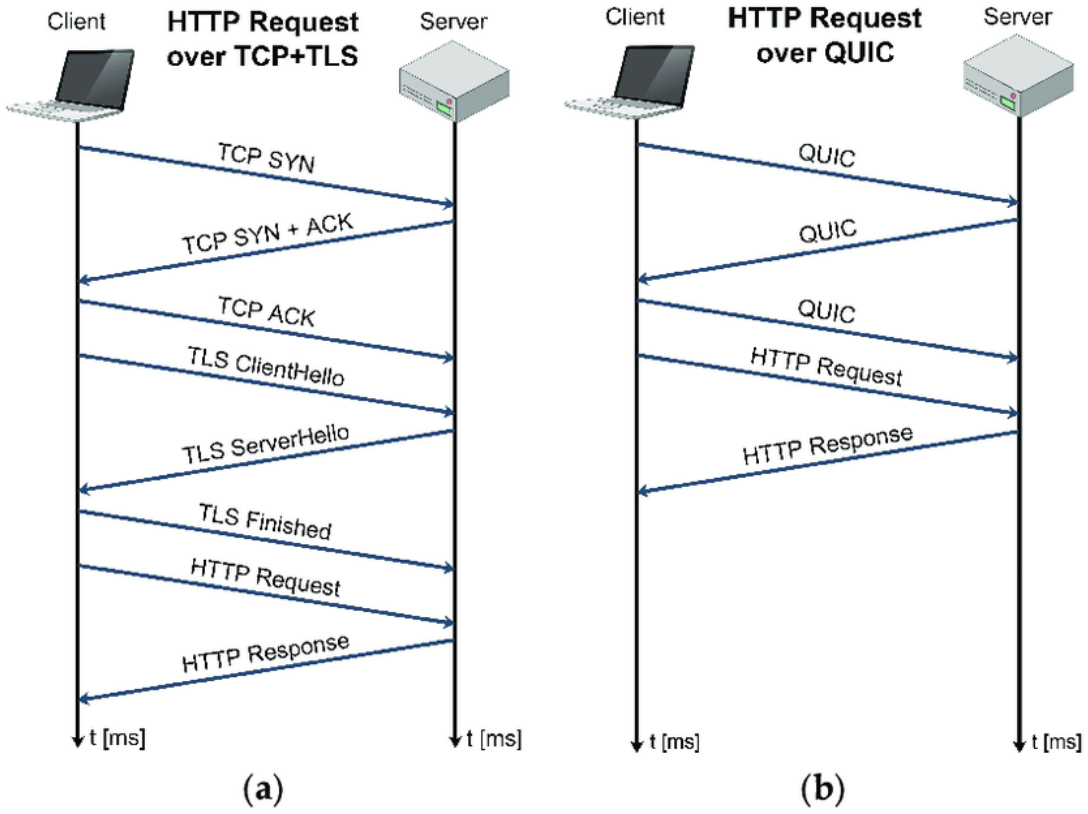
Εικόνα 5.1. Comparison of HTTP Versions

(Πηγή: https://www.researchgate.net/figure/Comparison-of-HTTP-versions_fig1_312560536)

Το 3-Way Handshake όμως είναι αρκετά χρονοβόρο, επειδή μετά από κάθε ξεχωριστό request η σύνδεση έκλεινε και έπρεπε να δημιουργηθεί από την αρχή, οπότε έπρεπε να δαπανώνται δύο RTT για νέα χειραψία. Με τη χρήση του Pipelining, η σύνδεση δεν είναι πλέον κλειστή μετά την ολοκλήρωση της τρίτης φάσης και ο client λαμβάνει πλέον ένα επιπλέον request: αυτό σημαίνει ότι η δεύτερη και η τρίτη φάση μπορούν πλέον να εκτελεστούν πολλές φορές, κάτι που μειώνει το συνολικό RTT.[25]

Με την αναβάθμιση σε HTTP/2, ήρθαν και νέες βελτιώσεις στη μετάδοση μηνυμάτων HTTP, με την ονομασία Multiplexing. Πλέον οι client και server μπορούν να αναλύουν ένα μήνυμα HTTP σε ανεξάρτητα πλαίσια, να τα παρεμβάλουν και στη συνέχεια να τα συναρμολογήσουν ξανά στο άλλο άκρο. Με αυτή τον τρόπο εξαλείφεται η ανάγκη για πολλαπλές συνδέσεις ώστε να γίνει δυνατή η παράλληλη επεξεργασία και παράδοση request και response, ωστόσο η αρχική χειραψία δεν επηρεάζεται. Για να ενισχυθεί η ασφάλεια στις μεταδόσεις αυτές η κρυπτογράφηση TLS σταμάτησε να είναι προαιρετική

και έγινε αναγκαστική. Όπως και με το HTTP/1.1, η TLS βρίσκεται αμέσως μετά την 3-Way Handshake, η οποία παραμένει στην αρχή της σύνδεσης και χρειάζεται δύο RTT, με την κρυπτογράφηση TLS να χρειάζεται ακόμα ένα RTT.



Εικόνα 5.2. Comparison of connection establishment and message exchange for: (a) TCP with TLS, (b) QUIC Protocol

(Πηγή: https://www.researchgate.net/figure/Comparison-of-connection-establishment-and-message-exchange-for-aTCP-with-TLS-b_fig3_352912159)

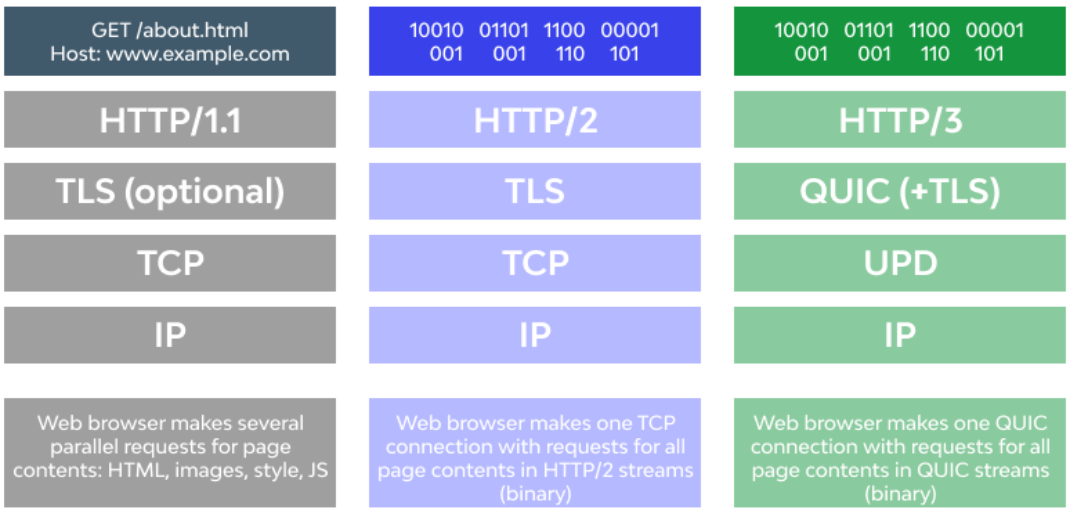
Ενώ το HTTP/2 αντιμετώπισε τον head-of-line αποκλεισμό στο πρωτόκολλο HTTP/1.1, ο αποκλεισμός σε επίπεδο TCP εξακολουθεί να προκαλεί προβλήματα. Αυτό είναι ένα από τα θέματα που θα βελτιωθεί αρκετά με το πρωτόκολλο HTTP/3. Το κύριο πρόβλημα με το TCP είναι ότι πριν από τη δημιουργία μιας συνεδρίας μεταξύ ενός client και του server, απαιτείται μια χειραψία TLS για την επαλήθευση μιας ασφαλούς συνεδρίας. Για να μειωθεί ο χρόνος για αυτές τις διεργασίες, το QUIC χρειάζεται μια μόνο χειραψία για να δημιουργήσει μια ασφαλή συνεδρία, η οποία στο σύνολό της χρειάζεται ένα RTT. Το επίπεδο QUIC εξακολουθεί να εκτελείται σε ένα κρυπτογραφημένο πρωτόκολλο μεταφοράς επειδή η TLS αποτελεί μέρος του QUIC. Το HTTP/3 μπορεί να

γίνει μόνο με ασφαλή και κρυπτογραφημένο τρόπο, ενώ η HTTP/2 έκδοση μπορεί να υλοποιηθεί χωρίς HTTPS.

