ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
Προηγμένα Πληροφοριακά Συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη Σύγχρονων Τεχνολογιών Επικοινωνίας Πελάτη-Εξυπηρέτη

Παρασκευή Δ. Ρήγα

Επιβλέποντες: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής
Βασίλειος Τσέτσος, Διδάκτωρ ΕΚΠΑ

ΑΘΗΝΑ
ΜΑΡΤΙΟΣ 2013

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μελέτη Σύγχρονων Τεχνολογιών Επικοινωνίας Πελάτη-Εξυπηρέτη

Παρασκευή Δ. Ρήγα

Α.Μ.: Μ952

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ευστάθιος Χατζηευθυμιάδης** Αναπληρωτής Καθηγητής
Βασίλειος Τσέτσος, Διδάκτωρ ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Ευστάθιος Χατζηευθυμιάδης,** Αναπληρωτής Καθηγητής

Μάρτιος 2013

ΠΕΡΙΛΗΨΗ

Η διπλωματική αυτή εργασία έχει ως στόχο τη μελέτη των διαφόρων τεχνολογιών επικοινωνίας που έχουν αναπτυχθεί για το μοντέλο πελάτη-εξυπηρέτη. Στο αρχικό μοντέλο την επικοινωνία ξεκινούσε ο πελάτης κάνοντας ρητές αιτήσεις και ο εξυπηρέτης έστελνε, εν συνεχεία, τις απαντήσεις του. Αυτός ο τρόπος επικοινωνίας είναι ξεκάθαρος και στο HTTP πρωτόκολλο το οποίο αποτελεί μέχρι και σήμερα το βασικό τρόπο επικοινωνίας ανάμεσα σε πελάτη και εξυπηρέτη.

Η ανάγκη για πιο δυναμικές ιστοσελίδες, οι οποίες θα ενημερώνονταν σε πραγματικό χρόνο, έδωσε ώθηση σε νέες τεχνολογίες και πρωτόκολλα, τα οποία δουλεύουν πάνω από το HTTP ή το αναβαθμίζουν. Κατ' αρχήν, μια πολύ σημαντική εξέλιξη ήταν η ανάπτυξη του AJAX. Η προσέγγιση αυτή αν και δεν άλλαξε την κατεύθυνση της επικοινωνίας πελάτη-εξυπηρέτη, βελτίωσε πολύ τον τρόπο που παρουσιάζονται οι σελίδες στο χρήστη - πλέον τα δεδομένα μοιάζουν να είναι πραγματικού χρόνου. Οι εφαρμογές, όμως, καθώς και η ανάγκη υποστήριξης πολλών ταυτόχρονων χρηστών, απαιτούσαν κάτι περισσότερο, μια τεχνολογία η οποία θα κάνει αυτόματη προώθηση των δεδομένων από τον εξυπηρέτη στον πελάτη. Οι δύο βασικότεροι πρωταγωνιστές σήμερα σε αυτή την προσπάθεια είναι η προσέγγιση COMET και τα API Server Sent Events και WebSocket της HTML5.

Στην παρούσα εργασία γίνεται μια επισκόπηση των σημαντικότερων τεχνολογιών όπως αυτές προτάθηκαν στο πέρασμα του χρόνου. Πιο συγκεκριμένα, ελέγχεται τι υποστήριξη υπάρχει στην πλευρά του πελάτη-φυλλομετρητή, ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα της κάθε τεχνολογίας και πώς υποστηρίζονται από τους διάφορους εξυπηρέτες εφαρμογών. Στην πλευρά του εξυπηρέτη έγινε ένα πείραμα ελέγχου φόρτου το οποίο επιβεβαιώνει το γεγονός ότι το πιο σύγχρονο API, το WebSocket, έχει χαμηλότερη καθυστέρηση και κλιμακώνεται πολύ καλύτερα από το COMET.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Εφαρμογές διαδικτύου

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: πελάτης, εξυπηρέτης, αυτόματη προώθηση, Ajax, COMET, WebSocket

ABSTRACT

This diploma thesis aims at studying the various communication technologies that have been developed to support the client-server model. In the original model, the communication was initiated explicitly by a client request, and the server responded with an answer. This way of communication is clear in the HTTP protocol, which remains the main way of communication between a client and a webserver until today.

The need for more dynamic websites, which will be updated in real time, gave impetus to new technologies and protocols, which work over HTTP or upgrade it. Firstly, a very important step was the development of AJAX. This approach, although it did not change the direction of the client-server communication, it greatly improved the way webpages are presented to the user - data seem to be real time. Current applications, however, and the need to support multiple concurrent users, required something more, a technology that will automatically forward data from the server to the client. The two main protagonists today in this effort are the COMET approach and two APIs introduced in HTML5, Server Sent Events and WebSocket API.

This thesis presents an overview of the major technologies presented through the years to support the client-server communication. More specifically, it studies the support given on the client-side (browser), the advantages and disadvantages of each technology, and how they it is implemented by different servers, application and dedicated ones. On the server side, a load test experiment was conducted which confirms the fact that the latest API, the WebSocket, has lower delay and scales much better than the COMET, which makes it the most promising technology for RIA.

SUBJECT AREA: Internet applications

KEYWORDS: client, server, push, Ajax, COMET, WebSocket

ΕΥΧΑΡΙΣΤΙΕΣ

Θέλω να ευχαριστήσω όλους όσους με βοήθησαν στην εκπόνηση της διπλωματικής μου εργασίας. Το Δρ. Βασίλειο Τσέτσο που μου έδωσε κατευθύνσεις και μου προσέφερε την πολύτιμη βοήθειά του καθ' όλη την ομολογουμένως μεγάλη διάρκειά της, τον Καθηγητή κο. Ευστάθιο Χατζηευθυμιάδη που με βοήθησε στη βελτίωση της παρούσας μελέτης και δεν έθιξε ποτέ το γεγονός ότι καθυστέρησα, αλλά και όλους τους συγγενείς και φίλους που μου θύμιζαν συνεχώς την ύπαρξή της. Κάποιοι από αυτούς βοήθησαν και στη διόρθωση του κειμένου, πέρνοντας μέρος της ευθύνης για το αποτέλεσμα. Ευχαριστώ.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1. ΕΙΣΑΓΩΓΗ

Τα τελευταία χρόνια, το Διαδίκτυο έχει καταστεί από σημαντικό έως αναγκαίο για τις επιχειρήσεις, την εκπαίδευση, και την κυβέρνηση. Η πιο γνωστή και ταχύτερα αναπτυσσόμενη πτυχή του Διαδικτύου (Internet) είναι ο Παγκόσμιος Ιστός (Web). Ο Ιστός διαθέτει ενισχυμένη τυπογραφία, φωτογραφίες, ήχο, ακόμα και βίντεο. Οι χρήστες μπορούν να πλοηγηθούν εύκολα από τη μία ιστοσελίδα στην άλλη, κάνοντας κλικ σε υπερσυνδέσεις. Ο Ιστός αποτελεί μία σημαντική πηγή εμπορικών πληροφοριών, ειδήσεων, κυβερνητικών δεδομένων και εκπαιδευτικών πόρων. Αν η σημερινή τάση συνεχιστεί, ίσως να γίνει το κυρίαρχο μέσο σε όλους αυτούς τους τομείς.

Αξίζει πραγματικά να αναφερθεί ότι οι διαδικτυακές εφαρμογές ήρθαν για να καλύψουν ήδη υπάρχουσες ανάγκες. Αν και τα περισσότερα σύγχρονα προϊόντα, φυσικά ή ψηφιακά, καλύπτουν ανάγκες που τα ίδια δημιουργούν, η εξέλιξη του Ιστού στη σημερινή της μορφή προέκυψε από την ανάγκη για άμεση και δυναμική αλληλεπίδραση. Από την ανάγκη να υποστηρίξει εφαρμογές οι οποίες επιτρέπουν στους ανθρώπους να μετάσχουν σε δραστηριότητες που μετείχαν και πριν, χωρίς όμως να είναι απαραίτητη η φυσική τους παρουσία σε συγκεκριμένο φυσικό χώρο. Ψωνίζουμε, παίζουμε, μαθαίνουμε, επικοινωνούμε με οποιονδήποτε, ενημερωνόμαστε για οτιδήποτε και όλα αυτά χωρίς να χρειάζεται να κουνηθούμε ούτε εκατοστό από την καρέκλα μας.

Ο Παγκόσμιος Ιστός (World Wide Web) [1] βρίσκεται, λοιπόν, σε διαρκή ανάπτυξη, όχι μόνο ως προς το περιεχόμενο και τους χρήστες, αλλά και ως προς τον τρόπο αλληλεπίδρασης. Από ένα αποθετήριο πληροφοριών στην αρχή της δεκαετίας του 1990, όπου ο χρήστης λάμβανε την πληροφορία που τον ενδιέφερε σε μια στατική σελίδα, στη συνέχεια άρχισε να εμπλουτίζεται με δυναμικά στοιχεία και να μετατρέπεται έτσι ραγδαία σε ένα μέσο άμεσης επικοινωνίας - οι χρήστες συμμετέχουν ενεργά μέσα από το φυλλομετρητή τους σε ιστοσελίδες-εφαρμογές και οι Πλούσιες Διαδικτυακές Εφαρμογές [2] (Rich Internet Applications – RIA) ανταποκρίνονται δυναμικά στις οποιεσδήποτε αλλαγές. Οι χρήστες χρησιμοποιούν τις RIA για να συνομιλούν με άλλους χρήστες, να εκθέτουν απόψεις, πληροφορίες και προϊόντα, να συμμετέχουν σε εμπορικές δραστηριότητες, να διασκεδάζουν, να μαθαίνουν τις ειδήσεις, να ενημερώνονται για την πορεία συγκεκριμένων δραστηριοτήτων ακόμη και για να παρακολουθούν ή να ελέγχουν συσκευές που δε βρίσκονται στο άμεσο περιβάλλον τους.

Σκοπός αυτής της εργασίας είναι να κάνει μια επισκόπηση των μεθόδων που χρησιμοποιεί ο Ιστός για την επικοινωνία Πελάτη-Εξυπηρέτη (Client-Server), να τις συγκρίνει και να παρουσιάσει κάποια API, framework και βιβλιοθήκες που έχουν αναπτυχθεί για να τις υποστηρίξουν. Έτσι, σε αυτό το σημείο, πριν περάσουμε σε μια πιο λεπτομερή ανάλυσή τους, θα δούμε επιγραμματικά κάποιες από τις τεχνολογίες που θεωρούνται σταθμοί στην επικοινωνία Client-Server.

1.1. Το πρωτόκολλο HTTP

Το Hypertext Transfer Protocol (HTTP) [3] σχεδιάστηκε στις πρώτες ημέρες του Διαδικτύου για τη μεταφορά οντοτήτων πληροφοριών όπως ιστοσελίδες ή αρχεία πολυμέσων μεταξύ πελατών και εξυπηρετών.

Το πρωτόκολλο εφαρμόζει ένα αυστηρό μοντέλο αίτησης-απάντησης (request-response), το οποίο σημαίνει ότι ο πελάτης στέλνει μια αίτηση HTTP και περιμένει έως ότου λάβει την απάντηση HTTP. Το ίδιο πρωτόκολλο, όμως, δεν έχει καμία υποστήριξη για την περίπτωση όπου ο εξυπηρέτης ξεκινά αλληλεπίδραση με τον πελάτη.

Γενικά, οι υπηρεσίες που προσφέρει κάποιο σύστημα είναι δυνατό να διακριθούν σε «Υπηρεσίες Ρητής Αίτησης» (Pull-based services) και «Υπηρεσίες Αυτόματης Προώθησης» (Push-based services). Μια Push υπηρεσία είναι μια εφαρμογή λογισμικού που γνωρίζει τι είδους περιεχόμενο ενδιαφέρει τον κάθε χρήστη και του το παρέχει όταν αυτό είναι διαθέσιμο ή τον πληροφορεί από που μπορεί να το πάρει. Αντίθετα, στις Pull υπηρεσίες ο χρήστης κάνει αίτηση για να λάβει κάποιο περιεχόμενο και το δίκτυο του στέλνει σαν απάντηση το περιεχόμενο αυτό ή ένα μήνυμα αποτυχίας. Το πρωτόκολλο HTTP υποστηρίζει μόνο Pull υπηρεσίες.

1.2. Η προσέγγιση AJAX

Για την ανάπτυξη πιο διαδραστικών εφαρμογών web, θα πρέπει το περιεχόμενο που βλέπει ο χρήστης να μεταβάλλεται δυναμικά και όχι μόνο μετά από κάθε ρητή του αίτηση. Η προσέγγιση AJAX [4] έχει καθιερωθεί ως μια δημοφιλής λύση για δυναμικό pulling δεδομένων από τον εξυπηρέτη. Με τη χρήση AJAX, η εφαρμογή του προγράμματος περιήγησης εκτελεί HTTP αιτήματα στο παρασκήνιο με ένα ασύγχρονο τρόπο, χωρίς να εμποδίζει το περιβάλλον εργασίας του χρήστη.

Ωστόσο η προσέγγιση AJAX δεν κάνει τίποτα για το push δεδομένων στον πελάτη.

1.3. Η προσέγγιση Comet

Αντίθετα με το πώς ξεκίνησε το Διαδίκτυο, πολλές από τις σημερινές διαδικτυακές εφαρμογές πρέπει να λαμβάνουν πληροφορίες, μόλις αυτές δημοσιευθούν στο server. Το Comet, επίσης γνωστό ως reversed Ajax, ενισχύει το πρότυπο επικοινωνίας AJAX ορίζοντας μια αρχιτεκτονική για το push των δεδομένων από τον εξυπηρέτη στον πελάτη.

Η προσέγγιση Comet επιτρέπει την ανάπτυξη διαδικτυακών εφαρμογών πραγματικού χρόνου, χωρίς να αλλάζει όμως το υποκείμενο HTTP πρωτόκολλο. Κάποια πολύ γνωστά πρωτόκολλα του Comet, τα οποία επιτρέπουν αμφίδρομη επικοινωνία μεταξύ client-server, είναι τα Bayeux [5] και BOSH [6]. Αυτά το πρωτόκολλα υλοποιούνται ωστόσο πάνω από το περιβάλλον της γλώσσας του φυλλομετρητή.

1.4. Η προσέγγιση της HTML5

Η HTML5 [7]επεκτείνει το πρότυπο HTML [HTML standard] για να υποστηρίζει καλύτερα άκρως διαδραστικές διαδικτυακές εφαρμογές. Η HTML5 μετατρέπει τα προγράμματα περιήγησης στο Web σε πλατφόρμες που μπορούν να υποστηρίξουν RIA στην πλευρά του πελάτη και είναι άμεσος ανταγωνιστής άλλων περιβαλλόντων πλατφόρμας πελάτη όπως η Adobe Flash και η Microsoft Silverlight.

Η HTML5 εφαρμόζει το πρότυπο επικοινωνίας Comet με τον ορισμό των Server-Sent-Events (SSE) [8]. Εδώ η επικοινωνία είναι μονοκατευθυντική και πιο απλή από αυτή των πρωτόκολλων Bayeux και BOSH. Αν και παρέχει λιγότερη λειτουργικότητα από τα προηγούμενα, το SSE αναμένεται ότι θα γίνει το κυρίαρχο πρωτόκολλο στις περιπτώσεις που δεν χρειάζεται αμφίδρομο κανάλι επικοινωνίας (που είναι και οι περισσότερες). Το SSE θα υποστηρίζεται εγγενώς από όλους τους HTML5-συμβατούς φυλλομετρητές και άρα δε θα χρειάζεται επιπλέον προγραμματισμός για την πλευρά του πελάτη.

Η HTML5 συστήνει ακόμη ένα API το οποίο πάει ένα βήμα πιο πέρα στην αμφίδρομη επικοινωνία. Πρόκειται για το WebSocket [9], το οποίο είναι η πλέον σύγχρονη επιλογή στην ασύγχρονη επικοινωνία. Όντας πρότυπο, αναμένεται ότι σε λίγο καιρό θα υποστηρίζεται από όλους τους σύγχρονους φυλλομετρητές και εξυπηρέτες.

Στο Κεφάλαιο 2 θα γίνει μια αναφορά στις πραγματικές εφαρμογές που έρχονται να στηρίξουν οι τεχνολογικές λύσεις και μια θεωρητική περιγραφή των προσεγγίσεων pull και push στις διαδικτυακές εφαρμογές. Έπειτα θα περιγραφούν οι τεχνολογικές λύσεις για pull (Κεφάλαιο 3) και θα συνεχίσουμε με μια σύντομη ανάλυση των λύσεων που έχουμε για push (Κεφάλαιο 4). Το Κεφάλαιο 5 κάνει αναφορά ενός νέου πρωτοκόλλου, το οποίο ενδέχεται να ενσωματωθεί στο HTTP 2.0. Η κάθε τεχνολογική λύση έχει φυσικά τα πλεονεκτήματα και τα μειονεκτήματά της. Προηγουμένως δε γίνεται ειδική μνεία σε αυτά, μιας και σκοπός είναι μια πρώτη ανάγνωση των διαθέσιμων τεχνολογιών. Θέλοντας να αναδείξουμε λοιπόν τα θεικά και τα αρνητικά κάθε τεχνολογίας δημιουργήθηκε το Κεφάλαιο 6, το οποίο κάνει μια επιγραμματική αναφορά στα πλεονεκτήματα και στα μειονεκτήματα κάθε λύσης και κλείνει με ένα συγκεντρωτικό συγκριτικό πίνακα. Αφού ολοκληρώθηκε η μελέτη για την πλευρά του πελάτη περνάμε στον εξυπηρέτη. Το Κεφάλαιο 7 δίνει έναν τρόπο αξιολόγησης της απόδοσης των εξυπηρετών και τα αποτελέσματα της εφαρμογής τέτοιων πραγματικών πειραμάτων και συνεχίζει με την υλοποίηση αυτόματης προώθησης από κάποιους ευρέως χρησιμοποιούμενους application server. Μια πλούσια διαδικτυακή εφαρμογή, όπως θα δούμε, δεν μπορεί πλέον να έχει αναπτυχθεί μόνο με μια συγκεκριμένη τεχνολογική λύση. Αν θέλουμε η εφαρμογή να είναι όσο καλύτερη γίνεται και ταυτόχρονα να μπορεί να τη χρησιμοποιήσει κάθε χρήστης ανεξαιρέτως, τότε θα πρέπει αυτή να υλοποιείται χρησιμοποιώντας πάνω από μία τεχνικές λύσεις. Προφανώς κάτι τέτοιο κάνει την ανάπτυξη και συντήρησή της πολύ δύσκολη, οπότε θα πρέπει να χρησιμοποιηθεί κάποιο framework (Κεφάλαιο 8) που επιτρέπει την ανάπτυξη μιας εφαρμογής σε ένα επίπεδο ανώτερο από αυτό των λύσεων που αναφέρονται στο Κεφάλαιο 4. Πέρα από τους application server οι οποίοι εξετάστηκαν στο Κεφάλαιο 7, υπάρχουν και άλλοι εξυπηρέτες και βιβλιοθήκες για COMET και WebSocket (Κεφάλαια 9 και 9.2).

Στο Παράρτημα 1 δίνονται τα πειραματικά αποτελέσματα της ενότητας **Error! Reference source not found.** και στο Παράρτημα 2 κάποιοι ελληνικοί και αγγλικοί όροι που χρησιμοποιούνται μέσα στο βασικό κείμενο.

2. ΤΕΧΝΙΚΕΣ ΡΗΤΗΣ ΑΙΤΗΣΗΣ ΚΑΙ ΑΥΤΟΜΑΤΗΣ ΠΡΟΩΘΗΣΗΣ ΣΤΙΣ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ

Η διάδοση των δεδομένων στον Ιστό γινόταν αρχικά με τη μορφή ρητών αιτήσεων από τον πελάτη προς τον εξυπηρέτη. Οι αιτήσεις που μπορεί να αφορούν σε δεδομένα ή και υπηρεσίες μεταδίδονται στη συνέχεια από τον εξυπηρέτη στον πελάτη. Αυτή η τεχνολογία είναι γνωστή ως "**client pull**", αφού ο πελάτης κάνει την αίτηση και στη συνέχεια «τραβά» τις πληροφορίες από τον εξυπηρέτη μέσω του καναλιού επικοινωνίας.

Η τεχνική pull χρησιμοποιείται ευρέως στο Διαδίκτυο για HTTP αιτήσεις σελίδων από δικτυακούς τόπους. Οι εφαρμογές που έχουν ρητή αλληλεπίδραση με το χρήστη συνοψίζονται κάτω από το όνομα Web 2.0. Ωστόσο, ο αριθμός των εφαρμογών οι οποίες βασίζονται στην άμεση αλληλεπίδραση μεταξύ πολλών χρηστών είναι σήμερα πολύ μεγαλύτερος από τον αριθμό των κλασικών εφαρμογών του Web 2.0.

Η κατάσταση πολλών διαδικτυακών εφαρμογών είναι εγγενώς ασταθής. Αλλαγές μπορεί να προέλθουν από διάφορες πηγές όπως άλλοι χρήστες, εξωτερικές ειδήσεις και δεδομένα, αποτελέσματα πολύπλοκων υπολογισμών, σκανδαλιστών (triggers) οι οποίοι βασίζονται στην τρέχουσα ώρα και ημερομηνία. Στις σύγχρονες εφαρμογές οι πιθανές ενέργειες ενός χρήστη εξαρτώνται από προηγούμενες ενέργειες άλλων χρηστών ή και αλλαγές που μπορεί να προέκυψαν. Στο πλαίσιο αυτό, ο χρόνος παίζει σημαντικό ρόλο όσον αφορά στη σχέση μεταξύ των μεμονωμένων δράσεων. Υπάρχουν, δηλαδή, πολλές περιπτώσεις όπου η σελίδα πρέπει να ανανεώνεται για να έχει χρονική συνοχή. Η γρήγορη ανταλλαγή μηνυμάτων μεταξύ πελάτη και εξυπηρέτη είναι αναγκαία, αλλά όχι ικανή συνθήκη, για να επιτρέψει την αλληλεπίδραση μεταξύ των χρηστών σε πραγματικό χρόνο. Πολλές εφαρμογές είναι stateful. Οι χρήστες, οι οποίοι αργούν να συμμετάσχουν σε μια αλληλεπίδραση ή που συνδέονται ακόμη και μετά το τέλος της, θα πρέπει να έχουν τη δυνατότητα να μοιραστούν το ίδιο πλαίσιο αλληλεπίδρασης με αυτούς που συμμετείχαν από την αρχή.

Στην περίπτωση των χρονικά μεταβαλλόμενων δεδομένων ο πελάτης δεν μπορεί να λειτουργεί ως μάντης και να ζητά ανανέωση της σελίδας που θα συμπέσει με την ενημέρωση των δεδομένων στον εξυπηρέτη. Ακόμη όμως και αν ήξερε κάθε πότε συμβαίνουν οι αλλαγές, δε θα πρέπει να σταματά τη δουλειά (task) που κάνει για να

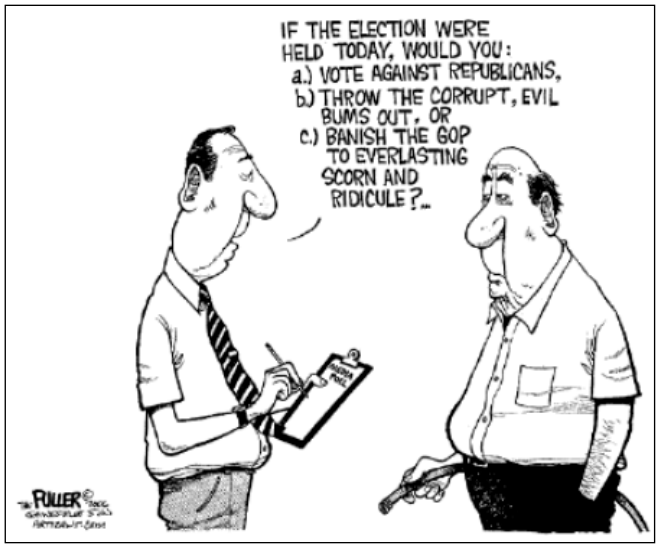
ασχοληθεί με την ανανέωση. Αντίθετα, θα πρέπει ο χρήστης να ενημερώνεται «ζωντανά» για τις αλλαγές της κατάστασης των χρονικά μεταβαλλόμενων δεδομένων τα οποία τον ενδιαφέρουν με μια διαδικασία ανανέωσης, η οποία θα γίνεται αυτόματα όσον αφορά στον ίδιο.

Μια πρώτη λύση για τα δυναμικά δεδομένα είναι να γίνεται συχνά και αυτόματα client pull με βάση τη δυναμική των δεδομένων και τις απαιτήσεις χρονικής συνοχής του χρήστη. Η τεχνική αυτή, αν και φαίνεται να λύνει το αρχικό πρόβλημα, οδηγεί σε νέα προβλήματα, που έχουν να κάνουν με το bandwidth, καθώς επιβαρύνει τόσο τον εξυπηρέτη όσο και το δίκτυο. Η κάθε νέα ενημέρωση μπορεί να γίνεται περιοδικά σε σταθερό χρόνο ή σε χρόνο που υπολογίζεται από τον εξυπηρέτη και έχει να κάνει με το πότε αναμένεται να αλλάξουν τα δεδομένα. Και στις δύο περιπτώσεις είναι σημαντική η παράμετρος της ελάχιστης χρονικής συνοχής που χρειάζεται ο χρήστης.

Η δεύτερη λύση είναι η τεχνολογία “**server push**”, όπου ο εξυπηρέτης μπορεί και διατηρεί πληροφορίες κατάστασης σχετικές με τους πελάτες και προωθεί αυτόματα σε κάθε χρήστη μόνο εκείνες τις αλλαγές που τον ενδιαφέρουν. Έτσι, η τεχνολογία server push αναφέρεται σε οποιαδήποτε τεχνική λύση, της οποίας στόχος είναι να αντιστραφεί το κλασικό παράδειγμα του Ιστού, όπου γίνεται ρητή αίτηση (pulling) των δεδομένων από το χρήστη, άμεσα ή έμμεσα.

Πώς προέκυψε όμως ο όρος server push;

Ο όρος «αυτόματη προώθηση» (push) επινοήθηκε το 1996 και έγινε σύντομα πολύ επιτυχής (από άποψη μάρκετινγκ) ως τρόπος αναφοράς στα νέα σύνορα του Διαδικτύου. Η φράση «διαλογή με προώθηση» (push polling) χρησιμοποιήθηκε από τα ΜΜΕ το 1996 κατά τη διάρκεια των αμερικάνικων εκλογών και αναφερόταν σε μια πλάγια τεχνική προεκλογικής εκστρατείας στην οποία αυτοί που αναζητούσαν ψήφους, προσποιούμενοι ότι διεξάγουν μια τηλεφωνική έρευνα, χρησιμοποίησαν την κλήση για να προωθήσουν τις αρετές του υποψηφίου τους (Εικόνα 1).



Εικόνα 1: Σκίτσο του πραγματικού push poll

Φυσικά, το “push polling” του 1996 έχει αρνητική έννοια αφού προωθούσε με δόλιο τρόπο τους πολιτικούς υποψηφίους. Το “server push” το οποίο συναντάμε στις διαδικτυακές εφαρμογές δεν αποσκοπεί στη δόλια προώθηση πληροφοριών. Σκοπός του, όπως θα δούμε στη συνέχεια, είναι η προώθηση επίκαιρης πληροφορίας και η αύξηση της διαδραστικότητας και της ευχρηστίας των εφαρμογών, έτσι ώστε να εξυπηρετούν καλύτερα το χρήστη.

Ως εδώ μιλήσαμε αρκετά γενικά για τους μηχανισμούς ενημέρωσης των δεδομένων που λαμβάνει ο χρήστης στο φυλλομετρητή του. Στη συνέχεια θα δούμε κάποιες ενδεικτικές διαδικτυακές εφαρμογές με χρονικά μεταβαλλόμενα δεδομένα που επιτάσσουν τη χρήση του συνεχούς client pull ή του server push και κάνουν εμφανές το ότι η τεχνολογία ρητής αίτησης πράγματι δεν καλύπτει όλες τις ανάγκες.

2.1. Παραδείγματα δυναμικών διαδικτυακών εφαρμογών

Οι εφαρμογές, οι οποίες απαιτούν συνεχή ενημέρωση του πελάτη, θα μπορούσαν χονδρικά να χωριστούν σε δύο κατηγορίες. Αυτές είναι οι εξής:

- η **παρακολούθηση (monitoring)**, όπου υπάρχει συνεχής ενημέρωση για κάτι που ενδιαφέρει το χρήστη, όπως είναι οι καιρικές συνθήκες, οι μετοχές ή η κατάσταση κάποιου συστήματος. Ο χρήστης χρησιμοποιεί την ιστοσελίδα σαν να είναι πίνακας ελέγχου της εφαρμογής. Χρειάζεται να ενημερώνεται τόσο για την εξέλιξη της

δραστηριότητας που τον ενδιαφέρει όσο και για τις ειδοποιήσεις του συστήματος (π.χ. ειδοποιήσεις ασφαλείας και παραβιάσεις SLA).

- οι **εφαρμογές πολλών χρηστών**, όπως η συνομιλία (chat), τα παιχνίδια και οι ιστοθέσεις διαδικτυακών αγορών (markets). Ο χρήστης χρησιμοποιεί την ιστοσελίδα για να επικοινωνήσει άμεσα με άλλους χρήστες ή για να δώσει δεδομένα και να κάνει επιλογές που επηρεάζονται ή επηρεάζουν τα δεδομένα και τις επιλογές των υπόλοιπων χρηστών της ίδιας εφαρμογής.

Στη συνέχεια γίνεται μια σύντομη παρουσίαση κάποιων ενδεικτικών εφαρμογών που υπάρχουν. Η πρώτη εφαρμογή αφορά στην παρακολούθηση και οι άλλες τρεις σε εφαρμογές πολλών χρηστών.

2.1.1. Τηλετυπώτης μετοχών (Stock ticker)

Μία εφαρμογή προώθησης η οποία γνώρισε μεγάλη επιτυχία τη δεκαετία του 1990 ήταν το PointCast Network [10]. Το PointCast ήταν το πρώτο σύστημα προώθησης που βασιζόταν σε κανάλια (βλέπε ενότητα 2.2.1) και παρέδιδε νέα και δεδομένα χρηματιστηριακών. Η Netscape και η Microsoft το ενσωμάτωσαν στο λογισμικό τους στην ακμή του πολέμου των φυλλομετρητών, αλλά αργότερα εξαφανίστηκε και αντικαταστάθηκε στη δεκαετία του 2000 με το RSS [11] (μία τεχνολογία pull).

Η μεταφορά των δεδομένων των τιμών της αγοράς με χαμηλή καθυστέρηση αποδείχθηκε βασική για τη δημιουργία υψηλής ποιότητας πυλών(portals) εμπορίου. Το περιεχόμενο, το οποίο προωθείται, αποτελείται από μια σειρά ενημερώσεων πραγματικού χρόνου μιας ιστοσελίδας. Η Εικόνα 2 δείχνει ένα τυπικό παράδειγμα των δεδομένων των μετοχών σε ένα χρηματιστήριο, όπως αυτά προβάλλονται και ενημερώνονται σε μια html σελίδα. Τα δεδομένα που προωθεί ο server συχνά τονίζονται για λίγο (ειδοποιήσεις) ώστε να τραβήξουν την προσοχή του χρήστη.

Name	Last	Time	Change	Min	Max
Anduct	3.20	3:45:49 ▲	+5.26%	2.79	3.33
Ations Eu	17.01	3:46:08 ▲	+5.71%	13.29	18.85
Bagies Co	6.35	3:46:08 ▼	-11.68%	6.22	8.46
BAY Corpo	3.77	3:45:23 ▲	+3.85%	3.50	3.82
CON Consu	6.94	3:46:08 ▼	-8.80%	6.43	8.46
Corcor PL	2.56	3:46:07 ▲	+11.30%	2.23	2.61
CVS Asia	15.19	3:46:09 ▼	-1.29%	12.61	16.99
Datio PLC	4.65	3:46:06 ▼	-12.42%	4.44	5.95

(GMT+01 Timezone)

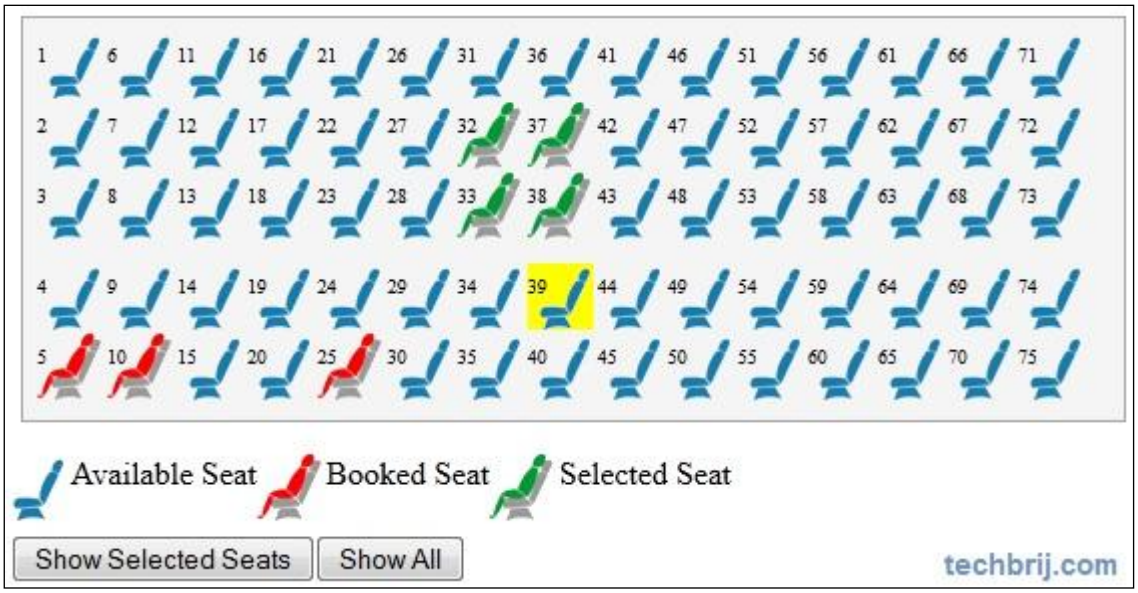
Εικόνα 2: Παράδειγμα χρήσης τεχνολογίας αυτόματης προώθησης με δείκτες μετοχών

Ανάλογες πραγματικές εφαρμογές παρακολούθησης αναπτύσσονται για αθλητικά αποτελέσματα, για αφίξεις αεροπλάνων και πλοίων, για την κατάσταση του καιρού αλλά και για τον έλεγχο πολύπλοκων συστημάτων, ιατρικών συσκευών και δικτύων αισθητήρων.

2.1.2. Διαδικτυακή κράτηση πτήσης

Πολλές εφαρμογές μπορεί να έχουν κάποιο περιβάλλον επισκέπτη (frontend) στο Web μέσω του οποίου τα δεδομένα στο εξυπηρέτη ενημερώνονται από πολλούς χρήστες. Τα ηλεκτρονικά συστήματα κράτησης πτήσεων αποτελούν ένα τέτοιο παράδειγμα. Εάν ένας πελάτης κάνει μια ενημέρωση, οι άλλοι δεν πρόκειται να το δουν αν δεν ανανεώνουν συνεχώς τις σελίδες τους. Αν και σε ορισμένες περιπτώσεις η ανανέωση της σελίδας αποτελεί μια απλή και εφαρμόσιμη λύση, σε άλλες περιπτώσεις οι χρήστες πρέπει να συγχρονιστούν με την ενημέρωση της σελίδας καθώς αυτή συμβαίνει.

Στην Εικόνα 3 βλέπουμε τη διεπαφή μιας διαδικτυακής εφαρμογής κρατήσεων πτήσης όπου ο χρήστης καλείται να επιλέξει συγκεκριμένες θέσεις από τις διαθέσιμες.



Εικόνα 3: Παράδειγμα χρήσης τεχνολογίας αυτόματης προώθησης θέσεων επιβατών σε κράτηση πτήσης

Στην ίδια κατηγορία εφαρμογών ανήκουν οι ηλεκτρονικές δημοπρασίες και τα στοιχήματα.

2.1.3. Διαδικτυακά συστήματα πολλών χρηστών

Η αυτόματη προώθηση βρίσκει ακόμη εφαρμογή όταν απαιτείται συνεργασία πραγματικού χρόνου μιας ομάδας ατόμων μέσω των φυλλομετρητών τους, όπως συμβαίνει στα διαδικτυακά παιχνίδια ή στο διαμοιραζόμενο μαυροπίνακα (blackboard). Ο προγραμματιστής μπορεί να δημιουργήσει παιχνίδια κίνησης-οθόνης (move-display) ή να φτιάξει ένα διαδικτυακό μαυροπίνακα. Ο εξυπηρέτης θα μπορούσε να εφαρμόζει τη λογική του παιχνιδιού και να προωθεί τις αλλαγές της κατάστασης του παιχνιδιού στους πελάτες και στην περίπτωση του μαυροπίνακα θα πρέπει να φροντίζει να ενημερώνονται άμεσα όλοι οι χρήστες για τις αλλαγές που γίνονται. Σε κάθε συνεργατικό σύστημα θα πρέπει όλοι οι χρήστες να ενημερώνονται για όλες τις αλλαγές με τη σειρά που γίνονται και χωρίς καθυστερήσεις.

Στην Εικόνα 4 βλέπουμε τη διεπαφή ενός διαδικτυακού παιχνιδιού χαρτιών πολλών χρηστών.

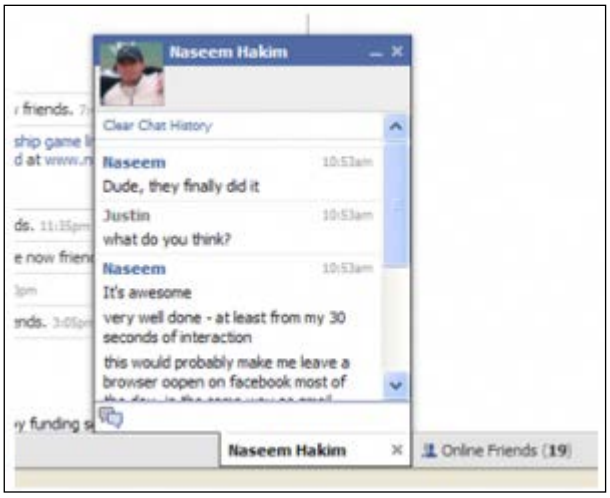
Εικόνα 4: Παράδειγμα online παιχνιδιού χαρτιών πολλών παιχτών που κάνει χρήση αυτόματης προώθησης

2.1.4. Συνομιλία πραγματικού χρόνου

Η σύγχρονη διάσκεψη (synchronous conferencing) και τα άμεσα μηνύματα (instant messaging) είναι τυπικά παραδείγματα εφαρμογών αυτόματης προώθησης. Τα μηνύματα συζήτησης, και κάποιες φορές τα αρχεία, προωθούνται στο χρήστη με το που λαμβάνονται από την υπηρεσία μηνυμάτων, αφού η ανταλλαγή πρέπει να γίνεται χωρίς καθυστερήσεις.

Πολλές εταιρείες όπως η AOL, η Yahoo και η Microsoft παρέχουν διαδικτυακές εφαρμογές chat στους χρήστες τους μέσω των φυλλομετρητών. Ακόμη, κάποια αποκεντρωμένα peer-to-peer προγράμματα (όπως το WASTE [12]) και συγκεντρωτικά προγράμματα (όπως τα IRC [13] και Jabber [14]) επιτρέπουν την προώθηση αρχείων από τον αποστολέα στον παραλήπτη.

Στην Εικόνα 5 βλέπουμε τη διεπαφή του Facebook Chat, όπου χρησιμοποιείται επιτυχώς το server push.



Εικόνα 5: Παράδειγμα διεπαφής μιας chat εφαρμογής σε φυλλομετρητή

2.1.5. Προώθηση ηλεκτρονικού ταχυδρομείου

Η αυτόματη προώθηση των μηνυμάτων ηλεκτρονικού ταχυδρομείου (Push email delivery), η οποία έγινε δημοφιλής από την υπηρεσία RIM/Blackberry, προσφέρεται τώρα από διάφορες υπηρεσίες web-based email, συμπεριλαμβανομένης της Gmail της εταιρίας Google.

2.1.6. Υπηρεσίες κοινωνικής δικτύωσης

Οι υπηρεσίες κοινωνικής δικτύωσης (social networking services) όπως το Twitter και το Facebook δίνουν ιδιαίτερα δυναμικές ιστοσελίδες οι οποίες μπορεί να ενημερώνονται αρκετές φορές μέσα σε ένα λεπτό. Η οθόνη του χρήστη αποτελείται από πολλά ανεξάρτητα τμήματα που αλλάζουν σύμφωνα με τις ενέργειες του ίδιου αλλά και άλλων χρηστών. Πρόκειται για εφαρμογές που ανήκουν ταυτόχρονα στην κατηγορία της παρακολούθησης και των πολλών χρηστών.

2.2. Θεωρητικό υπόβαθρο αυτόματης προώθησης

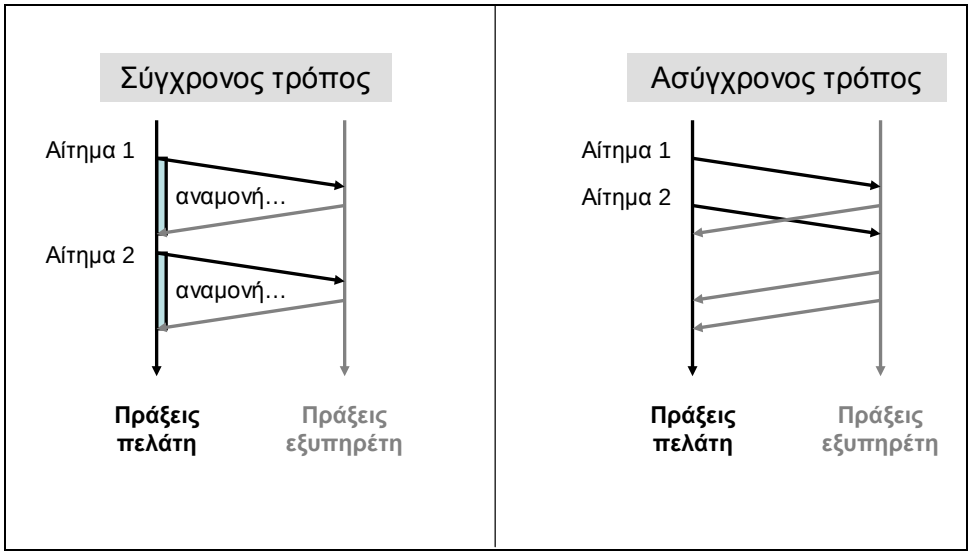
Στην Ενότητα 2.1 δόθηκαν κάποιες εφαρμογές της τεχνολογίας «αυτόματης προώθησης». Πέρα από το πρακτικό, υπάρχει και το πιο θεωρητικό κομμάτι της περιγραφής των τεχνικών ρητής αίτησης και αυτόματης προώθησης.

2.2.1. Ορισμοί ρητής αίτησης και αυτόματης προώθησης

Στα επιστημονικά έγγραφα, που πραγματεύονται τις δύο τεχνικές, οι ακόλουθοι όροι χρησιμοποιούνται ως συνώνυμα ή είναι μέρος των ίδιων λογικών κατηγοριών:

- **ρητή αίτηση** (client pull), σύγχρονος τρόπος (synchronous mode), αίτημα/απάντηση (request/response)
- **αυτόματη προώθηση** (server push), ασύγχρονος τρόπος (asynchronous mode), δημοσίευση/εγγραφή (publish/subscribe)

Η Εικόνα 6 παρουσιάζει γραφικά τις χαρακτηριστικές διαφορές μεταξύ των δύο κατηγοριών. Στο κλασικό παράδειγμα του Ιστού, που βασίζεται σε αιτήσεις, κάθε μήνυμα που παράγεται από τον εξυπηρέτη (server) πρέπει να ζητηθεί με ένα αίτημα πελάτη. Επιπλέον, αφού κάνει το αίτημα ο πελάτης, το νήμα εκτέλεσής του συνήθως μπλοκάρεται ενώ περιμένει μια απάντηση. Με την τεχνολογία push, ο πελάτης απλά παρουσιάζει ενδιαφέρον για κάποιου είδους πληροφορία (γίνεται μέλος στα αποκαλούμενα *θέματα* (topic, subject), *κανάλια* (channel), ή *στοιχεία* (item)) και έπειτα λαμβάνει ασύγχρονα μηνύματα από τον εξυπηρέτη. Αυτό σημαίνει ότι ο πελάτης δεν «μπλοκάρεται» ποτέ περιμένοντας απαντήσεις στα αιτήματά του και, προ πάντων, ο εξυπηρέτης μπορεί να παραδώσει τα μηνύματα κατά την κρίση του (δηλαδή όταν υπάρχουν πραγματικά νέα δεδομένα), αντί να αναμένει να ζητήσει ο πελάτης ενημερώσεις.



Εικόνα 6: Γραφική απεικόνιση σύγχρονου και ασύγχρονου τρόπου

2.2.2. Αρχιτεκτονικές αυτόματης προώθησης

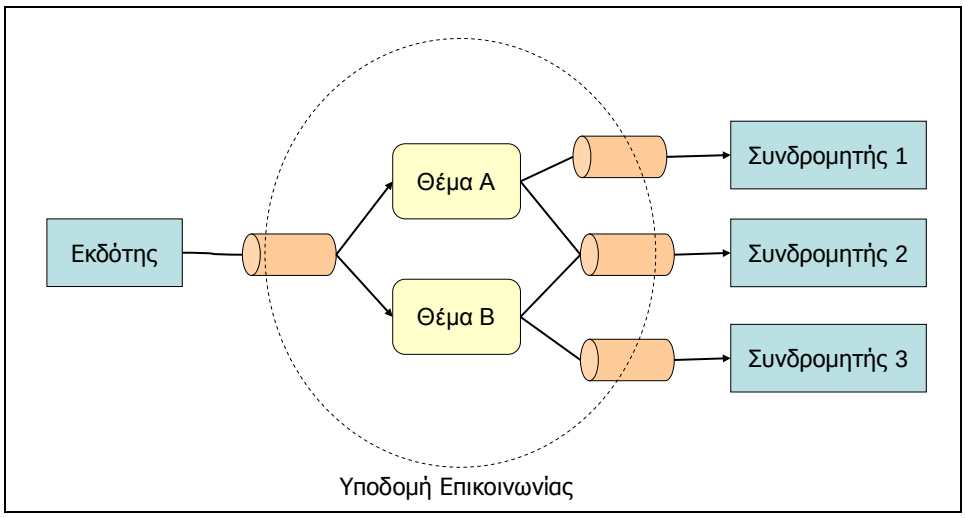
Οι εφαρμογές που είδαμε στην Ενότητα 2.1 μπορούν να διαχωριστούν στις κατηγορίες της παρακολούθησης και των εφαρμογών πολλών χρηστών [15]. Η κάθε κατηγορία έχει τη δική της μικροαρχιτεκτονική που δείχνει την αλληλεπίδραση των αντικειμένων στην επίλυση του προβλήματος (δηλ. την επίτευξη του εκάστοτε στόχου). Η πρώτη κατηγορία ακολουθεί το πρότυπο Observer (Παρατηρητή) και η δεύτερη το πρότυπο Model-View-Controller.

2.2.2.1. Πρότυπο Παρατηρητής (Observer)

Το πρότυπο Παρατηρητής ορίζει την εξάρτηση αντικειμένων ένα-προς-πολλά: όταν ένα αντικείμενο αλλάζει, τότε όλα τα εξαρτώμενα αντικείμενα ενημερώνονται για την αλλαγή. Στις εφαρμογές παρακολούθησης οι πελάτες πρέπει να παρατηρούν (δηλ. παρακολουθούν) τον εξυπηρέτη. Άρα το πρότυπο που ακολουθείται είναι το Observer, όπου ο εξυπηρέτης είναι το **υποκείμενο** (subject) και οι clients είναι οι **παρατηρητές** (observer).

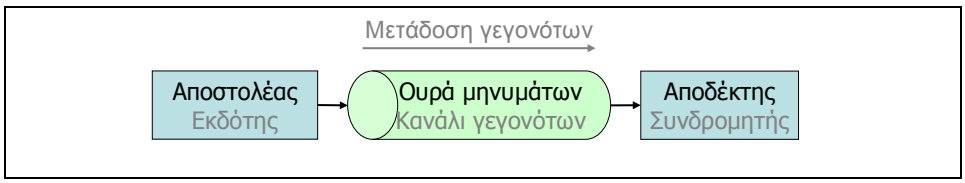
Το πρότυπο Παρατηρητή αποτελεί υποσύνολο του προτύπου *Δημοσίευσης/Εγγραφής* (Publish/Subscribe – Εικόνα 7). Η ιδέα της αυτόματης προώθησης πληροφορίας στους πελάτες αποτέλεσε τον καταλύτη για τους μηχανισμούς Δημοσίευσης/Εγγραφής [16].

Το *πρότυπο Δημοσίευσης/Εγγραφής* είναι ένα παράδειγμα μηνυμάτων όπου οι **εκδότες** (publisher) των μηνυμάτων δεν είναι προγραμματισμένοι για να στείλουν τα μηνύματά τους σε συγκεκριμένους **συνδρομητές** (subscriber) [17]. Αντί αυτού, τα δημοσιευμένα μηνύματα χαρακτηρίζονται σε κατηγορίες, χωρίς γνώση του ποιοι εκδότες μπορεί να υπάρχουν. Οι συνδρομητές εκφράζουν ενδιαφέρον (subscribe) για μια ή περισσότερες κατηγορίες (information channel), και στη συνέχεια ο εξυπηρέτης τούς προωθεί μόνο τα ενδιαφέροντα μηνύματα. Αυτή η *απόζευξη* (decoupling) των εκδοτών και των συνδρομητών μπορεί να επιτρέψει μεγαλύτερη *κλιμάκωση* (scalability) και μια δυναμικότερη τοπολογία δικτύου. Στις εφαρμογές παρακολούθησης μπορούμε να πούμε ότι οι πελάτες είναι οι συνδρομητές, οι οποίοι εκφράζουν εκ των προτέρων τις προτιμήσεις τους, και ο εξυπηρέτης είναι ο μόνος εκδότης.



Εικόνα 7: Γραφική απεικόνιση επικοινωνίας μοντέλου publish/subscribe

Το πρότυπο Δημοσίευσης/Εγγραφής είναι συγγενές του *μοντέλου Ουρών Μηνυμάτων* (message queue – Εικόνα 8) και είναι συνήθως κομμάτι ενός μεγαλύτερου, προσανατολισμένου στα μηνύματα, ενδιάμεσου λογισμικού (message-oriented middleware). Τα περισσότερα συστήματα μηνυμάτων υποστηρίζουν και τα δύο συγγενικά μοντέλα στη διεπαφή προγραμματισμού της εφαρμογής τους (API) (π.χ. Υπηρεσία Μηνυμάτων της Java (JMS)).



Εικόνα 8: Γραφική απεικόνιση επικοινωνίας μοντέλου ουράς μηνυμάτων

2.2.2.2. Πρότυπο Model-View-Controller (MVC)

Οι εφαρμογές πολλών χρηστών δεν είναι τόσο απλές όσο αυτές της παρακολούθησης. Συνεπώς και το πρότυπο της αρχιτεκτονικής τους, το *Model-View-Controller*, είναι πιο σύνθετο από αυτό του Observer.

Το πρότυπο Model-View-Controller (Εικόνα 9) έχει τις εξής τρεις βασικές λειτουργικότητες:

- **Μοντέλο πληροφοριών** (Model):

  Το Model διαχειρίζεται τη συμπεριφορά (business logic) και τα δεδομένα της εφαρμογής, απαντά σε ερωτήματα του View για πληροφορίες σχετικά με την κατάστασή

του και ανταποκρίνεται στις οδηγίες του Controller για να αλλάξει κατάσταση. Στην περίπτωση των event-driven συστημάτων (δηλ. συστήματα που βασίζονται σε γεγονότα) το Model ειδοποιεί τους Παρατηρητές, στέλνοντας ειδοποιήσεις (notification) στα αντίστοιχα View κάθε φορά που αλλάζει η πληροφορία. Το Model φροντίζει για τη συνέχιση (persistence) της εφαρμογής.

Στην υλοποίηση του Model οι ειδοποιήσεις μπορεί να γίνουν με `jQuery.trigger()` και η συνέχιση με cookies ή AJAX.

- **Αναπαράσταση πληροφοριών** (View):

  Το View αποδίδει (render) το Model σε μια μορφή κατάλληλη για αλληλεπίδραση, συνήθως σε ένα στοιχείο της διεπαφής του χρήστη που παρουσιάζεται στην οθόνη του. Για ένα Model, ανάλογα με το σκοπό, μπορούμε να πάρουμε πολλαπλά View.

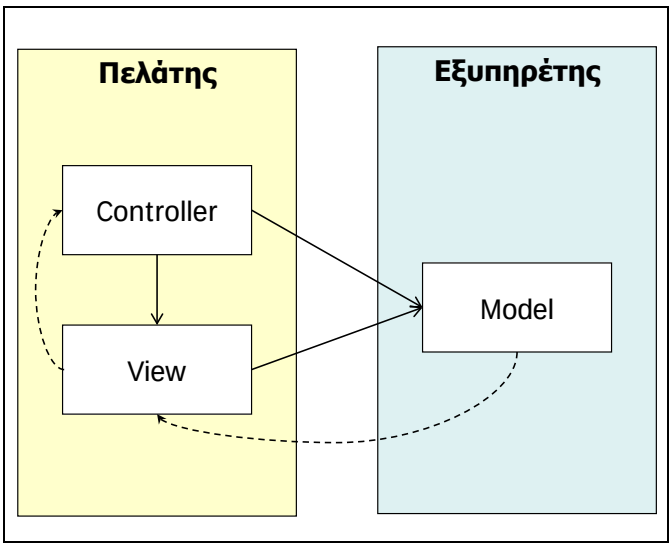
  Στις διαδικτυακές εφαρμογές το View είναι η (X)HTML που παράγεται από την εφαρμογή. Η υλοποίησή του μπορεί να γίνει, για παράδειγμα, εισάγοντας κόμβους HTML DOM ή αλλάζοντας τα CSS.

- **Διαχείριση πληροφοριών** (Controller):

  Το Controller δέχεται είσοδο από το χρήστη και στη συνέχεια δίνει οδηγίες στο Model και στο κατάλληλο View για να εκτελέσουν τις ενέργειες που απαιτούνται ώστε να δοθεί η απάντηση.

  Ο κώδικας του Controller κάνει την αρχικοποίηση του Model και συνδέει τα γεγονότα ανάμεσα στα στοιχεία HTML DOM του View και στο ίδιο και ανάμεσα στο Model και στο View με `jQuery.bind()`.

Ο στόχος του MVC είναι, μέσω της αποσύνδεσης του Model και του View, να μειωθεί η πολυπλοκότητα στον αρχιτεκτονικό σχεδιασμό και να αυξηθεί η *ευελιξία* και η *ευκολία συντήρησης* του κώδικα.



Εικόνα 9: Πρότυπο MVC (Η συνεχής γραμμή αντιπροσωπεύει μια άμεση σύνδεση ενώ η διακεκομμένη μια έμμεση (π.χ. μέσω ενός Παρατηρητή))

2.2.3. Φιλτράρισμα μηνυμάτων προτύπου Δημοσίευσης/Εγγραφής

Στο πρότυπο Publish/Subscribe οι συνδρομητές τυπικά λαμβάνουν μόνο ένα υποσύνολο των μηνυμάτων που δημοσιεύονται. Η διαδικασία επιλογής μηνυμάτων για λήψη και επεξεργασία καλείται *φιλτράρισμα* (filtering). Υπάρχουν δύο βασικές μορφές φιλτραρίσματος: βασισμένες-στο-θέμα και βασισμένες-στο-περιεχόμενο.

- Σε ένα σύστημα **βασισμένο-στο-θέμα**, τα μηνύματα δημοσιεύονται σε *θέματα* (topics) τα οποία λέγονται και *λογικά κανάλια*. Οι συνδρομητές σε ένα τέτοιο σύστημα λαμβάνουν όλα τα μηνύματα που δημοσιεύονται στα θέματα στα οποία εγγράφονται. Υπεύθυνος για τον καθορισμό των κατηγοριών μηνυμάτων, στις οποίες μπορούν να εγγραφούν οι συνδρομητές, είναι ο εκδότης.
- Σε ένα σύστημα **βασισμένο-στο-περιεχόμενο**, στο συνδρομητή παραδίδονται μόνο τα μηνύματα που οι ιδιότητες ή το περιεχόμενό τους ταιριάζει με τους περιορισμούς που έθεσε ο συνδρομητής. Αρμόδιος για την ταξινόμηση των μηνυμάτων είναι ο συνδρομητής.

Μερικά συστήματα υποστηρίζουν ένα **υβρίδιο** των δύο μορφών: οι εκδότες τοποθετούν τα μηνύματά τους σε κάποιο θέμα, ενώ οι συνδρομητές καταχωρούν τις βασισμένες-στο-περιεχόμενο εγγραφές (subscription) σε ένα ή περισσότερα θέματα.

