Thesis no: urn:nbn:se:bth-14497

Performance comparison of XHR polling, Long polling, Server sent events and Websockets

Rasmus Appelqvist
Oliver Örnmyr

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Contact Information:
Author(s):
Rasmus Appelqvist
E-mail: rasmus.a-91@hotmail.com

Oliver Örnmyr
E-mail: olor14@student.bth.se

University advisor:
Javier Gonzales Huerta PhD
Department of Software Engineering

Faculty of Computing	**Internet : www.bth.se**
Blekinge Institute of Technology	**Phone : +46 455 38 50 00**
SE-371 79 Karlskrona, Sweden	**Fax : +46 455 38 50 57**

ABSTRACT

Context. Many simultaneous clients that connects to a website where the client receives frequent updates from the server will put stress on the server. Selecting a performance efficient technology for sending updates to the client when building the web sites can allow for lesser server hardware requirements which can result in reduced server hardware costs. Knowledge about the difference between the technologies that can solve the problem of updating a client with information from the server is necessary to decide which technology to use to gain performance.

Objectives. This study considers what performance factors that affect XHR polling, Long polling, Server sent events and Websockets to allow web application developers to decide on when to use which technology. This study will also try to find out if the performance difference between the compared technologies are noticeable and which one is the best from a performance perspective. A performance impacting factor is defined in this study as a feature or property of a technology that will have an impact on the performance of the server.

Methods. The study is done in three steps. The first step is a summary of the specifications for the compared technologies and an analysis of this to identify performance impacting factors for each technology. The second step is an empirical study where the performance of each technology is tested by implementing each technology and measuring the system resources used by the server. Finally, a conclusion is made by comparing the results of the technology specification analysis with the results from the performance tests.

Conclusions. The results of this study found that the choice of technology does matter when it comes to web server performance. Websockets and Server sent events were measured to be the most performance efficient of the compared technologies in the experimental conditions used in this study. Factors impacting the server performance were found to be the amount of connections used, if requests had to be received and processed on the server and how much overhead data is needed. It was not determined if these are the only performance impacting factors or if there are more factors impacting the servers that were not found in this study.

Keywords: web server performance, performance comparison, web communication, dynamic web pages

LIST OF ABBREVIATIONS AND ACRONYMS

- CPU - Central processing unit
- GHz - Gigahertz
- GB - Gigabytes
- HTML - Hypertext markup language
- HTTP - Hypertext transport protocol (version 1.1 unless otherwise stated)
- ID - Identifier
- IETF - Internet Engineering Task Force
- JSON - JavaScript Object Notation
- KB - Kilobytes
- MiB - Mebibytes
- ms - milliseconds
- n.d - No date
- RFC - Request for comments
- RQ - Research question
- SSD - Solid state drive
- SSE - Server sent events
- TCP - Transmission control protocol
- TLS - Transport layer security
- URL - Uniform Resource Locator
- WHATWG - Web Hypertext Application Technology Working Group
- XHR - XMLHttpRequest
- XML - Extensible Markup Language
- XOR - Exclusive or (logical operation)

CONTENTS

ABSTRACT .... **I**

LIST OF ABBREVIATIONS AND ACRONYMS **1**

CONTENTS **2**

1 INTRODUCTION **3**

1.1 BACKGROUND 3
1.2 MOTIVATION 3
1.3 VALUE 3

2 RESEARCH QUESTIONS **4**

3 METHOD **5**

3.1 ANALYSIS OF THE PERFORMANCE FACTORS BASED ON TECHNOLOGY SPECIFICATIONS 5
3.2 EMPIRICAL STUDY 5

4 TECHNOLOGIES UNDER STUDY AND MAIN PERFORMANCE FACTORS **6**

4.1 WEBSOCKETS 6
4.2 XHR-POLLING 8
4.3 HTTP LONG POLLING 10
4.4 SERVER SENT EVENTS (SSE) 11
4.5 PERFORMANCE FACTOR IDENTIFICATION 13

5 PERFORMANCE TESTS **16**

5.1 PERFORMANCE METRICS USED 16
5.2 THE TEST SETUP 17
5.3 PAYLOAD 20
5.4 OVERHEAD SIZES 20
5.5 IDLE TEST 20
5.6 TEST CASE DESIGN 22
5.7 PERFORMANCE TEST 1 23
5.8 PERFORMANCE TEST 2 29
5.9 PERFORMANCE TEST 3 34

6 ANALYSIS **40**

6.1 ANALYSIS 40
6.2 RESEARCH QUESTION ANSWERS 42

7 CONCLUSION **43**

8 VALIDITY THREATS **44**

8.1 EXTENSIBILITY OF THE RESULTS 44
8.2 UNKNOWN GARBAGE COLLECTOR BEHAVIOR, NETWORK CONNECTION AND INACCURATE CLIENTS AFFECTING TEST RESULTS 44
8.3 SERVERS OR CLIENTS NOT FOLLOWING THE STANDARD 44

9 FUTURE WORK **45**

9.1 LOAD BALANCING 45
9.2 HTTP OVER TLS 45
9.3 HTTP/2 45
9.4 MORE POWERFUL AND VARIED PERFORMANCE TESTS 45

REFERENCES **46**

1 INTRODUCTION

1.1 Background

The issue of deciding which technology to use is commonly encountered when building software. It is therefore useful to know how the considered technologies differ from each other to make a good decision. One possible difference between technologies is performance and that is the focus of this study.