Για τα HTTP/1.1 και HTTP/2, το TCP είναι ο τρόπος μεταφοράς στην αρχιτεκτονική. Ωστόσο, το HTTP/3 χρησιμοποιεί το QUIC ως το επίπεδο μεταφοράς δικτύου του, το οποίο υλοποιεί τον έλεγχο συμφόρησης χώρου χρηστών μέσω του πρωτοκόλλου User Datagram Protocol (UDP).[26]

5.2. Αρχιτεκτονική

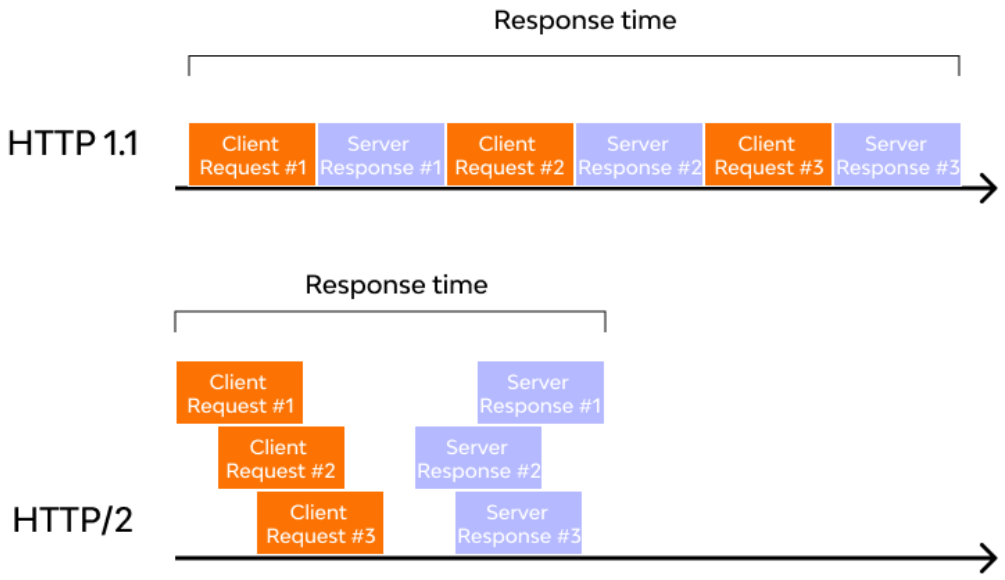
Το πρωτόκολλο HTTP αναπτύχθηκε το 1989 ως η κοινή γλώσσα που επιτρέπει την αλληλεπίδραση των μηχανών μεταξύ client και server. Τα εν λόγω requests και responses πρέπει να μεταφέρονται μεταξύ αυτών των μηχανημάτων έως ότου ο client λάβει όλους τους πόρους, απαραίτητους για τη φόρτωση μιας ιστοσελίδας στην οθόνη του τελικού χρήστη. Αυτή η ανταλλαγή request-response μπορεί να θεωρηθεί ως μια στοίβα IP που διαχειρίζεται το επίπεδο μεταφοράς και τα επίπεδα δικτύου πριν φτάσει τελικά στο επίπεδο εφαρμογής.



Εικόνα 5.3. Comparison of HTTP Architectures

(Πηγή: https://www.wallarm.com/what/what-is-http-2-and-how-is-it-different-from-http-1)

Το κύριο χαρακτηριστικό που διαφοροποιεί το HTTP/2 από το HTTP/1.1 είναι το επίπεδο δυαδικού πλαισίου. Σε αντίθεση με το HTTP/1.1, το οποίο στέλνει μηνύματα ως απλό κείμενο, το HTTP/2 χρησιμοποιεί ένα επίπεδο δυαδικού πλαισίου. Αυτό το επίπεδο ενσωματώνει μηνύματα – που έχουν μετατραπεί στο δυαδικό του ισοδύναμο – ενώ διασφαλίζει ότι η σημασιολογία του HTTP (λεπτομέρειες μεθόδου, πληροφορίες κεφαλίδας, κλπ.) παραμένει αδέσμευτη. Επίσης, αυτό το επίπεδο χωρίζει requests και responses σε μικροσκοπικά πακέτα δεδομένων και τα κωδικοποιεί. Λόγω αυτού, πολλαπλά requests και responses μπορούν να εκτελούνται παράλληλα με το HTTP/2 και οι πιθανότητες head-of-line blocking μειώνονται αισθητά. Ωστόσο, κατά καιρούς, πολλές ροές δεδομένων που απαιτούν τον ίδιο πόρο μπορεί να εμποδίσουν την απόδοση του HTTP/2. Για να επιτευχθεί καλύτερη απόδοση, το HTTP/2 έχει έναν άλλο τρόπο, τη δυνατότητα ιεράρχησης ροής.



Εικόνα 5.3. Stream Prioritization

(Πηγή: https://www.wallarm.com/what/what-is-http-2-and-how-is-it-different-from-http-1)

Όπως έχει ήδη συζητηθεί, ο client λαμβάνει μια σελίδα HTML κατά την αποστολή ενός GET request. Κατά την εξέταση των περιεχομένων της σελίδας, ο client καθορίζει ότι χρειάζεται πρόσθετους πόρους για την απόδοση της σελίδας και υποβάλλει περαιτέρω requests για την ανάκτηση αυτών των πόρων. Ως συνέπεια αυτών των αιτημάτων, ο χρόνος φόρτωσης σύνδεσης αυξάνεται. Εφόσον ο server γνωρίζει ήδη ότι ο client χρειάζεται πρόσθετα αρχεία, μπορεί να εξοικονομήσει χρόνο στον client στέλνοντας αυτούς τους πόρους πριν του ζητηθούν, προσφέροντας έτσι μια εξαιρετική λύση στο πρόβλημα.

Για να επιτευχθεί αυτό, το HTTP/1.1 έχει μια διαφορετική τεχνική που ονομάζεται resource inlining, όπου ο server περιλαμβάνει την απαιτούμενη πηγή στη σελίδα HTML ως response στο αρχικό αίτημα GET. Αν και αυτή η τεχνική μειώνει τον αριθμό των request που πρέπει να στείλει ο client, τα μεγαλύτερα αρχεία μορφής χωρίς κείμενο αυξάνουν το μέγεθος της σελίδας, το οποίο έχει ως αποτέλεσμα η ταχύτητα σύνδεσης να μειώνεται και το κύριο όφελος που προκύπτει από αυτή την τεχνική να ακυρώνεται. Ένα άλλο μειονέκτημα είναι ότι ο client δεν μπορεί να διαχωρίσει τους ενσωματωμένους πόρους από τη σελίδα HTML.