2.3. Αποφάσεις σχεδιασμού δυναμικών διαδικτυακών εφαρμογών

Έχοντας κατά νου την εφαρμογή την οποία θέλουμε να αναπτύξουμε, υπάρχουν κάποιες σημαντικές αποφάσεις που πρέπει να πάρουμε πριν ξεκινήσουμε να γράφουμε κώδικα. Για τις διαδικτυακές εφαρμογές υπάρχει ένα πλήθος τεχνολογιών που μπορούμε να χρησιμοποιήσουμε. Κάθε τεχνολογία βασίζεται σε κάποια τεχνική επικοινωνίας πελάτη/εξυπηρέτη και μπορεί να καλύψει διαφορετικές απαιτήσεις. Πώς θα διαλέξουμε την «καλύτερη» για εμάς;

Κάποιες σημαντικές ερωτήσεις είναι οι εξής:

- *Ποιες τεχνολογίες γνωρίζω;*

  Δεδομένης της εφαρμογής που σκέφτομαι να αναπτύξω, πρέπει να δω ποια ή ποιες τεχνολογίες με καλύπτουν. Σχεδιάζοντας την εφαρμογή σχεδιάζουμε αρχικά αυτό που θα λαμβάνει ο χρήστης και στη συνέχεια βλέπουμε πώς θα το υποστηρίξει ο εξυπηρέτης. Για κάθε τεχνολογία λοιπόν πρέπει να ξέρω ποιος είναι ο σκοπός της, καθώς και πώς υλοποιείται και από τος δύο πλευρές.

- *Ποιες θεωρούνται, από αυτούς που τις προτείνουν, κατάλληλες για το δικό μου τύπο εφαρμογής;*

- *Ποιες τεχνολογίες, από αυτές που ταιριάζουν στην εφαρμογή μου, φαίνεται να επικρατούν στο χώρο;*

  Αν υπάρχουν εφαρμογές παρόμοιας λειτουργικότητας με αυτή που σκοπεύω να αναπτύξω είναι πιο συνετό να μάθω τι τεχνολογίες χρησιμοποιούν. Αν τις θεωρώ «απαρχαιωμένες», τότε μπορώ να διαλέξω μια νέα τεχνολογία με την οποία ξέρω βέβαια ότι μπορώ να αναπτύξω την εφαρμογή μου. Αλλιώς θα διαλέξω την τεχνολογία που χρησιμοποιούν οι πιο διαδεδομένες ή ανερχόμενες εφαρμογές. Προφανώς έχουν σημαντικούς λόγους που τις επέλεξαν. Ακόμη και αν αυτό δεν ισχύει, τουλάχιστον θα συνεχίζεται η ανάπτυξη/βελτίωση των επιλεγμένων τεχνολογιών και δε θα «εξαφανιστούν» (μάλλον) πολύ σύντομα.

- *Πόσο πολύπλοκη είναι η εφαρμογή μου;*

- *Ποιες οι ανάγκες χρονικής συνέπειας;*

Η χρονική συνέπεια, όπως θα φανεί στο επόμενο κεφάλαιο, είναι αντιστρόφως ανάλογη της επιβάρυνσης του δικτύου και των συμμετεχόντων στις δικτυακές εφαρμογές. Για αυτό το λόγο είναι σκόπιμο να επιλέγεται το μέγιστο δυνατό χρονικό κενό ανάμεσα σε δύο διαδοχικές ενημερώσεις.

- *Πόσους ταυτόχρονους χρήστες θα κληθώ να υποστηρίξω;*

  Σε εφαρμογές όπου αναμένεται ή θέλω να επιτύχω πολλούς χρήστες ταυτόχρονα θα πρέπει να ξέρω ποια είναι τα όρια της κάθε τεχνολογίας, τι θα συμβεί σε περίπτωση που αυτά θα ξεπεραστούν και τι μπορώ να κάνω για να βγω από το αδιέξοδο. Το πλήθος των χρηστών που μπορεί να υποστηριχθεί είναι άμεση συνέπεια του εξυπηρέτη. Ανάλογα με την τεχνολογία κάποιοι συμπεριφέρονται καλύτερα. Αν δεν έχω μόνο ένα εξυπηρέτη, θα πρέπει να υπάρχει κάποιο πρωτόκολλο το οποίο θα υλοποιείται από την τεχνολογία και το οποίο θα επιτρέπει εύκολο προγραμματισμό της αρχιτεκτονικής και της λειτουργικότητας των εξυπηρετών.

- *Τι είδους απαντήσεις θα χρησιμοποιηθούν; [15]*

  Οι απαντήσεις έχουν σημαντική επίδραση στην απόδοση της εφαρμογής εξαιτίας της συχνότητάς τους, οπότε ο σχεδιασμός, όταν υπάρχει επιλογή, θα πρέπει να κλίνει σε απαντήσεις όσο το δυνατόν χαμηλότερου κόστους.

  Οι **απαντήσεις απλού κειμένου**, οι οποίες χρησιμοποιούν μια προσαρμοσμένη μορφή κωδικοποίησης για τα σύνθετα δεδομένα, μπορούν να χρησιμοποιηθούν για τη μείωση του επιπρόσθετου κόστους (overhead) των ετικετών XML. Ωστόσο, το κέρδος στην απόδοση είναι μάλλον μικρό, ειδικά όταν ο εξυπηρέτης συμπιέζει τα δεδομένα χρησιμοποιώντας το πρωτόκολλο gzip.

  Οι απαντήσεις που είναι πιο εξειδικευμένες (**application-specific**) μπορούν να κοπούν και να ραφτούν, έτσι ώστε να δώσουν στον φυλλομετρητή ακριβώς αυτό που χρειάζεται σε κάθε ανανέωση και τίποτα περισσότερο. Βελτιώνουν αισθητά την απόδοση, αφού μπορούν να δώσουν μόνο τις πρόσφατες αλλαγές αντί ολόκληρης της τρέχουσας κατάστασης.

  Οι **user-specific** απαντήσεις φτάνουν την απόδοση στο μέγιστο. Εντοπίζοντας τι γνωρίζει ο κάθε φυλλομετρητής, ο εξυπηρέτης μπορεί να περικόψει τις απαντήσεις ακόμη περισσότερο. Για να επιτευχθεί αυτό χρειάζεται να διατηρείται κάπου η

κατάσταση της αλληλεπίδρασης/συνομιλίας (stateful interactions). Αυτό, δυστυχώς, κοστίζει στον εξυπηρέτη, είναι πιο δύσκολο στην ανάπτυξη και βασίζεται σε cookies. Ένας καλός συμβιβασμός είναι κάθε αίτηση να περιλαμβάνει κάποιες επιπρόσθετες πληροφορίες σχετικά με την τρέχουσα κατάσταση του φυλλομετρητή, ώστε ο εξυπηρέτης να μπορεί να καθορίσει αν και τι πρέπει να στείλει.

- *Έχω ήδη κάποιον δικτυακό εξυπηρέτη όπου θα ανεβάσω την εφαρμογή;*

  Αν έχω ήδη κάποιον εξυπηρέτη, τότε θα πρέπει να μπορώ να ανεβάσω εκεί την εφαρμογή μου. Οπότε η τεχνολογία ανάπτυξης θα πρέπει να μου δίνει μια τελική εφαρμογή που θα υποστηρίζεται από τον εξυπηρέτη. Κάποιες τεχνολογίες, αν και μπορούν να δουλέψουν και σε έναν απλό web εξυπηρέτη, είναι καλύτερο να δουλεύουν σε εξυπηρέτες ειδικά σχεδιασμένους για να τις υποστηρίζουν, όπως θα δούμε και στη συνέχεια. Η *κλιμακωσιμότητα* (scalability) είναι μια παράμετρος πολύ σημαντική για τις διαδικτυακές εφαρμογές και σε μεγάλο βαθμό εξαρτάται από την επιλογή του εξυπηρέτη.

- *Σε τι ποιότητα δικτύου θα εκτελεστεί η εφαρμογή;*

  Μια εφαρμογή η οποία αφορά στην παρακολούθηση ενός εργοστασιακού συστήματος, για παράδειγμα, είναι αναμενόμενο ότι θα τρέχει σε κάποιον/ους σταθερούς υπολογιστές σε ένα σταθερό δίκτυο. Σε άλλες εφαρμογές όμως, όπου οι χρήστες μπορεί να χρησιμοποιούν ακόμη και smartphone, η επικοινωνία με το δίκτυο συχνά διακόπτεται (πέφτει το socket). Σε αυτές τις περιπτώσεις θα πρέπει να υπάρχει από την τεχνολογία κάποιος τρόπος *ανάκαμψης* της εφαρμογής. Ακόμη, αν αυτό είναι σημαντικό, θα πρέπει ο χρήστης μετά την επανασύνδεση να λαμβάνει τις ενημερώσεις που έχασε. Επιπρόσθετα σε δίκτυα όπου υπάρχει ογκοχρέωση των χρηστών θα πρέπει το περιεχόμενο που μεταφέρεται ανάμεσα στον πελάτη και στον εξυπηρέτη να είναι το ελάχιστο δυνατό.

- *Σε ποιους φυλλομετρητές θα είναι διαθέσιμη η εφαρμογή;*

  Κάποιες τεχνολογίες δεν υποστηρίζονται από όλους τους φυλλομετρητές. Αν η εφαρμογή αναφέρεται όμως σε πολλούς και άγνωστους εκ των προτέρων χρήστες θα πρέπει να μην αποκλείει τουλάχιστον τους πιο ευρέως χρησιμοποιούμενους φυλλομετρητές.

- *Σε τι συσκευές θα εκτελείται η εφαρμογή;*

  Ανάλογα με τη συσκευή αλλάζουν οι υπολογιστικές δυνάμεις του πελάτη, η ποιότητα του δικτύου που θα χρησιμοποιήσει, ο φυλλομετρητής που μπορεί να είναι εγκαταστημένος και τα αντίστοιχα plugin καθώς και το rendering της διεπαφής.

  Η τάση σήμερα δείχνει πως σε λίγα χρόνια θα επικρατήσει το *mobile web*, οπότε οποιαδήποτε διαδικτυακή εφαρμογή πολλών χρηστών θέλει να «επιζήσει», μάλλον οφείλει να χρησιμοποιήσει τεχνολογίες που υποστηρίζουν mobile web.

3. ΑΙΤΗΣΕΙΣ ΑΠΟ ΤΟΝ ΠΕΛΑΤΗ ΠΡΟΣ ΤΟΝ ΕΞΥΠΗΡΕΤΗ

Ξεκινώντας το Διαδίκτυο, όπως αναφέρθηκε και στην εισαγωγή, είχαμε αιτήσεις οι οποίες ξεκινούσαν από τον πελάτη και απαντήσεις που έστελνε με τη σειρά του εξυπηρέτης. Πιο συγκεκριμένα, στο μοντέλο πελάτη-εξυπηρέτη ο εξυπηρέτης παρέχει κάποια λειτουργία ή υπηρεσία σε έναν ή περισσότερους πελάτες και ο πελάτης το μόνο που κάνει είναι να ξεκινά τις αιτήσεις για αυτές τις υπηρεσίες (pull-based). Οι αιτήσεις αυτές μπορούν, όπως θα δούμε στη συνέχεια, να είναι είτε σύγχρονες (χρήση HTTP) είτε ασύγχρονες (χρήση AJAX).

3.1. Σύγχρονες αιτήσεις – HTTP

Το *Πρωτόκολλο Μεταφοράς Υπερκειμένου* (Hypertext Transfer Protocol, HTTP) [3] είναι η κύρια μέθοδος που χρησιμοποιούν τα πρωτόκολλα του Παγκοσμίου Ιστού για να μεταφέρουν δεδομένα ανάμεσα σε ένα εξυπηρέτη (server) και ένα πελάτη (client). Είναι ένα πρωτόκολλο επιπέδου εφαρμογής και προϋποθέτει την ύπαρξη του αξιόπιστου TCP πρωτόκολλου [18] για τη μεταφορά των δεδομένων. Η ανάπτυξη του HTTP έγινε υπό την εποπτεία του World Wide Web Consortium [19] και του Internet Engineering Task Force (IETF) [20].

Το HTTP χρησιμοποιείται για τη διεκπεραίωση αιτήσεων/απαντήσεων μεταξύ ενός υπολογιστή πελάτη (client) και ενός εξυπηρέτη (server). Πελάτης ονομάζεται ο τελικός χρήστης (που αλληλεπιδρά μέσω του φυλλομετρητή του) και ο εξυπηρέτης είναι η εκάστοτε ιστοσελίδα. Το πρωτόκολλο υλοποιεί ένα **αυστηρό μοντέλο αίτησης/απάντησης**, που σημαίνει ότι **ο πελάτης στέλνει μια αίτηση HTTP και περιμένει μέχρι να λάβει μια HTTP απάντηση**. Δεν υποστηρίζεται η περίπτωση να ξεκινά η αλληλεπίδραση από τον εξυπηρέτη προς τον πελάτη.

Για παράδειγμα ο πελάτης ζητά μια συγκεκριμένη σελίδα κάνοντας μια HTTP GET αίτηση. Η αίτηση περιέχει μόνο κεφαλίδα.

```
GET /html/rfc2616.html HTTP/1.1
Host: tools.ietf.org
User-Agent: xLightweb/2.11.3
```

Ο εξυπηρέτης επιστρέφει την οντότητα της πληροφορίας που ζητήθηκε με μια HTTP απάντηση. Η απάντηση αποτελείται από την κεφαλίδα και το σώμα χωρισμένα με μια κενή γραμμή.

```
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Debian)
Date: Tue, 09 Feb 2010 05:00:01 GMT
Content-Length: 510825
Content-Type: text/html; charset=UTF-8
Last-Modified: Fri, 25 Dec 2009 05:49:54 GMT
ETag: "302e72-7cb69-47b871ff82480"
Accept-Ranges: bytes

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en" xml:lang="en">
<head>
    <meta      http-equiv="Content-Type"      content="text/html;
charset=us-ascii" />
    <meta name="robots" content="index,follow" />
    <meta name="creator" content="rfcmarkup version 1.53" />
    <link rel="icon" href="/images/rfc.png" type="image/png" />
    <link     rel="shortcut     icon"     href="/images/rfc.png"
type="image/png" />
    <title>RFC    2616    Hypertext    Transfer    Protocol    --
HTTP/1.1</title>
[...]
</small></small></span>
</body></html>
```

Αφού το HTTP δεν έχει σχεδιαστεί για αμφίδρομη ανταλλαγή μηνυμάτων μεταξύ των χρηστών, θεωρείται ακατάλληλο για αλληλεπίδραση πραγματικού χρόνου. Μια επόμενη προσέγγιση αλληλεπίδρασης η οποία κάνει τις διαδικτυακές εφαρμογές πιο διαδραστικές και μας φέρνει πιο κοντά στο στόχο της αλληλεπίδρασης πραγματικού χρόνου, είναι η AJAX.

3.2. Ασύγχρονες αιτήσεις – AJAX

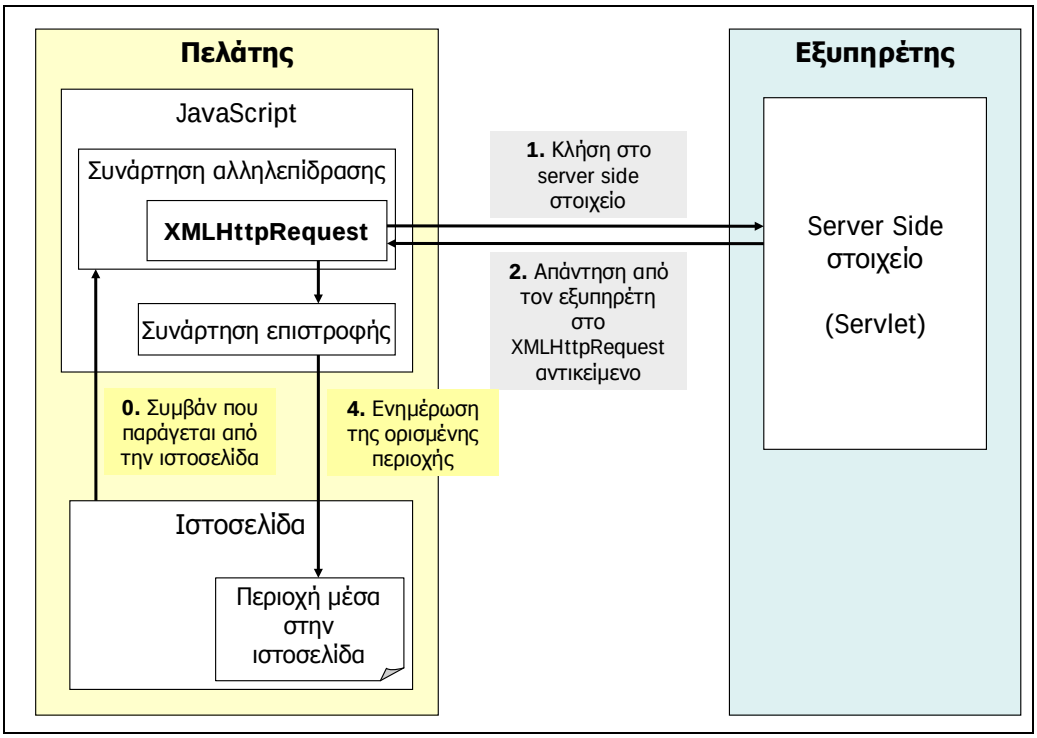
Το “web browsing” είναι γενικά μια απλή διαδικασία: οι χρήστες κάνουν κλικ στους συνδέσμους μιας σελίδας και οδηγούνται μέσω αυτών σε άλλες. Η λειτουργία αυτή, αν και βασική, δεν δίνει από μόνη της τη δυνατότητα στους χρήστες να ζητήσουν ή να ενημερώσουν στοιχεία χωρίς να αλλάξουν σελίδα. Μια επόμενη προσέγγιση της αλληλεπίδρασης στο Διαδίκτυο ήταν η *AJAX* (**A**synchronous **J**avaScript **A**nd **X**ML) [4], που ουσιαστικά προσφέρει ένα **στρώμα ασύγχρονης επικοινωνίας πάνω στην ήδη υπάρχουσα τεχνολογία**. Ανάμεσα στον πελάτη και τον εξυπηρέτη παρεμβαίνει η *Ajax engine* η οποία δέχεται ασύγχρονες αιτήσεις. Μέσω ασύγχρονων κλήσεων, ο browser του χρήστη μπορεί να επικοινωνήσει με τον εξυπηρέτη στο παρασκήνιο και να λάβει μια απάντηση χωρίς να χρειαστεί να γίνει ολόκληρη ανανέωση σελίδας. Εδώ γίνεται η μετάδοση όσο το δυνατόν μικρότερης ποσότητας πληροφοριών από και προς τον εξυπηρέτη, ώστε να φαίνεται στο χρήστη πιο *αποκριτικό* (responsive) το σύστημα.

Η τυπική μέθοδος αλληλεπίδρασης του χρήστη με δικτυακές εφαρμογές απαιτεί από το χρήστη την εισαγωγή κάποιων πληροφοριών (π.χ. συμπλήρωση μιας φόρμας), την κατάθεσή τους στον εξυπηρέτη και την αναμονή μιας ανανέωσης σελίδας ή ανακατεύθυνσης για να γίνει επιστροφή της απάντησης από τον εξυπηρέτη. Αυτό μπορεί κάποιες φορές να εκνευρίσει το χρήστη και είναι επίσης αρκετά διαφορετικό από το «desktop» στυλ του UI με το οποίο μπορεί να είναι πιο οικείος. Η τεχνολογία AJAX μας επιτρέπει να χτίζουμε δικτυακές εφαρμογές με *διεπαφές χρήστη* (user interfaces) που θυμίζουν περισσότερο εφαρμογές desktop, παρέχοντας μια καλύτερη εμπειρία για το χρήστη. Οι εφαρμογές αυτές λέγονται και **Rich Internet Applications** (RIA).

Στην τυπική μέθοδο η αίτηση του χρήστη στον εξυπηρέτη γίνεται μέσω μιας κανονικής HTTP POST ή GET αίτησης, όπως θα γινόταν με την υποβολή μιας φόρμας ή την επιλογή κάποιου υπερσυνδέσμου, ενώ ένα *Ajax script* κάνει την αίτηση στον εξυπηρέτη χρησιμοποιώντας το JavaScript *XMLHTTPRequest αντικείμενο* (XHR – Εικόνα 10).

Αυτό το αντικείμενο συμπεριφέρεται σχεδόν όπως ένα κανονικό JavaScript αντικείμενο. Με ένα JavaScript αντικείμενο εικόνας μπορούμε να αλλάξουμε δυναμικά το URL της πηγής της εικόνας χωρίς να κάνουμε ανανέωση της σελίδας. Το XMLHTTPRequest ανακτά πληροφορίες από τον εξυπηρέτη με έναν παρόμοιο αόρατο τρόπο. Πρόκειται για ένα

κανονικό HTTP Request με επιπλέον δυνατότητα αποστολής και λήψης XML κώδικα ή JSON.



Εικόνα 10: AJAX αλληλεπίδραση με το αντικείμενο XMLHttpRequest

Τα βήματα για την ανάπτυξη μιας εφαρμογής Ajax είναι λίγα και σχετικά εύκολα. Η περιγραφή που ακολουθεί περιγράφει τα κυριότερα σημεία.

- Πρώτον, η δημιουργία του XMLHTTPRequest αντικειμένου διαφέρει ανάλογα με το αν χρησιμοποιείται Internet Explorer με ενεργοποιημένο το ActiveX ή κάποιος άλλος φυλλομετρητής συμβατός με τα πρότυπα όπως ο Mozilla Firefox.

  Στον IE, η αίτηση γίνεται κάπως έτσι:

```
http = new ActiveXObject("Microsoft.XMLHTTP");
```

  ενώ για στον Firefox το αντικείμενο μπορεί να αρχικοποιηθεί απ' ευθείας:

```
http = new XMLHttpRequest();
```

- Δεύτερον, πρέπει να γραφεί ένα πρόγραμμα χειρισμού συμβάντων (event handler) το οποίο θα καλείται μέσω κάποιου συμβάντος στη σελίδα του χρήστη και θα χειρίζεται την αποστολή του αιτήματος για δεδομένα στον εξυπηρέτη.

Το πρόγραμμα χειρισμού συμβάντων χρησιμοποιεί διάφορες μεθόδους του XMLHTTPRequest αντικειμένου για να υποβάλει το αίτημα στον εξυπηρέτη, να ελέγξει πότε ο εξυπηρέτης λέει ότι έχει ολοκληρωθεί η αίτηση και να ασχοληθεί με τις πληροφορίες που επιστρέφονται από το εξυπηρέτη.

Το αίτημα στον εξυπηρέτη γίνεται με τη χρήση μιας GET μεθόδου στο κατάλληλο server-side script. Η συνάρτηση updateData είναι ένα παράδειγμα χειριστή συμβάντων (callback function):

```
function updateData(param) {
  var myurl = [εδώ εισάγεται το URL του server script];
  http.open("GET", myurl + "?id=" + escape(param), true);
  http.onreadystatechange = useHttpResponse;
  http.send(null);
}
```

Η συνάρτηση ακούει την ιδιότητα `onreadystatechange` του αντικειμένου XMLHTTPRequest και, κάθε φορά που αλλάζει η παράμετρος, ζητά περαιτέρω `useHttpResponse` λειτουργία.

Το server-side script το οποίο καλείται - κατ' ουσίαν μπορεί να είναι οποιαδήποτε ρουτίνα του εξυπηρέτη η οποία θα δημιουργήσει την απαιτούμενη έξοδο όταν θα κληθεί με το σχετικό URL που επισυνάπτεται και τις παραμέτρους, όπως και σε κάθε άλλη HTTP GET αίτηση. Προς χάριν του παραδείγματος περνάμε μια μεταβλητή που ονομάζεται `id` με τιμή `param` ως όρισμα στη συνάρτηση `updateData`.

- Τρίτον, πρέπει να γραφεί μια συνάρτηση useHttpResponse, η οποία θα καθορίζει πότε ο εξυπηρέτης έχει ολοκληρώσει το αίτημα και θα κάνει κάτι χρήσιμο με τα δεδομένα που έχει επιστρέψει:

```
function useHttpResponse() {
  if (http.readyState == 4) {
    var textout = http.responseText;
    document.write.textout;
  } }
```

Η συνάρτηση ελέγχει για την τιμή 4 στην ιδιότητα `readyState` - υπάρχουν διάφορες αριθμημένες καταστάσεις οι οποίες περιγράφουν την εξέλιξη μιας τέτοιας αίτησης, αλλά η 4 σημαίνει πως η αίτηση ολοκληρώθηκε και μπορούν να χρησιμοποιηθούν τα δεδομένα τα οποία επεστράφησαν.

Στην περίπτωση αυτή, οι πληροφορίες ελήφθησαν ως απλό κείμενο μέσω της ιδιότητας `responseText` του XMLHTTPRequest αντικειμένου. Οι πληροφορίες μπορούν, ωστόσο, να επιστρέφονται ως XML ή ως ιδιότητες ενός προκαθορισμένου αντικειμένου javascript.

Το σύνολο των τεχνολογιών που χρησιμοποιεί η AJAX, και ιδιαίτερα η JavaScript και το αντικείμενο XMLHttpRequest οι οποίες παρέχουν μια μέθοδο ανταλλαγής δεδομένων με ασύγχρονο τρόπο ανάμεσα στο φυλλομετρητή και στον εξυπηρέτη αποφεύγοντας πλήρεις ανανεώσεις σελίδας, επιτρέπουν στο φυλλομετρητή να κάνει αιτήσεις στο παρασκήνιο. Ο Πίνακας 1 δείχνει ποιο φυλλομετρητές υποστηρίζουν το XMLHttpRequest.

Τι γίνεται όμως στην περίπτωση που έχει να πει κάτι νέο ο εξυπηρέτης; Πώς θα το μάθει ο πελάτης; Σε αυτό το ερώτημα απαντά το επόμενο κεφάλαιο.

Πίνακας 1: Υποστήριξη του XMLHttpRequest από τους φυλλομετρητές [21]

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
πριν 11 εκδόσεις			4.0						
πριν 10 εκδόσεις			5.0						
πριν 9 εκδόσεις			6.0						
πριν 8 εκδόσεις		2.0	7.0						
πριν 7 εκδόσεις		3.0	8.0		9.0				
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6				
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1				
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5				
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0	
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2
τωρινή έκδοση	9.0	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3, 3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0
μελλοντικά	10.0	10.0	17.0	6.0	12.0				

4. ΕΙΔΟΠΟΙΗΣΕΙΣ ΑΠΟ ΤΟΝ ΕΞΥΠΕΡΕΤΗ ΠΡΟΣ ΤΟΝ ΠΕΛΑΤΗ

Στο προηγούμενο κεφάλαιο είδαμε τους δύο βασικότερους τρόπους ρητής αίτησης από τον πελάτη στον εξυπηρέτη. Στο τρέχον κεφάλαιο θα ασχοληθούμε με το αντίθετο, δηλαδή με την προώθηση δεδομένων-ειδοποιήσεων από τον εξυπηρέτη στον πελάτη. Αυτή η προώθηση όπως θα φανεί από τις διάφορες λύσεις που αναπτύχθηκαν δεν ήταν εξ αρχής αυτόματη, αλλά σταδιακά έφτασε να γίνει.

Υποθέτοντας πως υπάρχει κάποια εφαρμογή σε έναν εξυπηρέτη από την οποία πρέπει να ειδοποιηθούν οι πελάτες-φυλλομετρητές για νέα δεδομένα, οι λύσεις που υπάρχουν μπορούν να κατηγοριοποιηθούν σε “polling”, “server-side callbacks”, “messaging” και “streaming”.

4.1. Short Polling (Periodic Refresh)

Τη δεκαετία του ’90, οι περισσότερες ιστοσελίδες ήταν γραμμένες σε απλή HTML. Η τεχνική *short polling* [22] αποτέλεσε την πρώτη προσπάθεια παράδοσης πληροφορίας πραγματικού χρόνου στο Διαδίκτυο. Πρόκειται για μια τεχνική στην οποία ο πελάτης στέλνει HTTP αιτήσεις σε τακτά χρονικά διαστήματα και λαμβάνει αμέσως μια απάντηση. Αν και μπορεί να έχει γίνει μερική αλλαγή της σελίδας, ή και μηδενική, εφόσον δεν μπορούμε να προβλέψουμε πότε θα είναι έτοιμη μια ενημέρωση, πρέπει να ξανασταλεί όλο το περιεχόμενό της. Αυτό σημαίνει άχρηστες αιτήσεις, μεγαλύτερη **επιβάρυνση** (load) του εξυπηρέτη και **υπέρμετρη χρήση του εύρους ζώνης** (bandwidth). Οπότε, κατά το σχεδιασμό θα πρέπει ο στόχος να είναι να αυξηθεί όσο μπορεί η περίοδος και να μειωθεί το περιεχόμενο ανά ανανέωση.

Μια λύση polling αποτελεί το HTML Refresh.

4.1.1. HTML Refresh

Η απλούστερη λύση είναι μία «χρονομετρημένη ανανέωση σελίδας». Χρησιμοποιώντας την ετικέτα meta refresh στην κεφαλίδα του εγγράφου HTML, η σελίδα επαναφορτώνεται αυτόματα κάθε `N` δευτερόλεπτα.

```
<meta http-equiv="refresh" content="N">
```

Εάν εν τω μεταξύ έχει αλλάξει κάτι στο εξυπηρέτη, τότε λαμβάνεται το νέο περιεχόμενο ειδάλλως τα ίδια δεδομένα με πριν. Το σημαντικό εδώ είναι η επιλογή του κατάλληλου διαστήματος ανανέωσης. Αν οι χρόνοι άφιξης είναι ανεξάρτητοι και ομοιόμορφα κατανεμημένοι στο χρονικό διάστημα (0, t) και η περίοδος είναι T, τότε θα έχουμε μια μέση καθυστέρηση T/2. Επιπρόσθετα, αν n πελάτες συνδεθούν στον εξυπηρέτη, τότε ο εξυπηρέτης θα λάβει n*t/T αιτήσεις.

Αν και το W3C αποθαρρύνει τη χρήση του META Refresh καθώς οι μη αναμενόμενες ανανεώσεις σελίδας μπορούν να αποσυντονίσουν το χρήστη, ωστόσο χρησιμοποιείται ως σήμερα στις περιπτώσεις που θέλουμε να δώσουμε ενημερώσεις δυναμικών **σελίδων χωρίς να εξαρτόμαστε από τη Javascript**.

Το HTML Refresh είναι πανεύκολο στη χρήση και μπορεί να χρησιμοποιηθεί σε διαδικτυακές εφαρμογές όπως ο τηλετυπώτης μετοχών και η πρόγνωση του καιρού.

Ο Πίνακας 2 δείχνει πως όλοι οι φυλλομετρητές υποστηρίζουν το HTML Refresh.

Πίνακας 2: Υποστήριξη του HTML Refresh από τους φυλλομετρητές

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
πριν 11 εκδόσεις			4.0							
πριν 10 εκδόσεις			5.0							
πριν 9 εκδόσεις			6.0							
πριν 8 εκδόσεις		2.0	7.0							
πριν 7 εκδόσεις		3.0	8.0		9.0					
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6					
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1					
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5					
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0		
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1	
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2	
τωρινή έκδοση	9.0	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3,	3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0	
μελλοντικά	10.0	10.0	17.0	6.0	12.0					

4.1.2. Reverse AJAX

Το HTML Refresh σε κάθε περίοδο θα επιστρέψει ολόκληρη τη σελίδα, ανανεωμένη ή μη. Για να γίνει πιο αποκριτική η εφαρμογή και πάλι μπορεί να χρησιμοποιηθεί η τεχνολογία Ajax. Σε κάθε περίοδο θα έρχεται μικρότερη ποσότητα πληροφοριών από τον εξυπηρέτη, και αυτό θα **μειώσει την επιβάρυνση στο δίκτυο αλλά και στον εξυπηρέτη**.

Στο AJAX Polling [15], ο φυλλομετρητής περιοδικά (ανά `REFRESH_PERIOD_MILLIS` milliseconds) εκπέμπει μια κλήση XMLHttpRequest για να λάβει νέες πληροφορίες από την εξυπηρέτη.

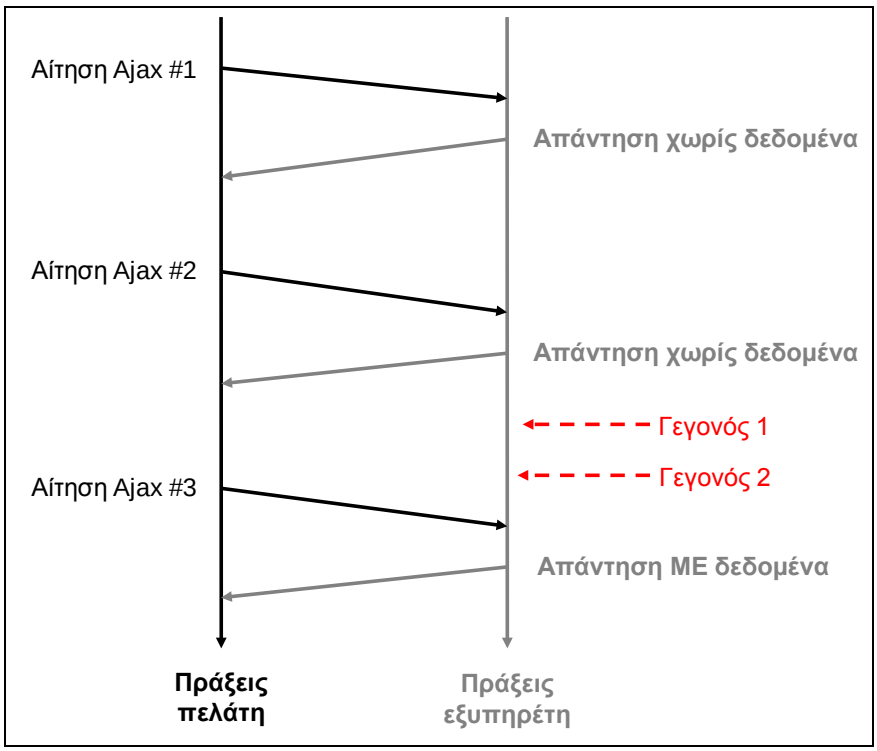
```
setInterval(callServer, REFRESH_PERIOD_MILLIS);
```

Για να μειωθεί η επιβάρυνση του δικτύου και του εξυπηρέτη θα πρέπει η περίοδος να γίνει όσο μεγαλύτερη γίνεται. Μια συνήθης τακτική είναι το timeout, όπου οι ανανεώσεις παύουν όταν το σύστημα ανιχνεύει πως ο χρήστης δεν είναι πλέον ενεργός. Ακόμη, το δίκτυο θα πρέπει να είναι ικανό να μεταδώσει όσες ενημερώσεις σταλούν. Οπότε το script του browser μπορεί να κάνει παρακολούθηση του δικτύου και να προσαρμόζει δυναμικά την περίοδο ώστε να μπορεί το δίκτυο να τα βγάζει πέρα με όλες τις εισερχόμενες ενημερώσεις.

Υπάρχουν τρεις τρόποι υλοποίησης για AJAX Polling:

4.1.2.1. HTTP Polling

Στο HTTP Polling ο πελάτης στέλνει μία HTTP Ajax αίτηση κάθε N χρονικές μονάδες (polling interval). Το χρονοδιάγραμμα στην Εικόνα 11 δείχνει πως για ορισμένες αιτήσεις δεν επιστρέφεται καμία πληροφορία. Ακόμη, ο πελάτης πρέπει να περιμένει το polling για να λάβει τα δύο γεγονότα που ελήφθησαν από τον εξυπηρέτη.



Εικόνα 11: Χρονοδιάγραμμα Reverse Ajax με HTTP polling

4.1.2.2. JSONP Polling

Η διαφορά του JSONP polling από το HTTP polling είναι πως με το JSONP μπορείς να κάνεις αιτήσεις σε διαφορετικά domain (**cross-domain**). Μια JSONP αίτηση αναγνωρίζεται από την παράμετρο στην callback και από το περιεχόμενο που επιστρέφεται (κώδικας JavaScript).

```
setInterval(function() {
    $.getJSON('events', function(events) {
        console.log(events);
    });
}, 2000);
```

Ο Πίνακας 3 δείχνει ποιοι φυλλομετρητές υποστηρίζουν Json Parsing.

Πίνακας 3: Υποστήριξη του Json Parsing από τους φυλλομετρητές

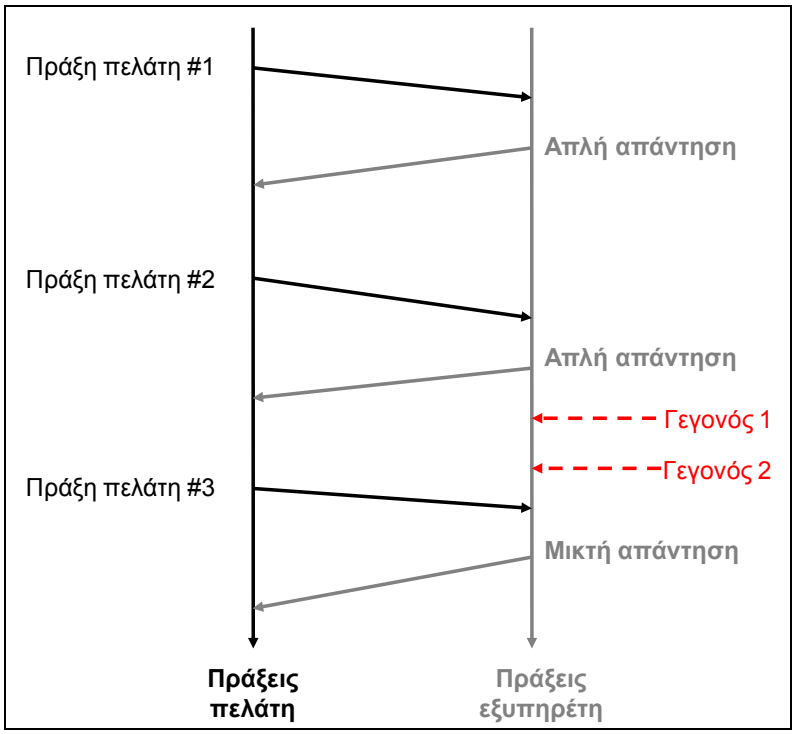
	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
πριν 11 εκδόσεις			4.0							
πριν 10 εκδόσεις			5.0							
πριν 9 εκδόσεις			6.0							
πριν 8 εκδόσεις		2.0	7.0							
πριν 7 εκδόσεις		3.0	8.0		9.0					
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6					
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1					
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5					
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0		
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1	
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2	
τωρινή έκδοση	9.	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3,	3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0	
μελλοντικά	10.0	10.0	17.0	6.0	12.0					

4.1.2.3. Piggyback Polling

Το Piggyback polling κάνει επίσης χρήση AJAX, αλλά **δεν έχει κάποια σταθερή περίοδο**. Οι αιτήσεις στέλνονται όταν ο πελάτης έχει να στείλει μια αίτηση στον εξυπηρέτη και η απάντηση αποτελείται από δύο μέρη 1) την απάντηση για αυτό που ζητήθηκε και 2) τα γεγονότα του εξυπηρέτη που πιθανώς συνέβησαν.

```
$('#submit').click(function() {
    $.post('ajax', function(data) {
        var valid = data.formValid;
        // process validation results
        // then process the other part of the response (events)
        processEvents(data.events);
    });
});
```

Η Εικόνα 12 δείχνει ένα παράδειγμα.



Εικόνα 12: Χρονοδιάγραμμα Reverse Ajax με piggyback polling

4.1.3. Αποφάσεις σχεδιασμού στο Short Polling

Επιλέγοντας την τεχνική short polling ο σχεδιαστής θα πρέπει να κάνει την αρχικά την επιλογή της λύσης που τον καλύπτει. Ανεξάρτητα όμως από αυτό, θα πρέπει να απαντήσει και στο εξής ερώτημα:

- *Πόση θα είναι η περίοδος ανανέωσης;*

Η περίοδος ποικίλλει και εξαρτάται από το πλαίσιο χρήσης. Σε γενικές γραμμές, το επίπεδο δραστηριότητας μπορεί να χωριστεί σε τρεις κατηγορίες:

- **Αλληλεπίδραση πραγματικού-χρόνου (χιλιοστά του δευτερολέπτου)**

Ο χρήστης αλληλεπιδρά ενεργά με το σύστημα και η είσοδός του εξαρτάται από την έξοδο του εξυπηρέτη –για παράδειγμα ένας χρήστης chat πρέπει να βλέπει τι λένε οι άλλοι, ένας έμπορος πρέπει να βλέπει τις τρέχουσες τιμές και ένας παίκτης πρέπει να ξέρει την τρέχουσα κατάσταση του παιχνιδιού. Σε αυτή την περίπτωση, το χρονικό διάστημα μπορεί να είναι από 1 millisecond σε ένα τοπικό δίκτυο ή 20-100 millisecond σε ένα παγκόσμιο δίκτυο.

- **Ενεργής παρακολούθηση (δευτερόλεπτα)**

Ο χρήστης βασίζεται στον εξυπηρέτη για εργασίες εκτός συστήματος –για παράδειγμα ο υπεύθυνος ασφαλείας παρακολουθεί κάποιους ανιχνευτές για ύποπτη δραστηριότητα ή ο διευθυντής παρακολουθεί τη διακύμανση των δεικτών των πωλήσεων σε ένα χρονικό παράθυρο. Στις περιπτώσεις που ο χρονισμός είναι σημαντικός οι απαντήσεις θα πρέπει να έρχονται σε λιγότερο από ένα δευτερόλεπτο. Σε άλλες περιπτώσεις, η ανατροφοδότηση είναι επαρκής όταν έρχεται ανά κάποια δευτερόλεπτα.

- **Περιστασιακή παρακολούθηση (λεπτά)**

Κάποιες εφαρμογές είναι έτσι σχεδιασμένες ώστε ο χρήστης να παρακολουθεί το παράθυρό τους σε όλη τη διάρκεια της ημέρας. Η πληροφορία δεν αλλάζει συχνά και δεν υπάρχει πρόβλημα αν ο χρήστης τη δει λίγο αργότερα. Οι βασικοί υποψήφιοι εδώ είναι τα portal και τα RSS aggregator. Η περίοδος ανανέωσης τέτοιου περιεχομένου μπορεί να είναι δέκα δευτερόλεπτα ή και παραπάνω.

4.2. Applet

Η **ασύγχρονη φόρτωση των περιεχομένων** μιας σελίδας έγινε δυνατή όταν παρουσιάστηκαν τα Java applet στην πρώτη έκδοση της γλώσσας Java το 1995. Τα applet (εφαρμογίδια) είναι μικρά προγράμματα Java τα οποία μπορούν να συμπεριληφθούν σε μια HTML σελίδα και επιτρέπουν τη φόρτωση δεδομένων ασύγχρονα, αφού φορτωθεί η ιστοσελίδα.

Τα applet τρέχουν σε φυλλομετρητές οι οποίοι υποστηρίζουν Java, ανεξάρτητα από το λειτουργικό σύστημα του υπολογιστή. Ο κώδικας του εφαρμογιδίου μεταφέρεται στο σύστημα του πελάτη και εκτελείται από τη Java Virtual Machine (**JVM**) του φυλλομετρητή.

Κάποια παραδείγματα εφαρμογών που χρησιμοποιούν applet είναι τα προγράμματα ανταλλαγής εγγράφων και το animation (απόδοση κίνησης σε εικόνα).

Προφανώς, τα applet επιτρέπουν στον πελάτη να ζητήσει νέο περιεχόμενο από τον εξυπηρέτη. Αλλά, και **ο εξυπηρέτης μπορεί πλέον να κάνει «αυτόματη επανάκληση»** όποτε έχει νέο περιεχόμενο προς αποστολή.

4.2.1. Αυτόματη επανάκληση από την πλευρά του εξυπηρέτη

Στην τεχνική *αυτόματης επανάκλησης από την πλευρά του εξυπηρέτη* (server-side callback), ένα *αντικείμενο* (object) του εξυπηρέτη κάνει αυτόματη επανάκληση στο applet του πελάτη χρησιμοποιώντας RMI (Remote Method Invocation) ή CORBA (Common Object Request Broker Architecture). Συνήθως, πρώτος **ο πελάτης περνά μια απομακρυσμένη αναφορά ενός αντικειμένου RMI ή CORBA στο εξυπηρέτη**. Ο εξυπηρέτης διατηρεί μια λίστα με αυτές τις αναφορές και τις καλεί διαδοχικά κατά την κοινοποίηση.

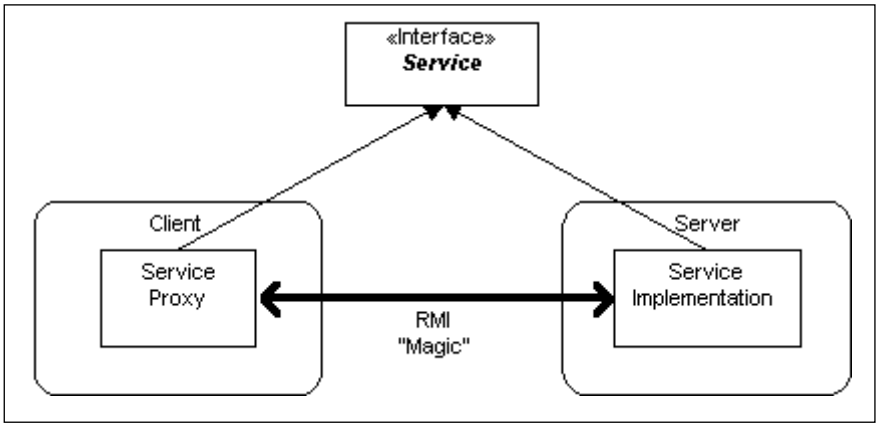
4.2.1.1. RMI

Η *RMI* (Remote Method Invocation) [23] είναι μια διεπαφή προγραμματισμού εφαρμογών Java η οποία αποτελεί το αντικειμενοστραφές ισοδύναμο του *RPC* (Remote Procedure Call).

Η Java RMI επιτρέπει στον προγραμματιστή να δημιουργεί **κατανεμημένες Java-προς-Java εφαρμογές**, στις οποίες οι μέθοδοι των απομακρυσμένων αντικειμένων Java μπορούν να κληθούν από άλλες JVM, ενδεχομένως σε διαφορετικούς host.

Για κάθε διεπαφή RMI (Εικόνα 13) εκτελούνται δύο κλάσεις: *η κλάση πληρεξούσιου* (proxy) τρέχει στη VM του πελάτη και *η κλάση της εφαρμογής* (implementation) τρέχει στη VM του εξυπηρέτη. Ως πελάτης αναφέρεται αυτός που ξεκινά την κλήση της RMI, ενώ ως εξυπηρέτης αναφέρεται αυτός που δέχεται και επεξεργάζεται την κλήση. Η *αυτόματη επανάκληση* (callback) είναι μια κλήση προς την αντίθετη κατεύθυνση.

Στις εφαρμογές client/server οι συναρτήσεις της διεπαφής χρήστη (UI) εκτελούνται στην εφαρμογή του πελάτη και το πρόγραμμα παροχής υπηρεσιών στο GUI τρέχει στον εξυπηρέτη.



Εικόνα 13: Αρχιτεκτονική RMI Client/Server εφαρμογής

Έστω ότι θέλουμε να αναπτύξουμε έναν τηλετυπώτη μετοχών. Τότε θα χρειαστούμε τα εξής:

Ο εξυπηρέτης θα πρέπει να παρέχει το `interface StockInfo`, το οποίο ορίζει τις μεθόδους `register()` και `unregister()`. Με αυτό τον τρόπο ο πελάτης λέει στον εξυπηρέτη πως υπάρχει.

```
package rmistock;import java.rmi.*;
public interface StockInfo extends java.rmi.Remote {
   void register(StockUpdate o) throws RemoteException;
   void unregister(StockUpdate o) throws RemoteException;
}
```

Ο εγγεγραμμένος πελάτης εκτελεί το `interface StockUpdate` και περνά μια αναφορά στο αντικείμενο του StockUpdate στη μέθοδο `register()`. Μέσω αυτής της αναφοράς θα μιλά

ο εξυπηρέτης με τον πελάτη (`update()`) όποτε χρειάζεται. Δηλαδή αυτό το αντικείμενο δρα ως το αντικείμενο “callback” της αρχιτεκτονικής.

```
package rmistock;import java.rmi.*;
public interface StockUpdate extends java.rmi.Remote {
   void update(String symbol, String price) throws RemoteException;
}
```

4.2.1.2. CORBA

Η *CORBA* (Common Object Request Broker Architecture) [24] είναι ένα πρότυπο ανάπτυξης κατανεμημένων εφαρμογών. Η CORBA αναπτύχθηκε για να **υποστηρίζει κώδικα ο οποίος τρέχει έξω από τα πλαίσια μιας JVM** και είναι *ουδέτερη ως προς τη γλώσσα* (language neutral). Αυτό σημαίνει πως τα αντικείμενά της (πελάτες και εξυπηρέτες) μπορεί να είναι γραμμένα σε διαφορετικές γλώσσες.

Μια εφαρμογή της αποτελείται από αντικείμενα CORBA εξυπηρετητών και CORBA πελατών οι οποίοι συνδέονται σε αυτούς τους εξυπηρέτες. Για να εντοπίσει ο πελάτης τον εξυπηρέτη θα πρέπει ο δεύτερος να εγγραφεί στο *CORBA Naming Service* ή σε ένα *CORBA Trader Service*. Μόλις εντοπίσει το αντικείμενο του εξυπηρέτη, ο πελάτης λαμβάνει μια αναφορά σε αυτό. Μέσω αυτής της αναφοράς ο πελάτης μπορεί να καλέσει (invoke) μεθόδους στο αντικείμενο του εξυπηρέτη και να κάνει τη δουλειά που θέλει.

Για τις περιπτώσεις που χρειάζεται ο εξυπηρέτης να καλέσει μια μέθοδο στο αντικείμενο ενός πελάτη, η CORBA **επιτρέπει «αυτόματες επανακλήσεις» στους πελάτες**. Σε αυτές τις περιπτώσεις, ο πελάτης πρέπει να περάσει μια αναφορά του εαυτού του στον εξυπηρέτη. Ο εξυπηρέτης πρέπει να αποθηκεύσει όλες τις αναφορές σε ένα language neutral φορμάτ και στη συνέχεια να καλεί όπως και όταν χρειάζεται τις μεθόδους στους κατάλληλους πελάτες.

Έστω ότι θέλουμε και πάλι να αναπτύξουμε έναν τηλετυπώτη μετοχών. Τότε θα χρειαστούμε τα εξής:

Το module `stock_ticker` ορίζει δύο interface: το `StockInfo` και το `StockUpdate`. Αυτά γράφονται σε *γλώσσα IDL* (Interface Definition Language). Το StockInfo δηλώνει μια CORBA μεταβλητή ακολουθίας για να κρατήσει τις αναφορές στα εγγεγραμμένα αντικείμενα

StockUpdate. Το interface StockInfo ορίζει τη μέθοδο `register()`. Με αυτή τη μέθοδο ο πελάτης λέει στον εξυπηρέτη πως υπάρχει. Το interface StockUpdate με τη σειρά του ορίζει τη μέθοδο `update()`. Αυτή δέχεται δύο παραμέτρους: το όνομα της μετοχής και τη νέα τιμή της.

4.2.2. Ενδιάμεσο λογισμικό προσανατολισμένο σε μηνύματα

Το *ενδιάμεσο λογισμικό προσανατολισμένο σε μηνύματα* (message-oriented-middleware, MOM) [25] είναι λογισμικό που διαχειρίζεται τις συναλλαγές μεταξύ πελατών και εξυπηρετών, **υποστηρίζει έμφυτα την αυτόματη προώθηση και το μοντέλο εγγραφής/δημοσίευσης**. **Ένα τυπικό κομμάτι του MOM περιλαμβάνει έναν αριθμό ουρών που περιέχουν μηνύματα και δεδομένα προερχόμενα είτε από έναν πελάτη είτε ένα εξυπηρέτη.** Το ενδιάμεσο λογισμικό μεσολαβεί μεταξύ των πελατών και εξυπηρετών. Αυτό το λογισμικό έχει ορισμένα πλεονεκτήματα: **οι πελάτες δεν χρειάζεται να είναι συνεχώς συνδεδεμένοι** - μπορούν απλά να τοποθετήσουν δεδομένα στο ενδιάμεσο λογισμικό και ν' αποσυνδεθούν, συλλέγοντας τις απαντήσεις αργότερα. Ως αποτέλεσμα η **κλιμακωσιμότητα** της εφαρμογής είναι μεγάλη. Επίσης, **το μοντέλο διεπαφής είναι απλό**: αποτελείται μόνο από τον πελάτη που τοποθετεί δεδομένα σε μια ουρά, που συντηρείται από το ενδιάμεσο λογισμικό, και από τον εξυπηρέτη που απομακρύνει τα δεδομένα, τα επεξεργάζεται και τοποθετεί μία απάντηση πίσω στην ουρά. Χάρη σ' αυτήν την απλότητα το MOM χρησιμοποιείται συχνά για την σύνδεση λογισμικού με κληροδοτημένα συστήματα.

Σε αυτή τη λύση ο πελάτης είναι ένα applet και ο εξυπηρέτης είναι ένας εξυπηρέτης μηνυμάτων ενδιάμεσου λογισμικού. Ο εξυπηρέτης ωθεί μηνύματα είτε σε μια σύνδεση TCP/IP (`java.net.Socket`) ή UDP χωρίς σύνδεση (`java.net.DatagramSocket`) ίσως ακόμη και multicast (`java.net.MulticastSocket`). Αφού τα μηνύματα δεν εκπέμπονται απαραίτητα ως πακέτα (TCP/IP), αποσυνδέεται πλέον η αποστολή από τη λήψη και μπορούν να λειτουργήσουν ασύγχρονα. **Ακόμη τα μηνύματα μεταδίδονται μία και μόνο φορά, ανεξαρτήτως σφαλμάτων ή προβλημάτων δικτύου**.

Αν επιλέξουμε να χρησιμοποιήσουμε ενδιάμεσο λογισμικό προσανατολισμένο σε μηνύματα, τότε μπορούμε να επιλέξουμε ένα προϊόν μηνυμάτων όπως το *iBus* (SoftWired) [26], το *MQSeries* (IBM) [27] ή το *WebLogic Events* (BEA) [28] ή να αναπτύξουμε ένα custom

προϊόν μηνυμάτων πάνω από socket (υποδοχές). Δυστυχώς δεν υπάρχει κάποιο ανοιχτό πρότυπο ή προδιαγραφή αλλά μόνο vendor specific υλοποιήσεις.

4.2.2.1. MQSeries

Από τα προαναφερθέντα, το κυριότερο ενδιάμεσο λογισμικό προσανατολισμένο σε μηνύματα είναι το MQSeries. Το ίδιο επεξεργάζεται τους εξής τέσσερις τύπους μηνυμάτων:

- Τα *Datagrams* είναι μονόδρομα μηνύματα χωρίς την επιστροφή απαντητικού μηνύματος, π.χ. το σήμα ότι ένας πελάτης είναι συνδεδεμένος.
- Τα *μηνύματα αίτησης* (Request Messages) χρησιμοποιούνται όταν ο αποστολέας περιμένει απάντηση, π.χ. όταν στέλνει ένα ερώτημα στην βάση δεδομένων.
- Τα *απαντητικά μηνύματα* (Reply Messages) είναι τα μηνύματα που αποστέλλονται ως απάντηση σε μηνύματα αίτησης. Και, τέλος,
- τα *μηνύματα αναφοράς* (Report Messages) χρησιμοποιούνται για να δώσουν σήμα σ' έναν πελάτη ότι κάποιο απρόβλεπτο γεγονός έχει συμβεί. Για παράδειγμα, ότι ένας εξυπηρέτης έχει βλάβη.

Το API των MQSeries είναι εξαιρετικά απλό και αποτελείται από μόνο 11 μεθόδους.

4.3. HTTP Server Push (HTTP Streaming)

Αφού η HTML προορίζεται για τη σχεδίαση, θα ήταν βολικό να αλλάζουν άμεσα τα μέρη του περιεχομένου της HTML με τα προσθετικά στοιχεία που προωθούνται από το εξυπηρέτη. Αυτό θα ήταν ένα ιδανικό σχήμα για τις δικτυακές εφαρμογές όπου το περιεχόμενο στο εξυπηρέτη μεταβάλλεται δυναμικά και η απαιτούμενη αλληλεπίδραση χρήστη-προς-εξυπηρέτη είναι ελάχιστη (π.χ. καθοδηγείται από HTML φόρμες).

Το πρωτόκολλο HTTP δε σχεδιάστηκε έχοντας κατά νου την αυτόματη προώθηση. Ωστόσο η ανάγκη για άμεση προώθηση δεδομένων στους χρήστες οδήγησε σε κάποιες λύσεις που ξεπερνούν αυτό το πρόβλημα.

Ο *HTTP server push* (γνωστός και ως HTTP streaming) [22] είναι ένας μηχανισμός αποστολής δεδομένων από έναν δικτυακό εξυπηρέτη σε ένα δικτυακό φυλλομετρητή. Στο μηχανισμό HTTP server push **ο εξυπηρέτης δεν τερματίζει μια σύνδεση αφού στείλει τα δεδομένα της απάντησης** σε κάποιον πελάτη. Αντιθέτως, αφήνει ανοικτή τη σύνδεση έτσι

ώστε αν γίνει κάποιο γεγονός ή κάποια αλλαγή στην κατάσταση να μπορεί να το στείλει άμεσα σε έναν ή περισσότερους πελάτες. Αλλιώς, τα δεδομένα θα έπρεπε να μπουν σε μια ουρά έως ότου ο πελάτης να κάνει μια νέα αίτηση. Εν τω μεταξύ, **ο φυλλομετρητής πρέπει να εξασφαλίσει ότι η διεπαφή χρήστη δείχνει τα νέα δεδομένα**.