Dynamically updating web pages during the users stay on the page is used in many web sites today and there are different technologies to solve the dynamic updates. The performance of web sites will be affected by what technologies is used to solve the dynamic updating. Better performing website software means that hardware resources can be used more efficiently and can lead to reduced hardware costs. In this study, the following technologies are compared to find the most performance efficient technology: HTTP Long polling, XHR-polling, Websockets and Server Sent Events. These technologies were chosen because they appear to be the most common occurring technologies encountered when searching the web for technologies that can dynamically update web pages.

1.2 Motivation

Many web sites on the internet are updating the web pages dynamically without having to reload the whole document. This is something that the studied technologies help to enable. Examples of this is chat applications, news feeds, web based email clients etc. Knowing what technology to use to gain performance can lead to reduced stress per client on the servers serving the web pages. This could be important for companies to save costs when expanding their server capacities. Less powerful hardware with performance efficient software can outperform more powerful hardware running performance heavy software.

1.3 Value

This study can help in making the decision of which technology to use when implementing dynamic web page updates and the performance matters. Better performing servers can increase user experience on the website.

Knowing which performance factors that impact the technologies can help in understanding which technology would be the most efficient in a specific situation can be useful to increase your server capacity without increasing the hardware related costs. Reduced hardware cost can enable businesses to lower server prices and gain more customers.

Knowing the factors that are responsible for limiting the performance of these technologies can help to develop better performing technologies in the future.

2 RESEARCH QUESTIONS

RQ1: Which of the compared technologies is the highest performing for server to client communication?

Motivation

It is necessary to know which technology is the highest performing to reduce the performance impact on the server when deciding which technology to use, and this is what this question aims to answer.

RQ2: Which factors, when considering server to client communication, of the compared technologies have an impact on server performance?

Motivation

The performance difference between the studied technologies might be dependent on the context they are used in. In this case, it is necessary to know what factors affects the performance of that technology when deciding on the technology choice.

3 METHOD

First an analysis of the technology specifications was done to identify performance factors affecting the studied technologies. Then an empirical study was done where performance tests were made. Afterwards the results from the analysis and performance tests are discussed which leads to the conclusion.

3.1 Analysis of the performance factors based on technology specifications

Finding and analyzing specifications of the studied technologies provides a good understanding of how the technologies work. This enables considering of every part of how the technologies handle communication between the client and server. The idea is to identify performance impacting factors by going through each step of the communication. Previous research identifying performance factors for the compared or similar technologies were searched for but not found.

Specifications of web technologies can be found as RFC (Request for Comments) entries hosted by the IETF (Internet Engineering Task Force). The IETF is an organization working with the evolution of the Internet architecture and the operation of the Internet. All the RFC's provided by the IETF is public and available on their website (Alvestrand 2004). This is one of the main sources used to retrieve specifications to understand on how the technologies work. The implementation of the different technologies can differ from the RFC's. Different browsers and web servers may have implemented the technologies differently, so it was also necessary to look for some more technical information for the different implementations as well.

The purpose of this analysis is to identify possible performance impacting factors.

3.2 Empirical study

To confirm that the result of the technology specification analysis is accurate, experiments were conducted in the form of performance tests on the technologies compared in this study. The goal of the empirical was to see if the results gathered is in line with the conclusions made from the technology summaries.

The experiments help to answer or reject the conclusions of the research questions in the following way:

RQ1: Which of the compared technologies is the highest performing for server to client communication?
If the experiments show different performance levels for different technologies, it should be possible to determine which is the highest performing.

RQ2: Which factors, when considering server to client communication, of the compared technologies have an impact on server performance?
The tests help to confirm or deny the validity of the factors identified in the technology specification analysis. For example, it could be concluded that one technology specifies that more overhead data is needed for one technology than another. The performance tests can confirm this by resulting in corresponding CPU, memory and network traffic differences etc.

A detailed description of how the tests were done can be found in section 5.2.

4 TECHNOLOGIES UNDER STUDY AND MAIN PERFORMANCE FACTORS

This section contains a summary and analysis of the technology specifications to identify the factors that have an impact on how the technologies perform on the server.