Καθώς το HTTP/2 υποστηρίζει πολλαπλά ταυτόχρονα responses στο αρχικό GET request του client, ο server παρέχει τον απαιτούμενο πόρο μαζί με τη ζητούμενη σελίδα HTML. Αυτό ονομάζεται διαδικασία server push, η οποία εκτελεί το resource inlining όπως το HTTP/1.1, ενώ διατηρεί τη σελίδα και τον προωθημένο πόρο χωριστά. Αυτή η διαδικασία διορθώνει το κύριο μειονέκτημα της ενσωμάτωσης πόρων, επιτρέποντας στον client να αποφασίσει να κάνει cache / απορρίψει τον προωθημένο πόρο ξεχωριστά από τη σελίδα HTML.

Παρά τα όλα προτερήματα του HTTP/2, κάποια από τα μειονεκτήματα που παραμένουν είναι:

- Ενώ το HTTP/2 μετρίασε τις επιπτώσεις του HOL blocking του προκάτοχού του, ο αποκλεισμός σε επίπεδο TCP εξακολουθεί να προκαλεί προβλήματα.
- Για clients που λειτουργούν σε αργό δίκτυο, τα πακέτα δεδομένων καθυστερούν και η ποιότητα του δικτύου υποβαθμίζεται σε μία μόνο σύνδεση HTTP/2. Εξαιτίας αυτού, η όλη διαδικασία επιβραδύνεται, εμποδίζοντας έτσι πολλή μεταφορά δεδομένων.
- Η αποτυχία ασφάλειας των cookies εξακολουθεί να μην αντιμετωπίζεται στο HTTP/2 όπως με τον πρόδρομό του. Τα cookies είναι αρχεία .txt που περιέχουν δεδομένα του client που λαμβάνονται από τον server και τον ιστότοπο. Ωστόσο, αυτά τα cookies

ενδέχεται να κλαπούν ή να παραβιαστούν από χάκερ, οι οποίοι μπορούν να έχουν πρόσβαση στα προσωπικά δεδομένα του χρήστη, ακόμη και χωρίς κωδικούς πρόσβασης.

Αυτά τα προβλήματα έρχεται να λύσει το HTTP/3, η πιο πρόσφατη έκδοση του HTTP που θα επηρεάσει την επικοινωνία μεταξύ clients και servers, με σημαντικές αναβαθμίσεις για την εμπειρία χρήστη, συμπεριλαμβανομένης της ασφάλειας, της αξιοπιστίας και της απόδοσης του API. Τόσο το HTTP/1.1 όσο και το HTTP/2 χρησιμοποιούν το TCP ως δίαυλο μεταφοράς, ενώ το HTTP/3 βασίζεται στο QUIC της Google - ένα πρωτόκολλο δικτύου επιπέδου μεταφοράς που εφαρμόζει τον έλεγχο συμφόρησης χώρου χρήστη μέσω UDP. Έχει έρθει με πολλές λύσεις, όπως μειωμένα αποτελέσματα απώλειας πακέτων, μηδενικό RTT, πιο ολοκληρωμένη κρυπτογράφηση και ταχύτερη εγκατάσταση σύνδεσης, για να διορθωθούν οι ελλείψεις του HTTP/2. [27]

Στη βάση του, το UDP δεν κάνει "χειραψία" όπως το TCP, ούτε απαιτεί επιβεβαίωση από τον παραλήπτη όταν λαμβάνονται πακέτα δεδομένων. Απλώς στέλνει τα πακέτα δεδομένων στον προορισμό. Η παράλειψη όλης αυτής της διαδικασίας καθιστά το UDP ταχύτερο από το TCP. Όμως, υπάρχει μια αντιστάθμιση. Στο UDP, τα πακέτα μπορεί να χαθούν. Επειδή δεν υπάρχει καμία επιβεβαίωση από τον παραλήπτη, ο αποστολέας δεν γνωρίζει ποτέ για την απώλεια του πακέτου και, επομένως, δεν υπάρχει τρόπος ανάκτησης. Το TCP, από την άλλη πλευρά, έχει σχεδιαστεί για να ανακτά τυχών απολεσθέντα πακέτα. Το TCP παρατηρεί ότι δεν έχει ληφθεί καμία επιβεβαίωση και στέλνει ξανά το πακέτο. Το TCP είναι πιο αργό, αλλά είναι ανεκτικό σε σφάλματα. Το UDP προσφέρει ταχύτητα. Το σημαντικό είναι να αποκτηθεί και ανοχή σφαλμάτων και ταχύτητα. Το HTTP/3 παρέχει και τα δύο, με την εισαγωγή του QUIC στο επίπεδο δικτύου. Ο λόγος για αυτή την εισαγωγή είναι ότι το QUIC παρέχει ανοχή σφαλμάτων για μετάδοση πακέτων δεδομένων υπό UDP. Το QUIC εφαρμόζεται μεταξύ των επιπέδων μεταφοράς (transport) και της εφαρμογής (application). Το UDP σε συνδυασμό με το QUIC, δίνει το καλύτερο και από τους δύο κόσμους: ταχύτητα με αξιόπιστη μετάδοση πακέτων. Το αποτέλεσμα είναι HTTP/3, το οποίο θα είναι σύντομα το τυπικό πρωτόκολλο και έχει ήδη δει μια τεράστια παρουσία σε βιβλιοθήκες, υποδομές και προγράμματα περιήγησης.[28]

5.3. Περιορισμοί HTTP

Ένα από τα σημεία συμφόρησης της απόδοσης HTTP είναι το άνοιγμα πάρα πολλών συνδέσεων TCP. Ένα μεγάλο μέρος των ροών δεδομένων HTTP αποτελείται από μικρές

(λιγότερο από 15 KB), εκρηκτικές μεταφορές δεδομένων σε δεκάδες διακριτές συνδέσεις TCP. Ωστόσο, το TCP έχει βελτιστοποιηθεί για συνδέσεις μεγάλης διάρκειας και ο χρόνος μετ' επιστροφής δικτύου (Round-Trip Time ή RTT) είναι επίσης ένας περιοριστικός παράγοντας στην ταχύτητα των νέων συνδέσεων λόγω των μηχανισμών ελέγχου συμφόρησης TCP. Εάν το HTTP προσπαθήσει να βελτιώσει την απόδοση ανοίγοντας περισσότερες συνδέσεις TCP και ο αριθμός των συνδέσεων είναι πολύ μεγάλος, μπορεί να προκύψει συμφόρηση δικτύου, με αποτέλεσμα υψηλή απώλεια πακέτων και τελικά χειρότερη απόδοση. Ο αριθμός των συνδέσεων γίνεται ακόμη μεγαλύτερος εάν τα πακέτα παραδίδονται από πολλά domains. Μια λύση αυτού του προβλήματος εισήχθη με τη διοχέτευση HTTP, αλλά ουσιαστικά απέτυχε λόγω προβλημάτων ανάπτυξης.

Ένας άλλος περιορισμός είναι ότι οι μεταφορές ιστού που βασίζονται στο HTTP ξεκινούν αυστηρά από τον client. Αυτό παρουσιάζει σοβαρό πρόβλημα επειδή βλάπτει σημαντικά την απόδοση στην περίπτωση φόρτωσης ενσωματωμένων (embedded) αντικειμένων. Οι servers πρέπει να περιμένουν ένα ρητό αίτημα από τον client, το οποίο μπορεί να σταλεί μόνο αφού ο client επεξεργαστεί τη γονική σελίδα. Επιπλέον, καθώς ένα τμήμα TCP δεν μπορεί να μεταφέρει περισσότερα από ένα HTTP requests ή responses, οι clients στέλνουν σημαντικό αριθμό περιττών δεδομένων με τη μορφή κεφαλίδων HTTP. Αυτό το γενικό κόστος είναι ιδιαίτερα μεγάλο εάν υπάρχουν πολλά μικρά ενσωματωμένα αντικείμενα σε μια σελίδα, επομένως τα αιτήματα και οι απαντήσεις είναι μικρές. Για μόντεμ ή συνδέσεις ADSL (Asymmetric Digital Subscriber Line) στις οποίες το εύρος ζώνης ανοδικής ζεύξης (uplink Bandwidth) είναι αρκετά χαμηλό, η καθυστέρηση που προκαλείται από αυτή την επιβάρυνση μπορεί να είναι σημαντική. Η developer κοινότητα προσπαθούσε να ελαχιστοποιήσει αυτό το αποτέλεσμα συνενώνοντας μικρά αρχεία με τον ίδιο τύπο δημιουργώντας μεγαλύτερα πακέτα και σε ορισμένες περιπτώσεις τα αρχεία ενσωματώνονται στο έγγραφο HTML (HyperText Markup Language) για να αποφευχθεί το request εντελώς. Ωστόσο, αυτές οι λύσεις μπορούν να αποτύχουν, καθώς η σύνδεση έχει αρνητικό αντίκτυπο στο caching και η κοινή μέθοδος για τη σύνδεση Cascading Style Sheets (CSS) και αρχείων JavaScript καθυστερεί την επεξεργασία.