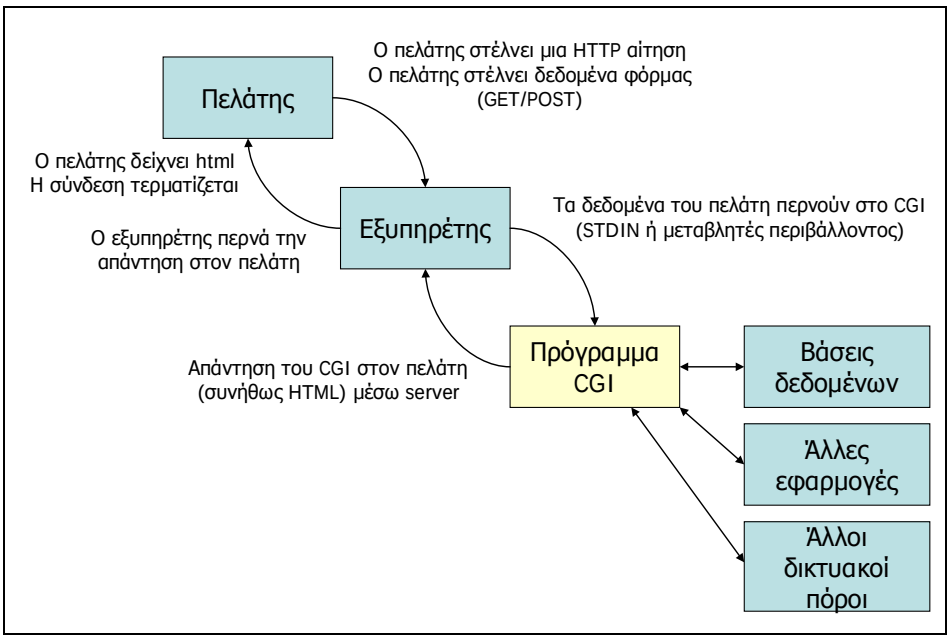
Αυτή η λειτουργικότητα μπορεί να επιτευχθεί με τις παρακάτω τεχνολογίες υλοποίησης:

4.3.1. CGI

Οι περισσότεροι εξυπηρέτες προσφέρουν τη λειτουργικότητα HTTP server push μέσω *CGI* (π.χ. Non-Parsed Headers scripts στον Apache), όπου δεν έχουμε ενδιάμεση αποθήκευση του αποτελέσματος (buffering) [29]. Η *CGI::Push* είναι μία υποκλάση του αντικειμένου CGI, η οποία χρησιμοποιείται για πράξεις αυτόματης προώθησης από τους εξυπηρέτες και επιτρέπει τη δημιουργία κινούμενων (animated) σελίδων το περιεχόμενο των οποίων αλλάζει σε τακτά χρονικά διαστήματα.

Η CGI::Push παίρνει ένα δείκτη σε μια υπορουτίνα η οποία σχεδιάζει μία σελίδα. Κάθε φορά που καλείται αυτή η υπορουτίνα, παράγεται και μια νέα σελίδα. Τα περιεχόμενα της σελίδας μεταδίδονται στο φυλλομετρητή με σκοπό να αντικαταστήσουν τα προηγούμενα. **Αυτή η τεχνική δουλεύει με HTML σελίδες καθώς και με αρχεία γραφικών, επιτρέποντας τη δημιουργία κινούμενων GIF.**

Η διαδικασία ροής δεδομένων στο CGI (Common Gateway Interface) είναι αυτή που δείχνει η Εικόνα 14.



Εικόνα 14: Διαδικασία ροής δεδομένων στο CGI

Το ακόλουθο είναι ένα παράδειγμα για το πώς γράφεται ένα CGI script, το οποίο κάνει χρήση της υποκλάσης CGI::Push. Το αποτέλεσμα είναι η προβολή ενός μετρητή (Counter) σε μία html σελίδα, ο οποίος μετρά από το 1 ως το 10 (συνάρτηση `next_page()`) και στη συνέχεια σταματά (συνάρτηση `last_page()`).

```
#!/usr/local/bin/perl5.00404
use CGI::Push qw(:standard);
do_push(-next_page=>\&next_page,
        -last_page=>\&last_page,
        -delay=>0.5);
sub next_page {
     my($q,$counter) = @_;
     if($counter >= 10){
          return undef;
     }
     return start_html("Counter"),
          h1("Counter"),"\n",
          "This page has been called ", strong($counter)," times",
          end_html();
}
```

Η `do_push()` παίρνει ως είσοδο την αναφορά σε δύο συναρτήσεις και σε μία καθυστέρηση. Η 1η συνάρτηση καλείται επανειλημμένως μέχρι να επιστρέψει `undef`, ενώ η 2η καλείται μία φορά.

Το `push` module θα περάσει δύο βαθμωτούς στη συνάρτηση. Ο 2ος βαθμωτός δίνει το πόσες φορές έχει κληθεί η συνάρτηση.

```
sub last_page {
     my($q,$counter) = @_;
     return start_html("Counter Done"),
          h1("Counter Finished"),
          strong($counter), " iterations.",
          end_html();
}
```

Το server push CGI πρόγραμμα πρέπει να αποθηκευτεί στο *cgi-bin* όπως και τα υπόλοιπα CGI προγράμματα. Για να προβληθεί σε μια ιστοσελίδα, το URL θα πρέπει να πρέπει να δηλώνει πως πρόκειται για *nph* (Non-Parsed Headers):

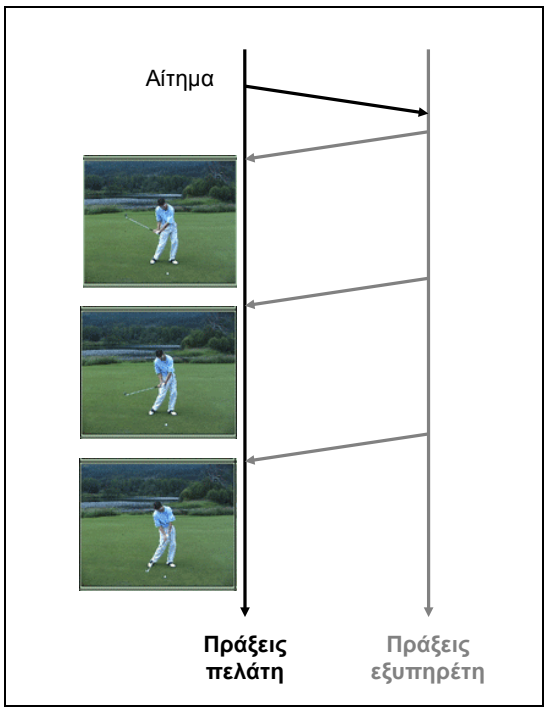
```
<P ALIGN=CENTER>
 <B> <font size="3" face="Arial">
  <a                      href="http://home-cgi.ust.hk/cgi-bin/nph-
cgiwrap/~horner/counter.cgi">
  server push: date </font>
 </B>
</P>
```

4.3.2. MIME

Το *MIME* (Multipurpose Internet Mail Extensions) [30] είναι ένα πρότυπο του Διαδικτύου που περιγράφει γενικότερα τον τύπο των περιεχομένων, αν και δημιουργήθηκε για την επέκταση του φορμάτ του ηλεκτρονικού ταχυδρομείου.

Πέρα από το CGI, ένας δεύτερος μηχανισμός HTTP server push σχετίζεται με έναν ειδικό τύπο του MIME που λέγεται `multipart/x-mixed-replace`, ο οποίος παρουσιάστηκε το 1995 από τη Netscape. Ο φυλλομετρητής καταλαβαίνει ότι η HTML σελίδα ή η εικόνα αλλάζει κάθε φορά που ο εξυπηρέτης ωθεί μια νέα έκδοση τους ως μέρος μιας πολυμερούς HTTP απάντησης, χρησιμοποιώντας τον τύπο περιεχομένων `multipart/x-mixed-replace`.

Το multipart/x-mixed-replace μπορεί να εφαρμοστεί σε HTML κείμενα και σε streaming εικόνες εφαρμογών webcam (Εικόνα 15).



Εικόνα 15: Χρονοδιάγραμμα server push με MIME

Για το σχεδιασμό μιας πεπερασμένης ακολουθίας κινούμενων σχεδίων, συμπεριλαμβάνεται στο Perl script ένας πεπερασμένος βρόχος ο οποίος ολοκληρώνεται στο τελευταίο πλαίσιο της εικόνας. Μια συνεχής σειρά κινουμένων σχεδίων από την άλλη δεν σταματά έως ότου ο χρήστης να κάνει κλικ στο «Stop» του προγράμματος περιήγησης ή να φύγει εντελώς από τη σελίδα. Όσο ο χρήστης είναι συνδεδεμένος σε μια σελίδα που έχει μια αδιάκοπη σειρά κινουμένων σχεδίων, **η θύρα της TCP σύνδεσης παραμένει ανοιχτή καθ' όλη τη διάρκεια της επίσκεψης του χρήστη**.

Το ακόλουθο είναι ένα παράδειγμα για το πώς γράφεται ένα server push CGI script. Υποθέτοντας ότι υπάρχουν μια σειρά από εικόνες, οι frame1.gif-frame10.gif, το script τις προβάλει κυκλικά και επ' άπειρον. Το σημαντικό σημείο είναι να δηλωθεί το έγγραφο ως `multipart/x-mixed-replace` και να οριστεί μια τυχαία συμβολοσειρά ως `boundary`.

```
#!/usr/local/bin/Perl
 use GD;
 $frmLoc="/user/bdeng/Web/docs/images";
 $header="Content-type: multipart/x-mixed-replace;" .
         "boundary=***Boundary_String***\n";
```

```
 $boundary="\n--***Boundary_String***\n";
 $giftype="Content-type: image/gif\n\n";
 print $header;
 print $boundary;
 $i=1;
 while (1) {
    sleep 1;
    print $giftype;
    open(GIFH,"< ${frmLoc}/frame${i}.gif");
    $img = newFromGif GD::Image(GIFH);
    close(GIFH);
    print $img->gif;
    print $boundary;
    if ($i == 10) {
       $i=1;
    } else {
       $i++
    } }
```

Για να προβληθεί αυτό το animation σε μία ιστοσελίδα, θα πρέπει το γνώρισμα `SRC` της ετικέτας `<IMG>` να δείχνει στο CGI script, κάπως έτσι:

```
<HTML>
 <BODY>
  <H1>Animated GIF using Server-push</H1>
  <IMG SRC=/cgi-bin/frames.pl>
  <P>This is a poor man's animation that downloads the next frame
continuously until you leave the page.
 </BODY>
 </HTML>
```

4.3.3. Hidden Iframe (COMET)

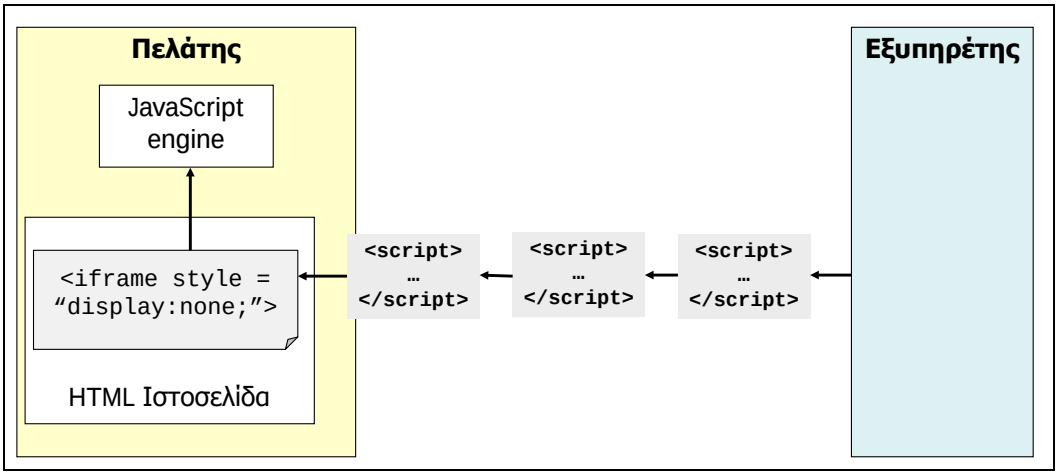
Το βασικό πρόβλημα με το XHR (το βασικό εργαλείο που χρησιμοποιείται από τις εφαρμογές Ajax για την επικοινωνία browser–server) είναι πως ήταν σχεδιασμένο έτσι ώστε η απάντηση να επιστρέφει αμέσως και να αντιμετωπίζεται από το φυλλομετρητή ως

ένα γεγονός. Για αυτό δεν μπορούσε να χρησιμοποιηθεί όπως ήταν για streaming. Ωστόσο, υπάρχει ένας μηχανισμός στο φυλλομετρητή που δεν κάνει αυτή την υπόθεση, η ετικέτα `<script>`. Όποτε τη συναντά, ο φυλλομετρητής εκτελεί αμέσως το script χωρίς να περιμένει να φορτώσει ολόκληρο το έγγραφο. Πώς μπορούν να σταλούν όμως πολλά script στο φυλλομετρητή; Τη λύση δίνει το HTML στοιχείο *hidden IFrame* (inline frame – επιτρέπει σε ένα ιστοχώρο να ενσωματώνει ένα HTML κείμενο μέσα σε ένα άλλο) [31]. Το hidden IFrame στέλνεται σαν ένα μεγάλο μπλοκ το οποίο είναι δηλωμένο ως απείρου μήκους (κάποιες φορές λέγεται «forever-frame»). Καθώς γίνονται τα γεγονότα, το IFrame σταδιακά γεμίζει με ετικέτες script, οι οποίες περιέχουν JavaScript, η οποία πρέπει να εκτελεστεί στο φυλλομετρητή. Κάθε script tag εκτελείται με το που λαμβάνεται και έτσι έχουμε τη λειτουργικότητα που ζητήθηκε.

Το σταδιακό γέμισμα με tags λέγεται *chunked encoding* και είναι ένα χαρακτηριστικό της προδιαγραφής HTTP 1.1. **Το chunked encoding επιτρέπει σε έναν εξυπηρέτη να ξεκινήσει την αποστολή μιας απάντησης πριν μάθει το συνολικό της μήκος.** Αυτό επιτρέπει στον εξυπηρέτη να σπάσει την πλήρη απάντηση σε μικρότερα «κομμάτια» και να τα στείλει σε σειρά. Οι απαντήσεις (response) είναι εύκολο να ανιχνευθούν επειδή στην κεφαλίδα τους περιέχουν το «`Transfer-Encoding: chunked`».

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked
23
This is the data in the first chunk
1A
and this is the second one
0
```

Οι φυλλομετρητές, όπως είπαμε, μπορούν να αποδώσουν αυξητικά τα κομματιασμένα κείμενα. Κάθε κομμάτι τυλίγεται σε ένα script block και εκτελείται με την κλήση μιας συνάρτησης στη βιβλιοθήκη του πελάτη Comet η οποία βρίσκεται στο γονικό έγγραφο (Εικόνα 16).



Εικόνα 16: Αλληλεπίδραση πελάτη-εξυπηρέτη στο Hidden Iframe

Δε συμπεριφέρονται, όμως, ακριβώς όπως ορίζεται σε αυτή την τεχνική όλοι οι φυλλομετρητές. Για παράδειγμα, ο IE απαιτεί ένα διερμηνευτικό στοιχείο όπως η ετικέτα `<bra>` και ο Safari απαιτεί 1KB δεδομένων (τα οποία στέλνονται συνήθως στη μορφή ενός λευκού κενού), για να κάνουν αυξητική απόδοση. Τα μεγάλα toolkit του Comet παρέχουν αυτόματα αυτούς τους εναλλακτικούς τρόπους αντιμετώπισης. Για την αποφυγή υπερβολικής χρήσης μνήμης, οι DOM κόμβοι οι οποίοι προστίθενται στο IFrame συνήθως αφαιρούνται αφού αποδοθούν.

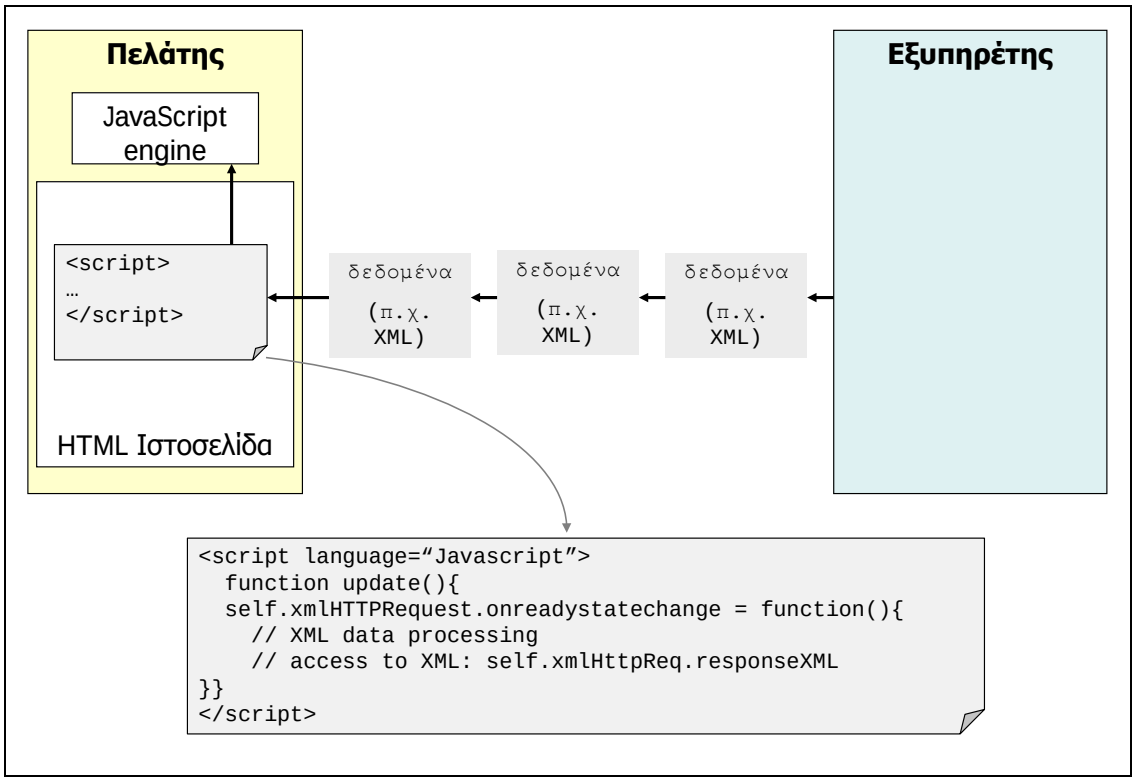
Πίνακας 4: Υποστήριξη του Iframe από τους φυλλομετρητές

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
πριν 11 εκδόσεις			4.0							
πριν 10 εκδόσεις			5.0							
πριν 9 εκδόσεις			6.0							
πριν 8 εκδόσεις		2.0	7.0							
πριν 7 εκδόσεις		3.0	8.0		9.0					
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6					
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1					
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5					
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0		
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1	
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2	
τωρινή έκδοση	9.0	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3,	3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0	
μελλοντικά	10.0	10.0	17.0	6.0	12.0					

4.3.4. Multipart XHR (COMET)

Στους σύγχρονους φυλλομετρητές είναι δυνατό να ξεκινήσεις HTTP αιτήσεις απ' ευθείας από τη JavaScript και να πάρεις το αποτέλεσμα σε JavaScript, χωρίς την ανάγκη hidden frames και άλλων τέτοιων κόλπων. Το XHR δίνει πλέον τη δυνατότητα παραλαβής δεδομένων από τον εξυπηρέτη αφού έχει φορτώσει η σελίδα και μπορεί να χρησιμοποιηθεί για τα server–browser Comet μηνύματα με διάφορους τρόπους.

Από το 2004 οι **φυλλομετρητές, οι οποίοι βασίζονται στο Gecko** (π.χ. ο Firefox), δέχονται **πολυμερείς απαντήσεις στο XHR** [32], το οποίο μπορεί για αυτό το λόγο να χρησιμοποιηθεί για μεταφορά streaming Comet. Στην πλευρά του εξυπηρέτη κάθε μήνυμα κωδικοποιείται σαν ένα ξεχωριστό κομμάτι της πολυμερούς απάντησης, ενώ στην πλευρά του πελάτη η συνάρτηση επιστροφής (ο οποία δίνεται στο XHR στην ιδιότητα `onreadystatechange`) καλείται καθώς φθάνει κάθε νέο μήνυμα (Εικόνα 17: Αλληλεπίδραση client-server στο multipart XHREικόνα 17).



Εικόνα 17: Αλληλεπίδραση client-server στο multipart XHR

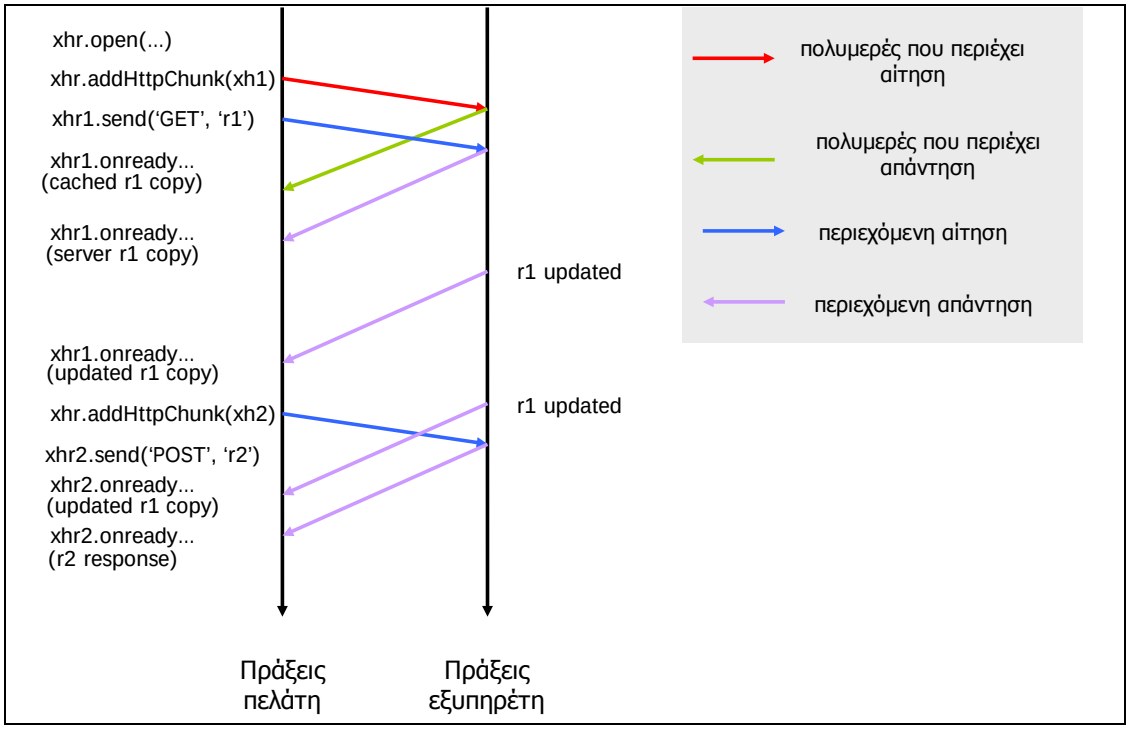
Για παράδειγμα, ο εξυπηρέτης μπορεί αρχικά να κάνει output ένα div το οποίο θα περιέχει τα τελευταία νέα.

```
  print ("<div id='news'></div>");
```

Αλλά αντί να κάνει `exit`, ξεκινά ένα βρόχο για να ενημερώνει το αντικείμενο κάθε 10 δευτερόλεπτα.

```
    <? while (true) { ?>
      <script type="text/javascript">
        $('news').innerHTML = '<?= getLatestNews() ?>';
      </script>
  <? flush(); // Ensure the Javascript tag is written out
//immediately
     sleep(10);  } ?>
```

Αντί της δημιουργίας μιας πολυμερούς απάντησης, και την εξάρτηση από το φυλλομετρητή για να αναλύσει διάφανα κάθε συμβάν, είναι επίσης εφικτή η δημιουργία ενός custom φορμάτ δεδομένων για την XHR απάντηση. Κάθε γεγονός αναλύεται στη συνέχεια από το φυλλομετρητή με τη χρήση JavaScript θεωρώντας απλώς πως πυροδοτείται η συνάρτηση επιστροφής (`callback function` - η οποία ορίζεται στην ιδιότητα `onreadystatechange` του αντικειμένου XHR) κάθε φορά που λαμβάνονται νέα δεδομένα (Εικόνα 18).



Εικόνα 18: Χρονοδιάγραμμα server push με multipart XHR

Πίνακας 5: Υποστήριξη του multipart XHR από τους φυλλομετρητές

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
πριν 11 εκδόσεις			4.0							
πριν 10 εκδόσεις			5.0							
πριν 9 εκδόσεις			6.0							
πριν 8 εκδόσεις		2.0	7.0							
πριν 7 εκδόσεις		3.0	8.0		9.0					
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6					
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1					
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5					
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0		
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1	
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2	
τωρινή έκδοση	9.	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3	3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0	
μελλοντικά	10.0	10.0	17.0	6.0	12.0					

4.3.5. Java Pushlet

Τα *Pushlet* [33] είναι ένας μηχανισμός που βασίζεται στα servlet, όπου τα δεδομένα προωθούνται άμεσα από τα αντικείμενα Java του εξυπηρέτη σε (Dynamic) HTML σελίδες μέσα σε ένα φυλλομετρητή **χωρίς τη χρήση Java applet ή plugin**. Ο εξυπηρέτης εκμεταλλεύεται τις επίμονες (persistent) HTTP συνδέσεις και αφήνει την απάντηση πάντα «ανοικτή» (δηλ. δεν τερματίζει ποτέ την απάντηση), «ξεγελώντας» έτσι το φυλλομετρητή ο οποίος συνεχίζει να φορτώνει. Στη συνέχεια, **ο εξυπηρέτης στέλνει περιοδικά μικρά τμήματα JavaScript, αντικείμενα Java ή XML** για να ενημερώσει τα περιεχόμενα της σελίδας. Αυτό επιτρέπει στη ιστοσελίδα να ενημερώνεται περιοδικά.

Η τεχνική pushlet αναπτύχθηκε αρχικά από τις δικτυακές εφαρμογές Java, αν και οι ίδιες τεχνικές μπορούν να χρησιμοποιηθούν και από άλλα πλαίσια ανάπτυξης. Ο πελάτης χρησιμοποιεί JavaScript/χαρακτηριστικά της Dynamic HTML που διατίθενται σε φυλλομετρητές όπως ο NetScape και ο IE. Ο υποκείμενος μηχανισμός χρησιμοποιεί μία servlet HTTP σύνδεση πάνω από την οποία ωθείται JavaScript κώδικας στον browser. Μέσα από ένα ενιαίο γενικό servlet (το Pushlet), οι πελάτες browser μπορούν να εγγραφούν σε θέματα από τα οποία επιθυμούν να λάβουν γεγονότα.

Στην πλευρά του εξυπηρέτη, εκτελείται ένα JSP όταν ζητείται η σελίδα. Αυτό το JSP προωθεί JavaScript στον πελάτη, όταν το ειδοποιεί ο εξυπηρέτης. Μόλις ο εξυπηρέτης λάβει μια εντολή όπως:

```
<script language=JavaScript >parent.push('Page 4')</script>
```

τότε η μηχανή JavaScript του φυλλομετρητή εκτελεί την εντολή και εμφανίζει τη σελίδα. **Κάθε φορά που ο εξυπηρέτης προωθεί ένα γεγονός, ειδοποιούνται οι εγγεγραμμένοι πελάτες στο σχετικό θέμα.** Τα αντικείμενα των γεγονότων μπορούν να σταλούν ως JavaScript (DHTML πελάτες), ως συνέχειες αντικειμένων Java (Java πελάτες) ή ως XML (DHTML ή Java πελάτες).

Για να πάρει κανείς μια μικρή γεύση των Pushlet μπορεί να δει το ακόλουθο παράδειγμα. Η σελίδα `push-js-stream.html` έχει δύο frame (πλαίσιο), το Pushlet Frame και το Display Frame. Το Pushlet Frame είναι κρυφό (μέγεθος 0) και περιέχει μία JSP (`push-js-stream-pusher.jsp`). Μέσα στη JSP προωθείται μία γραμμή JavaScript κάθε 3 δευτερόλεπτα. Αυτή η γραμμή καλεί τη συνάρτηση `push()` (μέσω του `parent.push()`) με το νέο

περιεχόμενο ως όρισμα. Η συνάρτηση `push()`θα ανανεώσει το περιεχόμενο του Display Frame γράφοντας στο κείμενό του. Το Display Frame είναι αρχικά μαύρο και γράφει «WAIT…» και στη συνέχεια δείχνει το περιεχόμενο που του προωθείται από τον εξυπηρέτη (π.χ. Server pushes: Page 1).

push-js-stream.html

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Pragma" content="no-cache">
<script LANGUAGE="JavaScript">
 var pageStart="<HTML><HEAD></HEAD><BODY BGCOLOR=blue TEXT=white><H2>Server pushes: <P>";
 var pageEnd="</H2></BODY></HTML>";
 // Callback function with message from server.
 // This function is called from within the hidden JSP pushlet frame
 function push(content) {
 // Refresh the display frame with the content received

window.frames['displayFrame'].document.writeln(pageStart+content+pageEnd);
 window.frames['displayFrame'].document.close();
 }
</script>
</HEAD>


<FRAMESET BORDER=0 COLS="*,0">
  <!-- frame to display the content pushed by the pushlet -->
  <FRAME SRC="push-js-stream-display.html" NAME="displayFrame" BORDER=0 SCROLLING=no>


  <!-- Hidden frame with the pushlet that pushes lines of JavaScript-->
```

```
  <FRAME     SRC="push-js-stream-pusher.jsp"     NAME="pushletFrame"
BORDER=0  SCROLLING=no>
</FRAMESET>


</HTML>
```

push-js-stream-pusher.jsp

```
<HTML>
<HEAD>
 <META  HTTP-EQUIV="Content-Type"  CONTENT="text/html;  charset=iso-
8859-1">
 <meta http-equiv="Pragma" content="no-cache">
</HEAD>
<BODY BGCOLOR="blue">
 <%
 /** Start a line of JavaScript with a function call to parent
frame. */
 String jsFunPre = "<script language=JavaScript >parent.push('";
 /** End the line of JavaScript */
 String jsFunPost = "')</script> ";
 try {
  // Every three seconds a line of JavaScript is pushed to the
client
  for (int i=1; i<10; i++) {
   // Push a line of JavaScript to the client
   out.print(jsFunPre+"Page "+i+jsFunPost);
   out.flush();
   // Sleep three secs
   try {
    Thread.sleep(3000);
   } catch (InterruptedException e) {
    // Let client display exception
    out.print(jsFunPre+"InterruptedException: "+e+jsFunPost);
```

```
   }
  }
 } catch (Exception e) {
  // Let client display exception
  out.print(jsFunPre+"Exception: "+e+jsFunPost);
 }
 out.print(jsFunPre+"DONE "+jsFunPost);
%>
</BODY>
</HTML>
```

push-js-stream-display.html

```
<HTML>
 <BODY BGCOLOR=black TEXT=white>
  <H1>WAIT...</H1>
 </BODY>
</HTML>
```

Ο **μηχανισμός είναι ελαφρύς**, υπό την έννοια ότι χρησιμοποιεί τη διαχείριση της σύνδεσης του εξυπηρέτη των servlet και νήματα, τα APIs javax.servlet και τυποποιημένα χαρακτηριστικά γνωρίσματα της Java όπως το παραγωγός/καταναλωτής μέσω των wait() και notify() μεθόδων του Object. Ακόμη, όταν χρησιμοποιείται JavaScript/DHTML στον πελάτη, αυτό προσφέρει ένα βολικό τρόπο για να αναπτυχθούν γρήγορα εφαρμογές μέσω scripting και για να ενσωματωθεί και να σχεδιαστεί το νέο περιεχόμενο με τα χαρακτηριστικά της HTML/CSS.

4.3.6. Server-Sent Events (HTML5)

Η πρόταση WHATWG [34] Web Applications 1.0 συμπεριλαμβάνει έναν μηχανισμό προώθησης περιεχομένου στον πελάτη. Στις 1-9-2006 ο φυλλομετρητής Opera υλοποίησε αυτή την πειραματική τεχνολογία σε ένα χαρακτηριστικό που λέγεται *Server-Sent Events (SSE)* [8]. Αυτή τη στιγμή το API είναι working draft και τυποποιείται ως μέρος του HTML5. Τα γεγονότα του SSE μοιάζουν με τα γεγονότα της Javascript τα οποία συμβαίνουν στο

φυλλομετρητή (π.χ. το γεγονός click), με τη διαφορά ότι στο SSE μπορούμε να ελέγξουμε το όνομά τους και τα δεδομένα με τα οποία συνδέονται.

Ο μηχανισμός SSE χρησιμοποιείται συνήθως για την αποστολή μηνυμάτων ενημέρωσης ή συνεχόμενων ροών δεδομένων με τη μορφή *συμβάντων DOM* σε κάποιο φυλλομετρητή πάνω από μια HTTP σύνδεση και έχει σχεδιαστεί για να ενισχύσει το **cross-browser streaming**. Η προσέγγιση event streaming ανοίγει μια μόνιμη σύνδεση στον εξυπηρέτη πάνω από την οποία στέλνει νέα πληροφορία στον πελάτη όταν αυτή είναι διαθέσιμη (και άρα σε πραγματικό χρόνο), μηδενίζοντας την ανάγκη για συνεχές polling. Για να επιτύχει το στόχο της, παρουσιάζει το JavaScript API `EventSource` και το `text/event-stream`.

- Το `EventSource` (στοιχείο HTML DOM) αντιπροσωπεύει την πλευρά του πελάτη ο οποίος λαμβάνει τα γεγονότα. Ο πελάτης, δημιουργώντας ένα `EventSource`, ανοίγει μία ροή γεγονότων (event stream). Αυτό το EventSource παίρνει ως όρισμα στον constructor το URL του εξυπηρέτη. Το πρόγραμμα χειρισμού συμβάντων `onmessage` θα καλείται κάθε φορά που λαμβάνονται νέα δεδομένα.
- Ο τύπος MIME `text/event-stream` ορίζει το φορμάτ του πλαισίου ενός γεγονότος.

Στο παράδειγμα:

```
REQUEST:
GET /Events HTTP/1.1
Host: myServer:8875
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE)
        AppleWebKit/532+ (KHTML, like Gecko) Version/4.0.4
        Safari/531.21.10
Accept-Encoding: gzip, deflate
Referer: http://myServer:8875/
Accept: text/event-stream
Last-Event-Id: 6
Accept-Language: de-DE
Cache-Control: no-cache
Connection: keep-alive
```

```
RESPONSE:
HTTP/1.1 200 OK
Server: xLightweb/2.12-HTML5Preview
Content-Type: text/event-stream
Expires: Fri, 01 Jan 1990 00:00:00 GMT
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Connection: close
```

Για την αποφυγή του caching, η κεφαλίδα της απάντησης περιέχει οδηγίες που απενεργοποιούν το caching. Εξ ορισμού άλλωστε οι ροές γεγονότων δε θα έπρεπε να αποθηκεύονται.

```
: time stream
retry: 5000

id: 7
data: Thu Sep 23 07:31:30 CET 2010

id: 8
data: Thu Sep 23 07:31:35 CET 2010

[...]
```

Τρία γεγονότα τα οποία οριοθετούνται από την αλλαγή γραμμής.

Γίνεται μία σύνδεση στον εξυπηρέτη `myServer` (θύρα 8875) και μόλις στείλει ο εξυπηρέτης μια ροή γεγονότων, τότε διαβιβάζεται ένα μήνυμα γεγονότος στο στοιχείο `EventSource`.

Αν ο εξυπηρέτης στείλει για παράδειγμα το ακόλουθο γεγονός (με τον τύπο text/event-stream MIME type):

```
id: 8\n
data: Thu Sep 23 07:31:35 CET 2010\n
\n
```

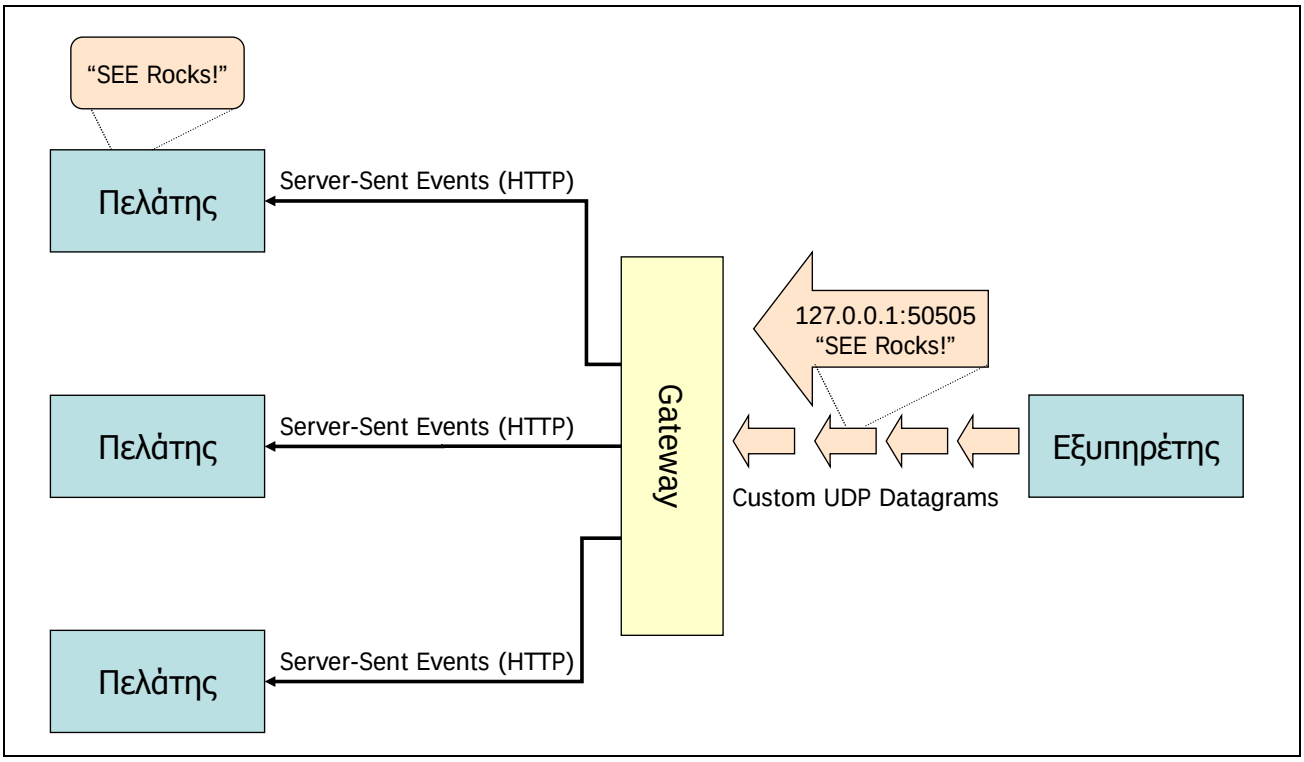
Τότε το στοιχείο `EventSource` στο φυλλομετρητή διαβιβάζει ένα γεγονός μηνύματος με ένα γνώρισμα δεδομένων το οποίο περιέχει τη συμβολοσειρά:

```
{Thu Sep 23 07:31:35 CET 2010}
```

Σε αυτή την περίπτωση λαμβάνεται από τον πελάτη η τρέχουσα ημερομηνία του εξυπηρέτη. Αντί της ημερομηνίας θα μπορούσε να στέλνεται η τιμή κάποιας μετοχής και στη συνέχεια αυτή να προβάλλεται στη σελίδα.

Επιπρόσθετα, αν ο εξυπηρέτης συμπεριλαμβάνει την κεφαλίδα `id` για ένα γεγονός (π.χ. `id: 7`) και ο πελάτης προσθέτει την κεφαλίδα `Last-Event-ID` όταν επανασυνδέεται, τότε μπορεί να εξασφαλιστεί η σωστή παράδοση των μηνυμάτων, αφού η ροή γεγονότων θα συνεχίζει χωρίς την επανάληψη ή την απώλειά τους όταν ολοκληρώνεται η απάντηση HTTP. Ακόμη, ο εξυπηρέτης μπορεί να ελέγξει το πόσο θα πρέπει να περιμένει ο πελάτης μέχρι να ξανασυνδεθεί ορίζοντας την προαιρετική επικεφαλίδα `retry` (π.χ. `retry: 5000`). Αν το instance του `EventSource` ανιχνεύσει ότι η σύνδεση έπεσε, τότε αυτή θα πρέπει να αποκατασταθεί αυτόματα μετά από μια καθυστέρηση ίση με το χρόνο επανασύνδεσης.

Στην Εικόνα 19 παρουσιάζεται η αρχιτεκτονική του SSE.



Εικόνα 19: Αρχιτεκτονική Server-Sent Events

Πίνακας 6: Υποστήριξη του SSE από τους φυλλομετρητές

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
πριν 11 εκδόσεις			4.0						
πριν 10 εκδόσεις			5.0						
πριν 9 εκδόσεις			6.0						
πριν 8 εκδόσεις		2.0	7.0						
πριν 7 εκδόσεις		3.0	8.0		9.0				
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6				
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1				
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5				
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0	
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2
τωρινή έκδοση	9.0	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3, 3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0
μελλοντικά	10.0 	10.0	17.0	6.0	12.0				

4.3.7. Αποφάσεις σχεδιασμού στο HTTP Streaming

Έχοντας επιλέξει να κάνουμε HTTP Streaming υπάρχουν κάποιες νέες αποφάσεις που πρέπει να πάρουμε πριν ξεκινήσουμε να γράφουμε κώδικα.

Τα νέα ερωτήματα τα οποία τίθενται είναι τα εξής:

- Πόσο καιρό θα μένει ανοικτή τη σύνδεση;

Είναι ανέφικτο να μείνει μια σύνδεση ανοικτή για πάντα. Θα πρέπει ο σχεδιαστής να αποφασίσει ένα εύλογο χρονικό διάστημα για να κρατήσει ανοικτή τη σύνδεση, και αυτό θα εξαρτηθεί από:

- Τους πόρους που απαιτούνται: εξυπηρέτης, δίκτυο, φυλλομετρητής και το λογισμικό που τα υποστηρίζει στην πορεία.
- Πόσοι πελάτες θα είναι συνδεδεμένοι ανά πάσα στιγμή. Όχι μόνο κατά μέσο όρο, αλλά στις περιόδους αιχμής.
- Πώς θα χρησιμοποιηθεί το σύστημα – πόσα δεδομένα θα γίνουν output, πώς θα αλλάζει αυτή η δραστηριότητα με την πάροδο του χρόνου.
- Ποιες είναι οι συνέπειες των πολλών ταυτόχρονων συνδέσεων. Για παράδειγμα, θα χάσουν κάποιοι χρήστες κρίσιμες πληροφορίες;

Είναι δύσκολο να δώσει κανείς ακριβή στοιχεία, αλλά φαίνεται σωστό να υποθέσουμε ότι μια μικρή εφαρμογή intranet θα μπορούσε να ανεχθεί μια σύνδεση λεπτών ή και ωρών, ενώ μια δημόσια dotcom εφαρμογή θα μπορούσε να προσφέρει αυτή την υπηρεσία μόνο για γρήγορες, εξειδικευμένες καταστάσεις.

Αν ο πελάτης θέλει να συνεχίσει να λαμβάνει ενημερώσεις αφού κλείσει η σύνδεση, τότε θα πρέπει να επανασυνδεθεί. Ο χρόνος επανασύνδεσης παίζει σημαντικό ρόλο στη βελτίωση της αξιοπιστίας στις περιπτώσεις που προκύπτουν σφάλματα δικτύου.

- Πότε θα κλείνει η σύνδεση;

Ένα web service έχει διάφορους τρόπους για να προκαλέσει το κλείσιμο μιας σύνδεσης:

- Έφθασε σε κάποιο χρονικό όριο.

- Έστειλε το πρώτο μήνυμα.
- Συνέβη ένα συγκεκριμένο γεγονός. Για παράδειγμα, το stream μπορεί να δείχνει την πρόοδο ενός πολύπλοκου υπολογισμού και να καταλήγει στο ίδιο το αποτέλεσμα.
- Ποτέ. Ο πελάτης πρέπει να τερματίσει την σύνδεση.

- Πώς θα κάνει διάκριση ο φυλλομετρητής μεταξύ των μηνυμάτων;

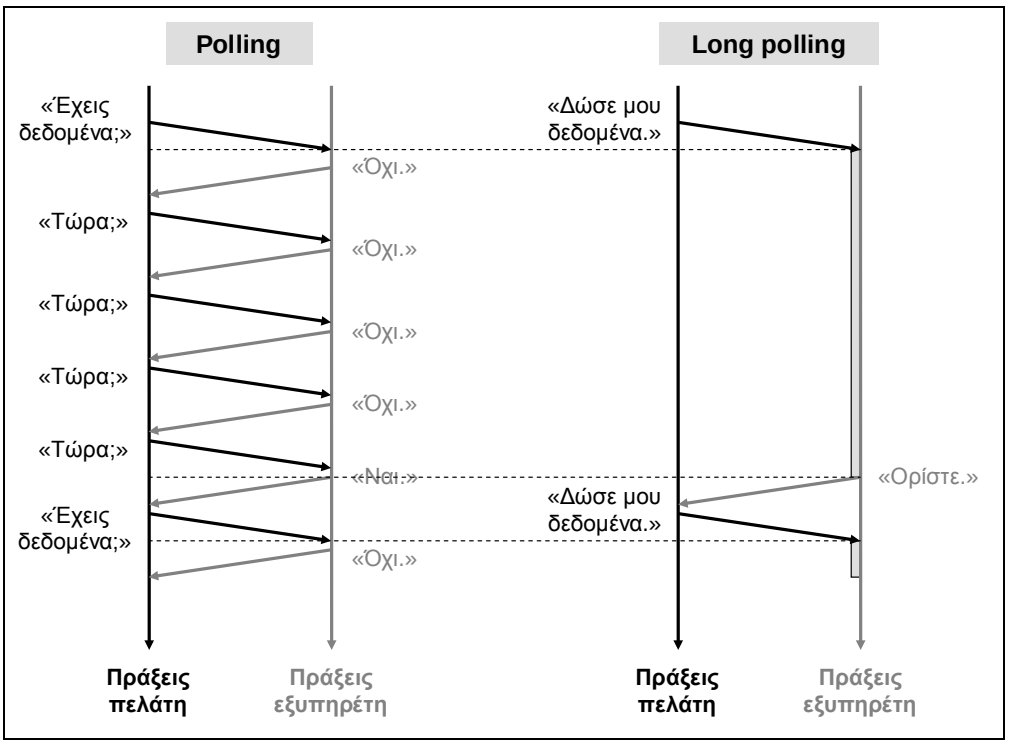
Αφού η υπηρεσία δεν μπορεί να διαγράψει ό,τι έχει κάνει output, συχνά χρειάζεται στην έξοδο μια σειρά από ξεχωριστά μηνύματα. Εάν η τεχνολογία δεν το κάνει ήδη, θα χρειαστεί κάποιο πρωτόκολλο για την οριοθέτηση των μηνυμάτων, π.χ. μηνύματα που αποστέλλονται με ένα τυποποιημένο πρότυπο, μηνύματα που χωρίζονται από ένα ειδικό token string, ή μηνύματα που συνοδεύεται από κάποια κομμάτια μεταδεδομένων (π.χ. μια επικεφαλίδα η οποία αναγράφει το μέγεθος των μηνυμάτων).

4.4. Long Polling (COMET)

Η τεχνική *long polling* [35] αποτελεί μια παραλλαγή της short polling (Εικόνα 20). Εδώ, αν ο εξυπηρέτης δεν έχει διαθέσιμη κάποια απάντηση για τον πελάτη, αντί να του στείλει μια άδεια απάντηση, κρατά την αίτηση έως ότου να γίνει διαθέσιμη κάποια πληροφορία. Μόλις η πληροφορία γίνει διαθέσιμη (ή μετά από κατάλληλο timeout) ο πελάτης λαμβάνει μια πλήρη απάντηση. Κανονικά ο πελάτης ξαναστέλνει αμέσως μια αίτηση, έτσι ώστε να υπάρχει πάντα μια αίτηση διαθέσιμη που περιμένει την αποστολή δεδομένων σε απάντηση κάποιου συμβάντος. Ιδανικά αν φτάσουν m μηνύματα στο χρονικό διάστημα (0, t), τότε ο πελάτης στέλνει το πολύ m αιτήσεις στον εξυπηρέτη.

Η long polling είναι καλύτερη από την απλή polling για τρεις λόγους:

- Η καθυστέρηση ανάμεσα σε δύο διαδοχικές ενημερώσεις είναι η ελάχιστη δυνατή.
- Το δίκτυο επιβαρύνεται όσο το δυνατόν λιγότερο.
- Το πλήθος των συνδέσεων που ανοίγονται μπορεί να μειωθεί σημαντικά.



Εικόνα 20: Χρονοδιάγραμμα long polling vs. polling

Αν ο πελάτης δεν είναι online εξ αρχής, αλλά υπήρξε κάποια δραστηριότητα για την οποία πρέπει να ενημερωθεί μπαίνοντας στην εφαρμογή, τότε θα πρέπει όλα τα μηνύματα της εφαρμογής να μπαίνουν σε μια ουρά.

Αν και η long polling δεν είναι μια τεχνολογία push, μπορεί να χρησιμοποιηθεί όταν δεν μπορεί να γίνει πραγματικό push. Κάποιες τεχνολογίες υλοποίησης long polling είναι οι κάτωθι:

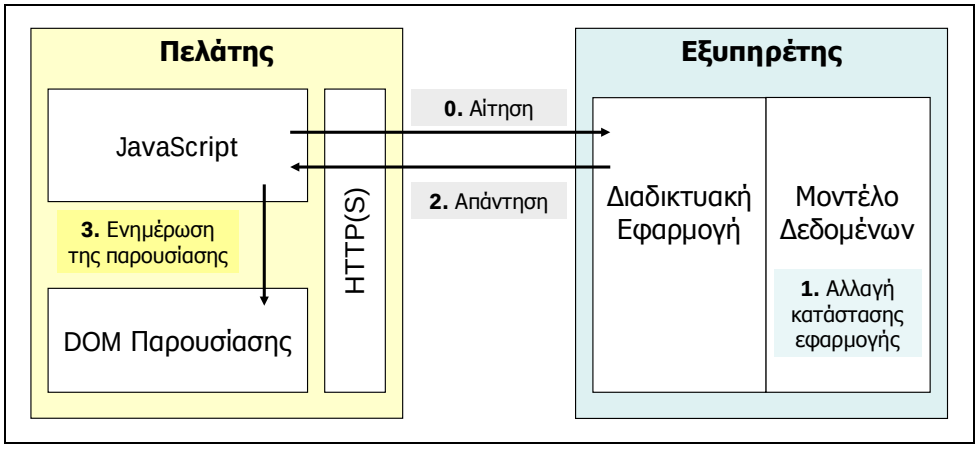
4.4.1. XHR

Μια ακόμη τεχνολογία του COMET είναι το XHR Long Polling. Στο μεγαλύτερο μέρος του, το XHR long polling δουλεύει όπως κάθε κανονική εφαρμογή του XHR. Ο φυλλομετρητής κάνει μια ασύγχρονη αίτηση στον εξυπηρέτη, ο οποίος μπορεί να περιμένει να γίνουν διαθέσιμα τα δεδομένα πριν να απαντήσει. Η απάντηση μπορεί να περιμένει κωδικοποιημένα δεδομένα (συνήθως XML ή JSON) ή JavaScript για να εκτελεστεί από τον πελάτη. Στο τέλος της επεξεργασίας της απάντησης, ο φυλλομετρητής δημιουργεί και στέλνει ένα ακόμη XHR, για να περιμένει το επόμενο γεγονός. Με αυτό τον τρόπο ο φυλλομετρητής πάντα κρατά μια αίτηση σε εκκρεμότητα με τον εξυπηρέτη, για να απαντηθεί καθώς συμβαίνουν τα γεγονότα.

Το ελάχιστο που απαιτείται για το XHR long polling, όπως φαίνεται και στην Εικόνα 21, είναι οι μηχανισμοί διαχείρισης των blocking αιτήσεων, οι οποίες σχετίζονται με το long polling, και ενημέρωσης της παρουσίασης, όταν έρχονται οι απαντήσεις.

Η διαδικασία η οποία ακολουθείται είναι αυτή:

0. Η αρχική blocking αίτηση στέλνεται με τη χρήση XHR για να ξεκινήσει η ακολουθία του long polling.
1. Κάποια αλλαγή κατάστασης στην εφαρμογή παράγει μια απάντηση ο οποία περιέχει ενημερώσεις στην παρουσίαση.
2. Η παραγόμενη απάντηση στέλνεται στον πελάτη.
3. Η JavaScript στην πλευρά του πελάτη χειρίζεται την απάντηση και ενημερώνει την παρουσίαση.
4. Γύρισε πίσω στην αρχική κατάσταση (0) όπου παράγεται μια άλλη blocking αίτηση.



Εικόνα 21: Αλληλεπίδραση πελάτη-εξυπηρέτη κατά το AJAX long polling

4.4.2. Script Tag

Ενώ μπορούμε να κάνουμε κάθε μεταφορά Comet να δουλέψει σε subdomains, καμία από τις άλλες μεταφορές COMET δεν μπορεί να χρησιμοποιηθεί σε διαφορετικά second-level domains (SLDs), εξαιτίας των πολιτικών ασφαλείας των φυλλομετρητών που έχουν σχεδιαστεί για να εμποδίζουν τις cross-site scripting επιθέσεις. Αυτό σημαίνει πως, αν η κύρια ιστοσελίδα σερβίρεται από ένα SLD και ο εξυπηρέτης Comet βρίσκεται σε κάποιο άλλο SLD, τότε τα συμβάντα του Comet δεν μπορούν να χρησιμοποιηθούν για να αλλάξει η

HTML και το DOM της κύριας σελίδας, χρησιμοποιώντας αυτές τις τεχνολογίες μεταφοράς. Αυτό το πρόβλημα μπορεί να αποφευχθεί με τη δημιουργία ενός εξυπηρέτη proxy μπροστά από τη μία ή και τις δύο πηγές, ο οποίος θα τις κάνει να φαίνονται πως ξεκινούν από το ίδιο domain. Ωστόσο, αυτό συχνά δεν είναι επιθυμητό για λόγους πολυπλοκότητας ή απόδοσης.

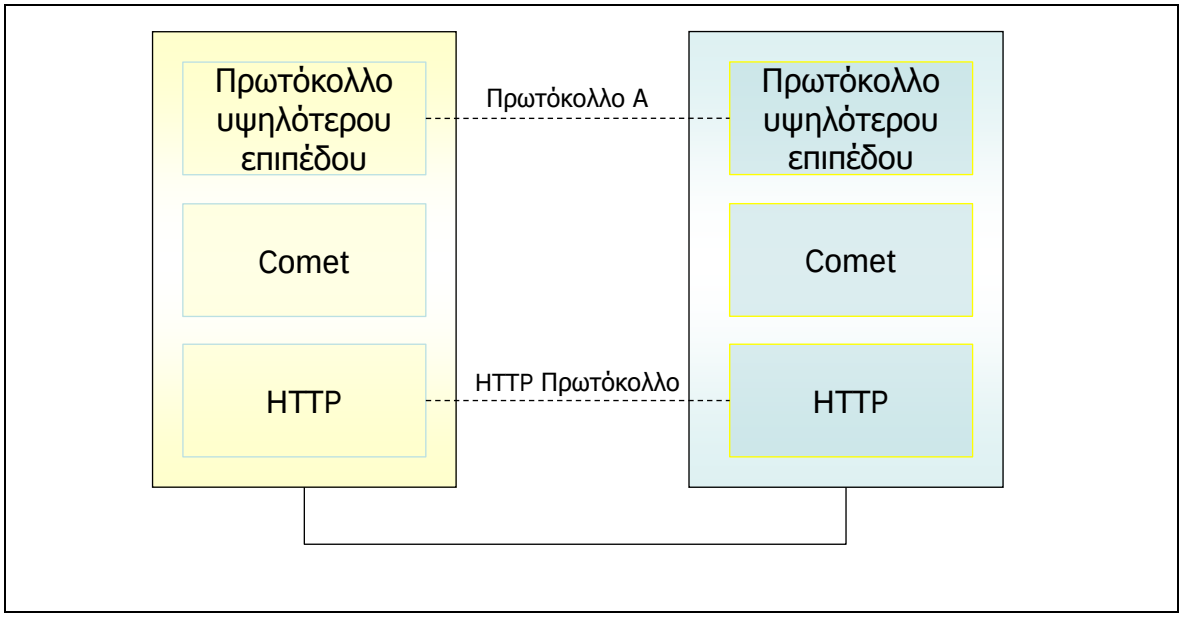
Σε αντίθεση με τα IFrame ή τα XHR αντικείμενα, οι ετικέτες `<script>` μπορούν να δείχνουν σε οποιοδήποτε URI, και ο κώδικας JavaScript της απάντησης θα εκτελείται στο τρέχον κείμενο HTML. Αυτό δημιουργεί ένα πιθανό ρίσκο ασφάλειας και για τους δύο εξυπηρέτες που εμπλέκονται, αν και ο κίνδυνος στα δεδομένα του πάροχου (στην περίπτωση αυτή του εξυπηρέτη Comet) μπορεί να αποφευχθεί με χρήση του JSONP.