Ένας ακόμα περιορισμός είναι τα θέματα ασφάλειας. Οι HTTP clients συχνά διαμοιράζονται έναν μεγάλο όγκο προσωπικών πληροφοριών, όπως το όνομα χρήστη, η τοποθεσία, η διεύθυνση αλληλογραφίας, οι κωδικοί πρόσβασης, τα κλειδιά κρυπτογράφησης κ.λπ. Επομένως, θα πρέπει να εξασκείται μεγάλη προσοχή ώστε να

αποτραπεί η ακούσια διαρροή αυτών των πληροφοριών μέσω του πρωτοκόλλου HTTP σε άλλες πηγές. Η τυχών αποκάλυψη της συγκεκριμένης έκδοσης λογισμικού του server μπορεί να επιτρέψει στον εν λόγω server να γίνει πιο ευάλωτος σε επιθέσεις εναντίον λογισμικού που είναι γνωστό ότι περιέχει κενά ασφαλείας.

Οι υπάρχων HTTP clients και οι user agents διατηρούν συνήθως τις πληροφορίες ελέγχου ταυτότητας επ' αόριστον. Οι παλαιότερες εκδόσεις HTTP δεν παρέχουν μια μέθοδο για έναν server να κατευθύνει τους clients να απορρίψουν αυτά τα αποθηκευμένα διαπιστευτήρια, κάτι που αποτελεί μεγάλο κίνδυνο ασφάλειας. Υπάρχουν πολλοί τρόποι αποφυγής γύρω αυτού του προβλήματος, και γι' αυτό συνιστάται για παράδειγμα η χρήση κωδικού πρόσβασης μετά από μεγάλα χρονικά διαστήματα αδράνειας.[29]

Ένας μεγάλος όγκος ερευνών ασχολείται με τη βελτίωση της ασφάλειας του Διαδικτύου παρέχοντας άμυνα έναντι επιθέσεων DoS ή διασφαλίζοντας την αυθεντικότητα και την ακεραιότητα των δεδομένων. Ενώ το HTTP δεν προστατεύει από επιθέσεις DoS σε επίπεδο IP, η ευρεία υποδομή caching και τα ιδιόκτητα προϊόντα αντιμετώπισης DoS που βασίζονται σε HTTP βελτιώνουν το status quo της προστασίας DoS. Από την άλλη πλευρά, οι εγγυήσεις αυθεντικότητας και ακεραιότητας δεδομένων μπορούν να υλοποιηθούν στο υπάρχον HTTP με διάφορους τρόπους: ενσωματώνοντας κατακερματισμούς περιεχομένου και ψηφιακές υπογραφές στην κεφαλίδα HTTP, χρησιμοποιώντας αυτοπιστοποιημένες διευθύνσεις URL ή χρησιμοποιώντας HTTPS.

Τις τελευταίες δύο δεκαετίες, έχουν προταθεί πολλές λύσεις για την επέκταση της IP για την υποστήριξη κινητικότητας, multicast, anycast, multi-path δικτύων και δικτύων ανοχής καθυστερήσεων (Delay Tolerant Networks ή DTN). Το HTTP μπορεί να προσφέρει απευθείας ορισμένες από αυτές τις υπηρεσίες. Το HTTP έχει χρησιμοποιηθεί με επιτυχία για την παροχή multicast σε μεγάλη κλίμακα μιας πηγής (π.χ. ζωντανές μεταδόσεις των Ολυμπιακών Αγώνων). Το HTTP μπορεί να παρέχει και anycast αξιοποιώντας τη λειτουργικότητα DNS anycast ή μέσω αντίστροφων HTTP proxies. Μέσω ενημερώσεων DNS, το HTTP υποστηρίζει την κινητικότητα ενός κεντρικού host. Ωστόσο, το HTTP δεν υποστηρίζει ταυτόχρονη end-host κινητικότητα ή multi-path επικοινωνίες. Πολλές προτάσεις οι οποίες στοχεύουν στη βελτίωση της αποτελεσματικότητας και της ασφάλειας του domain routing, καθώς και στην παροχή εγγυήσεων QoS. Με λίγες εξαιρέσεις, το HTTP δεν αντιμετωπίζει καμία από αυτές τις

προκλήσεις άμεσα. Επομένως, αυτός παραμένει ένας τομέας έρευνας που δεν επηρεάζεται σε μεγάλο βαθμό από το HTTP.

Εν κατακλείδι, παρά τα πολλά δυνατά του σημεία, το HTTP δεν είναι τέλειο. Αρχικά, το HTTP είναι ένα pull oriented πρωτόκολλο όπου οι δέκτες πρέπει να ζητούν ρητά νέα δεδομένα, το οποίο δεν είναι κατάλληλο για υπηρεσίες datagram και connection oriented υπηρεσίες, όπως η τηλεδιάσκεψη και εφαρμογές σε πραγματικό χρόνο. Επίσης, το HTTP επιφέρει ένα μη τετριμμένο γενικό κόστος σε σύγκριση με το IP. Τέλος, το HTTP δεν παρέχει ή παρέχει περιορισμένη υποστήριξη για QoS και για προστασία DoS επιπέδου δικτύου.[30]

5.4. Ζητήματα Απόδοσης

Σε αυτό το κεφάλαιο θα συγκρίνουμε την απόδοση του πρωτοκόλλου HTTP/2 με τον προκάτοχό του, το HTTP/1.1, στη φόρτωση ορισμένων ιστοσελίδων οι οποίες κατέγραψαν τη μεγαλύτερη επισκεψιμότητα μέσα στο έτος 2022. Οι μετρικές που χρησιμοποιήθηκαν περιλαμβάνουν:

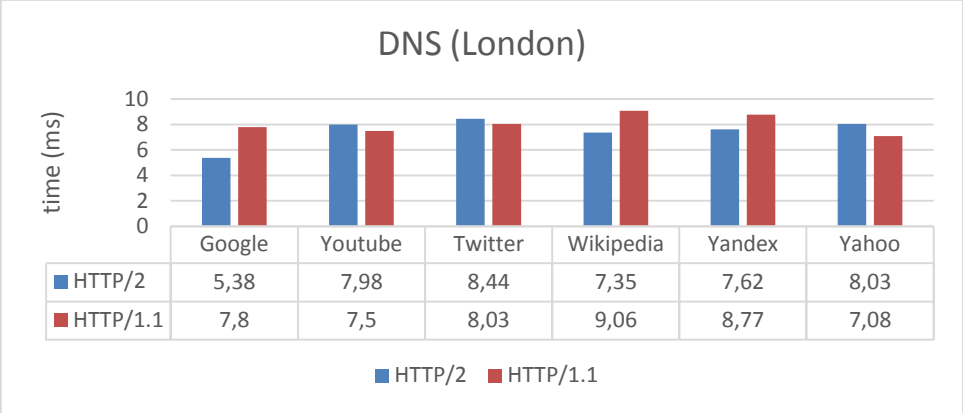
- το χρόνο αναζήτησης/απόκρισης **DNS**
- **Time to Connect**: στο χρόνο που χρειάζεται για να επιτευχθεί σύνδεση στον server, και
- **Time To First Byte**: ο χρόνος που μεσολαβεί από τη στιγμή που ένας client υποβάλλει ένα HTTP request για να λάβει το πρώτο byte δεδομένων από τον server.

Για αυτή τη σύγκριση θα χρησιμοποιήσουμε ένα εργαλείο από την ιστοσελίδα KeyCDN[18], το Performance Test. Αυτό το εργαλείο κάνει μια single asset δοκιμή απόδοσης που μπορεί να πραγματοποιηθεί προς διάφορες τοποθεσίες σε όλο τον κόσμο και επιτρέπει τη δοκιμή και τη μέτρηση της απόδοσης οποιουδήποτε URL. Για τη συγκεκριμένη μελέτη, θα λάβουμε μετρήσεις από 5 διαφορετικές τοποθεσίες στον κόσμο (Λονδίνο, Νέα Υόρκη, Ντάλας, Σύδνεϋ και Τόκυο) και για 6 διαφορετικές ιστοσελίδες (Google, Youtube, Twitter, Wikipedia, Yandex και Yahoo), χρησιμοποιώντας α) το πρωτόκολλο HTTP/1.1 και β) το πρωτόκολλο HTTP/2.

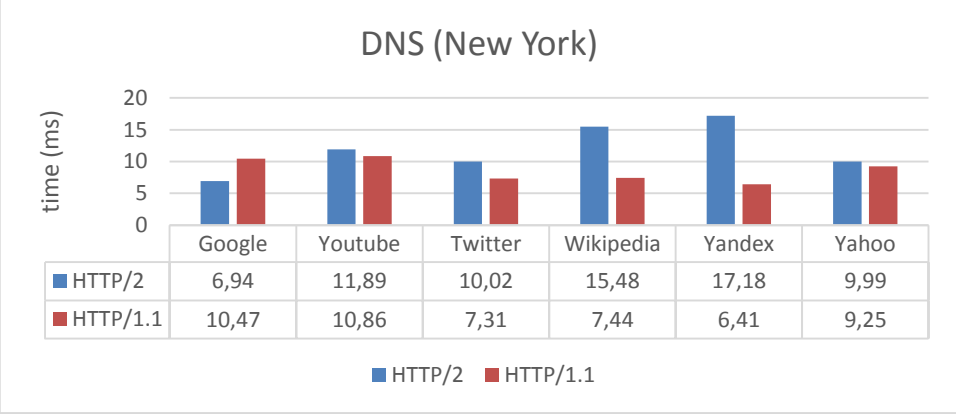
[18] https://tools.keycdn.com/performance

5.4.1. DNS

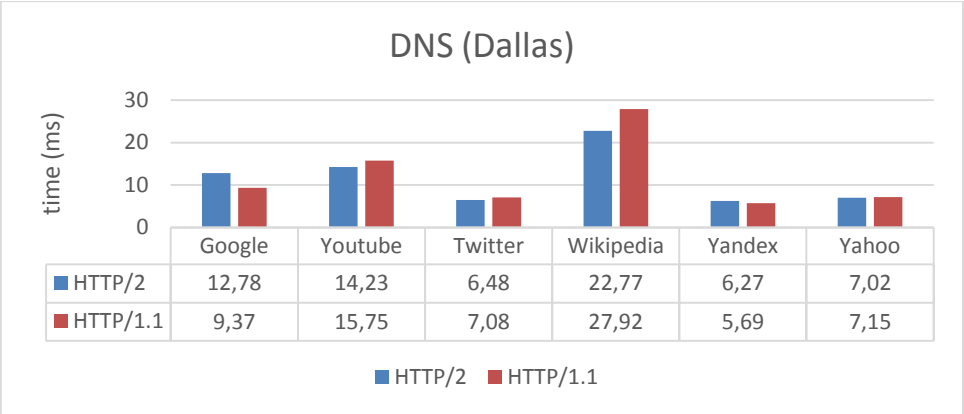
Πρώτη κατηγορία στη λίστα μας είναι ο χρόνος απόκρισης DNS. Παρακάτω παραθέτουμε τα γραφήματα με τους χρόνους αναζήτησης DNS για τις πόλεις Λονδίνο, Νέα Υόρκη, Ντάλας, Σύδνεϋ και Τόκυο.