Η long-polling Comet μεταφορά μπορεί να υλοποιηθεί με τη **δυναμική δημιουργία στοιχείων <script>**. Τα στοιχεία αυτά μπαίνουν στο `<head>` του εγγράφου και το `<src>` κομμάτι τους δείχνει στο κανάλι εγγραφής του Comet εξυπηρέτη. Όταν ο εξυπηρέτης έχει διαθέσιμα νέα δεδομένα, τότε στέλνει JavaScript (ή JSONP) η οποία πυροδοτεί μια υπάρχουσα μέθοδο στον πελάτη. Μόλις ολοκληρωθεί η επεξεργασία των δεδομένων ο φυλλομετρητής ανοίγει μια νέα σύνδεση στον εξυπηρέτη.

Αυτή η μέθοδος έχει το πλεονέκτημα πως είναι **cross-browser** ενώ επιτρέπει ακόμη τις **cross-domain** εφαρμογές. Όμως έχει και ένα μεγάλο πρόβλημα, την **έλλειψη ελέγχου επί της σύνδεσης**. Δεν υπάρχει ανατροφοδότηση σχετικά με την κατάσταση του load από τον εξυπηρέτη. Οι Firefox και Opera υποστηρίζουν τις `onload` συναρτήσεις επιστροφής, ο IE υποστηρίζει μόνον τις `onreadystatechange` και ο Safari δεν αναφέρει τίποτα. Η συνάρτηση επιστροφής μιας JSONP συμβολοσειράς κάνει δυνατή την ανίχνευση μιας πετυχημένης φόρτωσης, αλλά μια σύνδεση που ακυρώνεται ή λήγει χρονικά αποτυγχάνει αθόρυβα (δεν ειδοποιείται ο εξυπηρέτης).

4.5. Πρωτόκολλα του COMET

Όπως φάνηκε και από τις προηγούμενες ενότητες, ο όρος COMET χαρακτηρίζει τις τεχνικές αυτόματης προώθησης πληροφορίας από τον εξυπηρέτη προς τον πελάτη οι οποίες χρησιμοποιούν Ajax. Η ανάπτυξη διαδικτυακών εφαρμογών COMET είναι πολύπλοκη, οπότε για να μειωθεί αυτή η πολυπλοκότητα έχουν οριστεί κάποια πρωτόκολλα. Τα πρωτόκολλα αυτά είναι **υψηλότερου επιπέδου από το HTTP** (Εικόνα 22) και ορίζουν πώς θα είναι οι αιτήσεις και οι απαντήσεις σε μια διαδικτυακή εφαρμογή η οποία χρησιμοποιεί Comet.



Εικόνα 22: Αρχιτεκτονική των πρωτοκόλλων του Comet

Ένα σημαντικό στοιχείο αυτών των πρωτοκόλλων είναι ότι προσπαθούν να εισάγουν ένα **αμφίδρομο κανάλι επικοινωνίας**. Η αμφίδρομη επικοινωνία (bi-directional communication) δε σημαίνει απλά προώθηση μηνυμάτων από τον εξυπηρέτη προς τον πελάτη σε πραγματικό χρόνο, αλλά και το αντίστροφο. Είναι πλέον σημαντικό τα μηνύματα του πελάτη να φτάνουν στον εξυπηρέτη με μικρή καθυστέρηση και υψηλή συχνότητα.

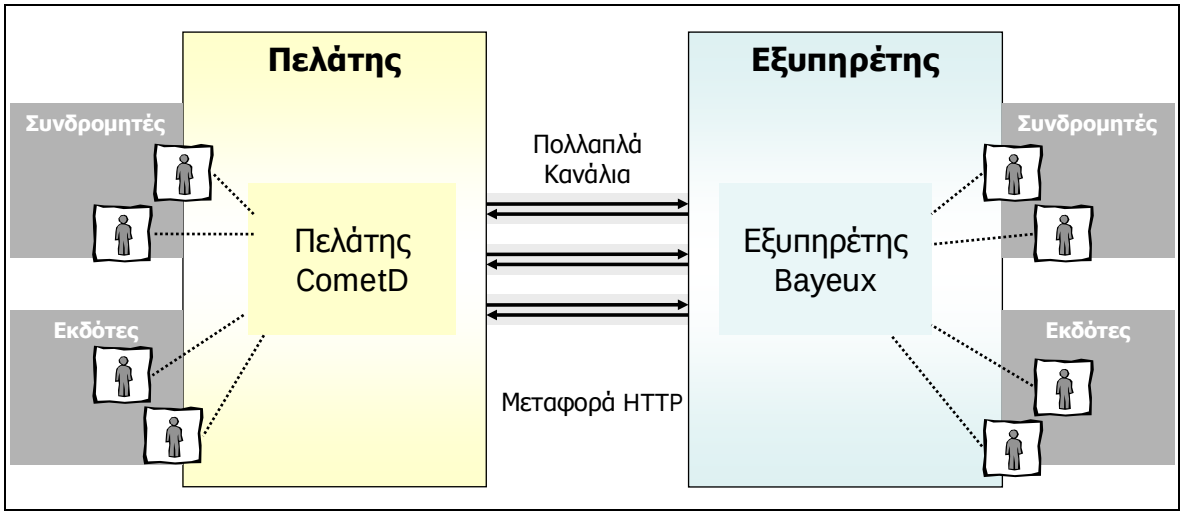
Στη συνέχεια παρουσιάζονται τρία σημαντικά πρωτόκολλα του COMET. Ο λόγος που γίνεται εδώ η περιγραφή τους, αν και δεν αποτελούν κάποια ξεχωριστή τεχνική προώθησης, είναι ότι κάθε εφαρμογή που θέλει να κάνει χρήση COMET είναι σχεδόν βέβαιο πως θα χρησιμοποιήσει ένα από αυτά. Ακόμη, η τεχνική προώθησης WebSocket,

την οποία παρουσιάζουμε στην επόμενη ενότητα, προτάθηκε για να καλύψει με πιο φυσικό τρόπο την ανάγκη αμφίδρομης επικοινωνίας.

4.5.1. Bayeux

Το *πρωτόκολλο Bayeux* [BAYEUX] αναπτύχθηκε κατά την περίοδο 2006-2007 από το ίδρυμα Dojo. Το Bayeux μπορεί να χρησιμοποιήσει τους μηχανισμούς **long polling και HTTP streaming**.

Παρέχει αφαίρεση πάνω από τη μεταφορά Comet, κατανέμοντας τα μεταδιδόμενα δεδομένα υπό όρους **καναλιών**, στα οποία μπορούν οι πελάτες να δημοσιεύουν και να εγγράφονται. Προκειμένου να επιτευχθεί αμφίδρομη επικοινωνία, ο πελάτης Bayeux χρησιμοποιεί δύο HTTP συνδέσεις σε ένα εξυπηρέτη Bayeux ώστε να μπορούν να σταλούν ασύγχρονα μηνύματα από τον εξυπηρέτη στον πελάτη και το αντίστροφο (Εικόνα 23). Η προδιαγραφή Bayeux δίνει τον ορισμό του πρωτοκόλλου. Το Cometd είναι ένα project του Dojo Foundation για την παραγωγή της προδιαγραφής και ενός συνόλου υλοποιήσεών της.



Εικόνα 23: Αλληλεπίδραση πελάτη-εξυπηρέτη στο Bayeux

Η προδιαγραφή Bayeux απαιτεί από την υλοποίηση να ελέγχει τη σωλήνωση των αιτήσεων HTTP, έτσι ώστε τα αιτήματα να μη σωληνώνονται ακατάλληλα (π.χ. ένα μήνυμα από τον πελάτη προς τον εξυπηρέτη σωληνώνεται πίσω από μία αίτηση long poll). Στην πράξη, για JavaScript πελάτες, αυτός ο έλεγχος στη σωλήνωση δεν είναι εφικτός στους σημερινούς φυλλομετρητές. Συνεπώς, οι εφαρμογές JavaScript του Bayeux προσπαθούν να τηρήσουν αυτή την απαίτηση περιορίζοντας τους εαυτούς τους σε **δύο το πολύ ξεχωριστές αιτήσεις**

HTTP σε οποιαδήποτε στιγμή, έτσι ώστε τα όρια σύνδεσης του φυλλομετρητή να μην εφαρμοστούν και να μη μπουν σε ουρά ή να σωληνωθούν οι αιτήσεις. Αν και σε γενικές γραμμές αποτελεσματικός, ο μηχανισμός αυτός μπορεί να διαταραχθεί αν ένας πελάτης που δε χρησιμοποιεί Bayeux JavaScript κάνει ταυτόχρονα αιτήσεις προς τον ίδιο host.

Για να γίνει η σύνδεση Bayeux, ο πελάτης κάνει μια διαπραγμάτευση με τον εξυπηρέτη μέσω των μηνυμάτων **χειραψίας**. Αυτά τους επιτρέπουν να συμφωνήσουν στον τύπο σύνδεσης, στη μέθοδο ελέγχου ταυτότητας (authentication) και σε άλλες παραμέτρους. Επιπλέον, κατά τη διάρκεια της φάσης της χειραψίας, ο πελάτης και ο εξυπηρέτης αποκαλύπτουν ο ένας στον άλλο τις *αποδεκτές αμφίδρομες τεχνικές* και ο πελάτης επιλέγει μία από την τομή των δύο συνόλων.

Τα μηνύματα από τον πελάτη προς τον εξυπηρέτη στέλνονται ως κωδικοποιημένη *JSON στις παραμέτρους της αίτησης στο URL*.

Τα μηνύματα από τον εξυπηρέτη προς τον πελάτη στέλνονται ως προγράμματα *JavaScript* τα οποία ενσωματώνουν το JSON μήνυμα με μια κλήση σε μια συνάρτηση JavaScript στην ήδη φορτωμένη εφαρμογή Bayeux.

```
// publish message
{
  "channel": "/some/name",
  "clientId": "83js73jsh29sjd92",
  "data": { "myapp" : "specific data", value: 100 }
}
```

× Το πρόβλημα με αυτό το πρωτόκολλο είναι πως **δεν αποτελεί πρότυπο της Java** και έτσι υπάρχουν διαφορετικά API για κάθε υλοποίηση.

× Υλοποιείται πάνω από το περιβάλλον της γλώσσας του προγράμματος περιήγησης και δεν είναι τόσο απλό όσο το SSE.

✓ Το καλό είναι πως υποστηρίζει ένα **αμφίδρομο κανάλι επικοινωνίας**.

✓ Επιπρόσθετα, το Bayeux μπορεί να χρησιμοποιεί **HTTP streaming καθώς και Long Polling**.

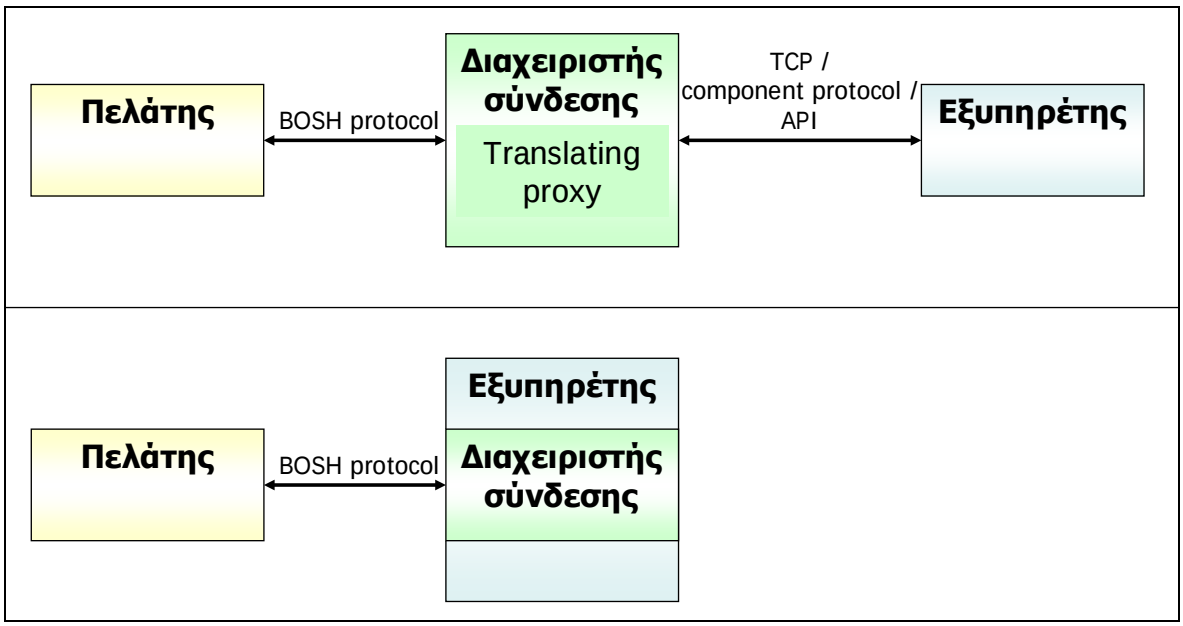
4.5.2. BOSH

Το *πρωτόκολλο BOSH* (Bidirectional-streams Over Synchronous HTTP) αποτελεί ένα draft standard του XMPP Standards Foundation. Πρόκειται για ένα πρωτόκολλο μεταφοράς το οποίο *μιμείται την αμφίδρομη ροή* μεταξύ δύο οντοτήτων (π.χ. ενός πελάτη και ενός εξυπηρετητή) με τη χρήση πολλαπλών σύγχρονων ζευγαριών αιτημάτων/απαντήσεων HTTP χωρίς να απαιτείται η χρήση polling ή ασύγχρονου chunking, αλλά με **long polling**.

Το σχετικό *πρότυπο XMPP Over BOSH* καθορίζει τον τρόπο με τον οποίο μπορεί να χρησιμοποιηθεί το BOSH για τη μεταφορά XMPP stanzas. Το αποτέλεσμα είναι ένα HTTP binding για XMPP επικοινωνίες που χρησιμοποιείται σε περιπτώσεις όπου η συσκευή ή ο πελάτης δεν είναι σε θέση να διατηρήσουν μια επίμονη TCP σύνδεση με ένα εξυπηρέτη XMPP.

Η προτεινόμενη αρχιτεκτονική **περιλαμβάνει έναν εξειδικευμένο διαχειριστή σύνδεσης** (Connection Manager - CM). Πρόκειται για έναν εξειδικευμένο εξυπηρέτη HTTP ο οποίος κάνει μετάφραση ανάμεσα στις αιτήσεις/απαντήσεις HTTP του πελάτη και στις ροές δεδομένων (ή API) του εξυπηρέτη με τον οποίο επικοινωνεί. Έτσι δίνει τη δυνατότητα στους πελάτες να συνδεθούν με τον εξυπηρετητή μέσω HTTP στη θύρα 80 ή στην 443 αντί για κάποια άλλη θύρα αφιερωμένη στην εφαρμογή.

Η προδιαγραφή του πρωτοκόλλου BOSH **καλύπτει μόνο την επικοινωνία μεταξύ του πελάτη και του διαχειριστή της σύνδεσης**. Δεν καλύπτει την επικοινωνία μεταξύ του διαχειριστή σύνδεσης και του εξυπηρέτη, δεδομένου ότι αυτή εξαρτάται από την εφαρμογή. Ο εξυπηρέτης μπορεί να υποστηρίζει εγγενώς τις συνδέσεις HTTP, οπότε ο διαχειριστής της σύνδεσης θα είναι μια λογική και όχι μια φυσική οντότητα. Εναλλακτικά, ο διαχειριστής σύνδεσης μπορεί να είναι ένας ανεξάρτητος μεσολαβητής (proxy) ο οποίος θα μεταφράζει, κάνοντας τον εξυπηρέτη να πιστεύει ότι μιλά απευθείας με τον πελάτη πάνω από το TCP, από ένα πρωτόκολλο συνιστωσών (component protocol) ή από ένα API οριζόμενο από την εφαρμογή του εξυπηρέτη. Οι δύο αρχιτεκτονικές παρουσιάζονται γραφικά στην Εικόνα 24.



Εικόνα 24: Οι δύο αρχιτεκτονικές του BOSH

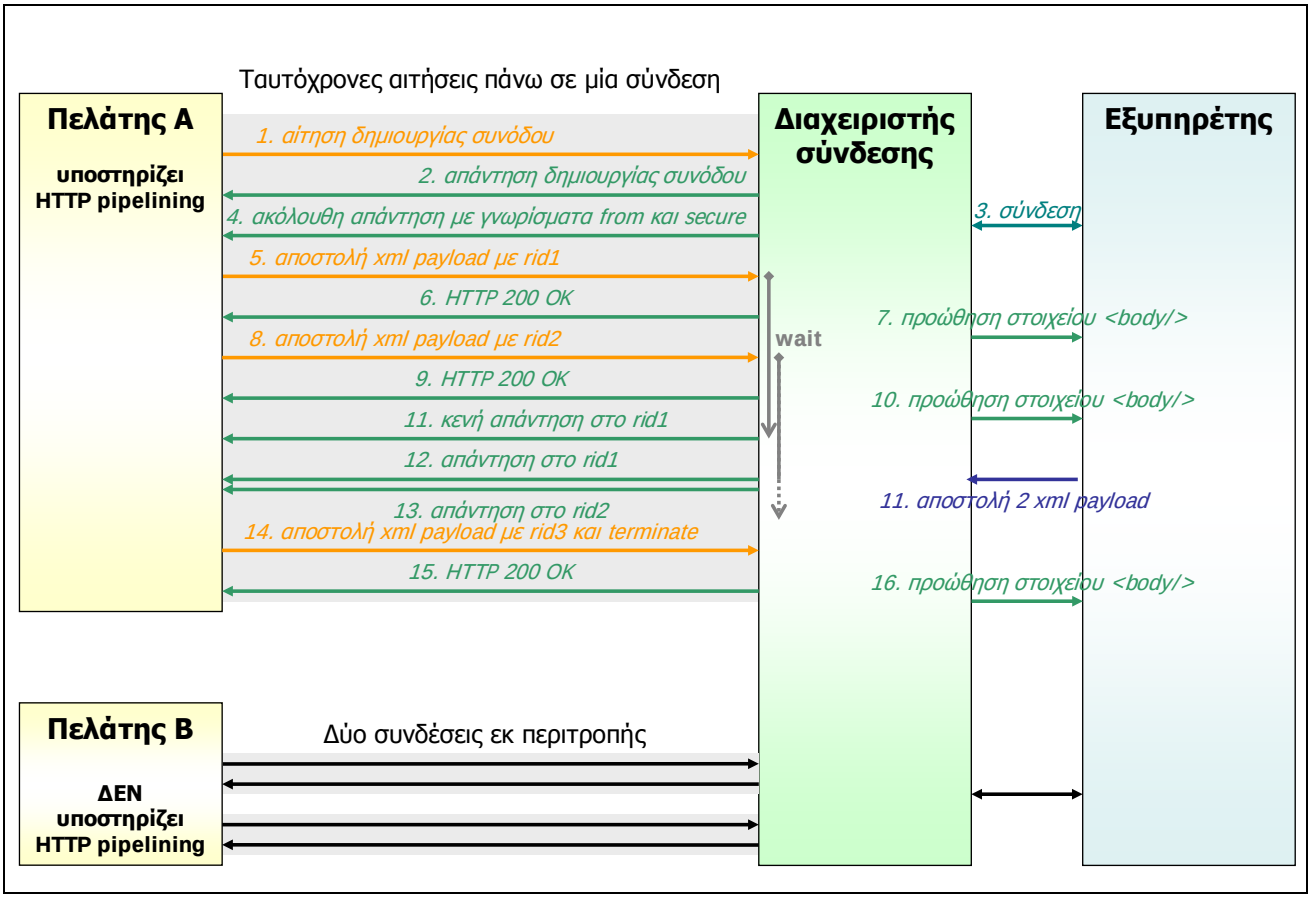
Για να ξεκινήσει η επικοινωνία μέσω BOSH ο πελάτης στέλνει ένα **αίτημα συνόδου BOSH**:

```
POST /webclient HTTP/1.1
Host: httpcm.example.com
Accept-Encoding: gzip, deflate
Content-Type: text/xml; charset=utf-8
Content-Length: 104

<body content='text/xml; charset=utf-8'
from='user@example.com'
hold='1'
rid='1573741820'
to='example.com'
route='xmpp:example.com:9999'
ver='1.6'
wait='60'
ack='1'
xml:lang='en'
xmlns='http://jabber.org/protocol/httpbind'/>
```

Ο εξυπηρέτης, εφόσον υποστηρίζει το πρωτόκολλο, απαντά:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 128

<body wait='60'
      inactivity='30'
      polling='5'
      requests='2'
      hold='1'
      ack='1573741820'
      accept='deflate,gzip'
      maxpause='120'
      sid='SomeSID'
      charsets='ISO_8859-1 ISO-2022-JP'
      ver='1.6'
      from='example.com'
      xmlns='http://jabber.org/protocol/httpbind'/>
```

Στην Εικόνα 25 παρουσιάζεται γραφικά ένα παράδειγμα BOSH επικοινωνίας. Στην περίπτωση που ο φυλλομετρητής δεν υποστηρίζει *σωλήνωση* (tunneling), χρειάζεται να ανοίξει μια νέα σύνδεση για κάθε νέα αίτηση προς τον εξυπηρέτη.



Εικόνα 25: Αλληλεπίδραση πελάτη-εξυπηρέτη στο BOSH

× Το πρόβλημα και με αυτό το πρωτόκολλο είναι πως **δεν αποτελεί πρότυπο της Java** και έτσι υπάρχουν διαφορετικά API για κάθε υλοποίηση.

× Υλοποιείται πάνω από το περιβάλλον της γλώσσας του προγράμματος περιήγησης και δεν είναι τόσο απλό όσο το SSE.

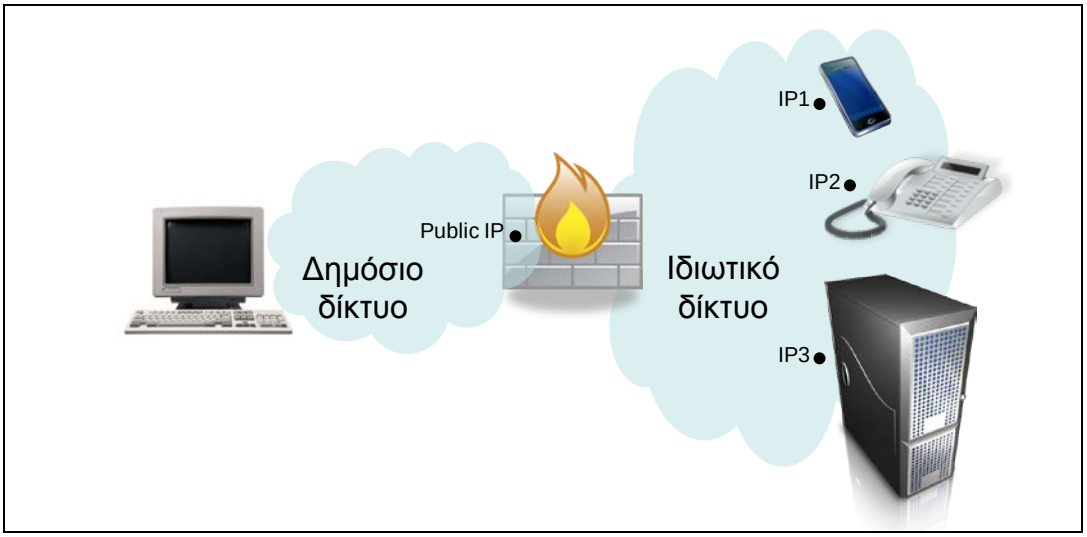
✓ Όπως και το Bayeux, έτσι και το BOSH πρωτόκολλο υποστηρίζει ένα **αμφίδρομο κανάλι επικοινωνίας**.

✓ Το BOSH, όμως, βασίζεται **μόνο στο long polling**.

- ✓ Κάθε μπλοκ δεδομένων που προωθείται από τον connection manager είναι μια πλήρης HTTP απάντηση. Έτσι το BOSH **δεν έχει πρόβλημα με τους ενδιάμεσους μεσολαβητές** οι οποίοι αποθηκεύουν τις μερικές HTTP απαντήσεις.
- ✓ Επίσης είναι **πλήρως συμβατό με το HTTP/1.0** το οποίο δεν υποστηρίζει chunked transfer encoding.

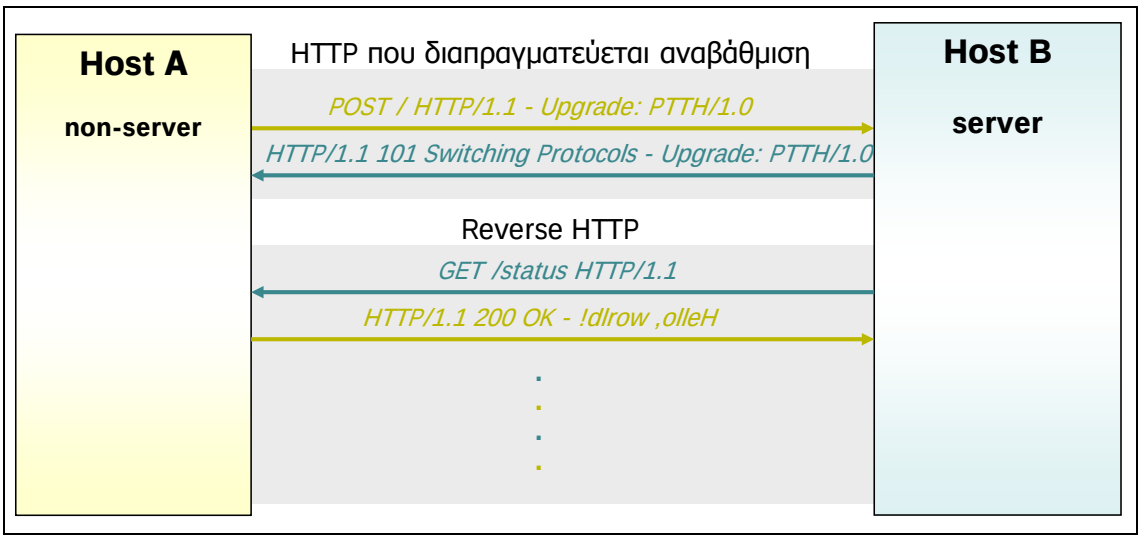
4.5.3. Reverse HTTP

Σε πολλές εφαρμογές, κάποιος host δεν μπορεί να λειτουργήσει σαν http **εξυπηρέτης**. Ο ίδιος μπορεί να **βρίσκεται πίσω από κάποιο τείχος προστασίας ή να μην έχει διεύθυνση IP που να είναι publicly routable** (Εικόνα 26). Σε αυτή την περίπτωση δεν μπορεί να χρησιμοποιηθεί συμμετρικό πρωτόκολλο στο οποίο ο εξυπηρέτης να ξεκινά TCP σύνδεση προς τον πελάτη με στόχο την αποστολή μιας αίτησης.



Εικόνα 26: Εξυπηρέτης πίσω από τείχος προστασίας και σε ιδιωτικό δίκτυο

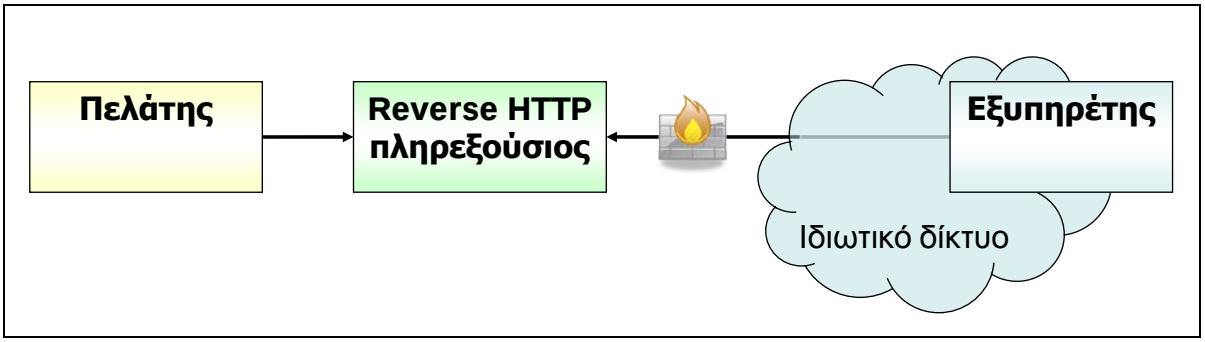
Το Reverse HTTP [36] είναι ένα πειραματικό πρωτόκολλο το οποίο εκμεταλλεύεται την κεφαλίδα Upgrade του HTTP/1.1 ώστε να αντιστρέφει το HTTP socket. Όταν ο πελάτης κάνει μια αίτηση σε έναν εξυπηρέτη με κεφαλίδα `Upgrade: PTTH/1.0`, τότε ο εξυπηρέτης μπορεί να απαντήσει με την κεφαλίδα `Upgrade: PTTH/1.0` και από εκεί και πέρα να αρχίσει ο ίδιος να χρησιμοποιεί το socket σαν πελάτης (ο πελάτης χρησιμοποιεί το socket σαν εξυπηρέτης – Εικόνα 27). Υπάρχει ακόμη ένα COMET-like πρωτόκολλο το οποίο θα δουλεύει με πελάτες HTTP/0.9 ή 1.1 οι οποίοι δε γνωρίζουν πώς να κάνουν αυτή την αναβάθμιση.



Εικόνα 27: Αλληλεπίδραση πελάτη-εξυπηρέτη στο Reverse HTTP

Με τον τρόπο αυτό, ο πελάτης μπορεί να εγκαταστήσει μια **αμφίδρομη επικοινωνία HTTP** σε έναν εξυπηρέτη (ή πληρεξούσιο) χρησιμοποιώντας δύο TCP συνδέσεις: Μία να τρέχει μια μόνιμη σύνδεση HTTP προς τον απομακρυσμένο υπολογιστή, και μια άλλη, μετά από τη διαπραγμάτευση για Reverse HTTP, να τρέχει μια μόνιμη σύνδεση HTTP από το εξυπηρέτη πίσω στον πελάτη.

Με αυτό το πρωτόκολλο δε χρειάζεται πλέον VPN (Virtual Public Network). Η ασφάλεια μπορεί να ικανοποιηθεί με HTTPS. Το κακό όμως είναι ότι **απαιτείται Reverse HTTP proxy** (Εικόνα 28).



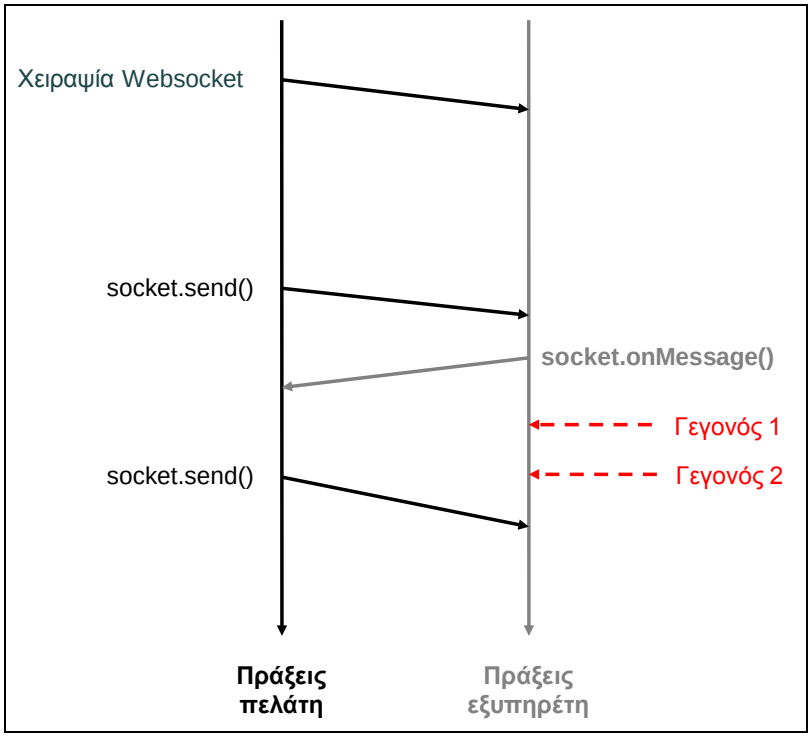
Εικόνα 28: Αρχιτεκτονική Reverse HTTP

4.6. WebSocket API (HTML5)

Οι προαναφερθείσες τεχνικές, που κάνουν χρήση Javascript και χρησιμοποιούνται κατά κόρον στις σημερινές υπηρεσίες, είναι ως επί το πλείστον μονόδρομες (π.χ. AJAX). Με λίγα λόγια ο χρήστης μπορεί να στείλει το αίτημα (request) στο website ασύγχρονα, αλλά όχι το αντίστροφο. Οι τεχνικές αυτές, αν και έχει αποδειχτεί ότι λειτουργούν στην πράξη, δεν μπορούν να χαρακτηριστούν ως «πραγματικού χρόνου» καθώς βασίζονται είτε σε συνεχή ερωτήματα από τον χρήστη στο website (polling) ή στην χρήση ενδιάμεσων λύσεων όπως το long polling ή το streaming. Στην ουσία καταναλώνουν πόρους ή δεσμεύουν bandwidth χωρίς λόγο. Ήταν καιρός λοιπόν να δημιουργηθεί ένα νέο κανάλι επικοινωνίας.

Το *WebSockets API* [9], που αποτελεί **μέρος της HTML5**, είναι μια πιο πρόσφατη τεχνική. Το WebSockets API **επιτρέπει στον εξυπηρέτη και τον πελάτη να επικοινωνούν πάνω από μια αμφίδρομη TCP σύνδεση πάνω από ένα socket**. Η προδιαγραφή του WebSockets API περιλαμβάνει ένα νέο αντικείμενο, το `<WebSocket>`.

Το χρονοδιάγραμμα στην Εικόνα 29 δείχνει πώς γίνεται η επικοινωνία όταν χρησιμοποιούνται WebSocket ενώ στην Εικόνα 8 παρουσιάζεται η αρχιτεκτονική του WebSockets API.



Εικόνα 29: Χρονοδιάγραμμα WebSocket

Η σύνδεση ανοίγει μέσω μιας αίτησης HTTP, η οποία ονομάζεται *WebSockets handshake* (χειραψία), με τη χρήση κάποιων ειδικών κεφαλίδων. Ουσιαστικά μέσω αυτής της αρχικής χειραψίας γίνεται **αναβάθμιση από το HTTP πρωτόκολλο στο WebSocket**. Στη συνέχεια η σύνδεση μένει ανοικτή και μπορούν να σταλούν και να ληφθούν δεδομένα σε Javascript σαν να χρησιμοποιούνταν ένα TCP socket. Η λήψη των δεδομένων γίνεται μέσω ενός *χειριστή συμβάντων* (event handler).

Αν ο πελάτης υποστηρίζει WebSocket, τότε θα χρησιμοποιήσει τον παρακάτω κώδικα JavaScript:

```
//Checking for browser support
if (window.WebSocket) {
document.getElementById("support").innerHTML = "HTML5 WebSocket is supported";
} else {
document.getElementById("support").innerHTML = "HTML5 WebSocket is not supported";
}


//Create new WebSocket
var mySocket = new WebSocket(“ws://www.websocket.org”);


// Associate listeners
mySocket.onopen = function(evt) {
  alert(“Connection open…”);
};
mySocket.onerror = function(evt) {
    ...
};
mySocket.onmessage = function(evt) {
  alert(“Received message: “ + evt.data);
};
mySocket.onclose = function(evt) {
  alert(“Connection closed…”);
};
```

Το `www.websocket.org` είναι το URL του εξυπηρέτη με την θύρα στην οποία «ακούει» τις συνδέσεις τύπου Web Socket. Το πρόθεμα `ws://` δείχνει πως πρόκειται για μια σύνδεση σε Web Socket.

Καλείται σε περίπτωση σφάλματος, π.χ. όταν κοπεί η σύνδεση

Καλείται όταν στείλει κάποιο μήνυμα ο εξυπηρέτης. Το evt.data περιέχει αυτό το μήνυμα.

```
// Sending data
mySocket.send(“Hello server! HTML5 WebSocket Rocks!”);

//Close WebSocket
mySocket.close();
```

Όταν δημιουργείται το αντικείμενο WebSocket, τότε γίνεται η αρχική χειραψία με την παρακάτω HTTP αίτηση από την πελάτη.

```
GET /text HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: www.websocket.org
Origin: http:// websocket.org
WebSocket-Protocol: sample
…\r\n
```

Αν ο εξυπηρέτης υποστηρίζει WebSocket θα απαντήσει:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: www.websocket.org
WebSocket-Location: ws:// websocket.org/demo
WebSocket-Protocol: sample
…\r\n
```

Δεν υπάρχει ακόμη κάποιο πρότυπο (specification) Java για την υποστήριξη του WebSocket. Οπότε, στην πλευρά του εξυπηρέτη, για να χρησιμοποιηθούν τα χαρακτηριστικά του web container (π.χ. του Tomcat ή του Jetty) για το WebSocket, θα πρέπει ο κώδικας της εφαρμογής να συνδέεται στενά με την αντίστοιχη βιβλιοθήκη του container. Αν χρησιμοποιούσαμε για παράδειγμα ένα Jetty container, τότε θα έπρεπε να χρησιμοποιήσουμε το WebSocket API του Jetty. Στο Jetty υπάρχουν διάφοροι τρόποι χειρισμού της χειραψίας WebSocket. Ο πιο εύκολος τρόπος είναι η δημιουργία μιας υποκλάσης της WebSocketServlet και η υλοποίηση της μεθόδου `doWebSocketConnect`.

```
public final class ReverseAjaxServlet extends WebSocketServlet {
    @Override
    protected    WebSocket    doWebSocketConnect(HttpServletRequest
request,
                                            String protocol) {
        return [...]
    }
}
```

Σε αυτή τη μέθοδο επιστρέφεται ένα στιγμιότυπο του WebSocket interface. Οπότε, πρέπει να υλοποιηθεί το interface και να επιστραφεί κάποιου είδους `endpoint` το οποίο θα αντιπροσωπεύει τη WebSocket σύνδεση.

```
class Endpoint implements WebSocket {
    Outbound outbound;
    @Override
    public void onConnect(Outbound outbound) {
        this.outbound = outbound;
    }
    @Override
    public void onMessage(byte opcode, String data) {
        // called when a message is received
        // you usually use this method
    }
    @Override
    public void onFragment(boolean more, byte opcode,
                           byte[] data, int offset, int length) {
        // when a fragment is completed, onMessage is called.
        // Usually leave this method empty.
    }
    @Override
    public void onMessage(byte opcode, byte[] data,
                          int offset, int length) {
        onMessage(opcode, new String(data, offset, length));
    }
```

```
    @Override
    public void onDisconnect() {
        outbound = null;
    }
}
```

Για να στείλει κάποιο μήνυμα ο εξυπηρέτης στον πελάτη, θα γράψει στο `outbound`.

```
if (outbound != null && outbound.isOpen()) {
    outbound.sendMessage('Hello client!');
}
```

Για να κάνει αποσύνδεση του πελάτη και να κλείσει τη σύνδεση WebSocket, καλεί την `outbound.disconnect()`.

Αφού γίνει η σύνδεση, μπορούν να στέλνονται πλαίσια δεδομένων WebSocket και προς τις δύο κατευθύνσεις, παράλληλα. Κάθε πλαίσιο κειμένου ξεκινά με το byte 0x00, τελειώνει με το byte 0xFF και περιέχει UTF-8 δεδομένα στο ενδιάμεσο (π.χ. `\x00Hello, WebSocket\0xff`). Τα δυαδικά πλαίσια από την άλλη χρησιμοποιούν ένα πρόθεμα με το μήκος τους. Αυτό σημαίνει πολύ **μικρό overhead** σε σχέση με το HTTP. Αν και το πρωτόκολλο μπορεί να υποστηρίξει διάφορα είδη πελατών (π.χ. JavaScript, Adobe Flex, JavaFX, Microsoft Silverlight), η προδιαγραφή του HTML5 ορίζει υποστήριξη μόνο για JavaScript. Η JavaScript, όμως, δεν υποστηρίζει τον τύπο δεδομένων byte και έτσι δεν μπορούν να σταλούν raw δυαδικά δεδομένα.

Ως προς τα **τείχη προστασίας** (firewalls) και τους **εξυπηρέτες μεσολάβησης** (proxy servers) δεν υπάρχει κάποιο πρόβλημα. Το WebSocket κάνει μια χειραψία συμβατή με το HTTP και έτσι οι εξυπηρέτες HTTP μπορούν να μοιραστούν τις θύρες τους (80 και 443) με έναν εξυπηρέτη WebSocket. Όταν το WebSocket εντοπίσει κάποιον proxy, τότε δημιουργεί αυτόματα μία δίοδο για να περάσει. Η δίοδος εγκαθιδρύεται με μια HTTP CONNECT εντολή στον proxy, η οποία ζητά να ανοίξει μια TCP/IP σύνδεση σε ένα συγκεκριμένο υπολογιστή (host) και port. Με την εγκατάσταση της διόδου, η επικοινωνία μπορεί να ρεύσει ανεμπόδιστα. Το HTTP/S δουλεύει παρόμοια, οπότε η υποστήριξη για SSL είναι έμφυτη.

Αν και, όπως και τα SSE, τα WebSocket έχουν μικρό overhead, ωστόσο δεν κάνουν πολλά πράγματα για την **αξιοπιστία** (χειρισμός επανασύνδεσης, συγχρονισμός μηνυμάτων και εξασφάλιση αποστολής μηνύματος). Αυτή αφήνεται στο επίπεδο της εφαρμογής. Το τρέχον πρωτόκολλο του WebSocket αποτελεί μόνο ένα χαμηλού-επιπέδου κανάλι επικοινωνίας.

Στην περίπτωση που έχουμε ένα γεγονός close για παράδειγμα, θα πρέπει η εφαρμογή να είναι σε θέση να ξανανοίξει τη σύνδεση αν αυτό έγινε αναπάντεχα (idle timeout, σφάλμα δικτύου). Δεν υπάρχει διαθέσιμο κάποιο timeout στο Websocket και ακόμη και η αποστολή μηνυμάτων keep-alive δε θα βοηθήσει όταν υπάρχει σφάλμα δικτύου (συνδέσεις Wi-Fi και κινητές συσκευές). Ένας τρόπος για να μη χαθούν μηνύματα είναι η υλοποίηση ουρών και προς τις δύο κατευθύνσεις. Αυτές όμως δεν μπορούν να μεγαλώνουν επ' άπειρον. Πρέπει να υπάρχει κάποιο timeout και κάποιο σαφές μήνυμα κλεισίματος για τις περιπτώσεις που αυτό είναι ηθελημένο. Αν όμως πέσει ο server, τότε ο πελάτης θα προσπαθεί συνεχώς να επανασυνδεθεί. Γι' αυτό θα πρέπει να υπάρξει κάποιος αλγόριθμος που θα προσπαθεί όλο και λιγότερο συχνά.

Για να εξασφαλιστεί η ποιότητα της υπηρεσίας θα πρέπει να στέλνεται κάποιο μήνυμα αποδοχής (acknowledge) για κάθε συμβάν που λαμβάνεται. Με αυτό τον τρόπο ακόμη αν ένα μήνυμα δε ληφθεί εξαιτίας κατάστασης ανταγωνισμού (π.χ. στέλνεται το μήνυμα και αμέσως μετά πέφτει η σύνδεση), αυτό δε θα χαθεί, αλλά θα ξανασταλθεί.

Ένα άλλο πρόβλημα που μπορεί να προκύψει είναι ότι αν **το μέγεθος ενός μηνύματος** ξεπερνά τα όρια κάποιου πόρου στον πελάτη ή τον εξυπηρέτη, τότε η σύνδεση θα πέσει. Αυτό μοιάζει με παροδικό σφάλμα δικτύου… Η σύνδεση αποκαθίσταται και γίνεται προσπάθεια να ξανασταλθεί το ίδιο μήνυμα από την ουρά. Ατέρμων βρόχος! Το μέγιστο μέγεθος δεν ορίζεται, αλλά η JavaScript δεν επιτρέπει δεδομένα μεγαλύτερα από 4GB, οπότε αυτό είναι πρακτικά ένα μέγιστο.

Το Web Sockets είναι διαθέσιμο στο Google Chrome από την έκδοση 4.0.249.0, στον FireFox 4 και στο Safari 5 και υπάρχει μια JavaScript βιβλιοθήκη από το Kaazing που το εξομοιώνει.

4.7. FlashSocket

Όταν κάποιος φυλλομετρητής δεν υποστηρίζει WebSocket, μερικές βιβλιοθήκες έχουν τη δυνατότητα να πέφτουν σε *FlashSocket* (Socket μέσω Flash) [37]. Οι βιβλιοθήκες αυτές

συνήθως παρέχουν επισήμως το ίδιο WebSocket API, αλλά το εφαρμόζουν κάνοντας ανάθεση κλήσεων σε ένα κρυφό συστατικό Flash, το οποίο περιλαμβάνεται στην ιστοσελίδα. Για παράδειγμα, το *WebSocketJS* παρέχει αυτή τη λειτουργικότητα. Απαιτεί τουλάχιστον Flash 10 και προσφέρει υποστήριξη για επικοινωνία μέσω WebSocket στους φυλλομετρητές Firefox 3, Internet Explorer 8 και Internet Explorer 9.

Πίνακας 7: Υποστήριξη του WebSocket από τους φυλλομετρητές

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser
πριν 11 εκδόσεις			4.0						
πριν 10 εκδόσεις			5.0						
πριν 9 εκδόσεις			6.0						
πριν 8 εκδόσεις		2.0	7.0						
πριν 7 εκδόσεις		3.0	8.0		9.0				
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6				
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1				
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5				
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0	
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2
τωρινή έκδοση	9.0 plugin	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3, 3.0
σύντομα	9.0 plugin	9.0	16.0	5.1	11.6				4.0
μελλοντικά	10.0	10.0	17.0	6.0	12.0				

Σημείωση: Το πρωτόκολλο WebSocket οριστικοποιήθηκε πρόσφατα ως RFC 6455 και δεν υποστηρίζεται ακόμη από όλους τους φυλλομετρητές. Επίσης, κατά την ανάπτυξή του πέρασε από διάφορες μη συμβατές εκδόσεις και κάποιοι φυλλομετρητές υποστηρίζουν μόνο παλιότερες εκδόσεις (π.χ. το draft76 ή hixie-76). Η μερική υποστήριξη του WebSocket σχετίζεται με τη χρήση μιας παλαιότερη έκδοσης του πρωτοκόλλου ή/και την απενεργοποίηση του από προεπιλογή (λόγω θεμάτων ασφάλειας στο παλαιότερο πρωτόκολλο). Η Microsoft προς το παρόν πειραματίζεται με αυτή την τεχνολογία.

5. ΑΝΑΒΑΘΜΙΣΗ ΤΟΥ HTTP

Σήμερα, τα βασικά πρωτόκολλα του Διαδικτύου (όσον αφορά τις εφαρμογές που δημιουργούμε σε αυτό) μπορούμε να πούμε ότι είναι τα HTTP και TCP. Το TCP είναι πρωτόκολλο επιπέδου μεταφοράς και χαρακτηρίζεται από την αξιοπιστία του, ενώ το HTTP είναι ένα πρωτόκολλο επιπέδου εφαρμογής. Σήμερα γίνεται μια προσπάθεια βελτίωσης της καθυστέρησης στο πρωτόκολλο επιπέδου εφαρμογής, το HTTP.

Το HTTP δε σχεδιάστηκε με γνώμονα τις μικρές καθυστερήσεις, αλλά καθώς οι ανάγκες των εφαρμογών άλλαζαν, κάποια χαρακτηριστικά του HTTP χρήζουν βελτίωσης.

Αυτά είναι τα εξής:

1. *Μία αίτηση ανά σύνδεση*

   Αφού το HTTP μπορεί να φέρει έναν πόρο τη φορά, δεν μπορεί να χρησιμοποιηθεί το ίδιο TCP κανάλι για επιπλέον αιτήσεις. Οι φυλλομετρητές, για να παρακάμψουν αυτό το πρόβλημα, χρησιμοποιούν ταυτόχρονα πολλές συνδέσεις. Από το 2008 οι περισσότεροι από 2 ανοίγουν μέχρι 6 συνδέσεις ανά domain.

2. *Αιτήσεις οι οποίες ξεκινούν αποκλειστικά από τον πελάτη*

   Στο HTTP, όπως είδαμε, μόνο ο πελάτης μπορεί να ξεκινήσει μια αίτηση. Ο εξυπηρέτης δεν έχει κάποιο τρόπο να πάρει πρωτοβουλία και να στείλει κάποια ειδοποίηση.

3. *Αποσυμπίεστες κεφαλίδες αίτησης και απάντησης*

   Οι κεφαλίδες αίτησης ποικίλουν σε μέγεθος από περίπου 200 bytes σε πάνω από 2KB. Καθώς οι εφαρμογές χρησιμοποιούν όλο και περισσότερο τα cookies και οι πράκτορες χρήστη (user agent) επεκτείνουν τα χαρακτηριστικά τους, το τυπικό μέγεθος κεφαλίδας είναι της τάξης των 700-800 bytes. Στα μόντεμ ή τις ADSL συνδέσεις, όπου το εύρος ζώνης άνω ζεύξης (uplink) είναι αρκετά χαμηλό, η καθυστέρηση που επιβάλλεται μπορεί να είναι σημαντική. Η μείωση των δεδομένων στις κεφαλίδες θα μπορούσε να βελτιώσει άμεσα την καθυστέρηση στην αποστολή των αιτήσεων.

4. *Περιττές κεφαλίδες*

Αρκετές κεφαλίδες στέλνονται επανειλημμένα στείλει με αιτήματα πάνω από το ίδιο κανάλι. Ωστόσο, οι κεφαλίδες όπως το User-Agent, Host, και Accept είναι γενικά στατικές και δεν χρειάζεται να στέλνονται ξανά και ξανά.

5. *Προαιρετική συμπίεση δεδομένων*

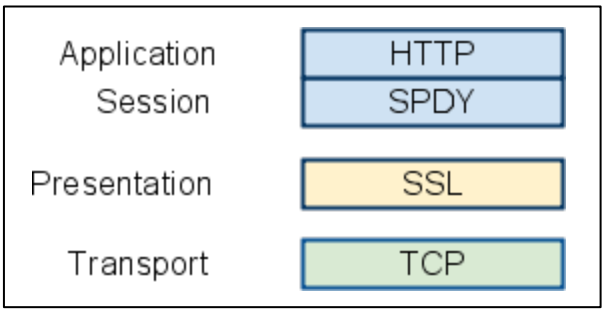
   Το HTTP κάνει προαιρετικά συμπίεση των δεδομένων. Το περιεχόμενο θα πρέπει πάντα να στέλνεται σε συμπιεσμένη μορφή.

Ασχέτως του πρωτοκόλλου μεταφοράς, τα προβλήματα του HTTP θα πρέπει να λυθούν. Αν και έχουν μελετηθεί κάποιες λύσεις [38] [39] [40] [41] [42], για πρακτικούς λόγους, είναι πολύ δύσκολο να πραγματοποιηθούν αλλαγές στο επίπεδο μεταφοράς,. Αυτό που είναι πραγματοποιήσιμο, είναι να αντιμετωπιστούν ένα προς ένα τα μειονεκτήματα του HTTP, μιας και οι αλλαγές στην υπάρχουσα υποδομή θα είναι πολύ μικρές.

5.1. SPDY

Το SPeeDY [43] είναι ένα πειραματικό πρωτόκολλο για ταχύτερο δίκτυο. Το υποκείμενο πρωτόκολλο μεταφοράς παραμένει το TCP. Το περιεχόμενο των ιστοσελίδων δε χρειάζεται να αλλάξει. Οι αλλαγές για να υποστηριχθεί το SPDY είναι στο φυλλομετρητή του χρήστη και στους εξυπηρέτες.

Το SPDY προσθέτει ένα **στρώμα συνόδου πάνω από το SSL**, το οποίο επιτρέπει **πολλαπλές ταυτόχρονες, εναλλασσόμενες ροές** πάνω από μια TCP σύνδεση. Το σύνηθες φορμάτ των μηνυμάτων HTTP GET και POST δεν αλλάζει, ωστόσο το SPDY ορίζει ένα **νέο φορμάτ για κωδικοποίηση και μετάδοση** των δεδομένων. Οι ροές είναι **διπλής κατεύθυνσης**, δηλαδή μπορεί να ξεκινήσει μια ροή από τον πελάτη και από τον εξυπηρέτη.



Εικόνα 30: Το SPDY μέσα στο OSI

Επιπλέον, το SPDY παρέχει ένα προηγμένο χαρακτηριστικό, **server-initiated streams**. Τα server-initiated streams μπορούν να χρησιμοποιηθούν για την αποστολή περιεχομένου στον πελάτη, χωρίς αυτός να χρειάζεται να τα ζητήσει. Αυτή η επιλογή μπορεί να ρυθμιστεί από τον προγραμματιστή της διαδικτυακής εφαρμογής με δύο τρόπους:

- *Server push (προώθηση εξυπηρέτη)*

  Το SPDY πειραματίζεται με μία επιλογή η οποία θα επιτρέπει στους εξυπηρέτες να ωθούν τα δεδομένα στους πελάτες μέσω της κεφαλίδας `X-Associated Content`. Αυτή η κεφαλίδα ενημερώνει τον πελάτη ότι ο διακομιστής ωθεί έναν πόρο στον πελάτη προτού τον ζητήσει ο πελάτης.

- Server hint (υπόδειξη εξυπηρέτη)

  Αντί να προωθεί τους πόρους αυτόματα στον πελάτη, ο εξυπηρέτης χρησιμοποιεί την κεφαλίδα `X-Subresources` για να υποδείξει στον πελάτη ότι πρέπει να ζητήσει συγκεκριμένους πόρους, σε περιπτώσεις όπου ο εξυπηρέτης γνωρίζει εκ των προτέρων τον πελάτη στον οποίο χρειάζονται αυτοί οι πόροι. Ο εξυπηρέτης θα περιμένει για την αίτηση του πελάτη πριν στείλει το περιεχόμενο.

Αυτά που υλοποιήθηκαν αρχικά είναι τα εξής:

- Ένας **εξυπηρέτης** [44] που επιστρέφει HTTP και SPDY απαντήσεις πάνω από TCP και SSL.
- Ένας τροποποιημένος **πελάτης** Google Chrome [45] που χρησιμοποιεί είτε HTTP ή SPDY πάνω από TCP και SSL.
- Μια **υποδομή για δοκιμές και συγκριτική αξιολόγηση** [46] η οποία επαληθεύει ότι οι σελίδες αναπαράγονται πιστά. Πιο συγκεκριμένα, επιβεβαιώνεται ότι το SPDY διατηρεί τις κεφαλίδες του εξυπηρέτη, τις κωδικοποιήσεις του περιεχομένου, τα URL κτλ.

Το πρωτόκολλο αυτό είναι ακόμη στην αρχή του και έχει πράγματα προς επίλυση. Ένα μειονέκτημά του, για παράδειγμα, είναι ότι η χρήση του SSL επιβάλει η ίδια καθυστέρηση και υπάρχουν προκλήσεις στην ανάπτυξη. Γίνονται πρόσθετα RTTs (round-trip time) για τη χειραψία του SSL, έχουμε κρυπτογράφηση, κάποια proxies κάνουν caching, οπότε χρειάζονται περισσότερες ρυθμίσεις στο SSL.

Πέρα από τη Google και την κοινότητα που δουλεύει πάνω στο SPDY, γίνονται προσπάθειες ενσωμάτωσής του σε διάφορα προϊόντα:

- *Σε εξυπηρέτες*

  Jetty, Apache, Node.js, NGINX. Ακόμη έχουν γίνει δύο υλοποιήσεις του SPDY server, μία σε Python και άλλη μία σε Ruby.

- *Σε φυλλομετρητές*

  το υποστηρίζουν οι Google Chrome και Mozilla Firefox και πειραματικά ο Opera.

- *Σε βιβλιοθήκες*

  έχουν γίνει υλοποιήσεις σε C, Java, Ruby, Go, Erlang και i-phone.

Οι διαδικτυακές υπηρεσίες που χρησιμοποιούν SPDY είναι κάποιες υπηρεσίες της Google (π.χ. Google search, Gmail και άλλες SSL-enabled services), το Twitter, το Cloudflare, το Facebook και το WordPress.com.

5.1.1. WebSocket ή SPDY

Το WebSockets και το SPDY είναι πανομοιότυπα σχεδόν σε όλα τα σημαντικά σημεία. Ο τρόπος που υλοποιούνται διαφέρει, αλλά το αποτέλεσμα μοιάζει. Οι ομοιότητές τους είναι οι εξής:

- ✓ Είναι και τα δύο ασύγχρονα, οπότε γλιτώνουμε από το polling.
- ✓ Και τα δύο χρησιμοποιούν μια σύνδεση TCP, οπότε μειώνεται το overhead στους εξυπηρέτες και την υποδομή του δικτύου, με αποτέλεσμα ο τελικός χρήστης να απολαμβάνει καλύτερη απόδοση.
- ✓ Και τα δύο μπορούν να κάνουν συμπίεση ώστε να μειωθούν τα δεδομένα που μεταφέρονται ιδίως στα δίκτυα της κινητής τηλεφωνίας, αν και το WebSocket το καταφέρνει με τη χρήση κάποιων extension.
- ✓ Και τα δύο πρωτόκολλα λειτουργούν «έξω» από το HTTP και χρησιμοποιούν ένα μηχανισμό αναβάθμισης πριν ξεκινήσουν. Το SPDY χρησιμοποιεί το Next Protocol Negotiation [47] (προτεινόμενη βελτίωση του TLS). Αυτός ο μηχανισμός δίνει καλύτερη συμβατότητα προς-τα-πίσω σε όλο τον ιστό, επιτρέποντας στις ιστοσελίδες να

υποστηρίζουν τις διαδικτυακές εφαρμογές επόμενης γενιάς, καθώς και το παραδοσιακό HTTP.

Όμως

- × Το framing του WebSocket έχει μικρότερο overhead από το SPDY.
- × Και η βασική διαφορά των δύο είναι πως το WebSockets δε χρησιμοποιεί κεφαλίδες, ενώ το SPDY το κάνει.

Αυτή η φαινομενικά μικρή διαφορά έχει αντιστρόφως ανάλογη επίπτωση επί της υποστήριξης των υποδομών και είναι ο λόγος που έχει ανοίξει μία συζήτηση ως προς το πιο είναι καλύτερο και θα επικρατήσει τελικά.

Τα antivirus και άλλες λύσεις σάρωσης κακόβουλου λογισμικού είναι πολύ καλά στο να εντοπίζουν ανωμαλίες σε συγκεκριμένους τύπους περιεχομένου. Το πρόβλημα είναι ότι χωρίς τον τύπο MIME, η ικανότητά τους να αναγνωρίζουν σωστά ένα συγκεκριμένο αντικείμενο γίνεται αμφίβολη. Αν και οι κεφαλίδες του HTTP μπορεί να λένε ψέματα, σε γενικές γραμμές η εφαρμογή που δίνει το αντικείμενο σπάνια ψεύδεται για το είδος των δεδομένων.

Δεν υπάρχει διεύθυνση URL που να σχετίζεται με τα αντικείμενα στο WebSocket και έτσι δεν μπορούμε να χρησιμοποιήσουμε το URL αντί για τις κεφαλίδες για να μάθουμε τον τύπο του περιεχομένου.

Το αποτέλεσμα είναι ότι τα λογισμικά ασφάλειας που έχουν σχεδιαστεί για να ψάχνουν συγκεκριμένες ανωμαλίες μέσα σε συγκεκριμένα είδη περιεχομένου, δεν έχουν τη δυνατότητα να το κάνουν πια. Δεν μπορούν να εξάγουν το αντικείμενο ενός WebSocket, μιας και δεν υπάρχει σήμανση της αρχής και του τέλους του.

Είναι όμως τόσο ίδια πια αυτά που προσφέρουν τα δύο πρωτόκολλα; Αν ήταν έτσι, τότε οι αλλαγές που επιβάλει η χρήση του WebSocket στις υποδομές (firewall, antivirus κτλ) θα έκαναν την επιλογή του απίθανη. Και τα δύο πρωτόκολλα ξεκίνησαν από τη Google. Είναι δυνατόν να δουλεύουν σε τόσες λύσεις ταυτόχρονα και να μην ξέρει η μία ομάδα τι κάνει η άλλη; Γιατί θα κάνανε το SPDY, τη στιγμή που το WebSocket μόλις έγινε πρότυπο;

Η απάντηση έγκειται στο **σκοπό** του κάθε πρωτοκόλλου.

- Το SPDY το χρησιμοποιούμε όταν θέλουμε παράδοση μιας σελίδας ή κάποιων αντικειμένων, δηλαδή στο κομμάτι αίτησης-απάντησης. Μιλάμε ακόμη σε επίπεδο browser και HTTP.
- Το WebSocket το χρησιμοποιούμε όταν θέλουμε ελαφριά και streaming παράδοση δεδομένων, δηλαδή επικοινωνία πραγματικού χρόνου. Πλέον μιλάμε σε επίπεδο εφαρμογής και javascript. Δεν έχει νόημα να «πειράξουμε» το HTTP για να μας δώσει ένα κανάλι δύο κατευθύνσεων.

Υπάρχει ένα draft υπό ανάπτυξη το οποίο εξηγεί πώς θα γίνει **layered** το WebSocket πάνω από SPDY. Αυτό που μπορούμε κατ' αρχήν να εκμεταλλευτούμε είναι η σύνδεση του SPDY. Και τα δύο πρωτόκολλα κρατούν με κάποιο τρόπο ανοικτές συνδέσεις, οπότε αν μια σελίδα τα χρησιμοποιεί και τα δύο, τότε θα χρειαστούν δύο συνδέσεις. Αν όμως έχουμε WebSocket πάνω από SPDY, τότε η σύνδεση θα είναι μία.

6. ΣΥΓΚΡΙΣΗ ΤΩΝ ΤΕΧΝΙΚΩΝ ΑΥΤΟΜΑΤΗΣ ΠΡΟΩΘΗΣΗΣ ΠΕΡΙΕΧΟΜΕΝΟΥ ΣΤΟ ΔΙΑΔΙΚΤΥΟ

Κάθε μια από τις λύσεις που προαναφέρθηκαν έχει τα δικά της πλεονεκτήματα και μειονεκτήματα. Αυτά έχουν να κάνουν με την πολυπλοκότητα, την ασφάλεια, την απόδοση, το scalability, τη συμβατότητα του φυλλομετρητή με τη Java και με περιορισμούς όπως τα τείχη προστασίας. Η βέλτιστη λύση εξαρτάται κυρίως από το στόχο της εφαρμογής.

Στη συνέχεια δίνονται κάποια πλεονεκτήματα και μειονεκτήματα των διάφορων λύσεων καθώς και ένας συγκριτικός πίνακας.

6.1. Μειονεκτήματα και πλεονεκτήματα κάθε τεχνικής

Κάθε τεχνική, από αυτές που αναφέρθηκαν στο προηγούμενο κεφάλαιο, έχει κάποια μειονεκτήματα και κάποια πλεονεκτήματα. Τα μειονεκτήματα μιας τεχνικής μπορεί να την κάνουν απαγορευτική για μια συγκεκριμένη διαδικτυακή εφαρμογή ενώ τα πλεονεκτήματά της μπορούν να την κάνουν μια καλή υποψήφια λύση. Στη συνέχεια, για κάθε τεχνική δίνονται επιγραμματικά τα μειονεκτήματα και τα πλεονεκτήματά της.

6.1.1. Short Polling

× Σε αυτή την πρώτη προσπάθεια παράδοσης πληροφορίας πραγματικού χρόνου στο Διαδίκτυο είχαμε **πολλές άχρηστες αιτήσεις**, αφού δεν μπορούμε να προβλέψουμε πότε θα είναι έτοιμα τα νέα μηνύματα.

× Δεν υποστηρίζει καλή χρονική συνοχή. Στην ιδανική περίπτωση η περίοδος ανάμεσα σε δύο ανανεώσεις θα έπρεπε να είναι μηδενική, πράγμα που θα σήμαινε στιγμιαίες ενημερώσεις. Εξαιτίας των καθυστερήσεων όμως, ο εξυπηρέτης πάντα θα υστερεί. Η **καθυστέρηση** αποτελεί πρόβλημα κυρίως στις εφαρμογές όπου ο χρήστης αλληλεπιδρά με πτητικά (volatile) δεδομένα από τον εξυπηρέτη. Για παράδειγμα, ο χρήστης θα μπορούσε να επεξεργάζεται ένα αντικείμενο χωρίς να γνωρίζει ότι κάποιος άλλος χρήστης το έχει ήδη διαγράψει.

× Η περιοδική ανανέωση έχει σημαντικό **κόστος**. Ακόμα και η μικρότερη αίτηση απαιτεί πόρους και στα δύο άκρα, μέχρι και το επίπεδο του λειτουργικού συστήματος. Ως προς

το δίκτυο, κάθε αίτηση συνεπάγεται κάποιο κόστος εύρους ζώνης, το οποίο μπορεί να αυξηθεί αν οι ανανεώσεις συμβαίνουν πολύ τακτικά.

× Χρησιμοποιείται σπάνια σε εφαρμογές αφού δεν εγγυάται **καθόλου καλή κλιμάκωση**.

× Όταν η περίοδος του polling είναι πολύ μικρή ή όταν το πλήθος των πελατών είναι μεγάλο ο εξυπηρέτης μοιάζει να δέχεται **επίθεση άρνησης υπηρεσίας** (Denial of Service attack - DoS).

✓ Είναι **εύκολο** στην υλοποίηση και δε χρειάζεται τίποτα το ιδιαίτερο από τον εξυπηρέτη.

✓ Δουλεύει σε **όλους τους φυλλομετρητές**.

6.1.1.1. META Refresh

× Το W3C αποθαρρύνει τη χρήση του META Refresh, καθώς οι μη αναμενόμενες ανανεώσεις σελίδας μπορούν να **αποσυντονίσουν** το χρήστη

✓ Χρησιμοποιείται ως σήμερα στις περιπτώσεις που θέλουμε να δώσουμε ενημερώσεις δυναμικών σελίδων **χωρίς να εξαρτόμαστε από τη Javascript**.

6.1.1.2. Reverse AJAX

× Τα δεδομένα που παρουσιάζονται με τεχνικές AJAX **δεν καταχωρούνται στις μηχανές αναζήτησης**.

✓ Οι αλλαγές του περιεχομένου μιας ιστοσελίδας με AJAX δεν καταγράφονται στο ιστορικό του browser, αφού δεν αλλάζει η σελίδα, και έτσι δεν μπορεί να χρησιμοποιηθεί το κουμπί «Πίσω» για να δούμε προηγούμενα δεδομένα.

6.1.1.3. Piggyback polling

× Ο πελάτης δεν μπορεί να ξέρει πότε συγκεντρώνονται τα γεγονότα στον εξυπηρέτη, αφού χρειάζεται μια πράξη (action) από το χρήστη για να ζητηθούν.

✓ Υπάρχει **μικρότερη κατανάλωση πόρων** αφού ο πελάτης ελέγχει πότε στέλνονται οι αιτήσεις.

6.1.2. Applet

× Τα applet είναι **πολύπλοκα** στην εφαρμογή.

× Είμαστε ακόμα μέσα σε ένα πρόγραμμα περιήγησης και, εκτός εάν το applet αποτελεί την πλήρη εφαρμογή πελάτη, είναι **δύσκολο να ενσωματώσουμε τις ενημερώσεις** που στέλνει ο εξυπηρέτης στον κώδικα HTML. Μια λύση είναι να ανανεώνουμε τη σελίδα καλώντας AppletContext.showDocument (URL) στη μέθοδο callback.

× Συχνά κάνουν τον πελάτη να είναι βαρύτερος στην εκκίνηση και **αυξάνουν την κατανάλωση πόρων**.

× Τα **plugin** που χρησιμοποιούν δεν περιλαμβάνονται εξ ορισμού σε όλους τους φυλλομετρητές.

× Έχουν περιορισμούς τείχους προστασίας. **Χρησιμοποιούν αφιερωμένες θύρες** οι οποίες μπορεί να μη λειτουργούν μέσω του τείχους προστασίας και οι περιορισμοί ασφαλείας του φυλλομετρητή μπορεί να αποτρέψουν callback ή την παραλαβή δεδομένων (data receipt) πάνω από UDP.

× Απαιτούν **πρόσθετη ανάπτυξη/συντήρηση του εξυπηρέτη**. Απαιτούν συχνά ένα αφιερωμένο προϊόν εξυπηρέτη ή τουλάχιστον ένα μητρώο το οποίο τρέχει RMI.

× **Επεκτασιμότητα** στα CORBA callback.

✓ Όταν οι χρήστες απαιτούν μια **άμεση αλληλεπίδραση με την κατάσταση**, όπως σε ένα διαμοιραζόμενο πίνακα, τα applet μπορεί να είναι μια ισχυρή τεχνική.

6.1.3. HTTP Server Push

× Η σύνδεση **καταναλώνει πόρους** στο εξυπηρέτη όσο είναι ανοιχτή. Αυτό μπορεί να αποδειχθεί δαπανηρό, ανάλογα με την ικανότητα (capacity) του εξυπηρέτη.

× Μία ενεργή σύνδεση μπορεί να εμποδίσει έναν πελάτη σε **κινητή συσκευή** να μπει σε κατάσταση αναστολής (sleep mode), έτσι δε θα γίνει εξοικονόμηση ενέργειας και θα μειωθεί ο χρόνος ζωής της μπαταρίας.

× Εξακολουθούν να υπάρχουν πολλά ερωτήματα όσον αφορά την εφικτότητα και την **επεκτασιμότητα** (scalability). Οι εξυπηρέτες δεν μπορούν να τα βγάλουν πέρα με πολλές ταυτόχρονες συνδέσεις. Η ταυτόχρονη εκτέλεση πολλών script, που το καθένα τρέχει στη δική του διαδικασία, βλάπτει το σύστημα. Ακόμη και αν ο εξυπηρέτης είναι πολυνηματικός, οι πόροι θα είναι περιορισμένοι.

- Οι streaming συνδέσεις είναι λιγότερο ανεκτικές σε εξυπηρέτες μεσολάβησης και τείχη προστασίας. Μερικές φορές, ένας **proxy** ο οποίος βρίσκεται μεταξύ του server και του browser θα κάνει **buffer τις απαντήσεις**, πράγμα που θα εμποδίζει τα δεδομένα να ρέουν στο φυλλομετρητή σε πραγματικό χρόνο. Αυτό μπορεί να παρακαμφθεί με τη χρήση HTTPS συνδέσεων οι οποίες περνούν διαφανώς από τους περισσότερους proxy [48]. **Το HTTPS αυξάνει όμως των επιβάρυνση στο πρωτόκολλο και στους υπολογιστικές ανάγκες** σε σχέση με το HTTP στο εξυπηρέτη και στον πελάτη. Ωστόσο, αν και το τρέχον HTTP RFC (RFC 2616 Hypertext Transfer Protocol - HTTP/1.1) δεν απαιτεί άμεση διαβίβαση των μερικών απαντήσεων, οι μεσολαβητές αποφεύγουν την αποθήκευση μεγάλου όγκου παραγόμενων δεδομένων, ώστε να ελαχιστοποιήσουν τη χρήση της μνήμης τους.
- Οι HTTP **proxy**, μέσω των οποίων συνδέονται οι περισσότεροι χρήστες, έχουν περιορισμένη μνήμη και θύρες TCP. Προκειμένου να εξυπηρετήσουν περισσότερους χρήστες, λοιπόν, ανακυκλώνουν συστηματικά τους αδρανείς πόρους. ‘Ετσι, κάποιοι εξυπηρέτες μεσολάβησης (proxy server) **ρίχνουν τη σύνδεση** όταν αυτή μένει αδρανής για κάποιο χρονικό διάστημα.
- Μια σύνδεση μεγάλης διάρκειας αναπόφευκτα κάποια στιγμή θα πέσει, οπότε θα πρέπει να υπάρχει κάποιο **σχέδιο αποκατάστασης**.
- Πρέπει να αποφασιστεί για πόσο θα μείνει ανοικτή η σύνδεση, πότε θα κλείνει η σύνδεση και πώς θα κάνει διάκριση ο φυλλομετρητής μεταξύ των μηνυμάτων.
- Οι φυλλομετρητές οριοθετούν το πλήθος των συνδέσεων ανά εξυπηρέτη.
- **Περίπλοκο** στην εφαρμογή.
- **Το Comet Streaming απαιτεί μια καλή βιβλιοθήκη** για το χειρισμό της επανασύνδεσης, του timeout, των αιτήσεων Ajax, των acknowledgments και των εναλλακτικών μεταφορών (Ajax long polling and jsonp short polling).
- Καμία από τις ροές μεταφοράς δε δουλεύει σε όλους τους σύγχρονους φυλλομετρητές χωρίς να προκαλεί αρνητικές παρενέργειες – αναγκάζοντας αυτούς που αναπτύσσουν κώδικα (developers) να υλοποιούν διάφορες **σύνθετες ροές μεταφοράς**, εναλλάσσοντάς τες ανάλογα το **φυλλομετρητή**.

- ✓ **Ο εξυπηρέτης έχει απόλυτο έλεγχο** στο πότε και πόσο συχνά στέλνονται νέα δεδομένα στον πελάτη.
- ✓ Μπορεί να είναι πιο αποτελεσματική από το polling, δεδομένου ότι δε χρειάζεται να ανοίγονται νέες συνδέσεις HTTP σε κάθε ανανέωση (**χαμηλότερη καθυστέρηση**).

6.1.3.1. CGI

- × Όπως και στο MIME, δε χρησιμοποιείται JavaScript. Ο εξυπηρέτης μπορεί να κάνει **μόνο προσθέσεις στη σελίδα** και όχι αλλαγές των στοιχείων της, έτσι, ο φυλλομετρητής και ο εξυπηρέτης είναι στενά συνδεδεμένοι και είναι δύσκολο να γραφεί μια πλούσια διαδικτυακή εφαρμογή.
- × Το υποστήριζε μόνο ο Netscape Navigator και όχι ο IE. Πλέον έχει εκλείψει.

6.1.3.2. MIME

- × Οι **Internet Explorer, Opera και Safari iOS** δεν υποστηρίζουν αυτή τη λειτουργία.
- × Χρησιμοποιείται για απλές ανανεώσεις (π.χ. γραφικών) και **όχι για πλούσιες διαδικτυακές εφαρμογές**.

6.1.3.3. Hidden Iframe (COMET)

- × Έλλειψη μιας αξιόπιστης μεθόδου **διαχείρισης σφαλμάτων**.
- × Αδυναμία εύρεσης της κατάστασης (**state**) της διαδικασίας που κάνει το αίτημα.
- × Κατάλληλη λύση **για εφαρμογές μικρής κλίμακας** (small-scale).
- × Τα στοιχεία Iframe **αργούν** πολύ να φορτώσουν.
- × Τα Iframe μοιράζονται την ίδια δεξαμενή συνδέσεων (connection pool) με το γονέα. Αυτό μειώνει σημαντικά το πλήθος άλλων συναλλαγών δεδομένων που μπορούν να εκτελεστούν στην ίδια σελίδα. Το **όριο των συνδέσεων ανά εξυπηρέτη** είναι 2 στον IE 7 και 6 στον IE 8.
- ✓ Στο hidden IFrame γίνεται **πραγματική προώθηση** αφού η τεχνική δε βασίζεται σε polling, στο XMLHttpRequest ή στον ορισμό κάποιας πηγής δυναμικού script.
- ✓ Δουλεύει **σε κάθε κοινό φυλλομετρητή**.

✓ Σχετικά **εύκολο** στην υλοποίηση.

✓ Στην περίπτωση που ο εξυπηρέτης χρησιμοποιεί php script, η ιστοσελίδα δε θα κάνει καλή **κλιμάκωση** ως προς τις παράλληλες συνδέσεις. Τα php script τρέχουν σε διαφορετικό instance για κάθε διαφορετική αίτηση πελάτη και έτσι η υλοποίηση πιθανώς να χρησιμοποιεί εντατικά τους πόρους του εξυπηρέτη. Κάποιες άλλες γλώσσες όπως η Python και τα Java Servlets είναι σχεδιασμένα για να διαχειρίζονται εκατοντάδες ή και χιλιάδες ταυτόχρονες συνδέσεις, οπότε αποτελούν καλύτερη επιλογή όταν έχουμε υψηλή χρήση (utilization) της σελίδας.

6.1.3.4. Multipart XHR (COMET)

× Υπάρχουν **άσχημες επιπλοκές στη μνήμη**, επειδή συσσωρεύονται πολλά script και ο φυλλομετρητής πρέπει να τα κρατήσει όλα αυτά στο μοντέλο της σελίδας του. Σε μια πλούσια εφαρμογή με πολλές ανανεώσεις το μοντέλο θα μεγαλώσει πολύ γρήγορα και κάποια στιγμή θα είναι απαραίτητη η ανανέωση της σελίδας προκειμένου να αποφευχθεί η χρήση του σκληρού δίσκου μαζί με τη RAM (hard drive swapping) ή κάτι ακόμη χειρότερο. Αυτό βέβαια ισχύει για κάθε τεχνική όπου συσσωρεύονται (java)script.

× Κάποιες ευρέως χρησιμοποιούμενες βιβλιοθήκες, όπως η CometD, αναφέρουν θέματα **buffering**. Για παράδειγμα, τα κομμάτια των δεδομένων (chunk) ενδέχεται να μπαίνουν σε buffer και να στέλνονται μόνο όταν η σύνδεση έχει ολοκληρωθεί ή όταν ο buffer είναι πλήρης. Αυτό μπορεί να δημιουργήσει μεγαλύτερη καθυστέρηση από την αναμενόμενη.

× Υλοποιείται μόνο στους **Gecko-based φυλλομετρητές** όπως ο Firefox.

✓ **Πιο αξιόπιστο** από το Hidden Iframe.

6.1.3.5. Java Pushlet

× Απαιτείται κάποια προσπάθεια για τη δημιουργία **cross-browser DHTML βιβλιοθηκών** οι οποίες λειτουργούν σε όλες τις εκδόσεις των προγραμμάτων περιήγησης και σε όλες τις πλατφόρμες.

× Ως προς την **επεκτασιμότητα**, όταν εκατοντάδες πελάτες συνδέονται μέσω pushlet, μπορεί να λαμβάνουν υπερβολική μερίδα των πόρων (π.χ. νήματα και socket). Σε αυτή

την κλίμακα θα ήταν καλύτερα να σερβίρονται pushlet από ένα **αφιερωμένο σε servlet εξυπηρέτη**, βελτιστοποιημένο για αυτό το είδος χρήσης του HTTP.

× Συνήθως οι εξυπηρέτες δεν είναι σχεδιασμένοι για **συνδέσεις μακράς διάρκειας** ζωής. Μία λύση, για να ξεπεραστεί αυτός ο περιορισμός, είναι η ίδια με εκείνη που εφαρμόζεται για την επεκτασιμότητα. Για τις εφαρμογές μεγάλης κλίμακας μπορεί να χρησιμοποιηθεί ένας πελάτης pushlet ο οποίος λαμβάνει **ειδοποιήσεις σε UDP μηνύματα**. Τότε το αίτημα HTTP χρησιμοποιείται μόνο για την εγγραφή.

× Κάποιοι εξυπηρέτες μεσολάβησης (proxy server) μπορεί να αποθηκεύουν δεδομένα HTTP, έχουμε δηλαδή και εδώ **proxy buffering**.

× Ο εξυπηρέτης **δεν παίρνει άμεσα επιβεβαίωση** ότι το γεγονός (event) υποβλήθηκε επιτυχώς σε επεξεργασία και έτσι δεν μπορεί να υπάρχει εγγύηση της ποιότητας των παρεχόμενων υπηρεσιών.

× Ο εξυπηρέτης δεν μπορεί να ελέγξει πότε θα κάνει ο φυλλομετρητής **timeout** (λήξη χρόνου) και για αυτό θα πρέπει να γίνει **ανανέωση της σελίδας στο φυλλομετρητή** όταν αυτό συμβεί.

✓ **Άμεση ενοποίηση με την DHTML στο φυλλομετρητή**: τα δεδομένα, τα οποία παράγονται από τον εξυπηρέτη, μπορούν να ενσωματωθούν αμέσως στο περιεχόμενο της σελίδας του φυλλομετρητή. Ως εκ τούτου, όλες οι δυνατότητες διάταξης της DHTML μπορούν να εφαρμοστούν άμεσα.

✓ **Standard HTTP θύρα και πρωτόκολλα**.

✓ Υπάρχει **χαμηλή επιβάρυνση στον πελάτη και στο πρωτόκολλο**.

✓ **Δεν χρειάζεται κάποιος ειδικός εξυπηρέτης**. Θεωρητικά τα Pushlet μπορούν να τρέξουν σε οποιαδήποτε μηχανή servlet, χρησιμοποιώντας λειτουργίες όπως η διαχείριση της σύνδεσης και ο πολυνηματισμός (που μπορεί να είναι ένα μειονέκτημα, όπως φαίνεται παραπάνω…).

✓ **Δεν αντιμετωπίζει πρόβλημα από τα τείχη προστασίας**.

6.1.3.6. Server-Sent Events (HTML5)

× Το SSE υποστηρίζει ένα **μονής κατεύθυνσης** κανάλι από τον εξυπηρέτη προς τον πελάτη.

✓ Το SSE είναι πολύ **απλό**. Για παράδειγμα, η ροή γεγονότων μπορεί να δοκιμαστεί με τη χρήση telnet. **Δε χρειάζεται να υλοποιηθούν πρωτόκολλα χειραψίας**. Η απλή αποστολή μιας HTTP GET αίτησης αρκεί για να ξεκινήσει η ροή γεγονότων.

✓ **Η σύνδεση γίνεται γρηγορότερα** απ' ότι στο WebSocket (δεν γίνονται χειραψίες και διαπραγματεύσεις) και έχει **λιγότερο overhead** (τα μηνύματα δε σπάνε σε πλαίσια ή πακέτα).

✓ Δουλεύει με τους στάνταρ **πληρεξούσιους** (nginx, Varnish κτλ) και τα **τείχη προστασίας**, αφού είναι συμβατό με την HTTP.

✓ Αποτελεί **πρότυπο της HTML5** και άρα στο μέλλον θα υποστηρίζεται από όλα HTML5-συμβατά προγράμματα περιήγησης.

✓ Ακόμη και αν κάποιοι φυλλομετρητές δεν το υποστηρίζουν, αφού δε χρησιμοποιείται κάποιο ειδικό πρωτόκολλο (όπως στο websocket) μπορούμε να χρησιμοποιήσουμε κάποιο **polyfill** [49] με σκέτη JavaScript

✓ Οι φυλλομετρητές οριοθετούν το πλήθος των συνδέσεων ανά εξυπηρέτη, οπότε για να μην υπερβούμε αυτό το νούμερο προτείνονται από το πρωτόκολλο δύο λύσεις.

✓ Η προδιαγραφή του SSE συνιστά την **περιοδική αποστολή του γεγονότος-σχολίου «keep-alive»** όταν δεν υπάρχουν δεδομένα προς αποστολή. Κάποιοι εξυπηρέτες μεσολάβησης (proxy server) ρίχνουν τη σύνδεση όταν αυτή μένει αδρανής για κάποιο χρονικό διάστημα, έτσι ώστε να μη σπαταλούν συνδέσεις. Η αποστολή ενός γεγονότος-σχολίου απενεργοποιεί αυτήν τη συμπεριφορά. Αν και το `EventSource` μπορεί και **αποκαθιστά τη σύνδεση αυτόματα**, με την περιοδική αποστολή σχολίων αποφεύγεται η άσκοπη επανασύνδεση.

✓ Αν ο server πιεστεί ως προς το πλήθος των συνδέσεων (**server load**), τότε μπορεί απλά να διακόψει κάποιες συνδέσεις. Αυτοί οι πελάτες δε θα έχουν πρόβλημα αφού θα ξανασυνδεθούν σε κάποια δευτερόλεπτα.

- ✓ Σε γενικές γραμμές, οι φυλλομετρητές οριοθετούν το πλήθος των συνδέσεων ανά εξυπηρέτη. Σε κάποιες περιπτώσεις, **η φόρτωση πολλών σελίδων που περιλαμβάνουν EventSource από τον ίδιο τομέα (domain) μπορεί να οδηγήσουν αντίστοιχα σε πολλές συνδέσεις**. Τότε, όπως είναι αναμενόμενο, ο μέγιστος αριθμός των συνδέσεων σύντομα υπερβαίνεται. Μια λύση στο πρόβλημα αποτελεί ένας **διαμοιραζόμενος WebWorker**, ο οποίος θα μοιράζει ένα μοναδικό αντικείμενο `EventSource`. Μια δεύτερη λύση είναι η **επαναχρησιμοποίηση μιας υπάρχουσας σύνδεσης**, αν το υπάρχον απόλυτο URL ταυτίζεται με αυτό που ζητείται. Σε αυτή την περίπτωση, την κατανομή των συνδέσεων θα πρέπει να διαχειριστεί η συγκεκριμένη υλοποίηση του `EventSource` στο φυλλομετρητή.
- ✓ Αν και το SSE έχει λιγότερη λειτουργικότητα από τα πρωτόκολλα του Comet, έχει τη δυνατότητα να καταστεί το κυρίαρχο πρωτόκολλο στις περιπτώσεις όπου απαιτείται απλά ένα μονής κατεύθυνσης κανάλι προώθησης από τον εξυπηρέτη.

6.1.4. Long Polling (COMET)

- × Η συντήρηση και ο συντονισμός αυτών των δύο συνδέσεων εισάγει **ακόμη μεγαλύτερη κατανάλωση πόρων και αυξάνει την πολυπλοκότητα.** Η αμφίδρομη επικοινωνία είναι **επιρρεπής σε σφάλματα, αυξάνει την καθυστέρηση, το network traffic και την απόδοση της CPU.**
- ✓ Πιο **εύκολο** να υλοποιηθεί στην πλευρά του φυλλομετρητή.
- ✓ Δουλεύει, το ελάχιστον, **σε κάθε φυλλομετρητή που υποστηρίζει XHR**. Το μόνο που κάνει είναι να χρησιμοποιεί το αντικείμενο XMLHttpRequest εκδίδοντας (issue) μια απλή αίτηση Ajax.
- ✓ **Οι συνδέσεις δεν είναι μόνιμες** και έτσι μπορεί να υποστηρίξει πολλούς ταυτόχρονους πελάτες.
- ✓ Είναι ιδανικό για εφαρμογές οι οποίες χρειάζονται **συχνές ενημερώσεις** όπως το chat. Άλλες εφαρμογές όπου μπορεί να βρει χρήση είναι αυτές των ηλεκτρονικών αγορών, όπου θέλουμε η διαθεσιμότητα και η τιμή του προϊόντος να μένει ενημερωμένη.
- ✓ Χάρη στα πρωτόκολλα του Comet προσομοιώνεται η **πλήρως αμφίδρομη επικοινωνία** χρησιμοποιώντας δύο HTTP συνδέσεις.

6.1.4.1. XHR

× Προεπιλεγμένα, το XMLHttpRequest **δεν επιτρέπει αιτήσεις σε άλλα domain** ή σε sub-domain.

× Όπως και οι άλλες τεχνικές, που έχουμε αναφέρει, βασίζεται σε μια **stateless HTTP σύνδεση**. Για αυτό το λόγο απαιτεί ειδικά χαρακτηριστικά από τον εξυπηρέτη για να μπορεί να αναστείλει προσωρινά τη σύνδεση.

✓ Είναι **αξιόπιστη** αφού έχει **καλό σύστημα χειρισμού σφαλμάτων** και **διαχείρισης των timeout**.

✓ Επιτρέπει το **συγχρονισμό** (round-trip) των συνδέσεων στον εξυπηρέτη, αφού αυτές δεν είναι μόνιμες (persistent).

6.1.4.2. Script Tag

× Όπως και στην τεχνική του Iframe, **δεν υπάρχει διαχείριση σφαλμάτων**, δεν μπορείς να ξέρεις σε ποια κατάσταση είσαι και δεν έχεις τη δυνατότητα να διακόψεις μια σύνδεση.

× Προκύπτουν **θέματα ασφάλειας** λόγω του cross-domain scripting.

✓ Είναι πολύ **εύκολο** στην εφαρμογή επειδή στηρίζεται σε HTML tag.

✓ Δουλεύει σε διαφορετικά domain (**cross-domain scripting**).

6.1.5. WebSockets API (HTML5)

× Το υποστηρίζουν πολλοί **φυλλομετρητές**, όπως οι Firefox, Google Chrome και Safari, αλλά **όχι όλοι**.

× Έχει μόνιμες (long-lived) συνδέσεις, όπως το HTTP Streaming. Αυτό σημαίνει πως οι εξυπηρέτες μεσολάβησης μπορεί να κάνουν **buffer** μη κωδικοποιημένες HTTP απαντήσεις, με αποτέλεσμα να εισάγουν απρόβλεπτη καθυστέρηση κατά τη διάρκεια του streaming της απάντησης (συμβατότητα δικτύου).

× **Η σύνδεση παραμένει ανοικτή** ακόμη και όταν δε μεταφέρονται δεδομένα.

× Ο εξυπηρέτης πρέπει να είναι πρόθυμος να δεχτεί **αφιερωμένη καταχώρηση μιας θύρας TCP/IP για κάθε ξεχωριστή σύνδεση** (δηλ. για κάθε στιγμιότυπό του που

χρησιμοποιείται), το οποίο μπορεί να είναι θέμα για τους εξυπηρέτες με περιορισμένο αριθμό θυρών TCP/IP.

× Αφού έχει μόνιμες συνδέσεις, δεν μπορεί να υποστηρίξει πολλούς χρήστες την ίδια στιγμή. Υπάρχει, δηλαδή, πρόβλημα στην **κλιμακωσιμότητα**.

× Το τρέχον πρωτόκολλο του WebSocket αποτελεί μόνο ένα **χαμηλού-επιπέδου κανάλι επικοινωνίας**. Ο χειρισμός επανασύνδεσης, ο συγχρονισμός των μηνυμάτων και η εξασφάλιση αποστολής μηνύματος, δηλαδή η αξιοπιστία, αφήνεται στο επίπεδο της εφαρμογής.

× Αν το **μέγεθος ενός μηνύματος** ξεπεράσει τα **όρια** κάποιου πόρου στον πελάτη ή τον εξυπηρέτη, τότε η σύνδεση θα πέσει. Αυτό μοιάζει με παροδικό σφάλμα δικτύου. Η σύνδεση αποκαθίσταται και γίνεται προσπάθεια να ξανασταλθεί το ίδιο μήνυμα από την ουρά. Ατέρμων βρόχος! Το μέγιστο μέγεθος δεν ορίζεται, αλλά η JavaScript δεν επιτρέπει δεδομένα μεγαλύτερα από 4GB, οπότε αυτό είναι πρακτικά ένα μέγιστο.

× Μόνο ο εξυπηρέτης ελέγχει τη ροή της πληροφορίας. Ως εκ τούτου, το μόνο που μπορεί να κάνει ο πελάτης είναι να ελέγχει τι παρουσιάζει στο χρήστη.

× **Δεν υποστηρίζει εύκολα υπηρεσίες βασισμένες στις αιτήσεις** αφού χρησιμοποιεί TCP socket και όχι HTTP request [50].

✓ Δεν έχει τα μειονεκτήματα του Comet streaming.

✓ Το API είναι πολύ **εύκολο** στη χρήση και δε χρειάζεται επιπρόσθετα layers.

✓ Επιτρέπει **αμφίδρομη επικοινωνία**.

✓ Έχει **χαμηλή καθυστέρηση** (low-latency).

✓ **Εύκολος χειρισμός των λαθών**.

✓ **Δε γίνονται πολλές συνδέσεις** όπως στο polling.

✓ Έχει **μικρό overhead**. Στο COMET για να πετύχουμε αμφίδρομη επικοινωνία χρειάζονταν πολλά επιπλέον δεδομένα στις κεφαλίδες. Τώρα που η σύνδεση είναι αμφίδρομη, τα KB άχρηστων κεφαλίδων αντικαθίστανται από 2-8 byte. Αυτό σημαίνει **λιγότερη επιβάρυνση στο δίκτυο**.

✓ **Διασχίζει τα τείχη προστασίας**.

✓ Ως προς τους πληρεξούσιους εξυπηρέτες, αν δεν υποστηρίζουν το “upgrade”, τότε μπορεί να χρησιμοποιηθεί το TLS.

6.1.6. FlashSocket

× Απαιτείται η εγκατάσταση του **Flash plugin**. Συνήθως, βέβαια, το διαθέτει ο φυλλομετρητής.

× **Χρειάζεται η θύρα 843 του τείχους προστασίας να είναι ανοιχτή**, έτσι ώστε να μπορέσει το στοιχείο (component) Flash να κάνει μια αίτηση HTTP για **να ανακτήσει ένα αρχείο πολιτικής ασφαλείας** το οποίο περιέχει τον κώδικα εξουσιοδότησης του domain. Εάν η θύρα 843 δεν είναι ανοιχτή, τότε θα πρέπει η βιβλιοθήκη να πάει σε άλλη τεχνική ή να επιστρέψει λάθος. Όλη αυτή **η επεξεργασία παίρνει χρόνο** (μέχρι 3 δευτερόλεπτα, ανάλογα με τη βιβλιοθήκη), και άρα επιβραδύνει την ιστοσελίδα.

× Αν ο πελάτης βρίσκεται πίσω από ένα **εξυπηρέτη μεσολάβησης**, τότε μπορεί να μη γίνει αποδοχή της σύνδεσης στη θύρα 843.

✓ Τα FlashSocket **παρέχουν διαφανώς τη δυνατότητα WebSocket**, ακόμη και σε φυλλομετρητές οι οποίοι δεν υποστηρίζουν το WebSocket του HTML5.

6.2. Συνοπτική σύγκριση

Μελετώντας κάποιος προσεκτικά τα παραπάνω μειονεκτήματα και πλεονεκτήματα, μπορεί να δει ότι υπάρχουν κάποια χαρακτηριστικά τα οποία χαρακτηρίζουν την κάθε τεχνική αυτόματης προώθησης και μπορούν να οριστούν ως τα κριτήρια επιλογής της καταλληλότερης. Το πόσο σημαντικό είναι το κάθε χαρακτηριστικό εξαρτάται από την ίδια την εφαρμογή αλλά και από την κρίση αυτού που θα την αναπτύξει. Στον Πίνακας 8 στην πρώτη στήλη παρατάσσονται οι τεχνικές και στην πρώτη γραμμή τα αντίστοιχα χαρακτηριστικά.

Το HTTP πρωτόκολλο δε σχεδιάστηκε για αμφίδρομη επικοινωνία. Οι μηχανισμοί HTTP long polling και HTTP streaming του Comet ξεφεύγουν από την αρχική σημασιολογία του HTTP προκειμένου να επιτύχουν αυτή την επικοινωνία. Αν και ως σήμερα αποτελούν τις πιο συνήθεις επιλογές, *το WebSocket είναι πλέον η καλύτερη επιλογή* που έχουμε αν θέλουμε να κάνουμε πραγματικά αυτόματη προώθηση περιεχομένου. Οι πελάτες που το υποστηρίζουν είναι γρήγοροι και παράγουν λιγότερες αιτήσεις. Η αμέσως επόμενη

καλύτερη επιλογή που έχουμε είναι το *SSE*. Το API του μοιάζει αρκετά με αυτό του WebSocket. Ακολουθούν οι βιβλιοθήκες του *Comet*. Ως προς το Comet επιλέγουμε πρώτα το streaming με multipart XHR και πάμε σε hidden Iframe αν ο φυλλομετρητής είναι IE. Για να μην έχουμε πρόβλημα από τους proxy πάμε σε long polling και όταν όλα τα άλλα αποτύχουν μπορούμε να επιστρέψουμε σε short polling.

Αφού δεν υποστηρίζουν όλοι οι φυλλομετρητές το WebSocket, αυτό δε σημαίνει ότι θα πρέπει να το απορρίψουμε. Τουναντίον, δεδομένου ότι θέλουμε να χρησιμοποιούμε WebSocket όποτε μπορούμε, θα πρέπει να κάνουμε μια ακόμη σχεδιαστική επιλογή. Θα πρέπει να επιλέξουμε κάποια βιβλιοθήκη για Reverse Ajax η οποία θα μπορεί να ανιχνεύει αν υπάρχει υποστήριξη για WebSocket και να πέφτει σε Comet (long polling) όταν δεν υπάρχει (Ενότητα 1).

Πίνακας 8: Συνοπτική σύγκριση τεχνολογιών αυτόματης προώθησης στο Διαδίκτυο

	Short polling	Applet	HTTP server push	Long polling	Websocket
Υποστήριξη φυλλομετρητών	ναι	μόνο αν βάλουμε plugin	όχι όλοι	ναι	όχι όλοι
Ευκολία χρήσης	ναι	όχι	όχι	ναι	
Χαμηλή καθυστέρηση	ανάλογα με το interval	στην αρχή της σύνδεσης είναι αργά	ναι	ναι	ακόμη καλύτερα
Μόνιμη σύνδεση	όχι		ναι	όχι	ναι
Αξιοποίηση της σύνδεσης	ναι		όχι	όχι	όχι
Αξιοπιστία (χειρισμός επανασύνδεσης, συγχρονισμός των μηνυμάτων και εξασφάλιση αποστολής μηνύματος)				ναι	αφήνεται στην εφαρμογή
Κλιμάκωση	όχι	ναι	απαιτεί νέους εξυπηρέτες	ναι	απαιτεί νέους εξυπηρέτες
Τείχη προστασίας (συμβατότητα δικτύου)	OK	πρόβλημα λόγω χρήσης άλλων θυρών	πρόβλημα	OK	OK
Πληρεξούσιοι εξυπηρέτες	OK	OK	OK	OK	όταν υπάρχουν προβλήματα, πάει σε long polling
Διατήρηση κατάστασης	όχι	ναι	όχι	όχι	όχι
Έλεγχος ανταλλαγής της πληροφορίας	στον πελάτη				στον εξυπηρέτη

7. ΥΛΟΠΟΙΗΣΗ ΕΙΔΟΠΟΙΗΣΕΩΝ ΣΤΟΝ ΕΞΥΠΗΡΕΤΗ

Οι λύσεις προώθησης από τον εξυπηρέτη προς τον πελάτη, οι οποίες παρουσιάστηκαν στο Κεφάλαιο 4 επικεντρώθηκαν, ως προς την υλοποίηση, κυρίως στην πλευρά του πελάτη. Ο πελάτης, και άρα ο φυλλομετρητής, μπορεί να υποστηρίξει διάφορες τεχνολογίες επικοινωνίας με τον εξυπηρέτη με σκοπό να δώσει δυναμικά περιεχόμενο στο χρήστη. Με βάση τις δυνατότητες της κάθε τεχνολογίας και τη σύγκριση η οποία έγινε στο Κεφάλαιο 5 καταλήξαμε στο ότι η λύση του WebSocket είναι η καλύτερη επιλογή. Δεδομένου όμως του γεγονότος πως δεν υποστηρίζουν όλοι οι φυλλομετρητές το WebSocket πρέπει να έχουμε και εναλλακτικές λύσεις. Η αμέσως επόμενη καλύτερη λύση λοιπόν είναι το Comet. Οπότε, για την υλοποίηση μιας εφαρμογής, ιδανικά θα πρέπει να επιλεγεί μια βιβλιοθήκη Reverse Ajax, η οποία θα δύναται να ανιχνεύει αν ο φυλλομετρητής υποστηρίζει WebSocket και αν δεν υπάρχει υποστήριξη τότε θα «καταφεύγει» σε Comet.

Εκτός από την κατάλληλη τεχνική προώθησης, ο σχεδιαστής θα πρέπει να επιλέξει και τον εξυπηρέτη που θα φιλοξενήσει την εφαρμογή του. Ο εξυπηρέτης θα πρέπει να έχει ικανοποιητική απόδοση (παράγραφος 7.1) και, φυσικά, να μπορεί να υποστηρίξει την τεχνική ή τις τεχνικές που έχει επιλέξει ο σχεδιαστής.

7.1. Απόδοση εξυπηρέτη και επίμονες συνδέσεις (NIO)

Σίγουρα, όπως προαναφέραμε, ο εξυπηρέτης θα πρέπει να μπορεί να υποστηρίξει τις τεχνολογίες που θέλουμε να υλοποιήσουμε. Επίσης, θα πρέπει να είναι αποδοτικός. Η απόδοση έγκειται στις εξής πέντε μετρικές:

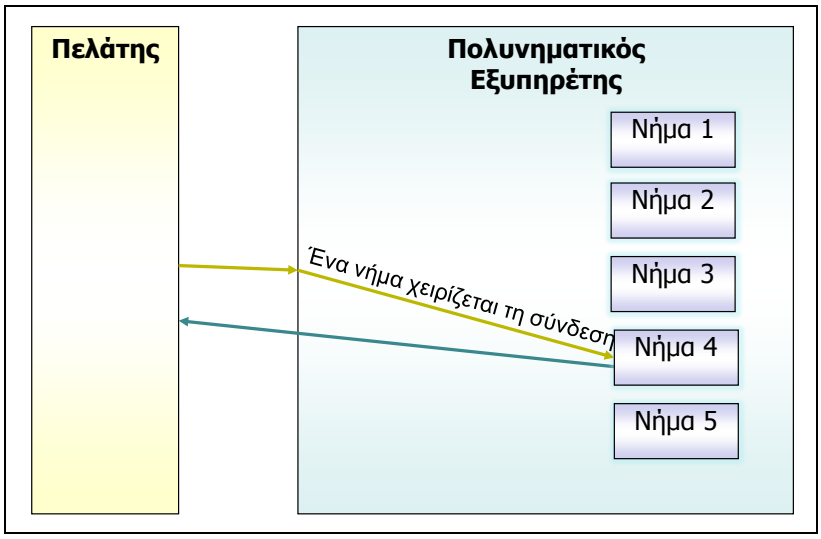
- *Κλιμάκωση* (scalability): Η κλιμάκωση ενός εξυπηρέτη καθορίζεται από το πλήθος των χρηστών που μπορούν να χρησιμοποιήσουν ταυτόχρονα ένα στιγμιότυπό του (instance). Για να κάνει ο εξυπηρέτης αποδοχή συνδέσεων οι οποίες χρησιμοποιούν Reverse Ajax, θα πρέπει να χρησιμοποιήσει συγκεκριμένες τεχνικές για το χειρισμό των επίμονων συνδέσεων. Για καλύτερη κλιμάκωση προτείνεται ένα νέο μοντέλο νημάτων. Αυτό το μοντέλο απαιτεί, όπως θα δούμε στη συνέχεια (Παράγραφος **Error! Reference source not found.**), ένα συγκεκριμένο Java API, μέσω του οποίου επιτυγχάνεται παύση στις αιτήσεις.

- *Καθυστέρηση* (latency): Η καθυστέρηση ορίζεται ως η διαφορά του χρόνου της στιγμής που δημιουργήθηκε το μήνυμα στον εξυπηρέτη από τη στιγμή που έφτασε στον πελάτη. Οι πιο σημαντικές στατιστικές τιμές εδώ είναι η μέση τιμή και η τυπική απόκλιση.
- *Κατανάλωση εύρους ζώνης* (bandwidth consumption): Για να πετύχει χαμηλή κατανάλωση εύρους ζώνης, θα πρέπει το πρωτόκολλο που χρησιμοποιείται από τον εξυπηρέτη να στέλνει τα δεδομένα στους χρήστες του όσο πιο αποδοτικά γίνεται, το οποίο επιτυγχάνεται αποκλειστικά μέσω προσθήκης της ελάχιστης δυνατής επιπλέον πληροφορίας (overhead) στο πραγματικά ωφέλιμο φορτίο (payload).
- *Ρυθμός αποστολής/παροχής* (throughput): Ο ρυθμός αποστολής ορίζεται ως το μέσο ποσοστό των δεδομένων που στέλνονται από τον εξυπηρέτη, π.χ. ο εξυπηρέτης Χ, που τρέχει σε ένα απλό (entry-level) μηχάνημα χρησιμοποιώντας 1 Gb Ethernet, είναι σε θέση να ωθήσει δεδομένα σε πραγματικό χρόνο με ένα ρυθμό 106 MB/δευτερόλεπτο σε 10000 χρήστες, όπου κάθε χρήστης λαμβάνει 5 μηνύματα/δευτερόλεπτο και κάθε μήνυμα έχει μέγεθος 2 KB.
- *Ποσοστό συνδέσεων* (Connection Rate): Το ποσοστό συνδέσεων ορίζεται απλά ως το ποσοστό των νέων συνδέσεων που μπορεί να δεχτεί ο εξυπηρέτης σε κάποιο χρόνο, προς το χρόνο αυτό. Η μετρική αυτή είναι σημαντική στην περίπτωση μιας εφαρμογής όπου συνδέονται συχνά πολλοί χρήστες. Ακόμη, σε μια ανεκτική στα σφάλματα εγκατάσταση εξυπηρετών που χρησιμοποιεί ταυτόχρονα πολλά στιγμιότυπα του εξυπηρέτη, αν κάποιο στιγμιότυπο αποτύχει, τότε θα πρέπει τα υπόλοιπα να μπορέσουν σύντομα να απορροφήσουν τους χρήστες του στιγμιότυπου που διακόπηκε.

Χρησιμοποιώντας αυτές τις μετρικές μπορούμε να συγκρίνουμε τους διαθέσιμους εξυπηρέτες ανεξάρτητα από την αρχιτεκτονική του καθενός. Πέρα όμως από τους εξυπηρέτες μπορούμε να συγκρίνουμε και τα μοντέλα που υποστηρίζουν ως προς την εξυπηρέτηση νέων αιτήσεων.

Τα Java servlet είναι σχεδιασμένα με βάση το **μοντέλο ενός-νήματος-ανά-σύνδεση**: ο εξυπηρέτης κατανέμει σε κάθε εισερχόμενη αίτηση ένα δικό της servlet thread (Εικόνα 31), επιτρέποντας έτσι την ταυτόχρονη εξυπηρέτηση πολλών ταυτόχρονων συνδέσεων. Αυτό είναι ιδανικό για τις απλές εφαρμογές όπου έχουμε σύντομες HTTP συνδέσεις. **Σκοπός των εφαρμογών αυτών είναι να αντιμετωπιστούν οι αιτήσεις όσο το δυνατόν ταχύτερα**

(μικρός χρόνος απόκρισης) και να ελαχιστοποιηθούν οι αιτήσεις οι οποίες βρίσκονται σε εκκρεμότητα.

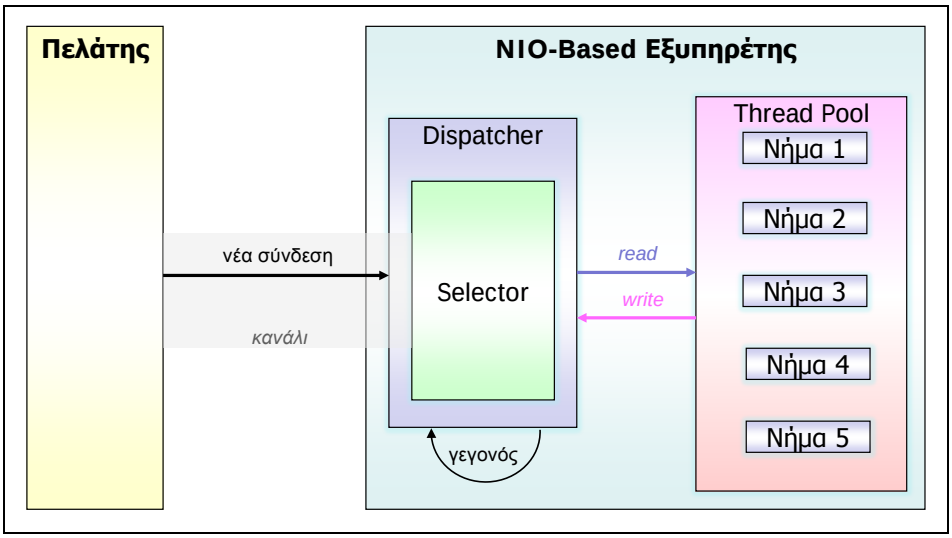


Εικόνα 31: Αλληλεπίδραση πελάτη και blocking εξυπηρέτη

Στην περίπτωση των επίμονων συνδέσεων, αν χρησιμοποιηθεί το μοντέλο ενός-νήματος-ανά-σύνδεση μειώνεται σημαντικά η δυνατότητα κλιμάκωσης των εφαρμογών. Όταν έχουμε επίμονες συνδέσεις, η δημιουργία (spawning) ενός νέου νήματος οδηγεί σε κατανάλωση μνήμης και σπατάλη πόρων, διότι δεν είναι εγγυημένο ότι το νέο νήμα θα χρησιμοποιηθεί. Πιο συγκεκριμένα, αν και μια σύνδεση έχει γίνει, είναι πιθανό να μη στέλνονται δεδομένα προς καμία κατεύθυνση. Ωστόσο, **η συγκεκριμένη σύνδεση μονοπωλεί το νήμα και είτε αυτό χρησιμοποιείται είτε όχι, καταναλώνει μνήμη και επεξεργαστική ισχύ (CPU) και επιπρόσθετα χρόνο λόγω εναλλαγών περιβάλλοντος λειτουργίας (context switch)**. Επιπρόσθετα, όταν ο εξυπηρέτης χρησιμοποιεί το threading model, θα πρέπει συνήθως να ορίζεται ένα **thread pool** (μέγιστος αριθμός νημάτων για την επεξεργασία των εισερχόμενων συνδέσεων). Εάν αυτή η τιμή δε ρυθμιστεί σωστά και είναι πολύ χαμηλή, καταλήγουμε σε **thread starvation**: οι αιτήσεις θα περιμένουν έως ότου κάποιο νήμα να είναι διαθέσιμο για την επεξεργασία τους και ο χρόνος απόκρισης θα αυξάνεται όσο πλησιάζει το μέγιστο πλήθος ταυτόχρονων συνδέσεων. Από την άλλη πλευρά, αν οριστεί μια υψηλή τιμή, τότε αυτό μπορεί να οδηγήσει σε **χρήση υπερβολικού μεγέθους μνήμης** (memory exception). Η δημιουργία πάρα πολλών νημάτων θα κατανάλωνε όλο το heap size της JVM και αυτό θα οδηγούσε σε συντριβή (crash) του εξυπηρέτη. Τέλος, ακόμη και όταν βρεθεί μια κατάλληλη

τιμή για το thread pool, το πλήθος των πελατών σύντομα θα ξεπεράσει το πλήθος των νημάτων που μπορεί να χειριστεί αποδοτικά ο εξυπηρέτης.

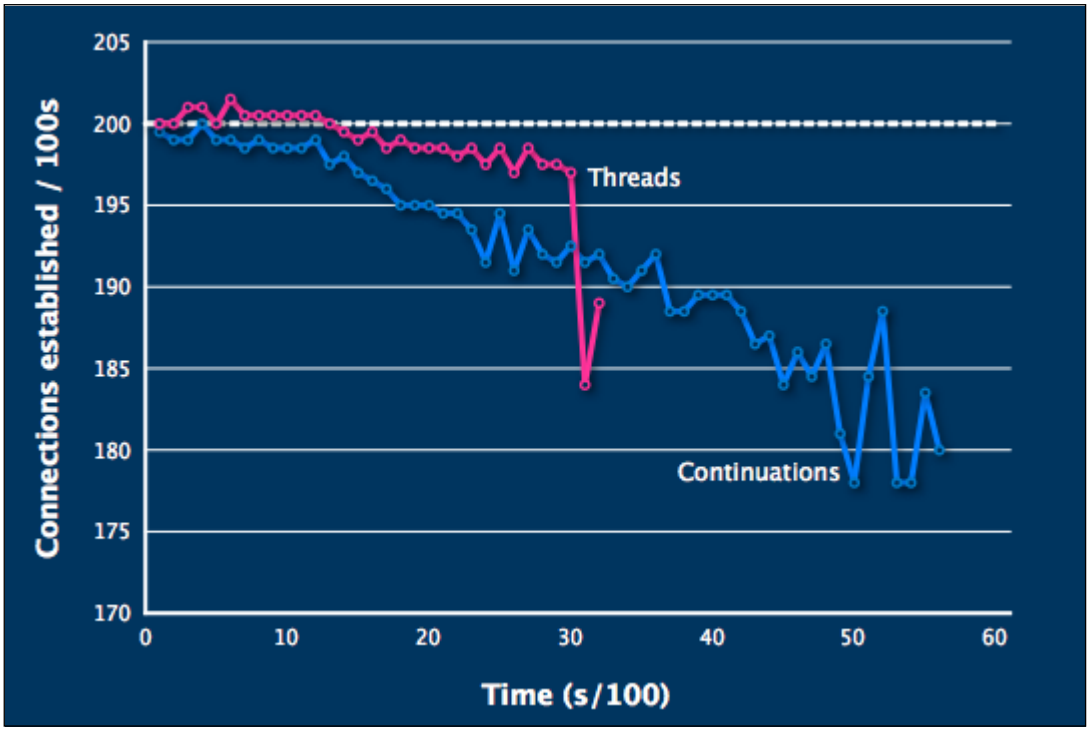
Η Java 4 εισήγαγε ένα νέο I/O API, το οποίο ονομάζεται *non-blocking I/O* (NIO) [51]. Αυτό το API χρησιμοποιεί έναν *επιλογέα* (selector) που αποφεύγει να δεσμεύει ένα νήμα κάθε φορά που γίνεται μια νέα HTTP σύνδεση στον εξυπηρέτη. Όταν έρχονται δεδομένα, τότε λαμβάνεται ένα γεγονός και διατίθεται ένα νήμα για την επεξεργασία της αίτησης. Αυτό ονομάζεται **μοντέλο ενός-νήματος-ανά-αίτηση** (Εικόνα 32). Επιτρέπει σε εξυπηρέτες, όπως ο WebSphere και ο Jetty, να κλιμακώνονται ώστε να χειρίζονται έναν αυξανόμενο αριθμό συνδέσεων χρήστη με ένα σταθερό αριθμό νημάτων. Χρησιμοποιώντας τον ίδιο εξοπλισμό, **οι εξυπηρέτες που ακολουθούν αυτό το μοντέλο έχουν μεν χειρότερο χρόνο απόκρισης, αλλά κλιμακώνονται πολύ καλύτερα απ' ότι στο μοντέλο ενός-νήματος-ανά-σύνδεση**.



Εικόνα 32: Αλληλεπίδραση πελάτη και NIO-based εξυπηρέτη

Στην Εικόνα 33 παρουσιάζονται τα αποτελέσματα δύο πειραμάτων που φιλοξενήθηκαν στο ίδιο μηχάνημα. Στο πρώτο πείραμα χρησιμοποιήθηκαν απλά νήματα για τον εξυπηρέτη, ενώ στο δεύτερο νήματα βασισμένα σε NIO Jetty Continuations [52]. Η σύγκριση των δύο γραφικών παραστάσεων επιβεβαιώνει τα παραπάνω. Στις πραγματικές εφαρμογές η δυνατότητα κλιμάκωσης εξαρτάται από πολλές μεταβλητές. Ο εξοπλισμός (hardware) του εξυπηρέτη, το λειτουργικό σύστημα, η JVM, η διαμόρφωση (configuration) του εξυπηρέτη, αλλά και ο σχεδιασμός της διαδικτυακής εφαρμογής και το προφίλ της κίνησης επηρεάζουν την απόδοση του μηχανισμού που χρησιμοποιείται υπό φορτίο (π.χ. Jetty Continuations).