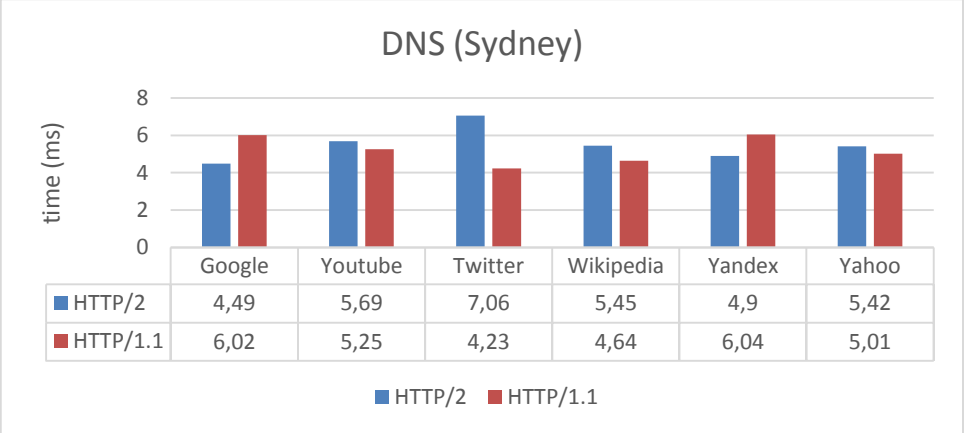
Σχήμα 1



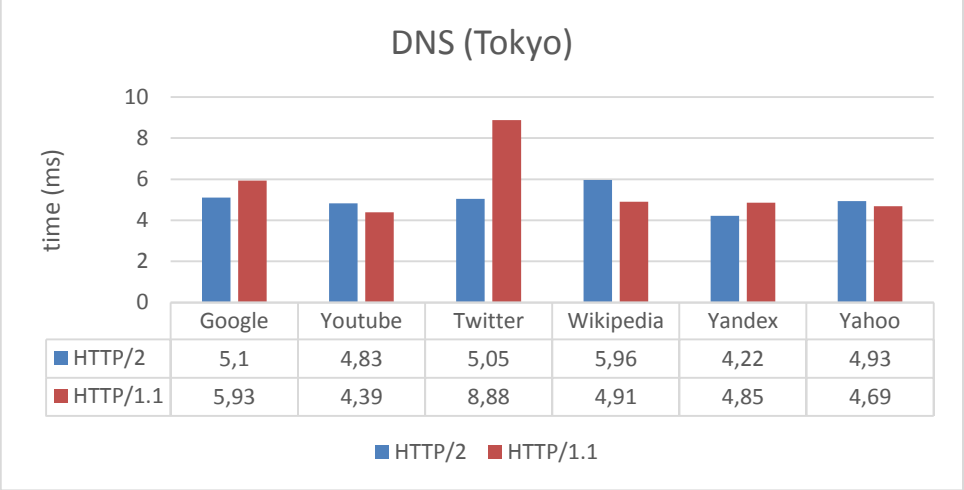
Σχήμα 2



Σχήμα 3



Σχήμα 4

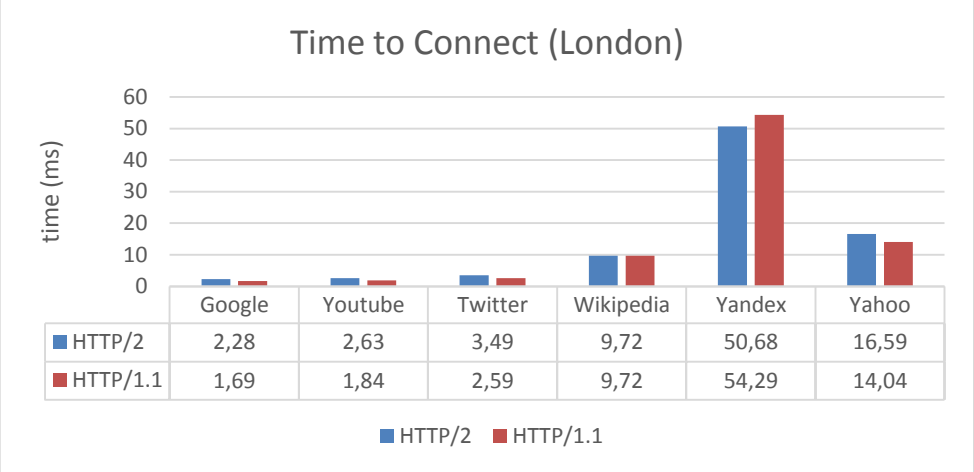


Σχήμα 5

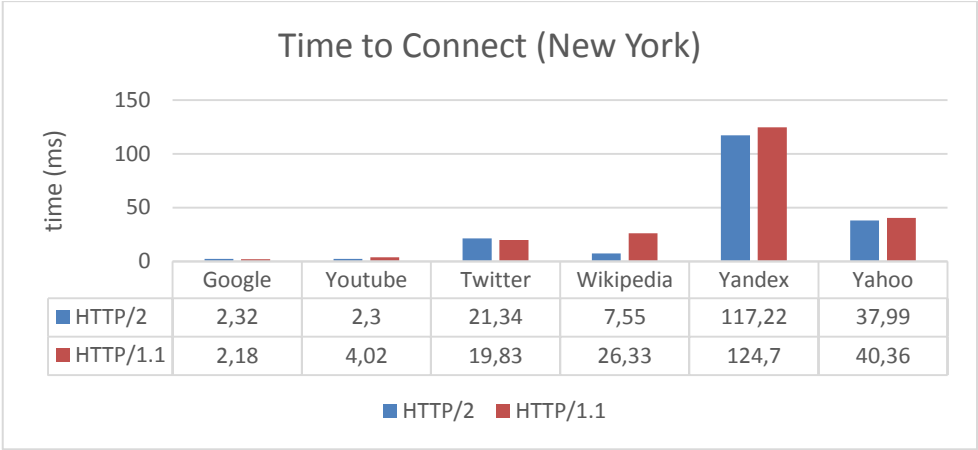
Τα σχήματα 1 έως 5 μας δείχνουν το χρόνο που χρειάστηκε για να βρεθεί ο DNS server για την κάθε ιστοσελίδα. Όπως μπορούμε να παρατηρήσουμε, δεν υπάρχουν μεγάλες διαφορές στους χρόνους ανάμεσα στα δύο πρωτόκολλα, με το HTTP/1.1 να είναι σε γενικές γραμμές λίγο πιο αργό από το HTTP/2, με εξαίρεση κάποιες περιπτώσεις όπως στο σχήμα 3, όπου το HTTP/2 χρειάστηκε σχεδόν το διπλάσιο χρόνο αναζήτησης του DNS για τα site Wikipedia και Yandex στην Νέα Υόρκη σε σχέση με το HTTP/1.1, παρόμοια και στο σχήμα 2, με το Twitter στο Σύδνεϋ.

5.4.2. Time to Connect

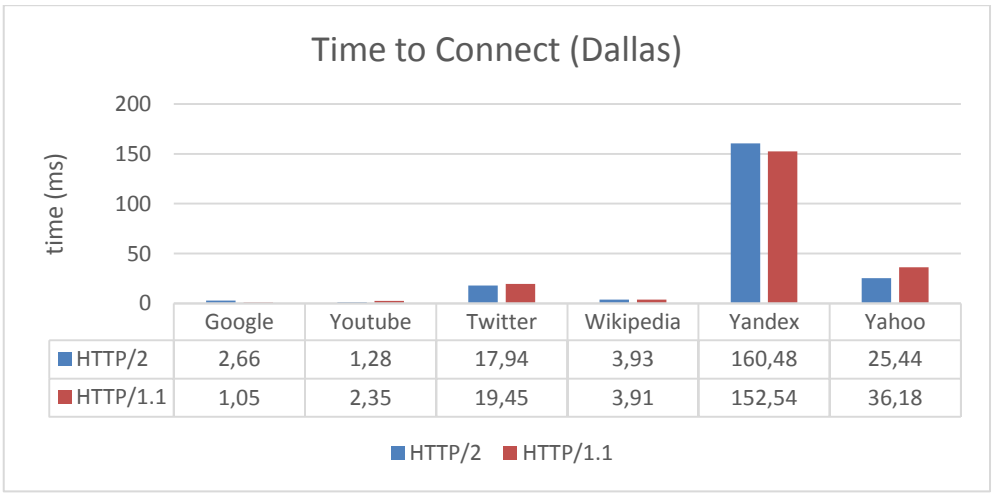
Η επόμενη κατηγορία μετρήσεων αφορά το Time to Connect. Παρακάτω παραθέτουμε τα γραφήματα Time to Connect για τις πόλεις Λονδίνο, Νέα Υόρκη, Ντάλας, Σύδνεϋ και Τόκυο.