Για αυτό το λόγο τα πειράματα του scalability θα πρέπει να γίνονται στις ίδιες συνθήκες όταν συγκρίνονται διαφορετικοί μηχανισμοί.



Εικόνα 33: Μετροπρόγραμμα (benchmark) των μοντέλων των νημάτων [35]

Στην επόμενη παράγραφο γίνεται μια σύντομη περιγραφή και σύγκριση των διαθέσιμων και πιο γνωστών εξυπηρετών εφαρμογών (application server) που υποστηρίζουν αυτόματη προώθηση.

7.2. Χρήση NIO σε εξυπηρέτες εφαρμογών

Για να διευκολυνθεί η χρήση NIO σε ένα servlet, χρειάζεται κάποιο API βασισμένο στα γεγονότα, το οποίο θα ξεκινά την κατάλληλη στιγμή τις ενέργειες διαβάσματος και γραψίματος πάνω από τις ανοικτές συνδέσεις. Το Continuations API, το οποίο αναφέρθηκε προηγουμένως, είναι ένα τέτοιο API. Στη συνέχεια θα δούμε μια σύντομη επισκόπηση των μηχανισμών υποστήριξης NIO ανά container, ξεκινώντας με το Continuations API στο οποίο αναφερθήκαμε.

7.2.1. Jetty Continuations API

Ο Jetty 6 σχεδιάστηκε για να κλιμακώνεται καλά όταν υπάρχει μεγάλο πλήθος ταυτόχρονων συνδέσεων. Για να χειριστεί τις επίμονες συνδέσεις ο Jetty παρουσίασε ένα νέο χαρακτηριστικό, το Continuations API [53] [54].

Για να χρησιμοποιηθεί το Continuations API, ο Jetty πρέπει να ρυθμιστεί ώστε να διαχειρίζεται τις αιτήσεις με την υποδοχή `SelectChannelConnector`. Αυτή η υποδοχή είναι κτισμένη πάνω στο java.nio API, πράγμα που της επιτρέπει να κρατά ανοικτές τις συνδέσεις χωρίς να καταναλώνει ένα νήμα για την καθεμιά. Όταν χρησιμοποιείται η SelectChannelConnector, τότε η μέθοδος `ContinuationSupport.getContinuation()` παρέχει ένα στιγμιότυπο της `SelectChannelConnector.RetryContinuation`. Όταν κληθεί η `suspend()` του RetryContinuation, τότε συμβαίνει ένα ειδικό runtime exception, το `RetryRequest`. Αυτό λαμβάνεται από το SelectChannelConnector, αλλά αντί να σταλεί κάποια απάντηση στον πελάτη ως αποτέλεσμα του exception, **η αίτηση μπαίνει στην ουρά με τα Continuations που εκκρεμούν και η σύνδεση HTTP παραμένει ανοιχτή**. Σε αυτό το σημείο, το νήμα που χρησιμοποιήθηκε για την εξυπηρέτηση της αίτησης επιστρέφεται στο `ThreadPool`, όπου μπορεί να χρησιμοποιηθεί για να εξυπηρετήσει μια άλλη αίτηση.

```
public interface Continuation {
  public boolean suspend(long timeout);
  public void resume();
  …
}
```

Η αίτηση που έχει ανασταλεί παραμένει στην ουρά των Continuations είτε μέχρι να λήξει το καθορισμένο χρονικό όριο είτε μέχρι να κληθεί η μέθοδος `resume()` στο Continuation της, οπότε και η αίτηση υποβάλλεται εκ νέου στο servlet (μέσω του filter chain). Ο κώδικας που εκτελέστηκε πριν για την αίτηση ξανατρέχει από την αρχή, αγνοώντας όμως αυτή τη φορά τη suspend() και συνεχίζοντας κανονικά. Αν δε θέλουμε να εκτελεστεί ο ίδιος κώδικας για δεύτερη φορά, θα πρέπει να χρησιμοποιήσουμε τη μέθοδο `isPending()`. Η μέθοδος αυτή επιστρέφει `true` αν έχει ήδη κληθεί το suspend μία φορά.

Έτσι, **ο μηχανισμός αυτός χρησιμοποιεί stateless χειρισμό της αίτησης HTTP για να προσομοιώσει την αναστολή και τη συνέχιση (continuation)**. Όταν γίνεται resume() η

αίτηση, τότε εισέρχεται εκ νέου στη filter/servlet chain και σε κάθε πλαίσιο ασφάλειας και συνεχίζεται ο χειρισμός κανονικά στο σημείο του continuation.

Επιπλέον, **η βιβλιοθήκη του Jetty Continuations δεν εξαρτάται από το Jetty container, αλλά είναι φορητή**. Η βιβλιοθήκη μπορεί να ανιχνεύσει ποιο container και ποια προδιαγραφή είναι διαθέσιμη. Έτσι, αν τρέχει σε ένα εξυπηρέτη Jetty, τότε θα χρησιμοποιηθεί το native Jetty API. Στην περίπτωση που ο container υποστηρίζει την Προδιαγραφή Servlet 3.0, τότε θα χρησιμοποιηθεί αυτό το κοινό API. Αλλιώς, θα χρησιμοποιηθεί μια λύση που δεν επιτρέπει κλιμάκωση.

7.2.2. Tomcat CometProcessor

Για να έχουμε ασύγχρονα servlet στον Tomcat, θα πρέπει να υλοποιείται ο *CometProcessor* [55]. Πρόκειται για μια προσέγγιση διαφορετική από αυτή του Jetty, ως προς το ασύγχρονο, non-blocking HTTP η οποία, αν και υποδεέστερή, είναι επαρκής.

Αντί των καθιερωμένων HTTP μεθόδων (doGet, doPost κτλ), το CometProcessor interface ορίζει τη μέθοδο επιστροφής (callback) event(CometdEvent), μέσω της οποίας γίνεται ασύγχρονα η επεξεργασία των γεγονότων Comet.

```
public interface CometProcessor extends Servlet{
{
  public     void     event(CometEvent)     throws     IOException,
ServletException;
}
```

7.2.3. GlashFish CometHandler

Το βασικό κομμάτι μιας εφαρμογής που βασίζεται σε Grizzly Comet είναι το *CometHandler* [56], το οποίο εφαρμόζει μια πολιτική προώθησης δεδομένων στο φυλλομετρητή, μεταφέροντας έμμεσα τη λογική του NIO στο Servlet.

Το interface CometHandler αντιπροσωπεύει μια σύνδεση (ή απάντηση) που έχει ανασταλεί. Περνώντας ένα στιγμιότυπο αυτής της κλάσης στη `CometContext.addCometHandler(com.sun.grizzly.CometHandler, Boolean)`, ενημερώνουμε το Grizzly Comet να αναστείλει την υποκείμενη σύνδεση και να καθυστερήσει την αποστολή της απάντησης. Δεδομένου ότι η απάντηση δεν αποστέλλεται,

η σύνδεση θεωρείται πως έχει γίνει suspend και μπορεί στη συνέχεια, μετά από κάποιο γεγονός, να γίνει resume καλώντας την `CometContext.resumeCometHandler(CometHandler)`, από την `onEvent(com.sun.grizzly.comet.CometEvent)`. Η `CometContext.resumeCometHandler(CometHandler)`, στέλνει τελικά την απάντηση.

```
public interface CometHandler{
  public void onEvent(CometEvent<String> ce){
    if(ce.getType() == CometEvent.READ){
      // Read Bytes Asynchronously
    }
  }
}
```

7.2.4. JBoss HttpEvent

Τα servlet που υλοποιούν το interface `org.jboss.servlet.http.HttpEventServlet` [57] έχουν τη δική τους μέθοδο γεγονότων και όχι τη συνήθη servlet method. Το πεδίο EventType δείχνει ποιο γεγονός συνέβη και το αντικείμενό του δίνει πρόσβαση στα αντικείμενα `request` και `response`. Η βασική διαφορά με τα απλά servlet είναι πως τα αντικείμενα αυτά τώρα παραμένουν έγκυρα και πλήρως λειτουργικά από τη στιγμή της επεξεργασίας του `BEGIN` μέχρι και την επεξεργασία του `END` ή του `ERROR`, ώστε να μπορούν να γίνουν οι ασύγχρονες λειτουργίες.

7.2.5. Servlet 3.0 AsyncListener

Η έλλειψη υποστήριξης ασύγχρονης λειτουργίας στην Προδιαγραφή Servlet 2.5 ανάγκασε τους προμηθευτές εξυπηρετών να σχεδιάσουν δικές τους λύσεις, αναπτύσσοντας δικές του κλάσεις και διεπαφές οι οποίες προωθούν τον κλιμακούμενο προγραμματισμό Comet στο Servlet 2.5 API. Για παράδειγμα, είδαμε ότι ο Tomcat δημιούργησε την κλάση CometProcessor, ο Jetty 6 διαθέτει το μηχανισμό Continuations και ο Grizzly τον CometHandler.

Αν και κάθε container έχει τη δική του υλοποίηση, οι εφαρμογές είναι προτιμότερο να μην είναι υλοποιημένες για συγκεκριμένο container. **Η Προδιαγραφή Servlet 3.0 παρέχει έναν**

πρότυπο μηχανισμό, ο οποίος περιλαμβάνει πρόσθετες μεθόδους για να κάνει αναστολή (suspend) και αργότερα συνέχιση (resume) μιας αίτησης [58], οπότε κάθε container που υποστηρίζει αυτή την προδιαγραφή μπορεί να υποστηρίξει και long polling Comet. Χρησιμοποιώντας τον πρότυπο μηχανισμό υποστήριξης event-based NIO του Servlet 3.0 μπορούμε λοιπόν να επιτύχουμε και **φορητότητα της εφαρμογής**.

```
<servlet>
  <servlet-name>AsyncMessageListenServlet</servlet-name>
  <servlet-class>…</servlet-class>
  <async-supported>true</async-supported>
</servlet>
<servlet-mapping>
  <servlet-name>AsyncMessageListenServlet</servlet-name>
  <url-pattern>/messageListen</url-pattern>
</servlet-mapping>
interface HttpServletRequest extends ServletRequest{
  AsyncContext startAsync();
  AsyncContext startAsync(ServletRequest, ServletResponse);
}

puplic interface AsyncContext{
  public ServletRequest getRequest();
  public ServletResponse getResponse();
}
```

7.3. Υποστήριξη ειδοποιήσεων σε εξυπηρέτες εφαρμογών

Οι τεχνολογίες του Comet και τα WebSocket δεν έχουν κάποιο συγκεκριμένο πρότυπο σε Java. Υπάρχουν διάφορα API που τις υποστηρίζουν και κάθε container υποστηρίζει ένα υποσύνολο αυτών των API, που σημαίνει πως **κάθε εξυπηρέτης έχει το δικό του native API για Comet και WebSocket.** Αυτό με τη σειρά του δυσχαιρένει την ανάπτυξη μιας φορητής (portable) διαδικτυακής εφαρμογής. Σε αυτή την ενότητα θα δούμε αν και πώς χρησιμοποιούνται συνολικά οι τεχνικές Comet και WebSocket στους εξυπηρέτες

εφαρμογών (application server) Jetty, Tomcat, Grizzly (Glassfish), Jboss και WebSphere (Πίνακας 9).

Όσον αφορά στο WebSocket, δεν υπάρχει ακόμα κάποιο πρότυπο σε Java και, ως εκ τούτου, θα πρέπει να χρησιμοποιηθεί το Native API για Websocket του container όπου θα φιλοξενηθεί η εφαρμογή.

Αν στο σχεδιασμό μιας εφαρμογής είναι ανοικτό το ζήτημα του ποιος εξυπηρέτης εφαρμογών θα χρησιμοποιηθεί, τότε αυτός που «κερδίζει», με βάση τις **δυνατότητες υποστήριξης των τεχνολογιών αυτόματης προώθησης γεγονότων**, είναι μάλλον ο **Jetty**. Η βιβλιοθήκη του Jetty Continuations έχει γίνει αποδεκτή και χρησιμοποιείται από όλους τους σύγχρονους ανωτέρω εξυπηρέτες, υποστηρίζει Servlet 3.0 και έχει Native API για WebSocket.

Πίνακας 9: Τεχνολογίες αυτόματης προώθησης στο Διαδίκτυο που υποστηρίζονται από κάθε container

Τεχνολογία	Non-blocking I/O	Servlet 2.5	Comet				WebSocket
			Jetty Continuations	Advanced I/O	Native API	Servlet 3.0	Native API
Container							
Jetty 6							
Jetty 7							Jetty
Jetty 8							Jetty
Tomcat 6							
Tomcat 7							
Glassfish 2					Grizzly		
Glassfish 3					Grizzly		Grizzly
Jboss 5					Jboss		
Jboss 6					Jboss		
Jboss 7					Jboss		
WebSphere 8							

7.4. Αξιολόγηση εξυπηρετών

Στις προηγούμενες ενότητες του κεφαλαίου, ειπώθηκε πως τα κριτήρια επιλογής ενός εξυπηρέτη αυτόματης προώθησης είναι κατά κύριο λόγο η απόδοσή του και οι τεχνικές προώθησης που μπορεί να υποστηρίξει. Φυσικά το κόστος, η ευκολία ανάπτυξης εφαρμογών, η ασφάλεια, η διάσχιση των proxy και των τειχών προστασίας, η δυνατότητα ανάκαμψης και απρόσκοπτης λειτουργίας μετά από fail-over ή σφάλμα σύνδεσης, η υποστήριξη που παρέχεται και το documentation είναι επίσης πολύ σημαντικοί παράγοντες, αλλά δε χρειάζεται κάποια πειραματική διαδικασία για να πάρουμε τις τιμές τους. Αντίθετα, ο υπολογισμός των μετρικών της απόδοσης απαιτεί την ανάπτυξη εφαρμογών αξιολόγησης.

Ως προς την αξιολόγηση, στη βιβλιογραφία συναντάμε τους εξής ορισμούς:

- **Έλεγχος απόδοσης** (performance testing): σύμφωνα με την IEEE και την ACM, είναι η βασική κατηγορία ελέγχου, με τις load testing και stress testing να είναι υποκατηγορίες της. Πιο γενικά, το performance testing ορίζεται ως ένας έλεγχος για να καθοριστεί η απόκριση ενός συστήματος ή μιας εφαρμογής. Ως εκ τούτου, ένας έλεγχος απόδοσης μπορεί να μετρήσει πόσο καλά εκτελείται μια λειτουργία (π.χ. ο χρόνος για να φορτώσει μια ιστοσελίδα, ο χρόνος για να εκτελεστεί μια συναλλαγή).
- **Έλεγχος φορτίου** (load testing): είναι η διαδικασία επιβολής ζήτησης (αιτήσεων) σε ένα σύστημα ή μια συσκευή και η μέτρηση της απόκρισής του. Ο έλεγχος φορτίου βοηθά στην κατανόηση της συμπεριφοράς ενός συστήματος κάτω από συγκεκριμένο φορτίο. Το *φορτίο* θα μπορούσε να είναι ο αναμενόμενος *αριθμός ταυτόχρονων χρηστών/πελατών* στην εφαρμογή οι οποίοι εκτελούν ένα συγκεκριμένο αριθμό συναλλαγών εντός κάποιου χρονικού διαστήματος. Ένα καλό εργαλείο ελέγχου φορτίου επιτρέπει στο χρήστη του *να αλλάζει συνεχώς το φορτίο* ώστε να προσομοιώσει ρεαλιστικά τους χρήστες της ιστοσελίδας. Μια άλλη σημαντική πτυχή ενός εργαλείου ελέγχου φορτίου είναι η υποβολής εκθέσεων (reporting) σε πραγματικό χρόνο, καθώς μπορεί, παραδείγματος χάριν, να δείχνει πώς αλλάζουν οι χρόνοι απόκρισης καθώς αυξάνεται το φορτίο ή από που προέρχονται οι καθυστερήσεις (δίκτυο, βάση δεδομένων, εφαρμογής κ.α.).

- **Δοκιμές υπό ακραίες συνθήκες** (stress testing): χρησιμοποιείται για να κατανοήσουμε τα όρια του συστήματος. Αυτό το είδος της δοκιμής καθορίζει την ευρωστία μιας ιστοσελίδας σε συνθήκες ακραίου φορτίου και βοηθά τους αρχιτέκτονες του συστήματος να αποδείξουν ότι η ιστοσελίδα θα αποδώσει ικανοποιητικά ακόμα και όταν ο φόρτος των χρηστών υπερβεί το αναμενόμενο μέγιστο.

Το σημαντικό σε κάθε περίπτωση είναι η ιστοσελίδα να λειτουργεί σωστά και να έχει καλό χρόνο απόκρισης στους χρήστες. Κάθε εξυπηρέτης υποστηρίζει διαφορετικές βιβλιοθήκες comet και websocket. Αυτό σημαίνει ότι είναι δύσκολο να μεταφέρουμε την εφαρμογή μας από έναν εξυπηρέτη σε έναν άλλο, αν διαπιστώσουμε ότι ο πρώτος δεν καλύπτει το στόχο της απόδοσης που έχουμε θέσει ή αν καταρρέει όταν έχει μεγάλο φόρτο.

Η απόδοση μιας εφαρμογής σε συνθήκες πραγματικής λειτουργίας μπορεί να μετρηθεί, όπως είδαμε, με τη χρήση **load testing**. Σκοπός αυτού του τεστ είναι η μοντελοποίηση των χαρακτηριστικών της απόδοσης μιας διαδικτυακής εφαρμογής στην παραγωγή. Πρέπει να εφαρμοστεί φόρτος στο σύστημα, το οποίο θα αντιπροσωπεύει το φορτίο που θα δέχεται στην πραγματικότητα, οπότε θα πρέπει να ληφθεί υπόψη **πώς θα αλλάζουν τα φορτία** των εικονικών χρηστών (Virtual User loads – VUs loads) με το χρόνο. Επιπρόσθετα, το load test πρέπει να προσομοιώνει μια **τυπική μίξη δραστηριοτήτων**. Αν θέλουμε μια καλή προσέγγιση πραγματικών χρηστών, τότε καλό θα είναι **οι VUs να διαφέρουν ελαφρώς** ως προς τα στοιχεία στο login, θα πρέπει να κάνουν παύσεις ανάμεσα στις ενέργειές τους (χρόνος για να σκεφτούν) κτλ. Για παράδειγμα, μια ιστοσελίδα για online gaming, έχει χρήστες οι οποίοι προσθέτουν χρήματα στο λογαριασμό τους, ελέγχουν τις αποδόσεις του στοιχήματος και ποντάρουν πριν από ένα αθλητικό γεγονός. Στην περίπτωση του εξυπηρέτη δεν έχουν τόση σημασία οι δραστηριότητες των χρηστών όσο έχει το φορτίο που μπορεί αυτός να επωμιστεί.

Δεδομένων των τεχνολογιών που υποστηρίζει ένας εξυπηρέτης και των υπόλοιπων χαρακτηριστικών του, μένει να δούμε αν καλύπτει τις απαιτήσεις μας σε φόρτο και να ελέγξουμε γενικά την απόδοσή του με load testing. Πολλοί εξυπηρέτες δίνουν κάποιες τιμές ως προς το πλήθος των παράλληλων χρηστών που μπορούν να υποστηρίξουν. Άλλοι, πέρα από τα νούμερα, δίνουν και κάποια βοηθητικά προγράμματα για load testing. Στη γενική περίπτωση, για να γίνουν πειράματα υπολογισμού της απόδοσης, θα πρέπει να αναπτυχθούν δύο βοηθητικά προγράμματα για κάθε εξυπηρέτη ο οποίος εξετάζεται. Όμοια

με τη δουλειά που έχει γίνει στον εξυπηρέτη Migratory [59], το πρώτο πρόγραμμα θα είναι το PushClient-Benchmark και το δεύτερο θα είναι το PushAgent-Benchmark.

Το PushClient-Benchmark θα συμπεριφέρεται όπως και ένας πελάτης του εξυπηρέτη, θα μπορεί δηλαδή να κάνει τα εξής:

- Θα συνδέεται στον εξυπηρέτη
- Θα κάνει εγγραφή σε μια λίστα από θέματα
- Θα λαμβάνει συνεχόμενα μηνύματα πραγματικού χρόνου για τα θέματα στα οποία έχει κάνει εγγραφή

Επιπρόσθετα, θα πρέπει να υποστηρίζει τις παρακάτω λειτουργίες:

- Θα μπορεί να ανοίξει όσες ταυτόχρονες συνδέσεις του ζητηθούν
- Θα υπολογίζει την καθυστέρηση των μηνυμάτων χρησιμοποιώντας την πληροφορία που υπάρχει στο time-stamp (χρόνος δημιουργίας μηνύματος) το οποίο περιέχεται σε κάθε μήνυμα που λαμβάνεται.

Στην πραγματικότητα, το PushClient-Benchmark θα είναι ένας προσομοιωτής πολλών ταυτόχρονων πελατών, των οποίων το πλήθος θα δίνεται ως παράμετρος (`ClientsNum`). Κάθε πελάτης θα μπορεί να κάνει εγγραφή σε `SubjectsPerClient` πλήθος θεμάτων από το σύνολο `Subjects` με τα θέματα.

Σκοπός του PushAgent-Benchmark είναι να τροφοδοτεί τον εξυπηρέτη με τα μηνύματα που θα καταναλώνει το PushClient-Benchmark. Το PushAgent-Benchmark θα συνδέεται στο στιγμιότυπο του εξυπηρέτη (παράμετρος `ServerHost`) και θα τον τροφοδοτεί με μηνύματα μεγέθους `Size` σε συχνότητα `Frequency` για μια διαμορφώσιμη λίστα θεμάτων (παράμετρος `Subjects`). Αυτό το πρόγραμμα είναι προαιρετικό, αφού τα μηνύματα θα μπορούσαν να είναι τυχαία και να φτιάχνονται δυναμικά από τους client.

Ο κάθε εξυπηρέτης μπορεί να φιλοξενηθεί σε μια σειρά από μηχανήματα. Η επιλογή του κατάλληλου μηχανήματος εξυπηρέτη είναι πολύ σημαντική για τα αποτελέσματα των πειραμάτων. Στην επόμενη υποενότητα αναφέρουμε τα πιο σημαντικά χαρακτηριστικά που μας ενδιαφέρουν σε αυτό το μηχάνημα.

7.4.1. Χαρακτηριστικά μηχανής

Όταν δοκιμάζουμε κάποιον ή κάποιους εξυπηρέτες, τότε οι δοκιμές θα πρέπει να γίνονται χρησιμοποιώντας πανομοιότυπες μηχανές. Τα τεχνικά χαρακτηριστικά που μας ενδιαφέρουν είναι τα εξής:

- *Το μοντέλο προμηθευτή (Vendor Model)*

Το μοντέλο προμηθευτή είναι σημαντικό, γιατί με αυτό μπορούμε να ελέγξουμε το BIOS και την κάρτα NIC (Network Interface Controller). Η κάρτα δικτύου συνδέει τον υπολογιστή στο δίκτυο. Οι Ethernet network controller τυπικά υποστηρίζουν Ethernet 10, 100 ή 1000 Mbit/s, το οποίο σημαίνει πως μπορούν να υποστηρίξουν μια θεωρητική ταχύτητα μεταφοράς 10, 100 ή 1000 Mbit το δευτερόλεπτο.

Για παράδειγμα, αν θέλουμε έναν εξυπηρέτη για COMET βασισμένο σε Linux ο οποίος θα χειρίζεται χιλιάδες ταυτόχρονες συνδέσεις long polling, τότε μπορούμε να χρησιμοποιήσουμε κάποιο μηχάνημα της Intel ή της Broadcom το οποίο έχει 1GB NIC με καλή υποστήριξη για τον πυρήνα Linux.

- *Το πλήθος CPU/ CPU core*

Η χρήση πολλαπλών CPU/CPU-Core δίνει στις εφαρμογές τη δυνατότητα να επεξεργάζονται παράλληλα αρκετές συνδέσεις (και αιτήσεις) πελατών, πετυχαίνοντας έτσι καλύτερη απόδοση. Η απόδοση, βέβαια, επιδέχεται βελτίωσης όταν η συνολική δουλειά της εφαρμογής μπορεί να σπάσει σε μικρότερα τμήματα που δεν αλληλοεξαρτώνται. Στην περίπτωση αυτή, ο κώδικας του εξυπηρέτη θα πρέπει να λαμβάνει υπόψη του όλες τις CPU, ώστε να διαμοιράζει τη δουλειά σε όλα τα core του επεξεργαστή. Στις εφαρμογές COMET, όπου έχουμε μόνο πελάτες και όλα τα νήματα του εξυπηρέτη κάνουν την ίδια δουλειά, μπορούμε να διαμοιράσουμε το φόρτο εργασία αρκετά ισορροπημένα στους CPU cores. Με αυτό τον τρόπο η εφαρμογή μπορεί να κλιμακώνεται γραμμικά με το πλήθος των CPU του εξυπηρέτη. Για παράδειγμα, στο documentation του Node.js η αρχική διεργασία γίνεται `fork()` πλήθος_CPU φορές και δημιουργείται έτσι ένα cluster.

- *Η ταχύτητα της CPU*

Η ταχύτητα της CPU είναι αυτή που καθορίζει την απόδοση ενός υπολογιστή, όταν του αναθέτουμε κάποιες εργασίες. Οι παράγοντας που καθορίζουν με τη σειρά τους την ταχύτητά της, και άρα μας ενδιαφέρουν, είναι, όπως αναφέρθηκε και προηγουμένως, τα

core, ο τύπος της (αφού διαφορετικοί τύποι, θα έχουν διαφορετικές επιδόσεις), το clock speed (συχνότητα λειτουργίας), η cache της CPU (τύπος και μέγεθος) και το bus speed.

- *Η μνήμη RAM*

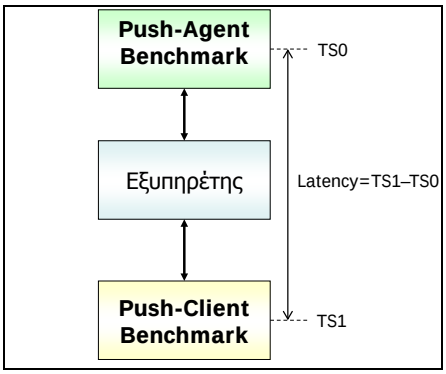
Ο εξυπηρέτης θα πρέπει να έχει καλή ταχύτητα, οπότε χρειάζεται μεγάλη μνήμη. Επίσης, θα πρέπει να λειτουργεί όσο γίνεται καλύτερα και χωρίς σφάλματα.

7.4.2. Λειτουργικό σύστημα

Ως προς το *λειτουργικό σύστημα*, αυτό θα πρέπει επίσης να είναι το ίδιο και να υποστηρίζει τον εξυπηρέτη που θέλουμε να εγκαταστήσουμε.

7.4.3. Καθυστέρηση

Η **καθυστέρηση** είναι εδώ ο χρόνος διάδοσης ενός μηνύματος από το PushAgent-Benchmark στο PushClient-Benchmark. Το PushAgent-Benchmark εισάγει σε κάθε μήνυμα ένα timestamp και το PushClient-Benchmark εξάγει αυτή την πληροφορία και υπολογίζει τη διαφορά ανάμεσα στον τρέχοντα χρόνο παραλαβής και στη χρονοσήμανση.



Έστω ότι για ένα χρονικό διάστημα T έχουμε τα k δείγματα λ_1, λ_2, ..., λ_k Οι στατιστικές τιμές που θα πρέπει να υπολογίζονται είναι οι κάτωθι τέσσερις:

- Ελάχιστη καθυστέρηση λ_{min}
- Μέγιστη καθυστέρηση λ_{max}
- Μέση καθυστέρηση $\mu_k = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_k}{k}$
- Τυπική απόκλιση της καθυστέρησης $\sigma_k = \sqrt{\frac{\sum_{i=1}^{k} (\lambda_i - \mu_k)^2}{k-1}}$

7.4.4. Μέσο φορτίο της CPU

Ο **φόρτος του συστήματος** (system load) [60] είναι μια μετρική του ποσού της εργασίας που εκτελεί το σύστημα. Το μέσο φορτίο αντιπροσωπεύει το μέσο φορτίο του συστήματος κατά τη διάρκεια μιας χρονικής περιόδου (π.χ. 1, 5 ή 15 λεπτών).

Τα συστήματα Unix αναφέρονται στο φορτίο της CPU ως το run-queue length, δηλαδή το πλήθος των διεργασιών που τρέχουν αυτή τη στιγμή συν αυτές που περιμένουν στην ουρά για να τρέξουν. Ιδανικά το νούμερο αυτό πρέπει να είναι κάτω από το 1,00 ώστε να μην υπάρχει αναμονή. Το 1,00 δεν αφήνει όμως περιθώρια, οπότε στην πράξη προτιμούμε να μένουμε κάτω από το 0,7. Αυτό ισχύει στην περίπτωση μιας single-core CPU. Όταν έχουμε πολλαπλούς επεξεργαστές το φορτίο είναι ανάλογο του πλήθους των core που έχουμε.

7.4.5. Εύρος ζώνης

Το **εύρος ζώνης** που διαβιβάζεται είναι η μετρική της ποσότητας των δεδομένων που μεταδίδονται ανά δευτερόλεπτο από τον εξυπηρέτη προς τους πελάτες.

7.4.6. Διαδικασία ελέγχου

Δεδομένης της μηχανής, του εξυπηρέτη και των μετρήσεων που θέλουμε να κάνουμε, θα πρέπει να δούμε τι είναι αυτό που θα ελέγξουμε τελικά. Στις προηγούμενες υποενότητες ορίσαμε τις μετρικές που μας ενδιαφέρουν, πλέον μπορούμε να ελέγξουμε πώς θα μεταβληθούν αλλάζοντας τις παραμέτρους που τις επηρεάζουν. Οι μεταβλητές που επηρεάζουν την καθυστέρηση, το μέσο φόρτο της CPU και το εύρος ζώνης είναι πιο συγκεκριμένα το **πλήθος των συνδεδεμένων πελατών**, το **μέγεθος των μηνυμάτων** και η **συχνότητά** τους. Όταν οι πελάτες εγγράφονται σε συγκεκριμένα θέματα, τότε σημασία έχει επίσης το **συνολικό πλήθος των θεμάτων** καθώς και το **πλήθος θεμάτων ανά πελάτη**. Μια ακόμη μετρική η οποία εξαρτάται από τις παραπάνω παραμέτρους και την οποία θα μπορούσαμε να μετράμε είναι το πλήθος των μηνυμάτων που στέλνει τελικά ο εξυπηρέτης στους πελάτες.

Στους ελέγχους που γίνονται στα πλαίσια της αξιολόγησης ενός εξυπηρέτη μπορούμε να δοκιμάσουμε τα παρακάτω:

- Μεγάλα μηνύματα (π.χ. 1KB, 2KB και 5KB) με 1000-16000 πελάτες, 2000 συνολικά θέματα, 100 μηνύματα/δευτερόλεπτο και 100 θέματα ανά πελάτη.

- Πολλούς πελάτες (π.χ. 25000-120000) με μικρά μηνύματα (50 byte), 1000 συνολικά θέματα, 500 μηνύματα ανά δευτερόλεπτο και 2 θέματα ανά πελάτη.
- Υψηλά ποσοστά ενημέρωσης (π.χ. 20000 μηνύματα/δευτερόλεπτο) με μικρά μηνύματα (50 byte), 2000-10000 πελάτες, 20000 συνολικά θέματα και 100 θέματα ανά πελάτη.

Ακόμη ο τρόπος που αλλάζει το φορτίο με το χρόνο είναι πολύ σημαντικός. Ο τρόπος που αλλάζει η ζήτηση σε μια εφαρμογή εξαρτάται από το είδος της. Η δοκιμή που γίνεται μάλλον σε κάθε load test δημιουργεί ένα σταθερό πλήθος VUs, αλλά αυτό το σενάριο είναι μάλλον το ιδανικό. Θα μπορούσαμε να έχουμε τα εξής μοτίβα δημιουργίας φόρτου [61]:

- Σταθερό (constant): δημιουργεί ένα σταθερό πλήθος VUs
- Κλιμάκωσης (ramp-up): δημιουργεί ένα πλήθος VUs που αυξάνει κατά τη διάρκεια της δοκιμής. Είναι χρήσιμο για τον έλεγχο της συμπεριφοράς του εξυπηρέτη υπό από ένα αυξανόμενο φορτίο.
- Αιχμής (peak): Δημιουργεί ένα σταθερό πλήθος VUs με περιοδικές φάσεις υψηλού φόρτου. Είναι χρήσιμο για τον έλεγχο του εάν ο εξυπηρέτης μπορεί να επανέλθει στην κανονική του συμπεριφορά μετά από μια αιχμή στο φορτίο (load peak).
- Προσαρμοζόμενο (custom): Επιτρέπει τη ρύθμιση του φορτίου που θα εφαρμοστεί δίνοντας μια καμπύλη μεταβολής VUs.

7.4.7. Έλεγχος φόρτου στον CometD

Το CometD project δίνει ένα εργαλείο για load testing. Στην παρούσα εργασία χρησιμοποιήθηκε αυτό το εργαλείο για δύο λόγους:

- για να γίνει μια σύγκριση με πραγματικά αποτελέσματα ανάμεσα στο COMET και στο WebSocket
- για να γίνει χρήση ενός πραγματικού εργαλείου benchmarking

Δεν είναι σκοπός μας να δούμε πόσο «αντέχει» ο εξυπηρέτης Jetty μιας και αυτό έχει ήδη γίνει χρησιμοποιώντας τον κατάλληλο εξοπλισμό. Σε κάθε περίπτωση, για να δούμε καλά αποτελέσματα θα πρέπει να έχουν γίνει σωστά οι ρυθμίσεις του λειτουργικού συστήματος, του TCP stack, του δικτύου, της JVM και φυσικά της εφαρμογής. Αν δε

ρυθμιστούν σωστά όλα αυτά, τότε το threshold στο οποίο θα φτάσουμε θα οφείλεται σε άλλους παράγοντες (bottlenecks) και όχι στον εξυπηρέτη.

7.4.7.1. Εγκατάσταση

Το setup πρέπει να γίνει και στον εξυπηρέτη αλλά και στους πελάτες. Ένα προτεινόμενο setup για Linux είναι το εξής:

```
$ ulimit -n 65536 # the maximum number of open file descriptors
$ ifconfig eth0 txqueuelen 8192 # replace eth0 with the ethernet interface you are using
$ /sbin/sysctl -w net.core.somaxconn=4096 # increase the size of the connection listening queue. Too high and you'll get resource problems as many will remain pending, and too low and you'll get refused connections.
$ /sbin/sysctl -w net.core.netdev_max_backlog=16384 # the size of the incoming packet queue for upper-layer (java) processing
$ /sbin/sysctl -w net.core.rmem_max=16777216 # receive socket memory
$ /sbin/sysctl -w net.core.wmem_max=16777216 # send socket memory
$ /sbin/sysctl -w net.ipv4.tcp_max_syn_backlog=8192 # how many SYN requests to keep in memory that we have yet to get the third packet in a 3-way handshake from
$ /sbin/sysctl -w net.ipv4.tcp_syncookies=1 # true
```

7.4.7.2. Εκτέλεση

Αυτό που χρειαζόμαστε είναι:

- JDK 5 ή παραπάνω, για να κάνουμε compile τον κώδικα Java
- Maven 3 ή παραπάνω, για το build

```
$ mvn clean install
$ cd cometd-demo
$ mvn jetty:run
```

Αν ανοίξουμε στο φυλλομετρητή τη διεύθυνση http://localhost:8080, πρέπει να φαίνεται η σελίδα CometD Demo. Εφόσον όλα πήγαν καλά ως εδώ, μπορούμε τώρα να τρέξουμε το δοκιμαστικό εξυπηρέτη.

```
$ cd cometd-java/cometd-java-examples
$ mvn -Pserver install exec:exec
```

Σε ένα άλλο μηχάνημα (στην περίπτωσή μας τερματικό), τρέχουμε τους πελάτες:

```
$ cd cometd-java/cometd-java-examples
$ mvn -Pclient install exec:exec
```

Το Pclient εξομοιώνει τους πελάτες σε μια εφαρμογή chat. Στην αρχή ζητείται να δοθούν τιμές για τις παρακάτω παραμέτρους:

```
server [localhost]:
port [8080]:
transports:
  0 - long-polling
  1 - websocket
transport [0]: 1
use ssl [false]:
max threads [256]:
context [/cometd]:
channel [/chat/demo]:
rooms [100]:
rooms per client [1]:
record latency details [true]:
-----
clients [100]: 1000
Waiting for clients to be ready...
Waiting for clients 999/1000
Clients ready
batch count [1000]:
batch size [10]: 1
batch pause (μs) [10000]:
message size [50]:
randomize sends [false]:
```

Στο default configuration συνδέονται 100 πελάτες στον εξυπηρέτη στο http://localhost:8080/cometd, έπειτα στέλνονται 1000 batches 10 μηνυμάτων (μεγέθους 50 bytes) με 10 ms παύση μετά από κάθε batch, επιλέγοντας 1 τυχαίο πελάτη ως αποστολέα του κάθε batch. Ο πελάτης στέλνει τα μηνύματα στα chat room όπου έχει εγγραφεί, τα μηνύματα φτάνουν στο server και ο server τα κάνει broadcast στους subscriber του chat room, οι οποίοι λαμβάνουν τελικά τα μηνύματα.

Εν τω μεταξύ μετράται η καθυστέρηση ανάμεσα στις αποστολές και τις λήψεις και στο τέλος παρουσιάζεται όπως παρακάτω:

```
Outgoing: Elapsed | Rate = 10273 ms | 973 messages/s - 97 requests/s
Waiting for messages to arrive 15815/15868
All messages arrived 15868/15868
Messages - Success/Expected = 15868/15868
Incoming - Elapsed | Rate = 10667 ms | 1487 messages/s
Messages - Latency Distribution Curve (X axis: Frequency, Y axis: Latency):
             @        _  13 ms (2436)
   @                  _  26 ms (581)
                  @   _  39 ms (3805)
                @     _  52 ms (3301)
         @            _  65 ms (1802)
        @             _  78 ms (1501)
     @                _  91 ms (1029)
   @                  _  104 ms (537)
 @                    _  117 ms (280)
 @                    _  130 ms (163)
 @                    _  142 ms (158)
 @                    _  155 ms (107)
@                     _  168 ms (76)
@                     _  181 ms (30)
@                     _  194 ms (17)
@                     _  207 ms (33)
```

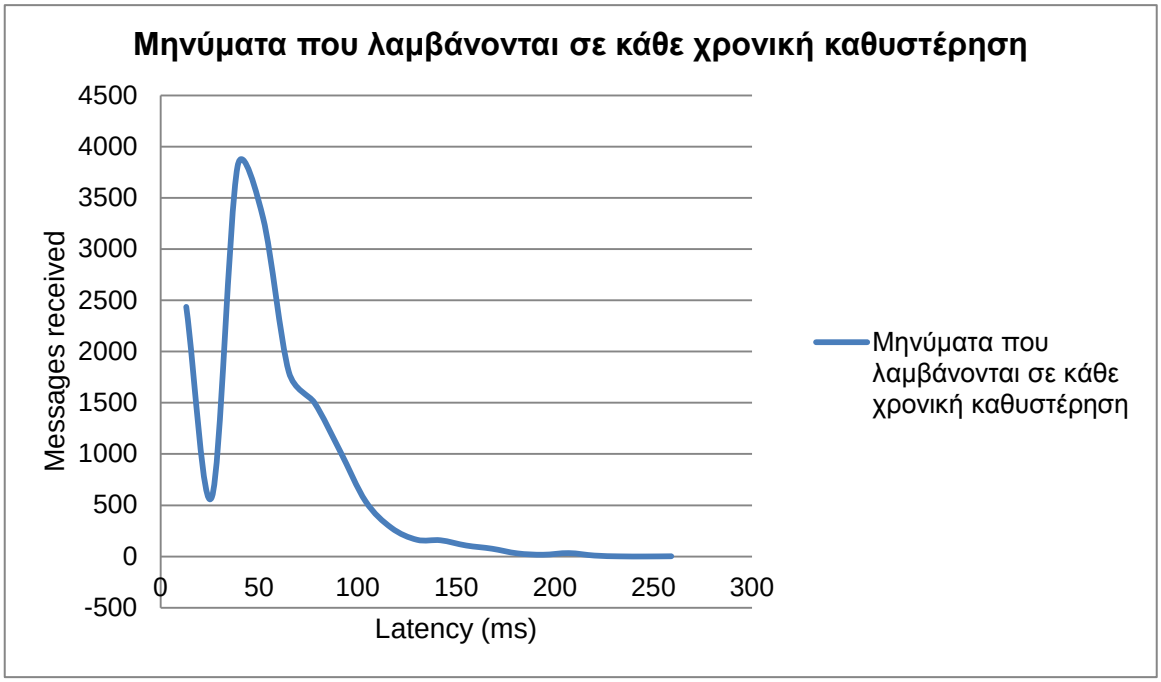
```
@                       _  220 ms (9)
@                       _  233 ms (1)
@                       _  246 ms (0)
@                       _  259 ms (2)
Messages - Latency Min/Ave/Max = 0/48/259 ms
```

Επιλέγοντας καθυστέρηση 10 ms ανάμεσα στα batches με 10 αποστολές κάθε φορά, αναμέναμε περίπου 1000 αποστολές μηνυμάτων/s και καταφέραμε 973 αποστολές/s.

Το πλήθος των μηνυμάτων εξαρτάται από το πόσοι πελάτες είναι εγγεγραμμένοι σε κάθε room. Στέλνοντας π.χ. 1 μήνυμα σε ένα chat room με 5 subscriber, θα γίνει broadcasted ένα μήνυμα πίσω σε κάθε subscriber. Στο παραπάνω παράδειγμα στείλαμε 1000 μηνύματα και λάβαμε 15868. Αυτό είναι λογικά αφού έχουμε 100 πελάτες εγγεγραμμένους τυχαία σε 100 room, οπότε κατά μέσο όρο έχουμε 1πελάτη/room (με κάποια δωμάτια άδεια και κάποια με πάνω από 1 συνδρομητές). Αν είχαμε 1 δωμάτιο, τότε θα στέλνονταν 10000*100 μηνύματα στους πελάτες.

Ο ρυθμός με τον οποίο επιστρέφουν τα μηνύματα είναι 1487 μηνύματα/s.

Η καμπύλη κατανομής της καθυστέρησης είναι καμπανοειδής με μια κορυφή στα 33ms περίπου (). Μέσα σε παρενθέσεις φαίνονται τα μηνύματα που λαμβάνονται σε κάθε χρονική καθυστέρηση.



Εικόνα 34: Καμπύλη κατανομής καθυστέρησης μηνυμάτων

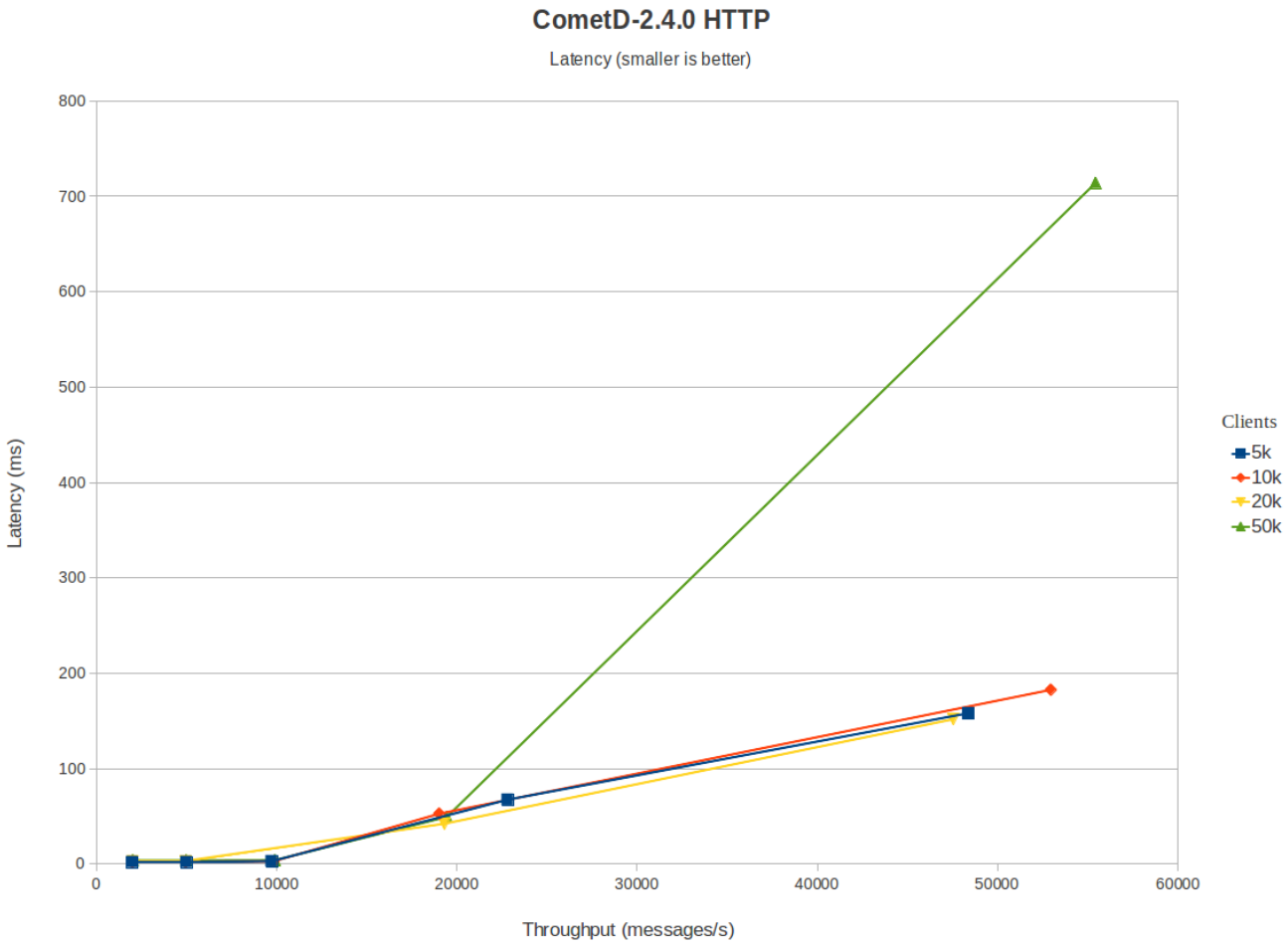
Τέλος, δίνονται η ελάχιστη, μέση και μέγιστη καθυστέρηση.

Στο ΠΑΡΑΡΤΗΜΑ Ι δίνονται τα πειραματικά αποτελέσματα που πήραμε εκτελώντας κατά γράμμα την παραπάνω δοκιμή. Παρατηρούμε ότι για τόσους λίγους πελάτες (1000) τα αποτελέσματα των τεχνικών COMET long polling και WebSocket είναι συγκρίσιμα.

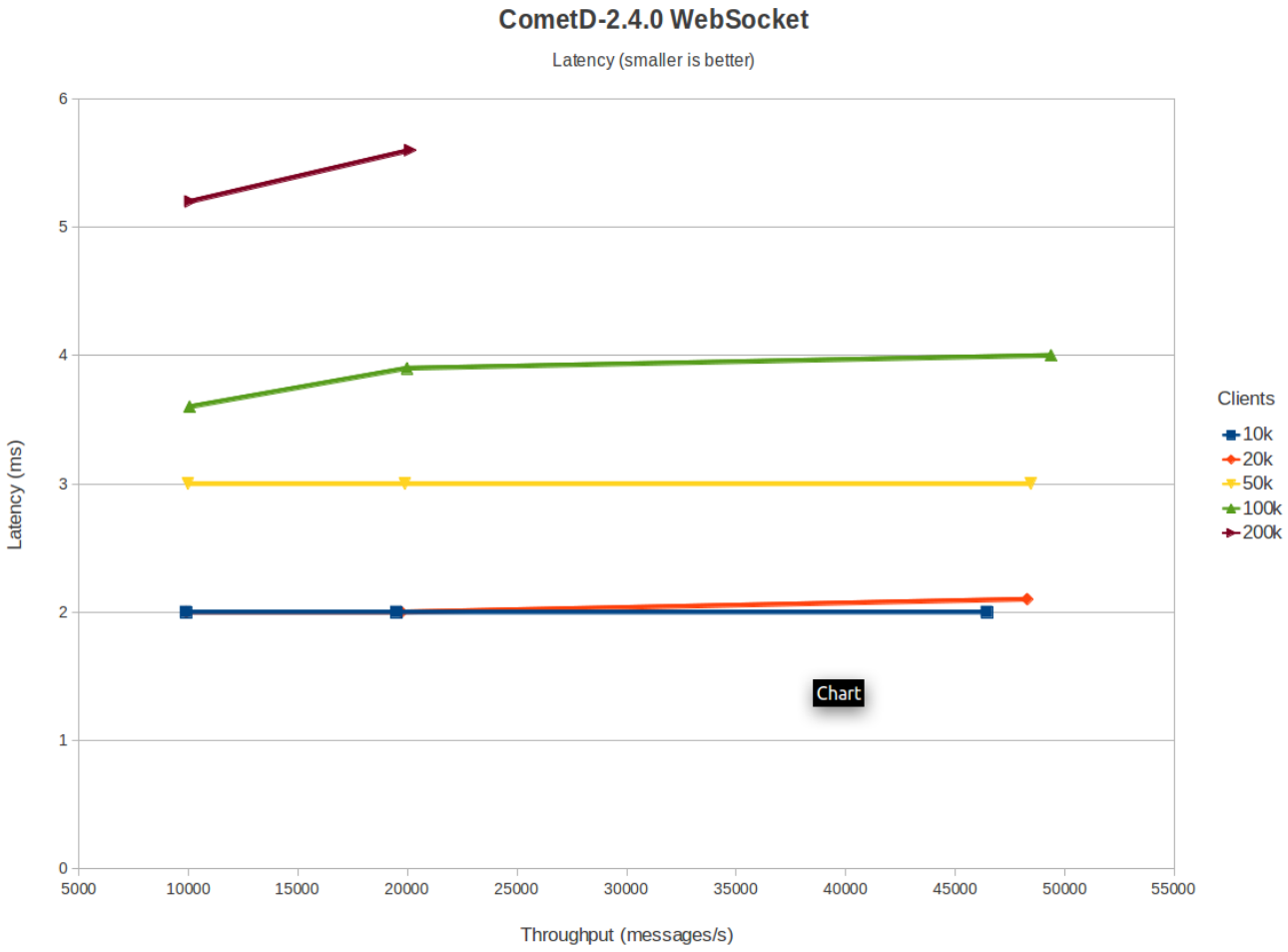
7.4.7.3. Αποτελέσματα

Τα πειραματικά αποτελέσματα [62] που έγιναν από τη WebTide έδωσαν τις γραφικές παραστάσεις των δύο παρακάτω εικόνων. Η Εικόνα 35 αναφέρεται σε χρήση long polling και η Εικόνα 36 σε χρήση WebSocket. Η κλίμακα που χρησιμοποιούν ως προς τους πελάτες και την καθυστέρηση είναι διαφορετική, οπότε δε γίνεται να μπουν στην ίδια γραφική παράσταση και οι δύο τεχνικές. Αν το κάναμε αυτό, τότε οι γραμμές του WebSocket θα έπεφταν στον άξονα των X...

Στο HTTP (long polling) η μέγιστη καθυστέρηση είναι 800 ms (άξονας των Y), ενώ στο WebSocket είναι 6 ms και, ενώ στο HTTP φτάνουμε στους 50.000 πελάτες ανά εξυπηρέτη, στο WebSocket πάμε στους 200.000.



Εικόνα 35: CometD με long polling



Εικόνα 36: CometD με WebSocket

Αν και αυτά τα αποτελέσματα δε δόθηκαν, στην ιστοσελίδα του CometD αναφέρεται πως έγιναν δοκιμές με διάφορα μεγέθη μηνυμάτων (50B, 500B, 2KB), αλλά τα αποτελέσματα έδειξαν πως το μέγεθος του payload έχει ελάχιστη, αν όχι μηδαμινή, επίδραση στις καθυστερήσεις.

Ακόμη η κατανάλωση μνήμης όταν οι πελάτες είναι σε κατάσταση αδράνειας (είναι συνδεδεμένοι, αλλά στέλνουν μόνο μηνύματα διατήρησης της σύνδεσης κάθε 30") έχει ως εξής:

- HTTP: οι 50.000 πελάτες απασχολούν περίπου 2,1 GB
- WebSocket: οι 50.000 πελάτες απασχολούν περίπου 1,2 GB και οι 200.000 περίπου 3,2 GB.

Σε κάθε περίπτωση είναι φανερό πως το WebSocket είναι πιο ελαφρύ πρωτόκολλο από το HTTP.

7.4.7.4. Ρυθμίσεις συστήματος

Εξυπηρέτης: EC2 instance τύπου “m2.4xlarge” (67 GB RAM, 8 cores Intel(R) Xeon(R) X5550 @2.67GHz) που έτρεχε Ubuntu Linux 11.04 (2.6.38-11-virtual #48-Ubuntu SMP 64-bit).

Πελάτες: 10 EC2 instances τύπου “c1.xlarge” (7 GB RAM, 8 cores Intel Xeon E5410 @2.33GHz) που έτρεχαν Ubuntu Linux 11.04 (2.6.38-11-virtual #48-Ubuntu SMP 64-bit).

Η **JVM** που χρησιμοποιήθηκε ήταν η Oracle’s Java HotSpot(TM) 64-Bit Server VM (build 21.0-b17, mixed mode) version 1.7.0 και για τον εξυπηρέτη και για τους πελάτες.

8. ΚΟΙΝΕΣ ΒΙΒΛΙΟΘΗΚΕΣ

Στο προηγούμενο κεφάλαιο είδαμε πώς επιτυγχάνονται οι δύο τρόποι επικοινωνίας Comet και WebSocket, δηλαδή οι δύο προτιμότεροι τρόποι δυναμικής προώθησης δεδομένων από τον εξυπηρέτη στον πελάτη μιας διαδικτυακής εφαρμογής, στην πλευρά του εξυπηρέτη. Για το Comet τα βασικότερα API είναι το Servlet 3.0 και το Jetty Continuations, ενώ, για το WebSocket υπάρχει μόνο native υποστήριξη και όχι κάποιο πρότυπο που να το υποστηρίζει. Οπότε, μια εφαρμογή, στη βέλτιστη περίπτωση, θα χρησιμοποιεί WebSocket μόνο αν αυτό υποστηρίζεται από το φυλλομετρητή του πελάτη. Το πρόβλημα βέβαια στην πλευρά του εξυπηρέτη είναι πως ο κώδικας για WebSocket δεν μπορεί να είναι συμβατός με άλλα containers πλην αυτού για το οποίο αναπτύχθηκε. Αν δεν υποστηρίζεται WebSocket, τότε, και πάλι στην καλύτερη περίπτωση, θα γίνει χρήση του Comet.

Είναι μάλλον δύσκολο να ξεκινήσει κάποιος από το μηδέν και να προγραμματίσει, για παράδειγμα, μια εφαρμογή σε JavaScript και Java που θα συνδέει όλες τις τεχνολογίες. Ακόμη και αν ο κώδικας για την πλευρά του πελάτη υλοποιηθεί με ευκολία, ο προγραμματιστής θα πρέπει να συνυπολογίσει και να προσθέσει στην εφαρμογή του τα *χρονικά όρια* (timeout), την *αποτυχία σύνδεσης* (connection failure), την *επανασύνδεση*, τις *επιβεβαιώσεις* (acknowledgement), τη *σειρά των μηνυμάτων* (ordering), το *buffering* και ούτω καθ' εξής. Τη λύση σε όλα αυτά δίνουν **κάποιες βιβλιοθήκες αφαίρεσης (abstraction library)**, μέσω των οποίων μπορεί να γίνει **διαφανής χρήση των Comet και WebSocket σε πολλαπλούς container**.

Το πρώτο κριτήριο επιλογής μιας βιβλιοθήκης είναι το μέγεθος και η ποιότητα της κοινότητας που την υποστηρίζει. Αν υπάρχει πολυπληθής και ποιοτική κοινότητα υποστήριξης, τότε αυτή η βιβλιοθήκη έχει μεγάλες πιθανότητες να μην εγκαταληφθεί. Έτσι, θα μπορούμε να ενημερώνουμε για περισσότερο καιρό την εφαρμογή μας και να λαμβάνουμε βοήθεια στη χρήση της βιβλιοθήκης, διαφορετικά θα καταλήξουμε με μια βιβλιοθήκη που πιθανότατα σε σύντομο χρονικό διάστημα δε θα καλύπτει τις ανάγκες μας και για την οποία δε θα μπορούμε να λάβουμε επιπλέον υποστήριξη, οπότε θα πρέπει να προχωρήσουμε σε μια εξ' ολοκλήρου νέα υλοποίηση.

Στη συνέχεια θα περιγράψουμε τρεις βασικές βιβλιοθήκες με πολύ καλές κοινότητες, τη Socket.IO, την Atmosphere και την CometD.

8.1. Socket.IO

Η *Socket.IO* [63] [64] είναι μια βιβλιοθήκη JavaScript η οποία χρησιμοποιείται στην πλευρά του πελάτη και παρέχει ένα ενιαίο API, παρόμοιο με το WebSocket, για να συνδέεται σε έναν απομακρυσμένο εξυπηρέτη και να στέλνει και να λαμβάνει ασύγχρονα μηνύματα. Η βιβλιοθήκη αυτή παρέχει ένα κοινό API, το οποίο υποστηρίζει διάφορες τεχνικές μεταφοράς: **WebSocket, Flash Socket, Long Polling, HTTP Streaming, hidden Iframe και JSONP polling**. Η Socket.IO ανιχνεύει τις δυνατότητες του φυλλομετρητή και προσπαθεί να επιλέξει, μέσω *failure detection*, την καλύτερη διαθέσιμη τεχνική. Είναι **συμβατή με σχεδόν όλους τους φυλλομετρητές**, συμπεριλαμβανομένων των παλιών (όπως ο IE 5.5) και των mobile (Πίνακας 10). Επίσης έχει χαρακτηριστικά, όπως τα πακέτα κτύπου καρδιάς (heartbeat), το χρονικό όριο, η αποσύνδεση και η αντιμετώπιση των λαθών (error handling).

Στην πλευρά του εξυπηρέτη η Socket.IO αρχικά απαιτούσε ύπαρξη του *Node.js*. Το NodeJS επιτρέπει την ενοποίηση των API του πελάτη και του εξυπηρέτη. Με το NodeJS δημιουργείται αρχικά ένας τυπικός HTTP εξυπηρέτης και στη συνέχεια το στιγμιότυπό του περνά στο Socket.IO. Από εκεί και πέρα μπορούν να δημιουργηθούν συνδέσεις, να γίνουν αποσυνδέσεις και να φτιαχτούν message listeners όμοια με την πλευρά του πελάτη.

Εκτός από το Node.js, η Socket.IO μπορεί να συνδυαστεί και με JVM εξυπηρέτες. Μάλιστα χρησιμοποιώντας το Socket.IO σε συνδυασμό με το πλαίσιο Atmosphere (Παράγραφος 8.3) ο προγραμματιστής δε χρειάζεται να ασχοληθεί πλέον με τους διαφορετικούς τρόπους μεταφοράς και την υλοποιησή τους όχι μόνον στους φυλλομετρητές, αλλά ούτε και στον εξυπηρέτη.

https://github.com/Atmosphere/atmosphere/wiki/Getting-Started-with-Socket.IO

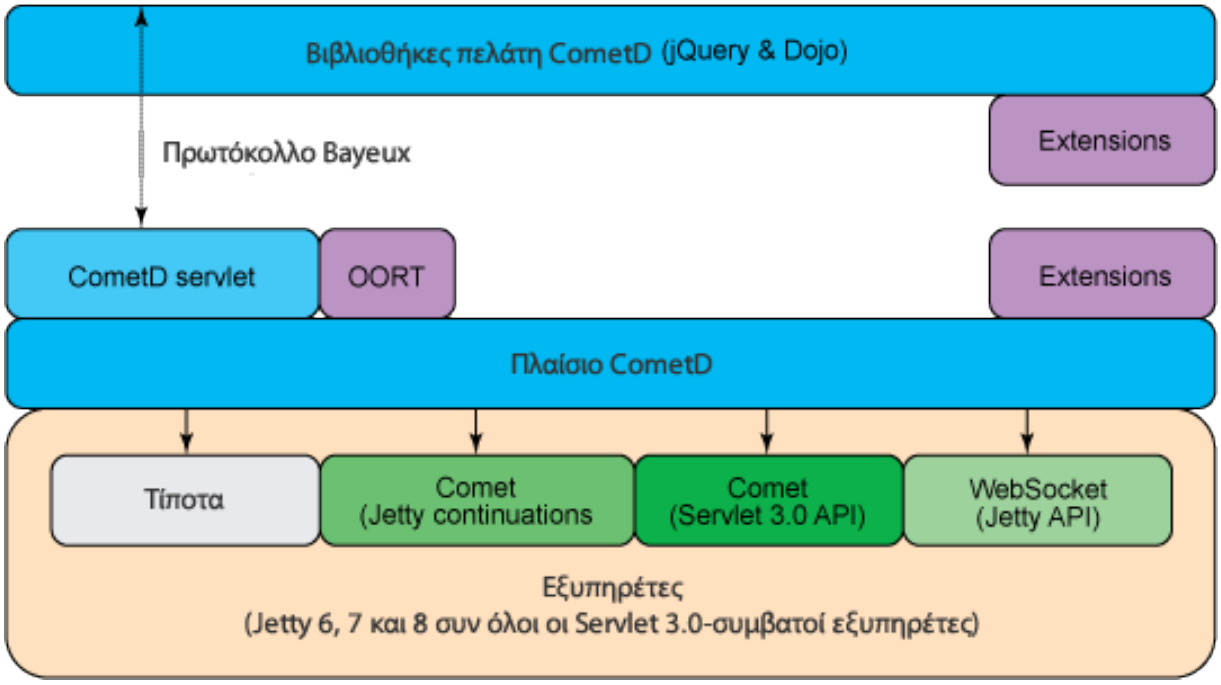
Πίνακας 10: Υποστήριξη φυλλομετρητών από τη βιβλιοθήκη Socket.IO

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Opera Mobile	Android Browser	
πριν 11 εκδόσεις			4.0							
πριν 10 εκδόσεις			5.0							
πριν 9 εκδόσεις			6.0							
πριν 8 εκδόσεις		2.0	7.0							
πριν 7 εκδόσεις		3.0	8.0		9.0					
πριν 6 εκδόσεις		3.5	9.0		9.5-9.6					
πριν 5 εκδόσεις		3.6	10.0		10.0-10.1					
πριν 4 εκδόσεις	5.5	4.0	11.0	3.1	10.5					
πριν 3 εκδόσεις	6.0	5.0	12.0	3.2	10.6	3.2		10.0		
πριν 2 εκδόσεις	7.0	6.0	13.0	4.0	11.0	4.0-4.1		11.0	2.1	
προηγούμενη έκδοση	8.0	7.0	14.0	5.0	11.1	4.2-4.3		11.1	2.2	
τωρινή έκδοση	9.	8.0	15.0	5.1	11.5	5.0	5.0-6.0	11.5	2.3,	3.0
σύντομα	9.0	9.0	16.0	5.1	11.6				4.0	
μελλοντικά	10.0	10.0	17.0	6.0	12.0					

8.2. CometD

Το πλαίσιο *CometD* [65], αποτελεί μια event-driven λύση επικοινωνίας που βασίζεται στο HTTP. Υπάρχει εδώ και χρόνια και η Έκδοση 2 προσέθεσε υποστήριξη για τη διαμόρφωση σχολιασμού (annotation configuration) και για WebSocket. Το πλαίσιο CometD (Εικόνα 37) δίνει ένα *κομμάτι εξυπηρέτη*, ένα *κομμάτι πελάτη* καθώς και βιβλιοθήκες πελάτη βασισμένες σε jQuery και Dojo. Το CometD χρησιμοποιεί το πρωτόκολλο επικοινωνίας **Bayeux**, επιτρέποντας στους προγραμματιστές να ενεργοποιούν κάποια extension για τα μηνύματα επιβεβαίωσης, τον έλεγχο ροής, το συγχρονισμό, το ομαδοποίηση και ούτω καθ' εξής.

Το CometD συνδέεται με τρεις τύπους μεταφοράς: JSON, JSONP και WebSocket. Αυτές εξαρτώνται από το *Jetty Continuations* και το *WebSocket API του Jetty*. Το CometD μπορεί να χρησιμοποιηθεί **σε Jetty 6, 7 και 8 καθώς και σε κάθε άλλο εξυπηρέτη που υποστηρίζει την Προδιαγραφή Servlet 3.0**. *Μπορούν όμως να προστεθούν και να αναπτυχθούν και άλλες μεταφορές*, όπως γίνεται και με τα extension. Για παράδειγμα ο προγραμματιστής θα μπορούσε να γράψει μια μεταφορά που υποστηρίζει το WebSocket API του Grizzly και να την προσθέσει κατά το configuration του εξυπηρέτη.



Εικόνα 37: Αρχιτεκτονική πλαισίου CometD

Η event-driven προσέγγιση του CometD ταιριάζει πολύ καλά στην ιδέα της event-driven ανάπτυξης διαδικτυακών εφαρμογών. Όπως συμβαίνει και στις παραδοσιακές διεπαφές χρήστη, τα στοιχεία (component) επικοινωνούν μέσω ενός *bus* για να στείλουν και να λάβουν γεγονότα. Επομένως, **όλη η επικοινωνία είναι ασύγχρονη**.

Το πλαίσιο CometD έχει τα εξής θετικά χαρακτηριστικά:

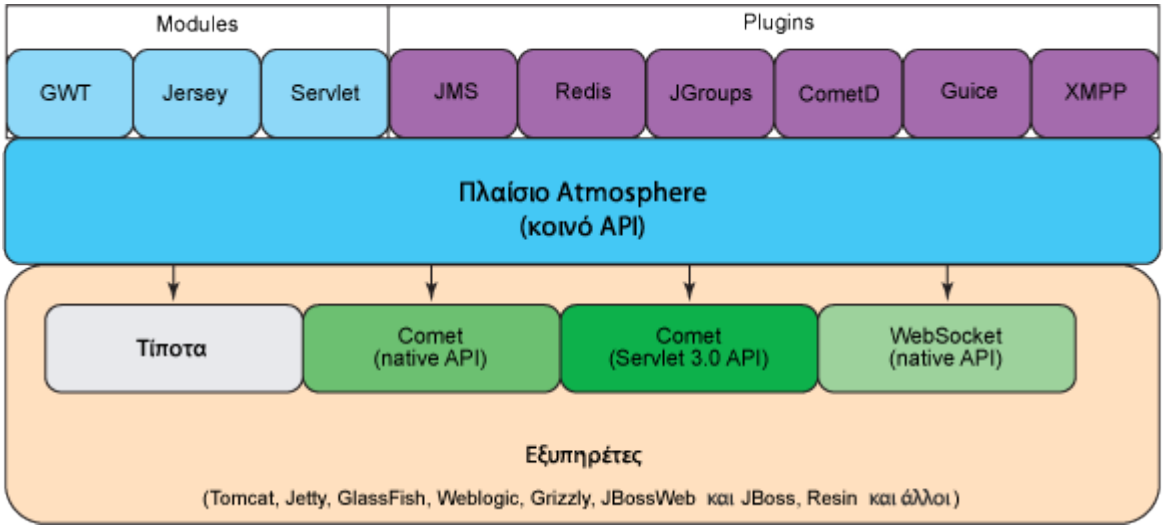
- ✓ Είναι καλά τεκμηριωμένο.
- ✓ Παρέχει δείγματα και αρχέτυπα Maven για να διευκολύνει την έναρξη ενός project.
- ✓ Έχει ένα καλά σχεδιασμένο API, το οποίο επιτρέπει την ανάπτυξη extension και επιπρόσθετων μεταφορών.
- ✓ Παρέχει ένα module για clustering, το Oort, το οποίο δίνει τη δυνατότητα να τρέξουν πολλοί εξυπηρέτες CometD ως κόμβοι σε ένα cluster πίσω από ένα σύμπλεγμα εξισορρόπησης φορτίου (load balancer), ώστε να μπορεί να γίνει κλιμάκωση σε ένα μεγαλύτερο αριθμό HTTP συνδέσεων.
- ✓ Υποστηρίζει πολιτικές ασφαλείας που επιτρέπουν λεπτομερή ρύθμιση του ποιος μπορεί να στέλνει μηνύματα πάνω από ποιο κανάλι.
- ✓ Ενοποιείται αρκετά καλά με το Spring και το Google Guice (Dependency Injection frameworks).

8.3. Atmosphere

Το πλαίσιο Atmosphere [66] είναι ένα πλαίσιο Java που παρέχει ένα κοινό API για τη χρήση Comet και WebSocket σε πολλούς εξυπηρέτες συμπεριλαμβανομένων των **Tomcat, Jetty, GlassFish, Weblogic, Grizzly, JBossWeb, JBoss και Resin**. Ακόμη υποστηρίζεται κάθε εξυπηρέτης ο οποίος υποστηρίζει την Προδιαγραφή Servlet 3.0. Σκοπός του είναι το Inversion of Control (IoC).

Το Atmosphere μπορεί να ανιχνεύσει το native API του εξυπηρέτη (για Comet και WebSockets) και στην περίπτωση του Comet να χρησιμοποιήσει τελικά το Servlet 3.0, εάν υποστηρίζεται. Αλλιώς θα πάει σε μια «διαχειριζόμενη» ασύγχρονη λειτουργία η οποία δεν είναι τόσο κλιμακώσιμη όσο η Jetty Continuations.

Το Atmosphere χρησιμοποιείται εδώ και δύο χρόνια και είναι ακόμη ενεργό. Στην Εικόνα 38 βλέπουμε την αρχιτεκτονική του.



Εικόνα 38: Αρχιτεκτονική πλαισίου Atmosphere

Το Atmosphere *δε χρησιμοποιεί κάποιο συγκεκριμένο πρωτόκολλο ανάμεσα στον πελάτη και τον εξυπηρέτη*, όπως η βιβλιοθήκη Socket.IO ή η CometD. Οι βιβλιοθήκες Socket.IO και CometD έχουν έναν πελάτη σε JavaScript και ένα servlet στον εξυπηρέτη, τα οποία επικοινωνούν χρησιμοποιώντας ένα συγκεκριμένο πρωτόκολλο (για τη χειραψία, την ανταλλαγή μηνυμάτων, τις επιβεβαιώσεις και τα πακέτα κτύπου καρδιάς). Στόχος του Atmosphere είναι να παρέχει ένα κοινό κανάλι επικοινωνίας στην πλευρά του εξυπηρέτη. **Αν η εφαρμογή χρειάζεται να χρησιμοποιεί κάποιο συγκεκριμένο πρωτόκολλο, όπως το Bayeux, τότε θα πρέπει ο προγραμματιστής να αναπτύξει το δικό του handler**. Το *plugin για το CometD* κάνει ακριβώς αυτό: εκμεταλλεύεται το API του Atmosphere για να κάνει suspend και resume τις αιτήσεις και αναθέτει στις κλάσεις του CometD τη διαχείριση της CometD επικοινωνίας χρησιμοποιώντας το πρωτόκολλο Bayeux.

Το Atmosphere περιλαμβάνει μια *jQuery βιβλιοθήκη πελάτη*, για να διευκολύνει την εγκατάσταση της σύνδεσης, η οποία είναι σε θέση *να ανιχνεύει αυτόματα την καλύτερη διαθέσιμη τεχνική μεταφοράς* (*WebSocket ή CometD*). Η χρήση του jQuery plugin του Atmosphere είναι παρόμοια αυτή του WebSocket API. Στην αρχή γίνεται η σύνδεση με τον εξυπηρέτη, καταχωρείται (register) η μέθοδος επανάκλησης για να λαμβάνονται τα μηνύματα και στη συνέχεια γίνεται προώθηση των δεδομένων.

Στα θετικά του Atmosphere συγκαταλέγονται τα εξής δύο:

- ✓ Αν πρέπει να γίνει ανάπτυξη μιας εφαρμογής η οποία θα μπορεί να χρησιμοποιηθεί από πολλούς εξυπηρέτες οι οποίοι δεν είναι γνωστοί εκ των προτέρων, τότε είναι πιο πιθανό να δουλέψει σωστά αν υλοποιείται πάνω από Atmosphere. Αυτό ισχύει λόγω του πλήθους των εξυπηρετών που υποστηρίζει το πλαίσιο.
- ✓ Επίσης, το Atmosphere αποτελεί μια καλή επιλογή όταν χρειάζεται ένα κοινό API πάνω από καθαρή Reverse Ajax επικοινωνία, χωρίς τον ορισμό κάποιου πρωτοκόλλου, επειδή θέλουμε να την αναπτύξουμε ή να την επεκτείνουμε.

Βλέπουμε όμως και τα εξής αρνητικά:

- × Το κακό είναι πως το Atmosphere έχει ελλείψεις ως προς το documentation της αρχιτεκτονικής, του project, των concept και των API. Το documentation είναι πολύ χρήσιμο όταν πρέπει να δεις τον πηγαίο κώδικα ή να αναλύσεις τα δείγματα που δίνονται. Το API είναι πολύ τεχνικό και κάποιες φορές ασαφές, σε σύγκριση με τα API των άλλων πλαισίων (π.χ. Socket.IO και CometD).
- × Αν και υπάρχει μια καλή αφαίρεση στην πλευρά του εξυπηρέτη, *δεν υπάρχει κάποια καλή βιβλιοθήκη πελάτη*. Δεν υπάρχει κανένα πρωτόκολλο, οπότε όλα τα επιπλέον χαρακτηριστικά αφήνονται στον προγραμματιστή. Η τρέχουσα βιβλιοθήκη είναι πολύ απλή για τις ανάγκες μιας μεγάλης, επεκτάσιμης διαδικτυακής εφαρμογής η οποία μπορεί να χρειάζεται προχωρημένη αναγνώριση των timeout, ενημερώσεις, back-off, cross-domain κοκ, ειδικά όταν εργαζόμαστε με κινητές συσκευές. Σε αυτή την περίπτωση, είναι πολύ πιο αξιόπιστο το CometD. Το CometD χρησιμοποιεί ένα πρωτόκολλο επικοινωνίας με το οποίο μπορούμε να έχουμε έλεγχο ροής και εντοπισμό σφαλμάτων.

Ο Πίνακας 11 δείχνει συγκεντρωμένα ποιες τεχνικές αυτόματης προώθησης γεγονότων χρησιμοποιεί η κάθε βιβλιοθήκη.

Πίνακας 11: Τεχνικές αυτόματης προώθησης στο Διαδίκτυο που υποστηρίζονται από κάθε βιβλιοθήκη

Τεχνική Μεταφοράς	JSONP Polling	Hidden Iframe	Multipart XHR	Long Polling	Adobe Flash Socket	WebSocket
Βιβλιοθήκη						
Socket.IO						
Atmosphere						
CometD						

Βασιζόμενοι στις παραπάνω περιγραφές καταλήγουμε στο συμπέρασμα ότι θα χρησιμοποιούσαμε το Atmosphere αν θέλαμε να αναπτύξουμε μια εφαρμογή την οποία θα μπορούσαν να σηκώσουν πολλοί διαφορετικοί εξυπηρέτες (υποστήριξη φορητότητας). Αν η ίδια εφαρμογή χρειαζόταν καλύτερη βιβλιοθήκη από αυτή που μπορεί να προσφέρει το Atmosphere στην πλευρά του πελάτη, τότε θα μπορούσε να χρησιμοποιηθεί το Socket.IO. Εναλλακτικά θα πορούσε να χρησιμοποιηθεί ο CometD JavaScript πελάτης μαζί με το Atmosphere CometD plugin. Αν ο εξυπηρέτης είναι δεδομένος - και αν ακόμη καλύτερα είναι ο Jetty στον οποίο καταλήξαμε στο προηγούμενο κεφάλαιο -, τότε η καλύτερη λύση είναι η βιβλιοθήκη CometD.

9. ΥΛΟΠΟΙΗΣΕΙΣ COMET-WEBSOCKET

Σκοπός αυτού του κεφαλαίου δεν είναι να δώσει έναν εξαντλητικό κατάλογο των υλοποιήσεων οι οποίες γίνονται αποκλειστικά με στόχο την υποστήριξη Comet ή WebSocket επικοινωνίας. Σκοπός είναι να γίνει μια σύντομη επισκόπηση των διαφόρων υλοποιήσεων όπου θα φανούν επιγραμματικά οι γλώσσες που υποστηρίζονται και οι δυνατότητες των διάφορων εξυπηρετητών.

Κανένας από τους δύο τρόπους επικοινωνίας δεν υποστηρίζεται πλήρως από όλους τους φυλλομετρητές ή χωρίς προβλήματα σε όλα τα δίκτυα. Εφόσον η τεχνολογία αυτή δεν είναι αρκετά ώριμη, αν θέλουμε να αναπτύξουμε μια εφαρμογή που θα δουλεύει σε όσο το δυνατόν περισσότερους φυλλομετρητές και χωρίς να αλλάξουμε εξυπηρέτη, τότε θα επιλέξουμε κάποιον Application Server και τη χρήση μιας κοινής βιβλιοθήκης. Αν από την άλλη θέλουμε να φτιάξουμε μια εφαρμογή που θα τρέχει σε πολύ συγκεκριμένα μηχανήματα και θέλουμε να πετύχουμε τη μέγιστη απόδοση, τότε για Comet και WebSocket επικοινωνία, θα χρησιμοποιήσουμε κάποιο Dedicated Server και ίσως κάποια βιβλιοθήκη.

9.1. Comet

Αν η υπό ανάπτυξη εφαρμογή αναμένεται να υποστηρίζει μόνο COMET επικοινωνία, τότε ο σχεδιαστής έχει στη διάθεσή του και μπορεί να επιλέξει κάποιες από τις υλοποιήσεις που ακολουθούν. Πρόκειται για εξυπηρέτες, βιβλιοθήκες και add-on. Κάποιοι από τους εξυπηρέτες είναι δωρεάν και κάποιοι όχι. Δεν έχει γίνει κάποια συγκριτική αξιολόγησή τους, αλλά όλοι, σχεδόν, δίνουν κάποια νούμερα σχετικά με την απόδοσή τους και τις διάφορες λειτουργίες που υποστηρίζουν π.χ. ως προς την ασφάλεια και την επανασύνδεση. Η επιλογή του κατάλληλου εξαρτάται από τις απαιτήσεις της υπό ανάπτυξη εφαρμογής.

9.1.1. WebSync

Η *WebSync* [67] είναι μια πλήρης εφαρμογή του πρωτοκόλλου **Bayeux** για την πλατφόρμα της **Microsoft** (.NET / IIS / SQL), η οποία επιτρέπει πλήρως αμφίδρομη επικοινωνία σε όλα τα **μεγάλα προγράμματα περιήγησης** (IE, Firefox, Chrome, Opera, Safari, Maxthon, Konqueror, Epiphany), στις **κινητές συσκευές** (iPhone, Android), καθώς και σε **πλαίσια (framework) κοινών γλωσσών**. Η WebSync κλιμακώνεται τόσο κάθετα όσο και οριζόντια και

είναι σε θέση να υποστηρίξει δεκάδες χιλιάδες ταυτόχρονους χρήστες σε κάθε server. Διαθέτει ενσωματωμένη υποστήριξη για φάρμες εξυπηρετών με χειρισμό της αυτόματης μετάβασης (failover). Η WebSync είναι εύκολη στη χρήση και παρέχει ένα απλό API για να κάνεις publish με **PHP** ή χρησιμοποιώντας ένα πελάτη σε **JavaScript, iOS ή .NET** για ελαφριά (thin) και βαριά (thick) ολοκλήρωση εφαρμογών. Είναι 100% συμβατή με τα πρότυπα, κάνοντας χρήση των χαρακτηριστικών της HTML5 στην περίπτωση που υποστηρίζονται από το φυλλομετρητή. Είναι διαθέσιμη τόσο ως **add-on module για το IIS** (εξυπηρέτης) και ως **υπηρεσία on demand** (SaaS). Το πρόβλημα είναι **πως η ελεύθερη έκδοση επιτρέπει μέχρι 10 ταυτόχρονους πελάτες**. Αν η εφαρμογή απαιτεί περισσότερους, τότε θα πρέπει να γίνει αγορά του προϊόντος.

9.1.2. APE

Η *Ajax Push Engine* (APE) [68] είναι μία ολοκληρωμένη λύση **ανοικτού κώδικα** η οποία περιέχει έναν streaming εξυπηρέτη. Σκοπός της είναι η προώθηση δεδομένων σε πραγματικό χρόνο με έναν ελαφρύ, υψηλής κλιμάκωσης και σπονδυλωτό τρόπο, χρησιμοποιώντας **μόνο τη Javascript** για την πλευρά του πελάτη. Βασιζόμενη στα Web Stardard, η APE είναι **πλήρως cross-browser (δουλεύει και σε κινητές συσκευές) και cross-subdomain**, χρησιμοποιεί τα νεότερα χαρακτηριστικά των φυλλομετρητών και παρέχει προς τα πίσω συμβατότητα για τα παλαιότερα. Ο APE Server είναι γραμμένος σε γλώσσα **C**, εγκαθίσταται σε **Linux** και έχει μικρό αποτύπωμα (footprint)

Η APE υποστηρίζει τις παρακάτω μεθόδους μεταφοράς:

- Long Polling (προκαθορισμένη)
- JSONP (για προώθηση σε διαφορετικά domain)
- XHRStreaming (για καλύτερη απόδοση και χαμηλή καθυστέρηση)
- WebSockets (για βέλτιστη απόδοση σε FF4, Chrome, Safari 5)

Ο APE **δεν μπορεί να αντικαταστήσει τους application server**. Είναι ένας streaming server που υλοποιεί τις μεθόδους POST και GET του HTTP πρωτοκόλλου και **καλείται μόνο όταν γίνονται επίμονες αιτήσεις AJAX**.

9.1.3. NGiNX HTTP Push Module

Το Nginx HTTP Push Module (NHPM) [69] είναι ένα **ελεύθερο** add-on για τον εξυπηρέτη Nginx [70], το οποίο τον μετατρέπει σε ένα **HTTP Push και Comet εξυπηρέτη**. Φροντίζει για όλα όσα αφορούν στη σύνδεση και δίνει μια απλή διεπαφή για τη μετάδοση μηνυμάτων στους πελάτες μέσω των απλών παλιών HTTP αιτήσεων. Αυτό επιτρέπει εύκολη ανάπτυξη ασύγχρονων δικτυακών εφαρμογών ζωντανής ενημέρωσης, δεδομένου ότι ο κώδικας δε χρειάζεται να διαχειρίζεται τις αιτήσεις με καθυστερημένες απαντήσεις. Το NHPM υλοποιεί το **Basic HTTP Push Relay Protocol** [71], το οποίο είναι απλούστερο από το Bayeux, και μπορεί να χρησιμοποιηθεί όταν δε θέλουμε να ασχοληθούμε με το CometD. Οι πελάτες γράφονται σε **JavaScript**.

9.1.4. Caplin Liberator

Ο *Liberator* [72] είναι η δωρεάν έκδοση του εμπορικού Comet εξυπηρέτη της εταιρία Caplin Systems. Μπορεί να διαχειριστεί **δεκάδες χιλιάδες ταυτόχρονους χρήστες** σε έναν μόνο εξυπηρέτη, ο οποίος μπορεί να στείλει μέχρι και **τέσσερα εκατομμύρια μηνύματα ανά δευτερόλεπτο** με πολύ χαμηλή καθυστέρηση μηνύματος. Είναι ένας **αυτόνομος εξυπηρέτης** που τρέχει σε **Linux**, και **χρησιμοποιείται ευρέως για οικονομικές εφαρμογές**. **Υποστηρίζει Ajax, Flex, Silverlight, Java και .Net πελάτες**.

Ο Caplin Liberator έχει καλή κλιμάκωση, είναι ανεκτικός σε σφάλματα, μπορεί να χειριστεί το failover και διαφορετικούς χρόνους σύνδεσης στο Internet. Ακόμη δεν αντιμετωπίζει πρόβλημα με τους router και τα firewall και μπορεί να δουλέψει με τη μέγιστη ποικιλία φυλλομετρητών.

9.1.5. Lightstreamer

Ο *Lightstreamer* [73] είναι ένας **αυτόνομος εξυπηρέτης** για HTTP Streaming. Τρέχει σε οποιαδήποτε πλατφόρμα είναι συμβατή με Java SE 5 και επάνω, υποστηρίζει HTML5 και παρέχει βιβλιοθήκες πελάτη σε **Java και .NET** για τις desktop εφαρμογές και σε **iOS, Android, BlackBerry, Windows Phone και Java ME** για τις φορητές πλατφόρμες. Υλοποιεί τις Java NIO κλάσεις και έτσι επιτυγχάνει καλή κλιμάκωση.

9.1.6. Migratory Push Server

Ο Migratory Push Server [59] είναι ένας **αυτόνομος Comet/WebSocket εξυπηρέτης** υψηλής απόδοσης, ανεκτικός στα σφάλματα και αξιόπιστος ακόμη και στην περίπτωση failover.

Η έκδοση αξιολόγησης του Migratory Push Server επιτρέπει μέχρι 100 ταυτόχρονους χρήστες και μπορεί να χρησιμοποιηθεί για 90 ημέρες. Διαθέτει πακέτα εγκατάστασης για Windows, Linux και Unix και μπορεί να υποστηρίξει πελάτες σε **JavaScript, ActionScript (Flash, Flex, Air) και Silverlight** για τις σταθερές πλατφόρμες και **σε Java J2SE & Android, iOS, Java J2ME & BlackBerry και .NET Compact Framework / Windows Mobile** για τις κινητές συσκευές.

Προσφέρει εξαιρετική κάθετη κλιμάκωση υποστηρίζοντας έως και **1.000.000 ταυτόχρονους χρήστες** σε έναν μόνο εξυπηρέτη. Επιπλέον, το σύστημα κλιμακώνεται οριζόντια με την ενσωματωμένη εξισορρόπηση φορτίου, ώστε να μπορεί να ανταποκριθεί σε οποιαδήποτε αύξηση στον αριθμό των πελατών.

9.1.7. Wt

Η Wt [74] είναι μια **C++** βιβλιοθήκη για την ανάπτυξη stateful και υψηλά διαδραστικών διαδικτυακών εφαρμογών. Μεταξύ άλλων περιέχει ένα απλό API με μια robust **cross-browser** υλοποίηση Server Push που κάνει χρήση του **Comet** ή του **WebSocket**. Κάνει αυτόματα σταδιακή υποβάθμιση (degradation) ή αναβάθμιση (enhancement), αναλόγως με το τι υποστηρίζει ο φυλλομετρητής του χρήστη (π.χ. υποστήριξη JavaScript ή μόνο απλή HTML). Η βιβλιοθήκη συνοδεύεται από έναν application server ο οποίος δρα είτε ως αυτόνομος Http(s)/WebSocket server ή ενσωματώνεται με άλλους server μέσω FastCGI.

Εκτός από τη C++, η Wt έχει υλοποιηθεί ή έχει συνδέσεις (binding) και στις εξής γλώσσες:

- native **Java** version (JWt)
- **Ruby** bindings (WtRuby)
- χρήση της JWt μέσα από **Clojure**, που είναι παραλλαγή της Lisp
- χρήση της JWt μέσα από **Jython**.

9.1.8. StreamHub

Ο StreamHub Comet Server [75] είναι ένας **cross-platform αυτόνομος εξυπηρέτης** αυτόματης προώθησης ο οποίος δίνει ένα **περιθώριο αξιολόγησης 60 ημερών στις εμπορικές του εκδόσεις και μια ελεύθερη έκδοση η οποία επιτρέπει μέχρι 10 ταυτόχρονους χρήστες**. Μπορεί να διαχειριστεί δεκάδες χιλιάδες χρήστες, ή εκατοντάδες χιλιάδες, όταν λειτουργούν ως μέρος ενός συμπλέγματος (cluster). Χρησιμοποιείται κυρίως σε οικονομικές εφαρμογές χαμηλής καθυστέρησης. Προς το παρόν υποστηρίζει **AJAX, Java, .NET, iPhone, Silverlight και GWT πελάτες**.

9.2. WebSocket

Στην περίπτωση που η εφαρμογή θα υλοποιηθεί μόνο με websocket, τότε υπάρχουν οι παρακάτω λύσεις (εξυπηρέτες και βιβλιοθήκες), οι οποίες είναι ως επί τω πλείστω πειραματικές.

9.2.1. Xsockets

Το Xsockets [76] είναι ένα framework το οποίο επιτρέπει την ανάπτυξη WebSocket στην πλατφόρμα **Microsoft.NET**. Υποστηρίζει διάφορες εκδόσεις του πρωτοκόλλου WebSocket και έχει υποστήριξη fallback η οποία δουλεύει στους περισσότερους φυλλομετρητές με τη βοήθεια Flash ή Silverlight plugin.

9.2.2. The Server Framework

Το Server Framework είναι ένα σύνολο **C++** βιβλιοθηκών για τα Microsoft Windows που επιτρέπουν την ανάπτυξη συστημάτων πελάτη και εξυπηρέτη. Το WebSockets Option Pack [77] δίνει τη δυνατότητα στις εφαρμογές να κάνουν establish και accept συνδέσεις WebSocket.

9.2.3. WebSocket++

Το websocket++ [78] είναι μια open source **C++** υλοποίηση του WebSocket η οποία περιλαμβάνει performance και stress testing εργαλεία.

9.2.4. EM-WebSocket

Ο EM-WebSocket [79] είναι EventMachine based, async, **Ruby** WebSocket εξυπηρέτης.

9.2.5. em-ws-client

Ο em-ws-client [80] είναι ένας WebSocket client γραμμένος σε **Ruby** που είναι συμβατός με το RFC 6455.

9.2.6. jwebSocket

Το jWebSocket [81] είναι μια open source Java και JavaScript υλοποίηση του WebSocket. Το jWebSocket package περιέχει τα εξής:

- *jWebSocket Server* **Java** WebSocket εξυπηρέτης για server-to-client (S2C) streaming και server controlled (C2C) client-to-client-communication.
- *jWebSocket Clients* **JavaScript** WebSocket πελάτης με διάφορα υποπρωτόκολλα και προαιρετική διαχείριση χρήστη, session και timeout. Δεν απαιτούνται plug-ins.
- *jWebSocket FlashBridge* **Flash** WebSocket Wrapper για cross-browser compatibility. Απαιτείται Flash plug-in.

9.2.7. WebSocket-Node

Η WebSocket-Node [82] είναι μία **JavaScript** υλοποίηση του WebSocket για τον εξυπηρέτη **Node.js** [83].

9.2.8. cWebsocket

Η cWebsocket [84] είναι μια ελαφριά βιβλιοθήκη εξυπηρέτη websocket γραμμένη σε **C**. Η βιβλιοθήκη αυτή περιλαμβάνει συναρτήσεις για εύκολη ανάπτυξη του εξυπηρέτη. Υλοποιεί το protocol draft 76 του websocket.

9.2.9. libwebsockets

Η libwebsockets [85] είναι μια ελαφριά βιβλιοθήκη GPL2 http και websocket εξυπηρέτη γραμμένη σε **C**.

9.2.10. pywebsocket

Το pywebsocket project [86] παρέχει ένα WebSocket εξυπηρέτη σε **Python** καθώς και ένα WebSocket extension για τον Apache HTTP Server, το mod_pywebsocket. Το pywebsocket έχει φτιαχτεί για ερευνητικούς και πειραματικούς σκοπούς.

9.2.11. ws4py

Το ws4py (WebSocket for Python) [87] είναι μια βιβλιοθήκη **Python** που υλοποιεί το WebSocket πρωτόκολλο όπως ορίζεται στο RFC 6455 [88] . Η απόδοσή του δεν είναι πολύ καλή όταν δουλεύει υπό φορτίο. Ο λόγος για αυτό είναι ότι σχεδιάστηκε για να υλοποιεί το πρωτόκολλο με απλότητα...

9.2.12. tornado.websocket

Το tornado.websocket [89] αποτελεί μια server-side υλοποίηση του πρωτοκόλλου WebSocket σε **python**.

9.2.13. AutobahnPython

Η AutobahnPython [90] είναι μια υλοποίηση του WebSocket η οποία μπορεί να χρησιμοποιηθεί για την ανάπτυξη client και server. Υλοποιεί το RFC6455 (και το Draft Hybi-10 έως 17).

9.2.14. AutobahnAndroid

Η AutobahnAndroid [91] είναι μία open source βιβλιοθήκη πελάτη websocket γραμμένη σε **Java** για την πλατφόρμα **Android 2.2** (= API level 8). Το AutobahnAndroid στηρίζεται σε Jackson, έναν JSON επεξεργαστή υψηλής απόδοσης.

ΠΑΡΑΡΤΗΜΑ Ι

Ακολουθούν τα πειραματικά αποτελέσματα χρήσης του load testing tool του COMETD.

PServer

```
listen port [8080]:
use ssl [false]:
acceptors [4]:
max threads [256]: 1000
record statistics [true]:
record latencies [true]:
detect long requests [false]: true
```

COMET long polling

PClient

```
server [localhost]:
port [8080]:
transports:
  0 - long-polling
  1 - websocket
transport [0]: 0
use ssl [false]:
max threads [256]:
context [/cometd]:
channel [/chat/demo]:
rooms [100]:
rooms per client [10]: 1
record latency details [true]:
-----
clients [100]: 1000
Waiting for clients to be ready...
Waiting for clients 998/1000
Clients ready
```

```
batch count [1000]:
batch size [10]: 1
batch pause (µs) [10000]:
message size [50]:
randomize sends [false]:
```

```
========================================
Statistics Started at Mon Oct 01 10:17:03 EEST 2012
Operative System: Linux 3.0.0-20-generic amd64
JVM : Oracle Corporation Java HotSpot(TM) 64-Bit Server VM runtime 23.0-b21 1.7.0_04-b20
Processors: 4
System Memory: 96.59848% used of 3.6733742 GiB
Used Heap Size: 94.069954 MiB
Max Heap Size: 1920.0 MiB
Young Generation Heap Size: 896.0 MiB
- - - - - - - - - - - - - - - - - - - -
Testing 1000 clients in 100 rooms, 1 rooms/client
Sending 1000 batches of 1x50 bytes messages every 10000 µs
- - - - - - - - - - - - - - - - - - - -
Statistics Ended at Mon Oct 01 10:17:15 EEST 2012
Elapsed time: 11059 ms
     Time in JIT compilation: 8373 ms
     Time in Young Generation GC: 0 ms (0 collections)
     Time in Old Generation GC: 0 ms (0 collections)
Garbage Generated in Young Generation: 265.98895 MiB
Garbage Generated in Survivor Generation: 0.0 MiB
Garbage Generated in Old Generation: 0.0 MiB
Average CPU Load: 114.47053/400
----------------------------------------
Outgoing: Elapsed = 11057 ms | Rate = 90 messages/s - 90 requests/s - ~0.034 Mib/s
All messages arrived 10146/10146
Messages - Success/Expected = 10146/10146
Incoming - Elapsed = 11046 ms | Rate = 918 messages/s - 918
```

```
responses/s(100.00%) - ~0.350 Mib/s
Messages - Wall Latency Distribution Curve (X axis: Frequency, Y axis: Latency):
                    @  _  2 ms (4756, 46.88%)
                @      _  4 ms (3662, 36.09%) ^50%
    @                  _  5 ms (1031, 10.16%) ^85%
  @                    _  7 ms (382, 3.77%) ^95%
 @                     _  8 ms (184, 1.81%)
@                      _  10 ms (76, 0.75%) ^99%
@                      _  12 ms (21, 0.21%)
@                      _  13 ms (10, 0.10%)
@                      _  15 ms (8, 0.08%)
@                      _  16 ms (4, 0.04%)
@                      _  18 ms (4, 0.04%) ^99.9%
@                      _  19 ms (5, 0.05%)
@                      _  21 ms (2, 0.02%)
@                      _  22 ms (0, 0.00%)
@                      _  24 ms (0, 0.00%)
@                      _  26 ms (0, 0.00%)
@                      _  27 ms (0, 0.00%)
@                      _  29 ms (0, 0.00%)
@                      _  30 ms (0, 0.00%)
@                      _  32 ms (1, 0.01%)
Messages - Wall Latency 50th%/99th% = 2/9 ms
Messages - Wall Latency Min/Ave/Max = 1/3/32 ms
Messages - Network Latency Min/Ave/Max = 0/2/31 ms
Thread Pool - Concurrent Threads max = 16 | Queue Size max = 23 | Queue Latency avg/max = 0/11 ms
```

WebSocket

PClient

```
server [localhost]:
```

```
port [8080]:
transports:
  0 - long-polling
  1 - websocket
transport [0]: 1
use ssl [false]:
max threads [256]:
context [/cometd]:
channel [/chat/demo]:
rooms [100]:
rooms per client [10]: 1
record latency details [true]:
-----
clients [100]: 1000
Waiting for clients to be ready...
Waiting for clients 999/1000
Clients ready
batch count [1000]:
batch size [10]: 1
batch pause (μs) [10000]:
message size [50]:
randomize sends [false]:
```

```
Statistics Started at Mon Oct 01 10:21:27 EEST 2012
Operative System: Linux 3.0.0-20-generic amd64
JVM : Oracle Corporation Java HotSpot(TM) 64-Bit Server VM runtime 23.0-b21 1.7.0_04-b20
Processors: 4
System Memory: 96.97369% used of 3.6733742 GiB
Used Heap Size: 80.28352 MiB
Max Heap Size: 1920.0 MiB
Young Generation Heap Size: 896.0 MiB
- - - - - - - - - - - - - - - - - - - -
Testing 1000 clients in 100 rooms, 1 rooms/client
```

```
Sending 1000 batches of 1x50 bytes messages every 10000 µs
- - - - - - - - - - - - - - - - - - - - -
Statistics Ended at Mon Oct 01 10:21:38 EEST 2012
Elapsed time: 11035 ms
     Time in JIT compilation: 2773 ms
     Time in Young Generation GC: 0 ms (0 collections)
     Time in Old Generation GC: 0 ms (0 collections)
Garbage Generated in Young Generation: 160.23317 MiB
Garbage Generated in Survivor Generation: 0.0 MiB
Garbage Generated in Old Generation: 0.0 MiB
Average CPU Load: 54.462097/400
----------------------------------------
Outgoing: Elapsed = 11032 ms | Rate = 90 messages/s - 90 requests/s - ~0.035 Mib/s
All messages arrived 9999/9999
Messages - Success/Expected = 9999/9999
Incoming - Elapsed = 10967 ms | Rate = 911 messages/s - 911 responses/s(100.00%) - ~0.348 Mib/s
Messages - Wall Latency Distribution Curve (X axis: Frequency, Y axis: Latency):
                    @  _  8 ms (9856, 98.57%) ^50% ^85% ^95%
@                      _  15 ms (70, 0.70%) ^99%
@                      _  22 ms (14, 0.14%)
@                      _  30 ms (1, 0.01%)
@                      _  37 ms (11, 0.11%)
@                      _  45 ms (11, 0.11%)
@                      _  52 ms (5, 0.05%)
@                      _  59 ms (1, 0.01%)
@                      _  67 ms (2, 0.02%)
@                      _  74 ms (6, 0.06%)
@                      _  82 ms (0, 0.00%)
@                      _  89 ms (0, 0.00%)
@                      _  96 ms (0, 0.00%)
@                      _  104 ms (0, 0.00%)
@                      _  111 ms (0, 0.00%)
```

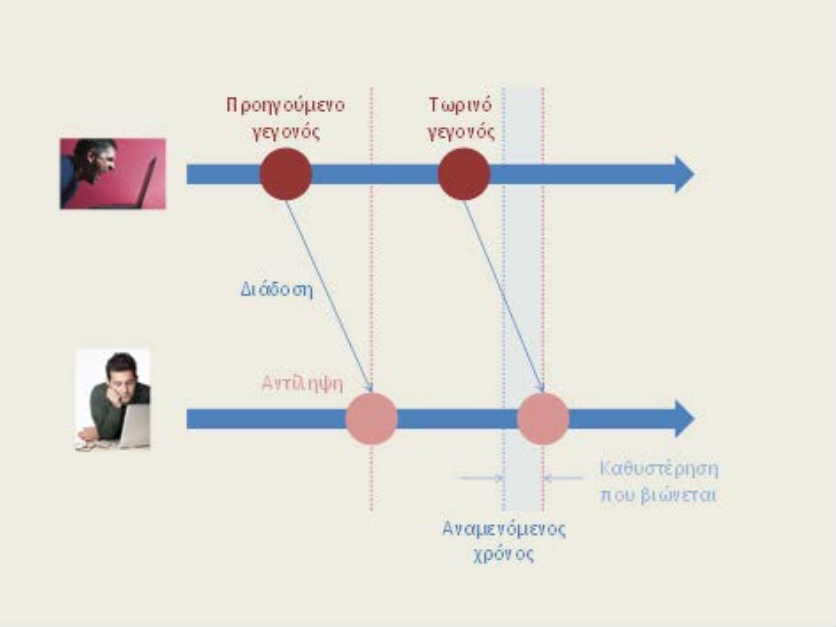
```
@                      _  119 ms (0, 0.00%)
@                      _  126 ms (9, 0.09%)
@                      _  134 ms (0, 0.00%)
@                      _  141 ms (0, 0.00%)
@                      _  148 ms (13, 0.13%) ^99.9%
Messages - Wall Latency 50th%/99th% = 1/9 ms
Messages - Wall Latency Min/Ave/Max = 0/2/148 ms
Messages - Network Latency Min/Ave/Max = 0/1/63 ms
Thread Pool - Concurrent Threads max = 15 | Queue Size max = 19 |
Queue Latency avg/max = 0/3 ms
```

ΠΑΡΑΡΤΗΜΑ ΙΙ

Πίνακας 12: Ελληνικοί Όροι

Όρος	Εξήγηση
Ανακυκλώσιμες Συνδέσεις (Connection pooling)	Το connection pooling στις διαδικτυακές εφαρμογές το χειρίζεται συνήθως ένας application server. Κάθε δυναμική ιστοσελίδα μπορεί στον κώδικά της να ανοίγει και να κλείνει κανονικά μια σύνδεση, όμως στην πραγματικότητα όταν ζητά μια νέα σύνδεση της επιστρέφεται μία από το connection pool που διατηρεί ο application server. Όμοια, όταν κλείνει μία σύνδεση στην πραγματικότητα επιστρέφει στο connection pool.
Γεγονός (Event)	Στην πληροφορική το «γεγονός» είναι μια ενέργεια που συνήθως ξεκινά έξω από τα όρια του προγράμματος και τη χειρίζεται ένα κομμάτι κώδικα μέσα στο πρόγραμμα. Τυπικά το πρόγραμμα έχει ένα ή περισσότερα ειδικά μέρη που χειρίζονται τα γεγονότα σύγχρονα (synchronous) με τη ροή του προγράμματος. Τα προγράμματα τα οποία αλλάζουν τη συμπεριφορά τους με βάση τα γεγονότα λέγονται καθοδηγούμενα από τα γεγονότα(event-driven) και έχουν ως στόχο τη διαδραστικότητα. Τα νέα γεγονότα τα επεξεργάζεται αρχικά ο event dispatcher μέσα από το framework. Συνήθως ελέγχει τις σχέσεις ανάμεσα στα γεγονότα και στους χειριστές γεγονότων (event handler) και μπορεί να βάλει σε ουρά τους χειριστές ή τα γεγονότα για μετέπειτα επεξεργασία. Οι event dispatcher μπορεί να καλέσουν απ’ ευθείας τους χειριστές ή να περιμένουν να βγουν από την ουρά τα γεγονότα με πληροφορίες σχετικά με το ποιος χειριστής πρέπει να εκτελεστεί.
Διαδίκτυο (Internet)	Ο όρος Διαδίκτυο (Internet) περιγράφει το παγκόσμιο πλέγμα διασυνδεδεμένων υπολογιστών και των υπηρεσιών και πληροφοριών που παρέχει στους χρήστες του. Το Διαδίκτυο μάς δίνει τη δυνατότητα

Όρος	Εξήγηση
	πρόσβασης σε νέα, πληροφορίες και βάσεις δεδομένων σε παγκόσμια κλίμακα. Επίσης, επιτρέπει τη χρήση πολλών και διαφορετικών εφαρμογών, που έχουν ως στόχο την επικοινωνία, όπως είναι το ηλεκτρονικό ταχυδρομείο (e-mail), οι ηλεκτρονικές ομάδες συζητήσεων (newsgroups), οι ηλεκτρονικές λίστες ανακοινώσεων (mailing lists), η επικοινωνία σε πραγματικό χρόνο (chat), οι τηλεδιασκέψεις (net-meeting) κ.ά. Όσον αφορά σε επιχειρησιακές εφαρμογές το Διαδίκτυο δίνει τη δυνατότητα για ηλεκτρονικό εμπόριο (e-commerce), εκπαίδευση και επιμόρφωση από απόσταση (e-learning & e-training), καθώς και εργασία από απόσταση, δηλαδή τηλεργασία (teleworking). Το Διαδίκτυο χρησιμοποιεί μεταγωγή πακέτων (packet switching) και τη στοίβα πρωτοκόλλων TCP/IP (Transmission Control Protocol/Internet Protocol).
Διαδραστική εφαρμογή πραγματικού χρόνου	Μια διαδικτυακή εφαρμογή λέγεται διαδραστική πραγματικού χρόνου εάν οι ενέργειες μεταξύ των χρηστών περνούν τόσο γρήγορα ώστε να μην διαταράσσεται η σημασιολογία της επικοινωνίας. Οποιαδήποτε ενέργεια χρήστη, από το πάτημα ενός κουμπιού ως τη δημοσίευση σε ένα microblog, μπορεί να οδηγήσει σε ένα γεγονός. Η προώθηση ενός γεγονότος σε όλους τους χρήστες που εμπλέκονται σε μια αλληλεπίδραση απαιτεί χρόνο για την επεξεργασία και τη μεταφορά. Αν αυτό το χρονικό διάστημα υπερβεί το χρονικό διάστημα που αναμένεται από το χρήστη, ο χρήστης βιώνει μια καθυστέρηση. Η καθυστέρηση αυτή μπορεί να διαταράξει την επικοινωνία ή ακόμα και να αλλάζει το νόημα των γεγονότων.

Όρος	Εξήγηση
	 **Εικόνα 39: Καθυστέρηση στις διαδραστικές εφαρμογές πραγματικού χρόνου**
Διευθύνσεις Ιστού (Web Addresses)	Κάθε ιστοσελίδα χαρακτηρίζεται με μοναδικό τρόπο από τη διεύθυνσή της, ή αλλιώς το URL (Uniform Resource Locator) της. Το URL είναι αρκετό για να εντοπιστεί μια ιστοσελίδα που βρίσκεται σε έναν εξυπηρέτη Ιστού οπουδήποτε στον κόσμο. Συνήθως αποτελείται από 5 μέρη: το πρωτόκολλο που χρησιμοποιείται για τη μεταφορά της, το όνομα περιοχής (domain name) του εξυπηρέτη Ιστού που την περιέχει, τη διαδρομή στο αρχείο της ιστοσελίδας και το όνομα του αρχείου της ιστοσελίδας. Η διεύθυνση ενός υπολογιστή στο διαδίκτυο εκτός από την ονομαστική του διεύθυνση, προσδιορίζεται και αριθμητικά. Αυτή η αριθμητική ταυτότητα ονομάζεται IP Διεύθυνση (IP Address). Αποτελείται από 4 ομάδες αριθμών που αντιστοιχούν στον αριθμό δικτύου και στον αριθμό υπολογιστή μέσα στο συγκεκριμένο δίκτυο (π.χ. 134.52.67.1).
Εγγραφή σε	Η αντιστοίχιση ενός χειριστή γεγονότος σε κάποιο γεγονός λέγεται

Όρος	Εξήγηση
γεγονός (Event Registration)	«εγγραφή σε ένα γεγονός». Κάθε γεγονός μπορεί να έχει πάνω από έναν χειριστές.
Ειδοποίηση γεγονότος (Event Notification)	Ο όρος «ειδοποίηση γεγονότος» χρησιμοποιείται σε συνδυασμό με λογισμικό επικοινωνίας για τη σύνδεση εφαρμογών, οι οποίες παράγουν μικρά μηνύματα (τα γεγονότα), με εφαρμογές οι οποίες παρακολουθούν τις σχετιζόμενες καταστάσεις (condition) και μπορεί να λάβουν μέτρα τα οποία προκαλούνται από τα γεγονότα.
Εξυπηρέτης (web server)	Κάθε ιστοσελίδα βρίσκεται με τη μορφή αρχείου σε κάποιον εξυπηρέτη Ιστού (web Server). Οι εξυπηρέτες Ιστού είναι ειδικοί υπολογιστές με ειδικό λογισμικό και κατάλληλες δικτυακές συνδέσεις, οι οποίοι επιτρέπουν τη διάθεση των ιστοσελίδων σε ολόκληρο τον κόσμο. Ο χρήστης του Διαδικτύου που θέλει να δει μια ιστοσελίδα, τη ζητάει από τον εξυπηρέτη Ιστού στον οποίο αυτή βρίσκεται, και ο εξυπηρέτης Ιστού με τη σειρά του την στέλνει. Πώς γίνεται όμως η ζήτηση και η διάθεση των σελίδων;
Εξυπηρέτης μεσολάβησης (proxy server)	Πρόκειται για έναν εξυπηρέτη ο οποίος δρα ως ενδιάμεσος ανάμεσα σε έναν πελάτη και σε έναν άλλο εξυπηρέτη (π.χ. έναν εξυπηρέτη στο Διαδίκτυο). Συνήθως χρησιμοποιείται για την αποθήκευση του περιεχομένου (content cashing), συνδεσιμότητα στο Διαδίκτυο, ασφάλεια και φιλτράρισμα του περιεχομένου μιας επιχείρησης. Ο εξυπηρέτης διαμεσολάβησης μπορεί να ελέγχει την κίνηση και να κλείνει μια σύνδεση όταν αυτή μένει ανοικτή για πολύ. Συνήθως τοποθετείται ανάμεσα σε ένα ιδιωτικό δίκτυο και το Διαδίκτυο.
Ιστοσελίδα (web page)	Οι πληροφορίες του Παγκόσμιου Ιστού εμφανίζονται μορφοποιημένες με τη γλώσσα HTML (Hypertext Markup Language) σε μορφή ιστοσελίδων (web pages) και με την κατάληξη .htm ή .html. Υπάρχουν όμως και διαφορετικές μορφοποιήσεις ιστοσελίδων, όπως για

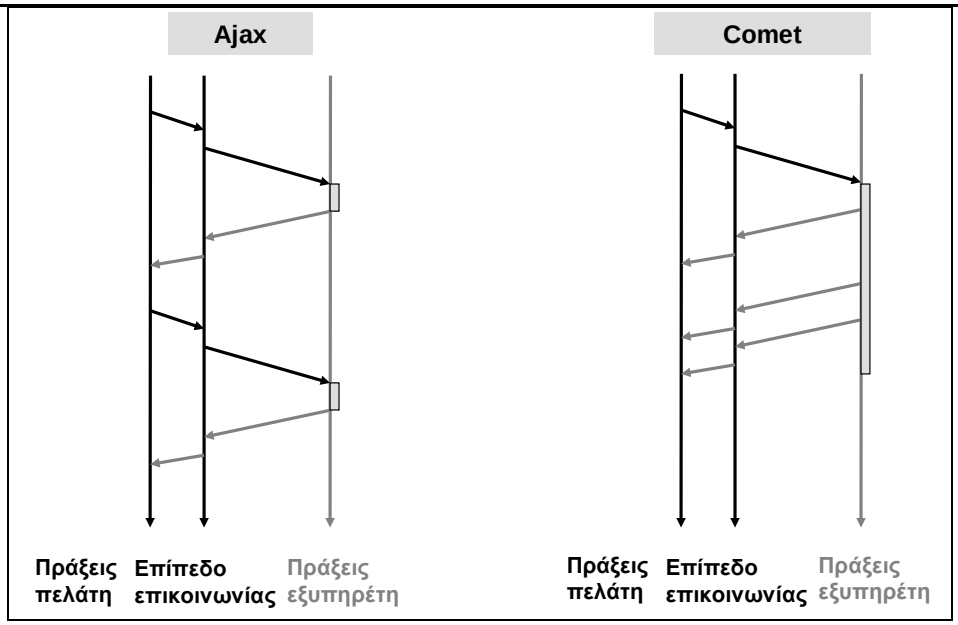
Όρος	Εξήγηση
	παράδειγμα .php. Οι ιστοσελίδες μπορεί να περιέχουν εκτός από στατικό κείμενο, εικόνες, video, ήχο, κινούμενες εικόνες (animation), δυναμικό κείμενο κτλ. Πού βρίσκονται όμως όλες αυτές οι ιστοσελίδες; Στον εξυπηρέτη.
Παγκόσμιος Ιστός (World Wide Web - WWW)	Δεν είναι λίγοι αυτοί που νομίζουν ότι οι όροι Διαδίκτυο και Παγκόσμιος Ιστός είναι ταυτόσημοι. Η αλήθεια είναι ότι ο Παγκόσμιος Ιστός (World Wide Web ή WWW) είναι ένα μέρος του Διαδικτύου. Αποτελεί όμως το μεγαλύτερο, το δημοφιλέστερο και το ταχύτερα αναπτυσσόμενο κομμάτι του. Συγκεκριμένα, ο Παγκόσμιος Ιστός είναι το μέσο για την εύκολη ανάκτηση του τεράστιου όγκου πληροφοριών που διατίθενται μέσω του Διαδικτύου. Χρησιμοποιεί ένα από τα πρωτόκολλα του Διαδικτύου, το Hypertext Transfer Protocol (HTTP).
Πλούσιες Εφαρμογές Διαδικτύου (Rich Internet Application - RIA)	Οι Πλούσιες Εφαρμογές Διαδικτύου είναι διαδικτυακές εφαρμογές οι οποίες λειτουργούν σαν παραδοσιακές desktop εφαρμογές, αλλά ο χρήστης χρειάζονται κάποιο φυλλομετρητή για να έχει πρόσβαση σε αυτές. Αν και δε χρειάζεται η εγκατάσταση του λογισμικού της εφαρμογής, συνήθως χρειάζεται η εγκατάσταση ActiveX, Java, Flash ή παρόμοιων τεχνολογιών στο μηχάνημα του χρήστη.
Πρόγραμμα Περιήγησης (web browser)	Το πρόγραμμα περιήγησης (web browser) ή αλλιώς ο φυλλομετρητής είναι ένα πρόγραμμα (πχ Netscape Navigator, Internet Explorer, Mozilla κτλ.), το οποίο χρησιμοποιεί ο χρήστης για να ζητήσει μια ιστοσελίδα από τον εξυπηρέτη Ιστού που την περιέχει. Ο εξυπηρέτης Ιστού λαμβάνει το αίτημα και εμφανίζει την ιστοσελίδα στο παράθυρο του προγράμματος περιήγησης του χρήστη. Πώς όμως καταλαβαίνει για ποια ιστοσελίδα πρόκειται;
Ροή Δεδομένων (Data Stream)	Στον τομέα των τηλεπικοινωνιών και της πληροφορικής, η ροή δεδομένων είναι μια σειρά από ψηφιακά κωδικοποιημένα συνεκτική

Όρος	Εξήγηση
	σήματα (πακέτα δεδομένων ή πακέτων δεδομένων) χρησιμοποιείται για την μετάδοση ή λήψη πληροφοριών που είναι στη διαδικασία που μεταδίδονται Η ροή δεδομένων είναι μια ακολουθία από ψηφιακώς κωδικοποιημένα συνεκτικά σήματα (πακέτα δεδομένων) η οποία χρησιμοποιείται για την εκπομπή ή τη λήψη πληροφορίας η οποία είναι σε διαδικασία μετάδοσης.
Τοποθεσία ιστού (web site)	Μια ομάδα ιστοσελίδων που αφορούν έναν ιδιώτη, μια επιχείρηση, έναν οργανισμό ή άλλες ομάδες αποτελεί μια τοποθεσία Ιστού ή ένα Web Site.
Τεχνολογία αυτόματης προώθησης (push technology)	Στον αντίποδα της τεχνολογίας ρητής αίτησης βρίσκουμε την τεχνολογία αυτόματης προώθησης, όπου ο εξυπηρέτης προωθεί δεδομένα στους πελάτες. Είναι πλέον δυνατή η ανάπτυξη εφαρμογών που βασίζονται στο φυλλομετρητή με υψηλή διαδραστικότητα με το χρήστη και χαμηλά αντιλαμβανόμενη καθυστέρηση από τον ίδιο. Τα δυναμικά δικτυακά δεδομένα πραγματικού χρόνου όπως οι τίτλοι ειδήσεων, οι δείκτες των μετοχών και οι ενημερώσεις των δημοπρασιών πρέπει να δημοσιοποιούνται στους χρήστες το συντομότερο δυνατόν. Αυτό δεν μπορεί να γίνει με την κλασική αρχιτεκτονική αιτήματος/απάντησης, αντιθέτως απαιτείται η αυτόματη προώθηση των δυναμικών δεδομένων.
Τεχνολογία ρητής αίτησης (pull technology)	Η τεχνολογία ρητής αίτησης ή αίτησης πελάτη είναι ένα είδος διαδικτυακής επικοινωνίας όπου η αρχική αίτηση για δεδομένα ξεκινά από τον πελάτη και στη συνέχεια δίνεται η απάντηση από τον εξυπηρέτη. Οι ρητές αιτήσεις αποτελούν τη βάση του διαδικτυακού υπολογισμού,

Όρος	Εξήγηση
	όπου πολλοί πελάτες ζητούν δεδομένα από κεντρικούς εξυπηρέτες. Στο Διαδίκτυο γίνονται ευρέως αιτήσεις HTTP σελίδων σε τοποθεσίες ιστού.
Υπερσύνδεσμος (hyperlink ή link)	Ένα από τα σημαντικότερα χαρακτηριστικά που διευκολύνουν την περιήγηση στον Παγκόσμιο Ιστό είναι η χρήση της δομής του υπερκειμένου (hypertext). Η ανάγνωση των πληροφοριών και η μετακίνηση μέσα στο υπερκείμενο γίνεται με τη βοήθεια των υπερσυνδέσμων (hyperlinks), οι οποίοι βρίσκονται σε διάφορα σημεία μιας ιστοσελίδας. Συνήθως πρόκειται για υπογραμμισμένο κείμενο με διαφορετικό χρώμα από το κείμενο της ιστοσελίδας, αλλά μπορεί να είναι και εικόνα. Αναγνωρίζεται από την μορφή που παίρνει ο δείκτης του ποντικιού όταν είναι επάνω του (γίνεται ένα «χέρι»).