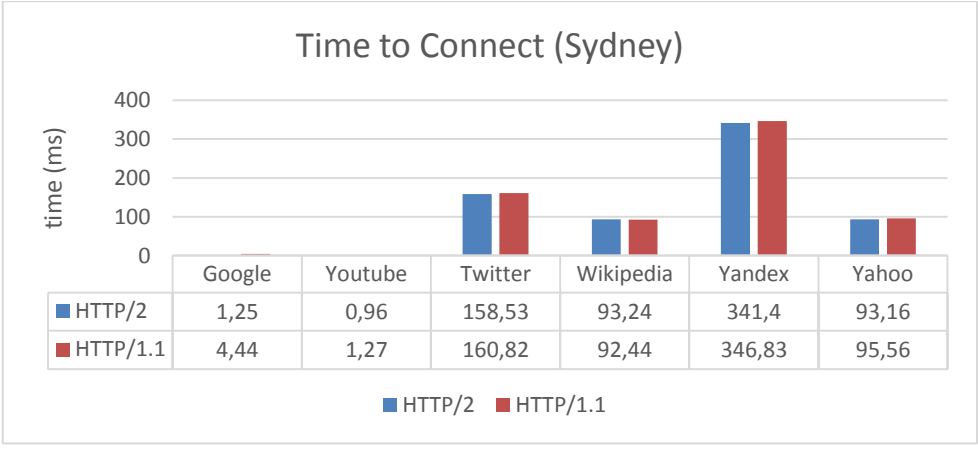
Σχήμα 6



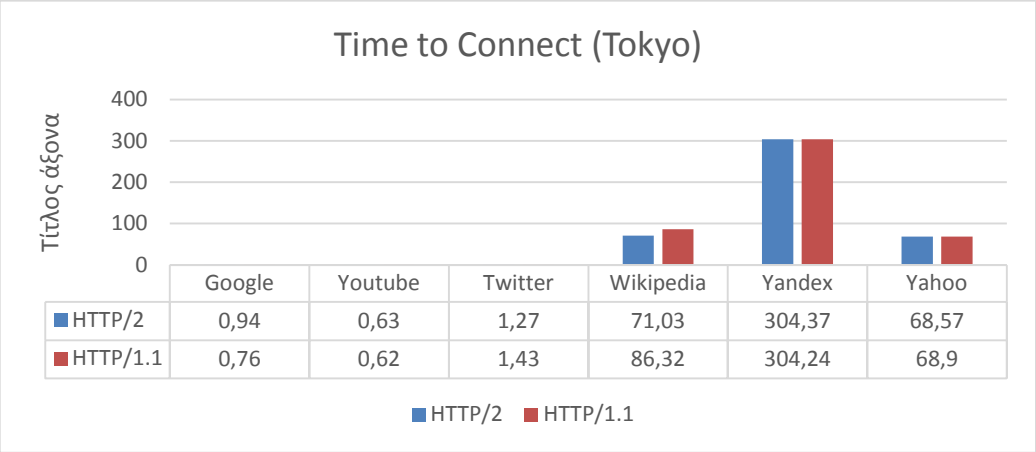
Σχήμα 7



Σχήμα 8



Σχήμα 9

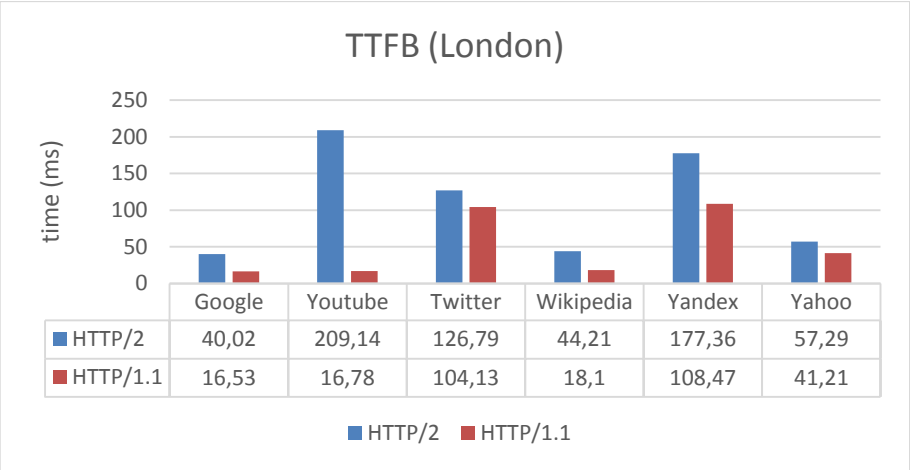


Σχήμα 10

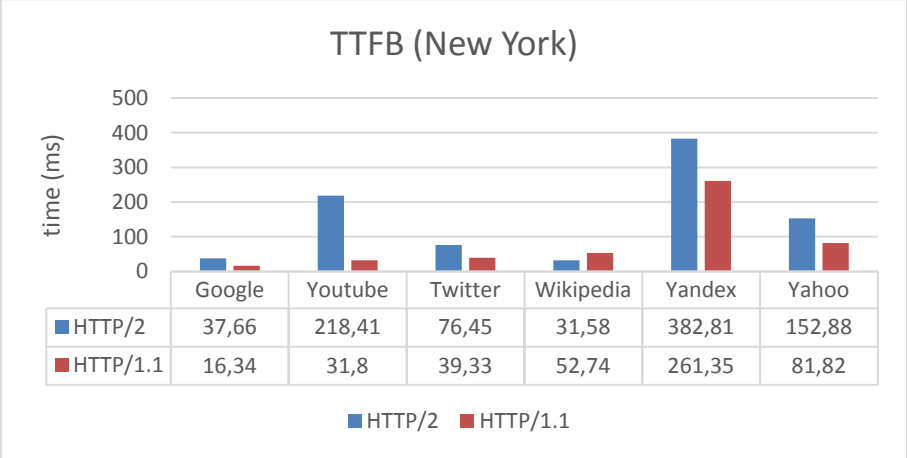
Τα σχήματα 6 έως 10 μας δείχνουν το χρόνο που χρειάστηκε για να επιτευχθεί η σύνδεση στις ίδιες τοποθεσίες. Όπως παρατηρούμε εδώ, οι χρόνοι ανάμεσα στα δύο πρωτόκολλα είναι σχεδόν πανομοιότυποι, παρόλο που οι χρόνοι που χρειάζεται η κάθε ιστοσελίδα είναι πολύ διαφορετικοί μεταξύ τους. Τη μεγαλύτερη διαφοροποίηση τη βρίσκουμε στο σχήμα 7, όπου για το Wikipedia στη Νέα Υόρκη παρατηρούμε ότι η σύνδεση επιτεύχθηκε στο HTTP/2 σχεδόν τέσσερις φορές πιο γρήγορα από ότι στο HTTP/1.1.

5.4.3. Time To First Byte

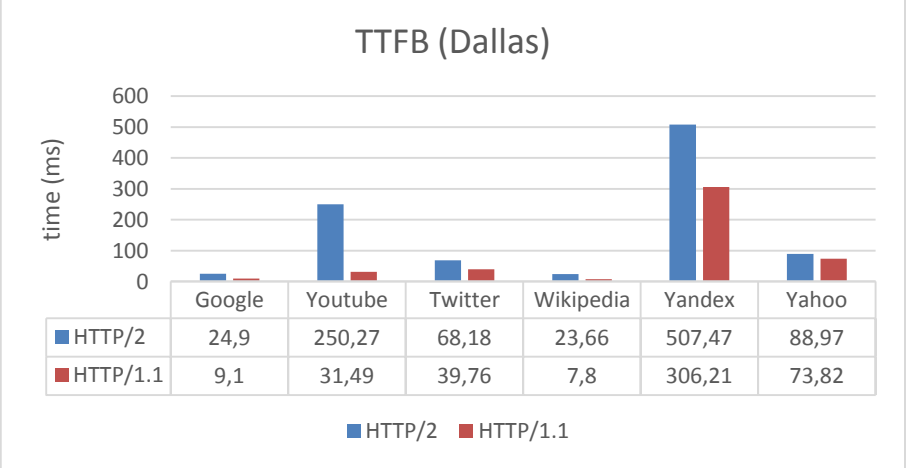
Η επόμενη κατηγορία με την οποία θα ασχοληθούμε είναι η TTFB (Time To First Byte). Παρακάτω παραθέτουμε τα γραφήματα TTFB για τις πόλεις Λονδίνο, Νέα Υόρκη, Ντάλας, Σύδνεϋ και Τόκυο.



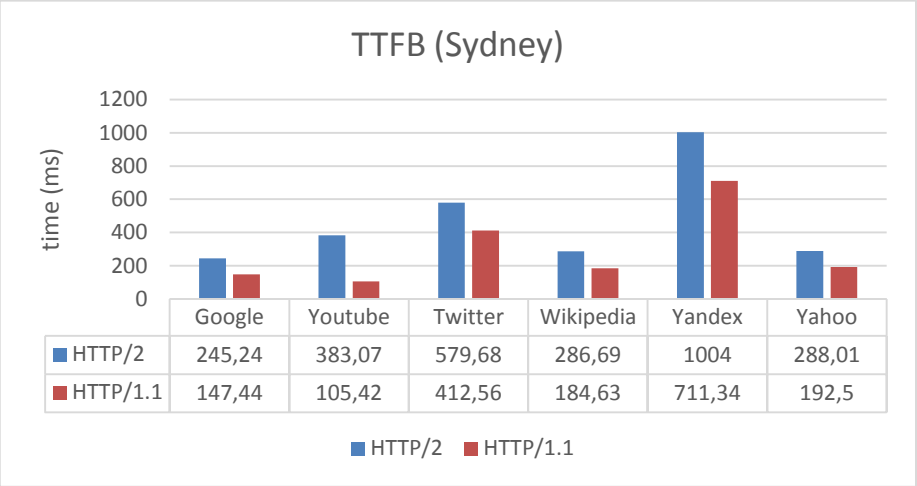
Σχήμα 11



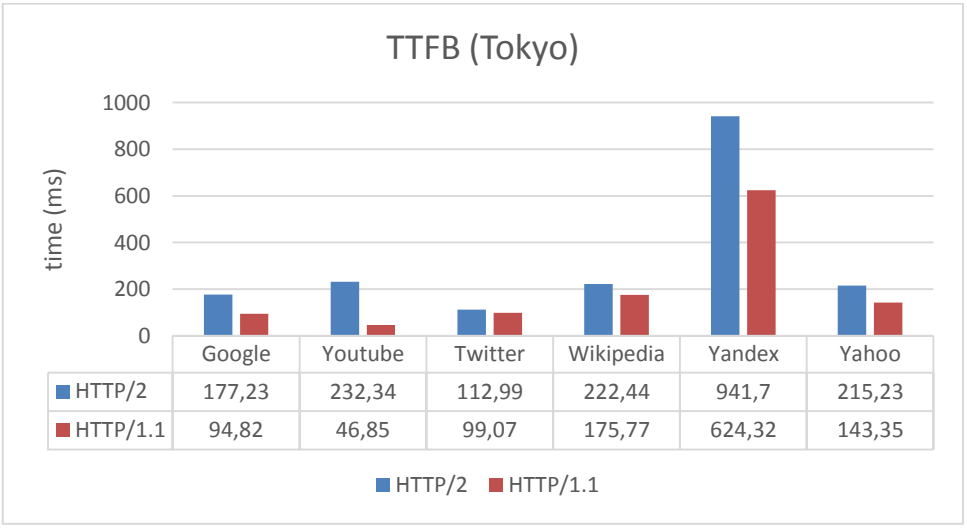
Σχήμα 12



Σχήμα 13



Σχήμα 14



Σχήμα 15

Τα σχήματα 11 έως 15 μας δείχνουν το χρόνο που χρειάστηκε για να φορτωθεί το πρώτο byte δεδομένων από τον κάθε ιστότοπο στις ίδιες τοποθεσίες. Παραδόξως, το HTTP/1.1 δείχνει τον ταχύτερο χρόνο για το πρώτο byte, σε όλα τα site και σχεδόν σε όλες

τις τοποθεσίες (μοναδική εξαίρεση αποτελεί το site της Wikipedia στη Νέα Υόρκη, όπως βλέπουμε στο σχήμα 12).

Δεν είμαστε ακριβώς σίγουροι τι θα προκαλούσε αυτή την αύξηση στη φόρτωση των δεδομένων, αλλά πιστεύουμε ότι μπορεί να έχει να κάνει με την υλοποίηση του κάθε web server. Επίσης παρατηρούμε ότι το HTTP/2 εμφανίζει εξαιρετικά υψηλή καθυστέρηση για τη λήψη του πρώτου byte σε ιστοσελίδες με πολλά μεγάλα αντικείμενα, όπως το Youtube. Αυτό πιθανότατα οφείλεται στην επιβάρυνση που σχετίζεται με τις τεχνικές multiplexing και compression του HTTP/2, οι οποίες μπορεί να προκαλέσουν καθυστερήσεις κατά τη φόρτωση μικρών πόρων. Ωστόσο, είναι σημαντικό να σημειωθεί ότι σε άλλες περιπτώσεις όπου μια ιστοσελίδα περιέχει μεγαλύτερους πόρους ή όταν υπάρχει υψηλό επίπεδο ταυτόχρονων αιτημάτων, το HTTP/2 μπορεί να είναι ταχύτερο. Σύμφωνα με τις μετρήσεις μας το latency που παρατηρούμε ήταν σταθερά υψηλό, επομένως ο μέσος όρος μας δεν έχει παραμορφωθεί από τυχών στιγμιαία προβλήματα στους server της κάθε ιστοσελίδας.

5.4.4. Συμπεράσματα

Συνολικά μπορούμε να συμπεράνουμε ότι το HTTP/2 είναι η καλύτερη επιλογή για τον ταχύτερο χρόνο φόρτωσης ιστοσελίδων. Ως επί το πλείστον, το HTTP/2 κατάφερε να ξεπεράσει το HTTP/1.1 στα δοκιμαστικά σενάρια χρόνου φόρτωσης ιστοσελίδας. Ήταν έκπληξη το γεγονός ότι το HTTP/1.1 είχε τον ταχύτερο χρόνο στο πρώτο byte. Σε περιπτώσεις όπου ένας ιστότοπος περιέχει μεγάλο αριθμό μικρών πόρων, η χρήση του HTTP/1.1 μπορεί να οδηγήσει σε ταχύτερους χρόνους φόρτωσης της σελίδας. Ωστόσο, σε άλλες περιπτώσεις, όπως αυτές με μεγάλους πόρους ή υψηλό concurrency, η χρήση του HTTP/2 μπορεί να είναι πιο ωφέλιμη.

6. Αναφορές

1. HTTP – Overview, https://www.tutorialspoint.com/http/http_overview.htm
2. Client-Server Model, https://www.geeksforgeeks.org/client-server-model/
3. https://reposit.haw-hamburg.de/bitstream/20.500.12738/12467/1/bachelorarbeit.pdf
4. https://dl.acm.org/doi/pdf/10.1145/306225.306230
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). RFC2616: Hypertext Transfer Protocol--HTTP/1.1.
6. https://dspace.cvut.cz/bitstream/handle/10467/76234/F8-DP-2018-Drhova-Klara-thesis.pdf
7. Kurose, J., & Ross, K. (2010). Computer networks: A top down approach featuring the internet.
8. Grigorik, I. (2013). *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.".

9. https://kinsta.com/blog/http3/
10. https://www.akamai.com/blog/performance/http3-and-quic-past-present-and-future
11. https://datatracker.ietf.org/doc/rfc8999/
12. https://datatracker.ietf.org/doc/rfc9000/
13. https://datatracker.ietf.org/doc/rfc9001/
14. https://datatracker.ietf.org/doc/rfc9002/
15. https://datatracker.ietf.org/doc/rfc9110/
16. https://datatracker.ietf.org/doc/rfc9111/
17. https://datatracker.ietf.org/doc/rfc9112/
18. https://datatracker.ietf.org/doc/rfc9113/
19. https://datatracker.ietf.org/doc/rfc9114/
20. https://datatracker.ietf.org/doc/draft-ietf-quic-version-negotiation/
21. https://datatracker.ietf.org/doc/rfc9308/
22. https://datatracker.ietf.org/doc/rfc9312/
23. https://datatracker.ietf.org/doc/draft-ietf-quic-load-balancers/
24. https://datatracker.ietf.org/doc/rfc9221/
25. https://developer.mozilla.org/en-US/docs/Web/HTTP/Session
26. https://www.section.io/engineering-education/http3-vs-http2/
27. https://www.wallarm.com/what/what-is-http-2-and-how-is-it-different-from-http-1
28. https://www.redhat.com/architect/http3
29. Megyesi, P., Kramer, Z. and Molnar, S. (2016) "How quick is Quic?," *2016 IEEE International Conference on Communications (ICC)*.
30. Popa, L., Ghodsi, A. and Stoica, I., 2010, October. HTTP as the Narrow Waist of the Future Internet. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (pp. 1-6).
31. Lorincz, J., Klarin, Z. and Ožegović, J., 2021. A comprehensive overview of TCP congestion control in 5G networks: Research challenges and future perspectives. *Sensors*, *21*(13), p.4510.