Πίνακας 13: Αγγλικοί Όροι

Όρος	Εξήγηση
COMET	Η προσέγγιση Comet αναφέρεται σε τεχνολογίες που προσπαθούν να εξαφανίσουν τους περιορισμούς που επιβάλλει το δικτυακό μοντέλο σελίδα-προς-σελίδα και το παραδοσιακό polling. Οι Comet-like εφαρμογές προσφέρουν αλληλεπίδραση πραγματικού χρόνου. Βασίζονται σε επίμονες HTTP συνδέσεις (ή όπου αυτό δεν είναι δυνατό σε μακράς διάρκειας HTTP συνδέσεις) για να δίνουν στο φυλλομετρητή τις ενημερώσεις καθώς αυτές δημιουργούνται από την εφαρμογή. Η Comet είναι, δηλαδή, μια «push» προσέγγιση, η οποία βελτιώνει σε μεγάλο βαθμό την απόδοση και μειώνει την καθυστέρηση και το φορτίο του εξυπηρέτη. Οι μέθοδοι υλοποίησης του Comet εμπίπτουν στις κατηγορίες του Streaming και του Long Polling αντίστοιχα. Η Εικόνα είναι χρήσιμη για μία σύγκριση των μεταφορών δεδομένων Ajax και Comet.



Εικόνα 40: Μεταφορά δεδομένων Ajax και Comet

Σε αντίθεση με τους παραδοσιακούς διαδικτυακούς εξυπηρέτες, οι οποίοι παραδίδουν ένα ενιαίο φορτίο (payload) και στη συνέχεια κλείνουν τη σύνδεση με τον πελάτη, οι Comet εξυπηρέτες πρέπει να διατηρούν μια συνεχή σύνδεση με κάθε πελάτη όσο διαρκεί η σύνοδος. Η προσαρμογή ενός παραδοσιακού εξυπηρέτη στη μέθοδο Comet δεν μπορεί να γίνει, αφού δεν μπορεί να επιτευχθεί η απαιτούμενη κλιμάκωση και συχνά αποτυγχάνει μετά από μερικές χιλιάδες ταυτόχρονων ανοιχτών συνδέσεων. Μια πραγματική εφαρμογή Comet απαιτεί ένα πολύ διαφορετικό είδος αρχιτεκτονικής για να είναι αποτελεσματική και επεκτάσιμη.

Cookies

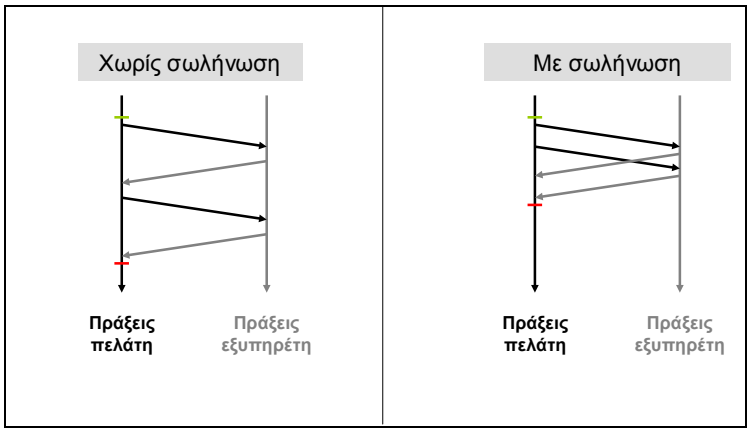
Τα Cookies είναι μικρά "αρχεία" που περιέχουν πληροφορίες τις οποίες χρησιμοποιούν οι ιστοσελίδες για την αναγνώρισή των πελατών. Όταν ο χρήστης κάνει login, ένα Cookie καταγράφει τα στοιχεία του, έτσι ώστε η ιστοσελίδα να γνωρίζει ότι έχει κάνει login από τον συγκεκριμένο Η/Υ. Ταυτόχρονα μπορεί να καταγράφει και κάποιες άλλες πληροφορίες σχετικά με την δραστηριότητά του στο site, όπως για παράδειγμα σε ποιες ψηφοφορίες έλαβε μέρος (έτσι ώστε να του απαγορέψει να ψηφίσει δύο φορές), ή κάποιες προσωπικές ρυθμίσεις που χρησιμοποιεί (π.χ. χρωματικά θέματα). Εάν ο browser δεν μπορεί να δεχτεί τα Cookies, οι

	πιθανοί λόγοι μπορεί να είναι: • Είναι απενεργοποιημένη η υποστήριξη Cookies στον browser. • Η ημερομηνία του υπολογιστή είναι λάθος. • Ο browser που χρησιμοποιείται δεν υποστηρίζει καθόλου Cookies.
Dependency Injection (DI)	Όταν εφαρμόζεται η DI, τα αντικείμενα λαμβάνουν παθητικά τις εξαρτήσεις τους (dependencies) αντί να τις δημιουργούν ή να τις αναζητούν μόνα τους. Αντί ένα αντικείμενο να ψάχνει μόνο του για τις εξαρτήσεις του σε έναν container, ο container δίνει τις εξαρτήσεις στο αντικείμενο χωρίς να περιμένει πρώτα να ερωτηθεί. Η τεχνική αυτή αποτελεί το κλειδί για τη *χαλαρή διασύνδεση (loose coupling)*. Αν ένα αντικείμενο γνωρίζει για τις εξαρτήσεις του μόνο μέσω μίας διεπαφής (όχι μέσω της υλοποίησής τους ή μέσω του τρόπου που δημιουργούνται), η εξάρτηση μπορεί να αλλαχτεί πολύ εύκολα από μια διαφορετική υλοποίηση χωρίς το αντικείμενο να γνωρίζει για τη διαφορά.
Direct Web Remoting (DWR)	Η DWR είναι μια Java βιβλιοθήκη ανοικτού κώδικα για την ανάπτυξη Ajax Διαδικτυακών εφαρμογών. Επιτρέπει στη JavaScript στο φυλλομετρητή να αλληλεπιδρά με τον κώδικα Java στον εξυπηρέτη.
DOM	Το DOM (Document Object Model - Μοντέλο Αντικειμένου Εγγράφου) αποτελεί την προδιαγραφή (specification) για το πώς αναπαριστούνται τα αντικείμενα (κείμενο, εικόνες, κεφαλίδες, σύνδεσμοι, κ.α.) σε μια ιστοσελίδα. Το DOM ορίζει ποια χαρακτηριστικά (attribute) σχετίζονται με κάθε αντικείμενο και το πώς μπορεί να χειριστεί κάποιος τα αντικείμενα και τα χαρακτηριστικά. Η Δυναμική HTML (DHTML) βασίζεται στο DOM για να αλλάζει δυναμικά την εμφάνιση των ιστοσελίδων αφού έχουν κατέβει στο φυλλομετρητή κάποιου χρήστη. Δυστυχώς, οι διαφορετικοί φυλλομετρητές χρησιμοποιούν διαφορετικά DOM. Αυτός είναι ένας λόγος για τον οποίο οι αντίστοιχες υλοποιήσεις τους για τη DHTML είναι τόσο διαφορετικές.

Dynamic HTML	Κάνει χρήση Javascript για να χειρίζεται την είσοδο του χρήστη και να αλλάζει το DOM. Μπορεί να αντιδρά μέσα στη σελίδα με αποτέλεσμα οι σελίδες να κάνουν περισσότερα πράγματα σε σχέση με πριν. Ακόμη είναι διαδραστική τοπικά, χωρίς να χρειάζεται επικοινωνία με τον εξυπηρέτη.
Flush	Για να βελτιώσει την απόδοση, η JavaScript αποθηκεύει (buffer) τη σελίδα HTML που κατασκευάζει. Η συνάρτηση flush() στέλνει αμέσως τα δεδομένα από τον εσωτερικό buffer στον πελάτη. Αν δεν κληθεί η flush(), τότε η Javascript στέλνει δεδομένα στον πελάτη αφού έχει φτιάξει 64 KB δεδομένων.
Glassfish [92]	Ο Glassfish αποτελεί την εφαρμογή αναφοράς των προδιαγραφών J2EE 6. Χρησιμοποιεί ένα παράγωγο του Apache Tomcat ως servlet container και ένα πρόσθετο συστατικό το οποίο ονομάζεται Grizzly και χρησιμοποιεί NIO για επεκτασιμότητα και ταχύτητα και υποστηρίζει Comet και WebSocket. Ο Glassfish αναπτύσσεται από την Oracle Corporation και είναι ανοιχτού κώδικα.
Google Guice [93]	Η Google δημιούργησε το δικό της πλαίσιο εφαρμογών Java ανοικτού κώδικα με DI, το Guice. Το πλαίσιο αυτό φαίνεται να είναι πιο γρήγορο από το Spring και οι τελικές εφαρμογές έχουν λιγότερο κώδικα.
Host	Ο Host είναι ένας dedicated υπολογιστής δικτύου που παρέχει ορισμένου είδους υπηρεσίες. Συνήθως ο όρος αυτός αναφέρεται σε έναν υπολογιστή που αποθηκεύει αρχεία ιστοσελίδων και περιλαμβάνει έναν web server, ο οποίος μοιράζεται το περιεχόμενο του με το υπόλοιπο του Διαδικτύου, και έναν mail server, ο οποίος δέχεται μηνύματα ηλεκτρονικού ταχυδρομείου και τα στέλνει στους αντίστοιχους παραλήπτες.
HTTP Pipelining	Η HTTP pipelining είναι μια τεχνική κατά την οποία πολλαπλές αιτήσεις HTTP γράφονται σε ένα μόνο socket χωρίς να περιμένουν για τις αντίστοιχες απαντήσεις. Η pipelining υποστηρίζεται μόνο στην HTTP/1.1,

όχι στην 1.0.

Η HTTP pipelining απαιτεί πρέπει να υποστηρίζεται τόσο από τον πελάτη όσο και από τον εξυπηρέτη. Οι HTTP/1.1 εξυπηρέτες απαιτείται να υποστηρίζουν pipelining. Αυτό δε σημαίνει ότι είναι υποχρεωμένοι να σωληνώνουν τις απαντήσεις, αλλά θα πρέπει να μην αποτυγχάνουν (fail) εάν ο πελάτης επιλέγει να σωληνώνει τις αιτήσεις.



Εικόνα 41: HTTP Pipelining

Η HTTP pipelining είναι απενεργοποιημένη στους περισσότερους **φυλλομετρητές**.

- Ο Opera έχει προεπιλεγμένα ενεργοποιημένη τη σωλήνωση. Χρησιμοποιεί ευριστικά για τον έλεγχο του επιπέδου της σωλήνωσης ανάλογα με το συνδεμένο εξυπηρέτη.
- Ο Mozilla Firefox 3 υποστηρίζει τη διασωλήνωση, αλλά είναι απενεργοποιημένη από προεπιλογή. Χρησιμοποιεί κάποιο heuristics, ειδικά για την απενεργοποίησή της σε εξυπηρέτες IIS.
- Ο Camino κάνει το ίδιο πράγμα με το Firefox.
- Ο SeaMonkey υποστηρίζει pipelining μετά την αρχική έκδοση και μπορεί εύκολα να (απ)ενεργοποιηθεί το μενού προτιμήσεων.
- Konqueror 2.0 υποστηρίζει σωλήνωση, αλλά είναι απενεργοποιημένη από προεπιλογή.

	• Ο Interner Explorer 8 δε σωληνώνει τα αιτήματα, λόγω ανησυχιών σε σχέση με προβληματικούς πληρεξούσιους και head-of-line μπλοκάρισμα. • Ο Google Chrome δεν υποστηρίζει σωλήνωση, αν και μπορεί να εφαρμοστεί στο εγγύς μέλλον. Οι περισσότεροι **πληρεξούσιοι** HTTP δεν σωληνώνουν τις εξερχόμενες αιτήσεις. • Ορισμένες εκδόσεις του Squid σωληνώνουν μέχρι δύο εξερχόμενες αιτήσεις. Αυτή η λειτουργία έχει απενεργοποιηθεί στις τελευταίες εκδόσεις για «λόγους διαχείριση εύρους ζώνης και καταγραφής πρόσβασης (access logging)." Ο Squid υποστηρίζει πολλαπλές αιτήσεις από τους πελάτες. • Ο Polipo σωληνώνει τις εξερχόμενες αιτήσεις.

Iframe	Τα πλαίσια (frame) επιτρέπουν στο παράθυρο του HTML φυλλομετρητή να σπάσει σε τμήματα, κάθε ένα εκ των οποίων μπορεί να δείχνει ένα διαφορετικό έγγραφο. Το Iframe (inline frame) τοποθετεί ένα άλλο έγγραφο HTML μέσα στο πλαίσιο. Παρουσιάστηκε από το Microsoft IE το 1997, έγινε πρότυπο στην HTML 4.0 Transitional και επιτρέπεται στην HTML5.
	Υπάρχουν δύο είδη Iframe, αυτά που βρίσκονται μέσα στον HTML κώδικα και φορτώνονται στη σελίδα:
	`<html>`
	`  <body>`
	`  <iframe id="testFrame" src="http://www.google.com`
	`  </body>`
	`</html>`
	και εκείνα που δημιουργούνται δυναμικά, μέσω του DOM και της JavaScript, αφού έχει τελειώσει το φόρτωμα της σελίδας:
	`<html>`

	<head>
	<script language="JavaScript">
	function initialize() {
	var testFrame = document.createElement("IFRAME");
	testFrame.id = "testFrame";
	testFrame.src = "http://www.google.com";
	document.body.appendChild(testFrame);
	}
	</script>
	</head>
	<body onload="initialize()"> </body>
	</html>

Inversion of Control (IoC)	Η IoC (Αντιστροφή του Ελέγχου) είναι μια προσέγγιση στην ανάπτυξη λογισμικού η οποία ευνοεί την αφαίρεση των σφραγισμένων (sealed) εξαρτήσεων μεταξύ των κλάσεων, έτσι ώστε ο κώδικας να γίνει πιο απλός και ευέλικτος. Προωθούμε τον έλεγχο της δημιουργίας μιας εξάρτησης έξω από την κλάση κάνοντας την εξάρτηση ταυτόχρονα ρητή (explicit). Η χρήση της IoC επιτρέπει γενικά τη δημιουργία εφαρμογών που είναι πιο ευέλικτες, unit-testable, απλές και μακροπρόθεσμα συντηρήσιμες.
Jboss [94]	Ο Jboss είναι ένας βασισμένος σε Java EE εξυπηρέτης εφαρμογών κτισμένος πάνω από τον Tomcat. Υποστηρίζει NIO και Comet από την έκδοση 5. Ο Jboss αναπτύσσεται από τη Red Hat και είναι ανοικτού κώδικα.
Jetty [95]	Ο Jetty είναι ένας pure Java HTTP εξυπηρέτης και servlet container. Υποστηρίζει την Προδιαγραφή Servlet 3.0, τα WebSocket και πολλές άλλες προδιαγραφές ολοκλήρωσης (integration specification). Ο Jetty είναι: • Ισχυρός και ευέλικτος • Εύκολα ενσωματώσιμος • Υποστηρίζει virtual host, ομαδοποίηση συνεδριών (session

	clustering) και πολλά ακόμη χαρακτηριστικά τα οποία μπορούν εύκολα να ρυθμιστούν μέσω κώδικα Java ή XML. • Χρησιμοποιείται στην υπηρεσία φιλοξενίας Google App Engine. Το core project του Jetty φιλοξενείται από το Eclipse Foundation και είναι ανοιχτού κώδικα.
JQUERY	Η JQUERY είναι μία βιβλιοθήκη της Javascript. Μία συλλογή δηλαδή από έτοιμες ρουτίνες γραμμένες σε Javascript, τις οποίες μπορούμε να χρησιμοποιήσουμε για να εκτελέσουμε συγκεκριμένες λειτουργίες. Η ενσωμάτωσή της στον κώδικα είναι εύκολη καθώς έχει σχεδιαστεί με βάση την δομή και την φιλοσοφία των HTML και CSS. Με την JQUERY μπορούμε να κάνουμε τα εξής: • να έχουμε άμεση πρόσβαση σε οποιοδήποτε στοιχείο της ιστοσελίδας, • να αλλάξουμε την εμφάνιση μιας ιστοσελίδας χωρίς να ανησυχούμε για τις ασυμβατότητες των διαφόρων browsers, • να αλλάξουμε δυναμικά το περιεχόμενο της ιστοσελίδας ή ακόμα και ολόκληρη την ιστοσελίδα, • να εφαρμόσουμε διάφορα οπτικά εφέ όπως κίνηση, σκίαση κ.α., • να χρησιμοποιούμε λειτουργία “Drag and Drop” με διάφορα αντικείμενα της σελίδας και • να εφαρμόσουμε τεχνικές AJAX και πολλές άλλες λειτουργίες.
JSON [96]	Το μορφότυπο JSON (JavaScript Object Notation), είναι ένα «ελαφρύ» πρότυπο ανταλλαγής δεδομένων. Βασίζεται σε ένα υποσύνολο της Γλώσσας Προγραμματισμού JavaScript. Το πρότυπο JSON χρησιμοποιείται συχνά για την σειριακή τοποθέτηση (serialization) και την μετάδοση δομημένων δεδομένων μέσω μιας σύνδεσης δικτύου. Χρησιμοποιείται κυρίως για την μετάδοση δεδομένων σε

διαδικτυακές εφαρμογές, χρησιμεύονται ως μια εναλλακτική επιλογή για την XML.

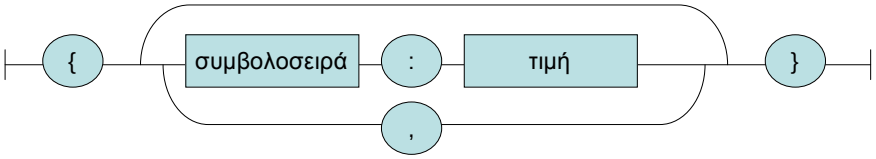
Είναι ένα πρότυπο που βασίζεται σε κείμενο (text-based) και όχι σε κάποια συγκεκριμένη γλώσσα, είναι αναγνώσιμο από τον άνθρωπο (human-readable), και χρησιμοποιείται για την αναπαράσταση απλών δομών δεδομένων και προσεταιριστικών συστοιχιών (associative arrays) που ονομάζονται αντικείμενα. Το JSON είναι χτισμένο σε δύο δομές:

- Μια συλλογή από ζευγάρια όνομα/τιμή. Ανάλογα με τη γλώσσα, αυτό υλοποιείται ως αντικείμενο, εγγραφή, δομή, λεξικό, πίνακας κατακερματισμού, λίστα με κλειδί ή συσχετιστικός πίνακας.
- Μια διατεταγμένη λίστα τιμών. Στις περισσότερες γλώσσες, αυτό υλοποιείται ως πίνακας, διάνυσμα, λίστα ή ακολουθία.

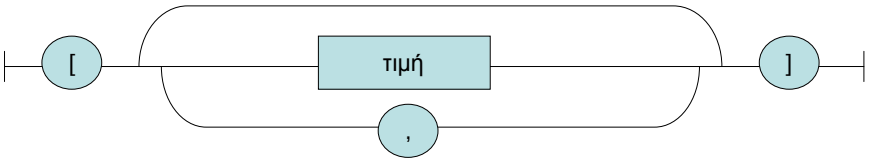
Αυτές είναι οι καθολικές δομές δεδομένων. Είναι λογικό πως μία μορφή δεδομένων που είναι ανταλλάξιμη με τις γλώσσες προγραμματισμού, θα βασίζεται σε αυτές τις δομές.

Στο JSON, παίρνουν αυτές τις μορφές:

Ένα αντικείμενο είναι ένα μη διατεταγμένο σύνολο ζευγαριών ονόματος/τιμής. Ένα αντικείμενο ξεκινάει με { (αριστερό άγκιστρο) και τελειώνει με } (δεξί άγκιστρο). Κάθε όνομα ακολουθείται από : (άνω και κάτω τελεία) και τα ζεύγη όνομα/τιμή χωρίζονται με , (κόμμα).



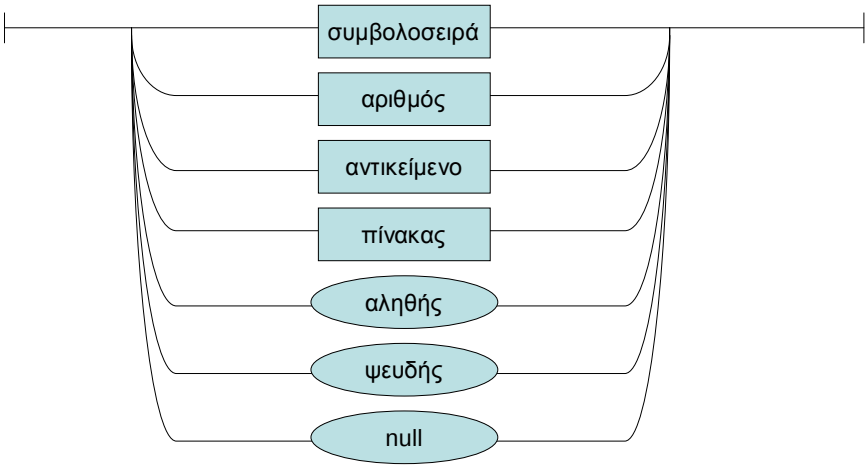
Ένας πίνακας είναι μια διαταγμένη συλλογή τιμών. Ο πίνακας ξεκινά με [(αριστερή αγκύλη) και τελειώνει με] (δεξιά αγκύλη). Οι τιμές χωρίζονται από , (κόμμα).



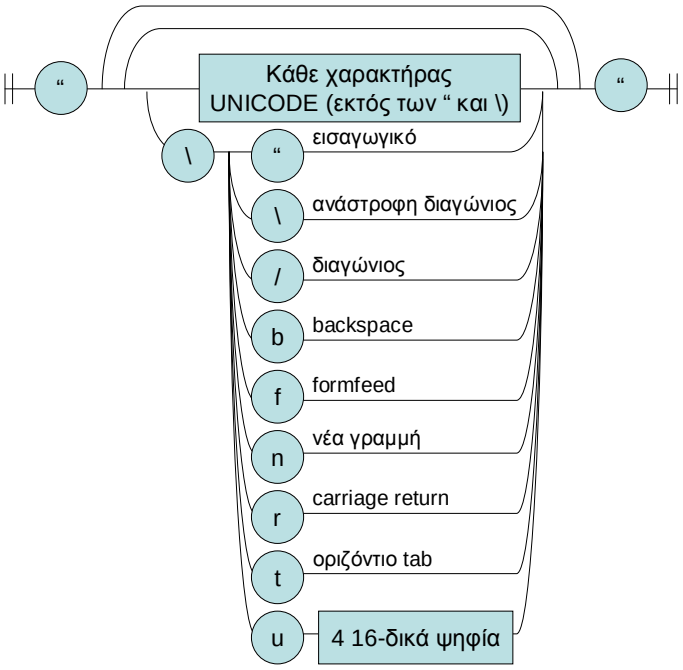
Μια τιμή μπορεί να είναι:

- μια συμβολοσειρά σε διπλά εισαγωγικά,
- ένας αριθμός,
- αληθής, ψευδής ή null,
- ένα αντικείμενο ή
- ένας πίνακας.

Οι δομές αυτές μπορεί να είναι εμφωλιασμένες.



Μια συμβολοσειρά είναι μια συλλογή από μηδέν ή περισσότερους χαρακτήρες Unicode, μέσα σε διπλά εισαγωγικά, χρησιμοποιώντας backslash χαρακτήρες διαφυγής. Ένας χαρακτήρας παρουσιάζεται ως μια συμβολοσειρά ενός χαρακτήρα.



Ένας αριθμός είναι όπως θα τον βρούμε στη C ή την Java, αλλά δε χρησιμοποιούνται η οκταδική και η δεκαεξαδική μορφή.

Online

Γενικά, η λέξη online υποδηλώνει μια κατάσταση συνδεσιμότητας. Οι χρήστες θεωρούνται online όταν είναι συνδεδεμένοι σε κάποια υπηρεσία μέσω ενός modem, ενώ οι πληροφορίες και οι εφαρμογές θεωρούνται online όταν μπορεί κάποιος να έχει πρόσβαση σε αυτές μέσω Διαδικτύου.

PHP [97]

Η PHP (Hypertext PreProcessor) είναι μια γλώσσα προγραμματισμού για τη δημιουργία σελίδων web με δυναμικό περιεχόμενο. Μια σελίδα PHP περνά από επεξεργασία από ένα συμβατό εξυπηρέτη του Παγκόσμιου Ιστού (π.χ. Apache), ώστε να παραχθεί σε πραγματικό χρόνο το τελικό περιεχόμενο, που θα σταλεί στο πρόγραμμα περιήγησης των επισκεπτών σε μορφή κώδικα HTML.

Η PHP μπορεί να χρησιμοποιηθεί σε όλα τα κύρια λειτουργικά συστήματα και σε πολλούς εξυπηρέτες (Apache, Microsoft Internet Information Server, Personal Web Server, Netscape και iPlanet servers, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd κ.ά.). Ένα από τα πιο δυνατά και

	σημαντικά χαρακτηριστικά της PHP είναι η υποστήριξη που έχει για ένα μεγάλο σύνολο βάσεων δεδομένων.
Script	Η εξάπλωση του Διαδικτύου και η χρήση των σελίδων HTML για δυναμική αλληλεπίδραση με τους χρήστες δεν μπορούσε να πραγματοποιηθεί μόνο μέσω της HTML. Η HTML είναι μια γλώσσα μορφοποίησης υπερκειμένου, οπότε για να ελεγχθούν προγραμματιστικά όλα τα αντικείμενα, που μπορεί να περιέχει μια ιστοσελίδα, απαιτείται η χρήση κάποιας γλώσσας προγραμματισμού για το Διαδίκτυο. Αρχικά, η Netscape ανέπτυξε τη γλώσσα JavaScript η οποία μεταφράζεται από τον Web Browser κατά την εμφάνιση της σελίδας. Η Microsoft ανέπτυξε μια δική της έκδοση της JavaScript την οποία ονόμασε JScript καθώς και μια έκδοση της γλώσσας Basic που ονόμασε VBScript. Τα προγράμματα που προκύπτουν από αυτές τις γλώσσες λέγονται «script».
Spring [98]	Το Spring, προϊόν της εταιρείας SpringSource, είναι ένα ελαφρύ Java/J2EE DI πλαίσιο εφαρμογών που βασίζεται σε κώδικα που δημοσιεύθηκε στο βιβλίο «Expert One-on-One J2EE Design and Development» (ανοικτού λογισμικού) (R. Johnson, J2EE Design and Development, ISBN 0-7645-4385-7, Wiley Publishing Inc., Indianapolis, 2003) από τον βασικό συντελεστή του framework, Rod Johnson. Αναπτύχθηκε κατά κύριο λόγο, για να αντιμετωπίσει την πολυπλοκότητα ανάπτυξης enterprise εφαρμογών. Το Spring κάνει εφικτή την χρήση απλών JavaBeans για την επίτευξη πραγμάτων που προηγουμένως μπορούσαν να γίνουν μόνο μέσω EJBs. Παρ΄ όλα αυτά, το Spring δεν είναι χρήσιμο μόνο για την ανάπτυξη server-side εφαρμογών. Η χρήση του μπορεί να βοηθήσει στην απλοποίηση του κώδικα, τον ευκολότερο και πιο αποτελεσματικό έλεγχο και τη χαλαρή διασύνδεση (loose coupling) κάθε Java εφαρμογής.
TLS	Το TLS (Transport Layer Security) είναι ένα πρωτόκολλο που εγγυάται ότι κατά την επικοινωνία εξυπηρέτη - πελάτη (server -client) μέσω του

	Διαδικτύου δεν πρόκειται να μεσολαβήσει κάποιος τρίτος που θα "υποκλέψει" το περιεχόμενο της επικοινωνίας. Το TLS αποτελεί το διάδοχο του Secure Sockets Layer (SSL).
Tomcat [99]	Ο Tomcat είναι ένας pure Java HTTP εξυπηρέτης και servlet container και ίσως είναι ο πιο γνωστός εξυπηρέτης. Χρησιμοποιείται εδώ και πολλά χρόνια και ήταν ενσωματωμένος ως container στις αρχικές εκδόσεις του εξυπηρέτη JBoss. Ο Tomcat χρησιμοποιήθηκε επίσης ως η εφαρμογή αναφοράς για την προδιαγραφή servlet. Σταμάτησε στο servlet API 2.5, όταν αυξήθηκε η ζήτηση εναλλακτικών επιλογών οι οποίες βασίζονταν σε non-blocking I/O (όπως το Jetty). Τέλος, υποστηρίζει την Προδιαγραφή Servlet 3.0. Ο Tomcat αναπτύσσεται από το Apache Software Foundation (ASF) και είναι ανοιχτού κώδικα.
WebSphere [100]	Ο WebSphere είναι ένας εξυπηρέτης εφαρμογών. Από την έκδοση 8 παρέχει υποστήριξη για το Servlet 3 API και περιέχει το προτυποποιημένο ασύγχρονο API για Comet. Ο WebSphere αναπτύσσεται από την IBM.
Web Container	Οι διαδικτυακές εφαρμογές αποτελούνται από στοιχεία Διαδικτύου (Web component) και άλλα δεδομένα όπως είναι οι HTML σελίδες. Τα Web component μπορεί να είναι servlet, σελίδες JSP, φίλτρα (Web filters) και ακροατές γεγονότων (Web event listener). Τα στοιχεία αυτά συνήθως εκτελούνται σε έναν εξυπηρέτη και μπορεί να αντιστοιχούν σε HTTP αιτήσεις πελατών. Τα servlet, οι σελίδες JSP και τα φίλτρα μπορεί να χρησιμοποιηθούν για τη δημιουργία HTML σελίδων. Εναλλακτικά, μπορούν να χρησιμοποιηθούν για τη δημιουργία δεδομένων (π.χ. σε XML) τα οποία χρησιμοποιούνται στη συνέχεια από τα στοιχεία της εφαρμογής. Το Web container αποτελεί μέρος του εξυπηρέτη και παρέχει το περιβάλλον εκτέλεσης (runtime environment) για τα components Servlet και JSP. Κάθε εξυπηρέτης εφαρμογών υλοποιεί στο Web container τις δικές του κλάσεις για τα interface αυτών των component. Το Web container

	διαχειρίζεται τον κύκλο ζωής τους, το γενικό πλαίσιο της ονοματοδοσίας (naming context – αντιστοιχεί ένα URL σε ένα συγκεκριμένο component και διασφαλίζει πως αυτός που ζήτησε το URL έχει τα απαιτούμενα δικαιώματα), την ασφάλεια, το συγχρονισμό (concurrency), τη συναλλαγή (transaction), την ανάπτυξη (deployment) και άλλες υπηρεσίες.
XMPP [101]	Το Extensible Messaging and Presence Protocol (XMPP – πρώην Jabber) είναι ένα σύνολο ανοιχτών πρωτοκόλλων για επικοινωνία σε πραγματικό χρόνο, η οποία τροφοδοτεί ένα ευρύ φάσμα εφαρμογών, συμπεριλαμβανομένων των εξής: • άμεσης επικοινωνίας (instant messaging), βασισμένα στην XML • άμεσα μηνύματα, • παρουσία, • ταυτόχρονο chat πολλών χρηστών, • κλήσεις φωνητικές και βίντεο, • τη συνεργασία, • το ενδιάμεσο λογισμικό lightweight, • διανομή περιεχομένου (syndication), και • τη γενικευμένη δρομολόγηση των δεδομένων XML. Το δίκτυο Jabber ακολουθεί τη λογική πελάτη/εξυπηρέτη (δυο προγράμματα πελάτες δεν επικοινωνούν απ' ευθείας το ένα με το άλλο), παραμένοντας όμως αποκεντρωμένο (δεν υπάρχει κάποιος κεντρικός εξυπηρέτης). Ένας χρήστης προσδιορίζεται με ένα όνομα χρήστη και ένα όνομα εξυπηρέτη. Τα δυο πεδία αυτά συνδέονται με ένα @ και αποτελούν το λεγόμενο Jabber ID, ή JID. Ένα μοναδικό χαρακτηριστικό του συστήματος Jabber είναι αυτό των μεταφορών, επίσης γνωστές ως πύλες, οι οποίες επιτρέπουν στους

χρήστες την πρόσβαση σε δίκτυα βασισμένα σε άλλα πρωτόκολλα. Αυτά μπορεί να είναι πρωτόκολλα άμεσης επικοινωνίας, αλλά και πρωτόκολλα όπως το SMS ή το ηλεκτρονικό ταχυδρομείο. Σε αντίθεση με τα προγράμματα πελάτες που υποστηρίζουν ταυτοχρόνως πολλά πρωτόκολλα, το Jabber παρέχει αυτήν την υπηρεσία στο επίπεδο του εξυπηρέτη, διαμέσου των ειδικών πυλών που τρέχουν σ' αυτόν. Οποιοσδήποτε χρήστης Jabber μπορεί να «καταχωρηθεί» (register) σε μια από αυτές τις πύλες, παρέχοντας τις πληροφορίες που απαιτούνται για να καταγραφεί στο δίκτυο με τον οποίο τον συνδέει, και μπορεί έπειτα να επικοινωνήσει με τους χρήστες του δικτύου αυτού σαν να ήταν χρήστες Jabber (και αντιστρόφως). Αυτό σημαίνει ότι οποιοδήποτε πρόγραμμα πελάτης που υποστηρίζει πλήρως το πρωτόκολλο Jabber μπορεί να χρησιμοποιηθεί για οποιοδήποτε δίκτυο για το οποίο ο εξυπηρέτης Jabber παρέχει μια πύλη, χωρίς πρόσθετο κώδικα στον πελάτη.

Το XMPP, πέρα από την υλοποίηση basic Jabber instant messaging υποστηρίζει επίσης pubsub (publish and subscribe), ένα χαρακτηριστικό το οποίο επιτρέπει σε XMPP πελάτες να εγγράψουν XML data σε κάποιον XMPP εξυπηρέτη και στη συνέχεια δίνει το δικαίωμα σε άλλους πελάτες να εγγραφούν στον κόμβο και να λαμβάνουν ενημερώσεις οποτεδήποτε τα δεδομένα αυτά μεταβάλλονται.

ΑΝΑΦΟΡΕΣ

[1] T. Berners-Lee, Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web, HarperBusiness, 2000.

[2] P. Fraternali, G. Rossi και F. Sánchez-Figueroa, «Rich Internet Applications,» *IEEE Internet Computing,* τόμ. 14, αρ. 3, pp. 9-12, 2010.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach και T. Berners-Lee, «RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1,» 1999. [Ηλεκτρονικό]. Available: http://tools.ietf.org/html/rfc2616. [Πρόσβαση 2012].

[4] J. J. Garrett , «Ajax: A New Approach to Web Applications,» 2005. [Ηλεκτρονικό]. Available: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. [Πρόσβαση 2012].

[5] A. Russell, G. Wilkins, D. Davis και M. Nesbitt, «The Bayeux Specification, Bayeux Protocol -- Bayeux 1.0.0,» Dojo Foundation, 2005. [Ηλεκτρονικό]. Available: http://svn.cometd.org/trunk/bayeux/bayeux.html. [Πρόσβαση 2012].

[6] I. Paterson, D. Smith, P. Saint-Andre και J. Moffitt, «XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH),» XMPP Standards Foundation, 2010. [Ηλεκτρονικό]. Available: http://xmpp.org/extensions/xep-0124.html. [Πρόσβαση 2012].

[7] M. Pilgrim, HTML5: Up and Running, O'Reilly, 2010.

[8] W3C, «Server-Sent Events,» [Ηλεκτρονικό]. Available: http://dev.w3.org/html5/eventsource/. [Πρόσβαση 12 2 2013].

[9] «The WebSocket API, Candidate Recommendation,» W3C, 2012. [Ηλεκτρονικό]. Available: http://www.w3.org/TR/websockets/. [Πρόσβαση 2012].

[10] S. Ramakrishnan και V. Dayal, «The Pointcast Network,» σε *ACM SIGMOD International Conference on Management of Data*, NY, 1998.

[11] B. Hammersley, Content syndication with RSS, O'Reilly, 2003.

[12] W. D. Team, «waste: anonymous, secure, encrypted sharing,» [Ηλεκτρονικό]. Available: http://waste.sourceforge.net/. [Πρόσβαση 12 2 2013].

[13] J. Oikarinen και D. Reed, «Internet Relay Chat Protocol,» 5 1993. [Ηλεκτρονικό]. Available: https://tools.ietf.org/html/rfc1459. [Πρόσβαση 12 2 2013].

[14] P. Saint-Andre, «Extensible Messaging and Presence Protocol (XMPP): Address Format,» 3 2011. [Ηλεκτρονικό]. Available: http://tools.ietf.org/html/rfc6122. [Πρόσβαση 12 2 2013].

[15] M. Mahemoff, Ajax Design Patterns, O'Reilly , 2006.

[16] K. Birman και T. Joseph, «Exploiting virtual synchrony in distributed systems,» σε *Eleventh ACM Symposium on Operating Systems Principles*, NY, 1987.

[17] P. Eugster, P. Felbe, R. Guerraoui και A. Kermarrec, «The many faces of publish/subscribe,» σε *CSUR ACM Computing Surveys*, NY, 2003.

[18] «RFC 793 - Transmission Control Protocol,» [Ηλεκτρονικό]. Available: http://www.ietf.org/rfc/rfc793.txt. [Πρόσβαση 2012].

[19] «W3C,» MIT, ERCIM, Keio, [Ηλεκτρονικό]. Available: http://www.w3.org/. [Πρόσβαση 2012].

[20] «The Internet Engineering Task Force (IETF),» Internet Society (ISOC), [Ηλεκτρονικό]. Available: http://www.ietf.org/. [Πρόσβαση 2012].

[21] «When can I use...,» based on data from StatCounter GlobalStats, 2012. [Ηλεκτρονικό]. Available: http://caniuse.com/. [Πρόσβαση 2012].

[22] S. Loreto, P. Saint-Andre, S. Salsano και G. Wilkins, «Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP,» Internet Engineering Task Force (IETF), 2011. [Ηλεκτρονικό]. Available: http://www.hjp.at/doc/rfc/rfc6202.html#sec_2.1, IETF. [Πρόσβαση 2012].

[23] «Javaworld: An in-depth look at RMI callbacks,» Infoworld, 1999. [Ηλεκτρονικό]. Available: http://www.javaworld.com/javaworld/javaqa/1999-04/05-rmicallback.html?page=3. [Πρόσβαση 2012].

[24] P. Sarang, «Java: Article: Callbacks In CORBA,» SYS-CON Media, 1999. [Ηλεκτρονικό]. Available: http://java.sys-con.com/node/36132. [Πρόσβαση 2012].

[25] «Ενδιάμεσο Λογισμικό,» Parallel and Distributed Processing Laboratory, [Ηλεκτρονικό]. Available: http://pdplab.it.uom.gr/teaching/ince_2e_gr/Text/C3/Middleware_3.htm. [Πρόσβαση 2012].

[26] «iBus//MessageBus Programmer's Manual,» SoftWired AG, 1999. [Ηλεκτρονικό]. Available: http://www.scis.ulster.ac.uk/~kevin/ibus.pdf. [Πρόσβαση 2012].

[27] «MQ,» MQSeries.net , [Ηλεκτρονικό]. Available: http://www.mqseries.net/. [Πρόσβαση 2012].

[28] «Overview of WebLogic Events,» BEA Systems, 2001. [Ηλεκτρονικό]. Available: http://docs.oracle.com/cd/E13222_01/wls/docs61/event/overview.html. [Πρόσβαση 2012].

[29] P. 5. Porters, «Perl Programming Documentation: CGI::Push,» [Ηλεκτρονικό]. Available: http://perldoc.perl.org/5.12.1/CGI/Push.html. [Πρόσβαση 2012].

[30] N. Freed και N. Borenstein, «Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,» Network Working Group, 1996. [Ηλεκτρονικό]. Available: http://tools.ietf.org/html/rfc2045. [Πρόσβαση 2012].

[31] R. Gravelle, «Comet Programming: the Hidden IFrame Technique,» QuinStreet, [Ηλεκτρονικό]. Available: http://www.webreference.com/programming/javascript/rg30/index.html. [Πρόσβαση 2012].

[32] T. Powell και F. Schneider, JavaScript: The Complete Reference, Third Edition, McGraw-Hill, 2012.

[33] J. v. d. Broecke, «Pushlets,» 2007. [Ηλεκτρονικό]. Available: http://www.pushlets.com/. [Πρόσβαση 2012].

[34] «The WHATWG,» [Ηλεκτρονικό]. Available: http://wiki.whatwg.org/wiki/FAQ#What_is_the_WHATWG.3F. [Πρόσβαση 2012].

[35] P. McCarthy, «Comet & Java: Threaded Vs Nonblocking I/O,» [Ηλεκτρονικό]. Available: http://iobound.com/2008/11/comet-nio/. [Πρόσβαση 2012].

[36] T. Garnock-Jones, «ReverseHttp,» LShift Ltd., 2009. [Ηλεκτρονικό]. Available: http://reversehttp.net/. [Πρόσβαση 2012].

[37] «Push technology: Flash XMLSocket relays,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Push_technology#Flash_XMLSocket_relays. [Πρόσβαση 2012].

[38] «Stream Control Transmission Protocol (SCTP),» 2004. [Ηλεκτρονικό]. Available: http://www.sctp.org/. [Πρόσβαση 2012].

[39] P. Natarajan, P. Amer, J. Leighton και F. Baker, «Using SCTP as a Transport Layer Protocol for HTTP,» Network Working Group , 2008. [Ηλεκτρονικό]. Available: http://tools.ietf.org/html/draft-natarajan-http-over-sctp-00. [Πρόσβαση 2012].

[40] «Structured Stream Transport,» SST is a part of the UIA project at the Parallel and Distributed Operating Systems Group in MIT's Computer Science and Artificial Intelligence Laboratory, 2007. [Ηλεκτρονικό]. Available: http://pdos.csail.mit.edu/uia/sst/. [Πρόσβαση 2012].

[41] H. F. Nielsen, «MUX,» 2000. [Ηλεκτρονικό]. Available: http://www.w3.org/Protocols/MUX/. [Πρόσβαση 2012].

[42] J. Gettys και H. F. Nielsen, «SMUX,» 1998. [Ηλεκτρονικό]. Available: http://www.w3.org/TR/WD-mux. [Πρόσβαση 2012].

[43] Chromium Developer Documentation, «SPDY: An experimental protocol for a faster web,» Google, 2010. [Ηλεκτρονικό]. Available: http://dev.chromium.org/spdy/. [Πρόσβαση 2012].

[44] «SPDY Server tool,» Google, 2012. [Ηλεκτρονικό]. Available: http://src.chromium.org/viewvc/chrome/trunk/src/net/tools/flip_server/. [Πρόσβαση 2012].

[45] «SPDY Chromium client implementation,» Google, 2012. [Ηλεκτρονικό]. Available: http://src.chromium.org/viewvc/chrome/trunk/src/net/spdy/. [Πρόσβαση 2012].

[46] «The Chromium Project: Benchmarking Extension,» Google, 2010. [Ηλεκτρονικό]. Available: http://dev.chromium.org/developers/design-documents/extensions/how-the-extension-system-works/chrome-benchmarking-extension. [Πρόσβαση 2012].

[47] A. Langley, «TLS Next Protocol Negotiation,» Google, 2011. [Ηλεκτρονικό]. Available: https://technotes.googlecode.com/git/nextprotoneg.html. [Πρόσβαση 2012].

[48] E. Rescorla, «HTTP Over TLS,» Network Working Group , 2000. [Ηλεκτρονικό]. Available: http://www.ietf.org/rfc/rfc2818.txt. [Πρόσβαση 2012].

[49] «Polyfill,» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Polyfill. [Πρόσβαση 2012].

[50] M. Carbou, «Reverse Ajax, Part 2: WebSockets,» IBM, 2011. [Ηλεκτρονικό]. Available: http://www.ibm.com/developerworks/web/library/wa-reverseajax2/#N10221. [Πρόσβαση 2012].

[51] G. Roth, «Architecture of a Highly Scalable NIO-Based Server,» Oracle, 2007. [Ηλεκτρονικό]. Available: http://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html. [Πρόσβαση 2012].

[52] G. Wilkins, «Interface Continuation,» Mort Bay Consulting , 2009. [Ηλεκτρονικό]. Available: http://jetty.codehaus.org/jetty/jetty-6/apidocs/org/mortbay/util/ajax/Continuation.html. [Πρόσβαση 2012].

[53] G. Wilkins, «Jetty Continuations,» 2008. [Ηλεκτρονικό]. Available: http://docs.codehaus.org/display/JETTY/Continuations. [Πρόσβαση 2012].

[54] P. McCarthy, «Inside Jetty's Continuations mechanism,» IBM, 2007. [Ηλεκτρονικό]. Available: http://www.ibm.com/developerworks/web/library/j-jettydwr/index.html#N10136. [Πρόσβαση 2012].

[55] «Apache Tomcat 6.0: Advanced IO and Tomcat,» Apache Software Foundation, 2011. [Ηλεκτρονικό]. Available: http://tomcat.apache.org/tomcat-6.0-doc/aio.html. [Πρόσβαση 2012].

[56] «GlassFish: Grizzly,» Oracle, [Ηλεκτρονικό]. Available: http://grizzly.java.net/.

[57] «Advanced IO and JBoss Web,» Red Hat, [Ηλεκτρονικό]. Available: http://docs.jboss.org/jbossweb/latest/aio.html. [Πρόσβαση 2012].

[58] «JSR 315: JavaTM Servlet 3.0 Specification,» Oracle, [Ηλεκτρονικό]. Available: http://jcp.org/en/jsr/detail?id=315. [Πρόσβαση 2012].

[59] «Migratory Push Server,» Migratory Data Systems, [Ηλεκτρονικό]. Available: http://migratorydata.com/products/migratory-push-server.html. [Πρόσβαση 2012].

[60] «Load (computing),» [Ηλεκτρονικό]. Available: http://en.wikipedia.org/wiki/Load_(computing). [Πρόσβαση 2012].

[61] «Load Testing - Simulating Users,» Altica, 2012. [Ηλεκτρονικό]. Available: http://www.altica.co.uk/load-testing-simulating-users. [Πρόσβαση 2012].

[62] «CometD 2.4.0 WebSocket Benchmarks,» Webtide Blogs, 2011. [Ηλεκτρονικό]. Available: http://webtide.intalio.com/2011/09/cometd-2-4-0-websocket-benchmarks/. [Πρόσβαση 2012].

[63] G. Rauch, «Introducing SOCKET IO V.9,» LearnBoost Labs, 2012. [Ηλεκτρονικό]. Available: http://socket.io/. [Πρόσβαση 2012].

[64] «LearnBoost / socket.io,» GitHub Inc., [Ηλεκτρονικό]. Available: https://github.com/learnboost/socket.io. [Πρόσβαση 2012].

[65] «CometD Project,» The Dojo Foundation, [Ηλεκτρονικό]. Available: http://cometd.org/. [Πρόσβαση 2012].

[66] «Welcome to Atmosphere: The Asynchronous WebSocket/Comet Framework,» [Ηλεκτρονικό]. Available: https://github.com/Atmosphere/atmosphere. [Πρόσβαση 2012].

[67] «What is WebSync?,» Frozen Mountain Software, 2010. [Ηλεκτρονικό]. Available: http://www.frozenmountain.com/websync/. [Πρόσβαση 2012].

[68] «Ajax Push Engine (APE) - Complete Comet solution,» weelya, 2012. [Ηλεκτρονικό]. Available: http://www.ape-project.org/. [Πρόσβαση 2012].

[69] Leo P, «NGiNX HTTP Push Module,» slact, 2010. [Ηλεκτρονικό]. Available: http://pushmodule.slact.net/. [Πρόσβαση 2012].

[70] «nginx,» 2012. [Ηλεκτρονικό]. Available: http://nginx.org/en/. [Πρόσβαση 2012].

[71] «Basic HTTP Push Relay Protocol,» slact, [Ηλεκτρονικό]. Available: http://pushmodule.slact.net/protocol.html. [Πρόσβαση 2012].

[72] «Caplin Platform: Caplin Liberator,» Caplin Systems, [Ηλεκτρονικό]. Available: http://www.caplin.com/caplin-platform. [Πρόσβαση 2012].

[73] «Lightstreamer,» weswit, 2012. [Ηλεκτρονικό]. Available: http://www.lightstreamer.com/. [Πρόσβαση 2012].

[74] «wt: a C++ Web Toolkit,» emweb, 2012. [Ηλεκτρονικό]. Available: http://www.webtoolkit.eu/wt. [Πρόσβαση 2012].

[75] «StreamHub: The Reverse Ajax & Comet Server,» 2011. [Ηλεκτρονικό]. Available: http://www.stream-

hub.com/. [Πρόσβαση 2012].

[76] «XSockets,» [Ηλεκτρονικό]. Available: http://www.xsockets.net/. [Πρόσβαση 2012].

[77] «ServerFramework: The WebSockets Option Pack,» JetByte Limited , [Ηλεκτρονικό]. Available: http://www.serverframework.com/products---the-websockets-option.html. [Πρόσβαση 2012].

[78] «WebSocket++,» Zaphoyd Studios, 2012. [Ηλεκτρονικό]. Available: http://www.zaphoyd.com/websocketpp/. [Πρόσβαση 2012].

[79] I. Grigorik, «EM-WebSocket,» 2012. [Ηλεκτρονικό]. Available: https://github.com/igrigorik/em-websocket. [Πρόσβαση 2012].

[80] D. Simpson, «em-ws-client,» 2012. [Ηλεκτρονικό]. Available: https://github.com/dansimpson/em-ws-client. [Πρόσβαση 2012].

[81] «jWebSocket,» jWebSocket.org , 2012. [Ηλεκτρονικό]. Available: http://jwebsocket.org/. [Πρόσβαση 2012].

[82] «WebSocket-Node,» [Ηλεκτρονικό]. Available: https://github.com/Worlize/WebSocket-Node. [Πρόσβαση 2012].

[83] «Node.js,» Joyent, Inc, 2012. [Ηλεκτρονικό]. Available: http://nodejs.org/. [Πρόσβαση 2012].

[84] «cWebsocket,» 2012. [Ηλεκτρονικό]. Available: https://github.com/m8rge/cwebsocket. [Πρόσβαση 2012].

[85] «libwebsockets,» 2012. [Ηλεκτρονικό]. Available: http://git.warmcat.com/cgi-bin/cgit/libwebsockets/. [Πρόσβαση 2012].

[86] «pywebsocket,» 2012. [Ηλεκτρονικό]. Available: http://code.google.com/p/pywebsocket/. [Πρόσβαση 2012].

[87] «WebSocket-for-Python,» 2012. [Ηλεκτρονικό]. Available: https://github.com/Lawouach/WebSocket-for-Python. [Πρόσβαση 2012].

[88] I. Fette και A. Melnikov, «http://tools.ietf.org/html/rfc6455,» Internet Engineering Task Force (IETF), 2011. [Ηλεκτρονικό]. Available: http://tools.ietf.org/html/rfc6455. [Πρόσβαση 2012].

[89] «tornado.websocket,» 2012. [Ηλεκτρονικό]. Available: http://www.tornadoweb.org/documentation/websocket.html. [Πρόσβαση 2012].

[90] «AutobahnPython,» Tavendo GmbH , 2012. [Ηλεκτρονικό]. Available: http://autobahn.ws/python/reference. [Πρόσβαση 2012].

[91] «AutobahnAndroid,» Tavendo GmbH, 2012. [Ηλεκτρονικό]. Available: http://autobahn.ws/android/reference. [Πρόσβαση 2012].

[92] «Glassfish,» Oracle Foundation, [Ηλεκτρονικό]. Available: http://glassfish.java.net/. [Πρόσβαση 2012].

[93] «Google Guice,» Google , [Ηλεκτρονικό]. Available: http://code.google.com/p/google-guice/. [Πρόσβαση 2012].

[94] «JBOSS,» Red Hat, [Ηλεκτρονικό]. Available: http://www.jboss.org/.

[95] «Jetty,» [Ηλεκτρονικό]. Available: http://www.eclipse.org/jetty/. [Πρόσβαση 2012].

[96] «JSON,» [Ηλεκτρονικό]. Available: http://www.json.org/.

[97] «PHP,» [Ηλεκτρονικό]. Available: http://www.php.net/manual/en/index.php. [Πρόσβαση 2012].

[98] «Spring,» SpringSource, [Ηλεκτρονικό]. Available: http://www.springsource.org/. [Πρόσβαση 2012].

[99] «Apache Tomcat,» Apache , [Ηλεκτρονικό]. Available: http://tomcat.apache.org/. [Πρόσβαση 2012].

[100] «IBM Software - WebSphere,» IBM, [Ηλεκτρονικό]. Available: http://www-01.ibm.com/software/websphere/. [Πρόσβαση 2012].

[101] «XMPP,» XMPP Standards Foundation, [Ηλεκτρονικό]. Available: http://xmpp.org/about-xmpp/. [Πρόσβαση 2012].

[102] M. Mahemoff, Ajax Design Patterns, O'Reilly, 2006.

[103] D. Schiemann, «The forever-frame technique,» Comet Daily, LLC, 2007. [Ηλεκτρονικό]. Available: http://cometdaily.com/2007/11/05/the-forever-frame-technique/. [Πρόσβαση 2012].

[104] «HTML Living Standard: Server Sent Events,» [Ηλεκτρονικό]. Available: http://www.whatwg.org/specs/web-apps/current-work/#server-sent-events. [Πρόσβαση 2012].

[105] P. McCarthy και D. Crane, Comet and Reverse Ajax: The Next-Generation Ajax 2.0, Firstpress, 2008.
[106] T. Berners-Lee, Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web, HarperBusiness, 2000.