4.1 Websockets

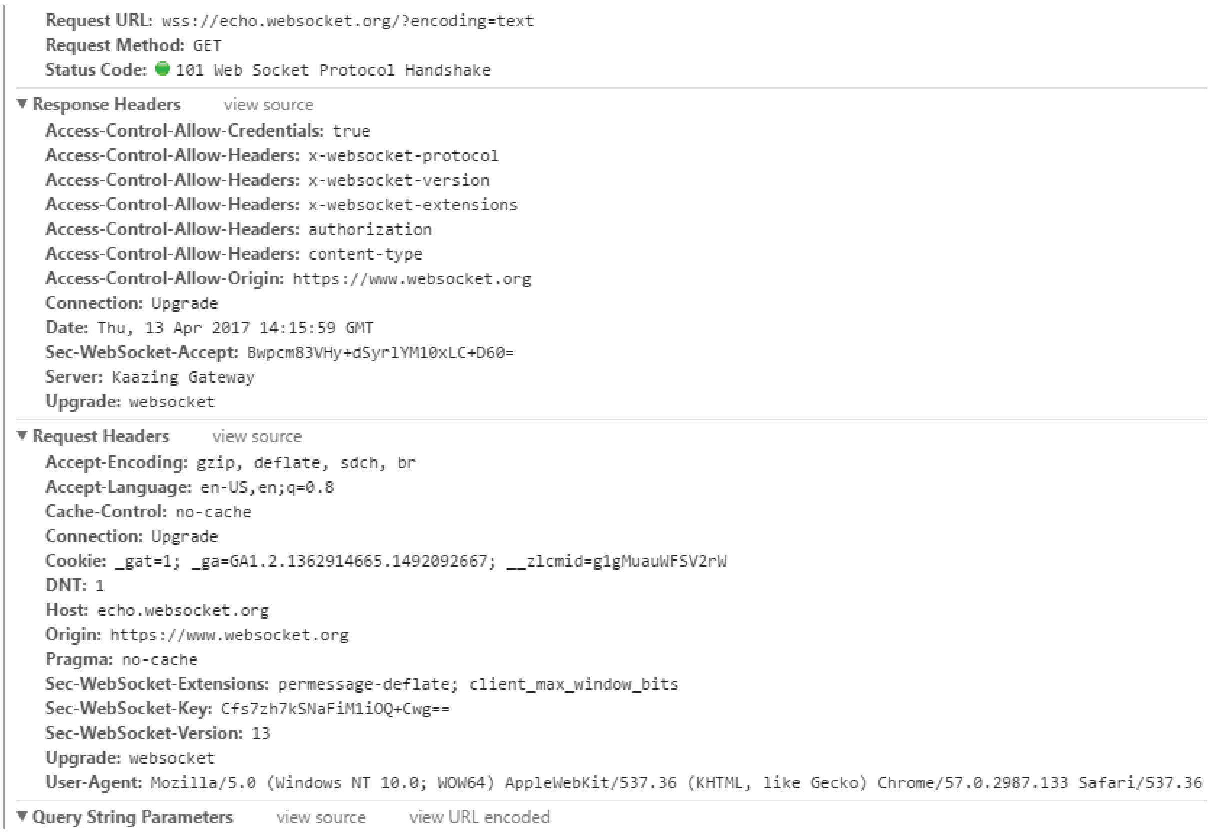
This section is split into two parts. The first part is describing how the connection is established between the client and server. The second part is describing how data is being sent over the connection described in the first

4.1.1 Establishing a connection

The client initiates the Websocket connection by establishing a TCP connection with the server and sending a HTTP request. The HTTP request will include headers stating that the client wishes to establish a Websocket connection (MDN 2017).

If the server supports the Websocket protocol it will inform the client that the server accepts the request by sending a HTTP response. These are the only HTTP requests and responses needed for communicating over the Websocket protocol. (Fette and Melnikov, 2011).

The figure below (Figure 1) illustrates the HTTP request and response sent between the client and server to establish a Websocket connection. The figure is a screenshot taken from Google Chrome Developer tools version 57.0.2987.133 when establishing a Websocket connection to a website.



```
Request URL: wss://echo.websocket.org/?encoding=text
Request Method: GET
Status Code: ● 101 Web Socket Protocol Handshake
▼ Response Headers    view source
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: x-websocket-protocol
Access-Control-Allow-Headers: x-websocket-version
Access-Control-Allow-Headers: x-websocket-extensions
Access-Control-Allow-Headers: authorization
Access-Control-Allow-Headers: content-type
Access-Control-Allow-Origin: https://www.websocket.org
Connection: Upgrade
Date: Thu, 13 Apr 2017 14:15:59 GMT
Sec-WebSocket-Accept: Bwpcm83VHy+dSyrlYM10xLC+D60=
Server: Kaazing Gateway
Upgrade: websocket
▼ Request Headers    view source
Accept-Encoding: gzip, deflate, sdch, br
Accept-Language: en-US,en;q=0.8
Cache-Control: no-cache
Connection: Upgrade
Cookie: _gat=1; _ga=GA1.2.1362914665.1492092667; __zlcmid=g1gMuauWFSV2rW
DNT: 1
Host: echo.websocket.org
Origin: https://www.websocket.org
Pragma: no-cache
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
Sec-WebSocket-Key: Cfs7zh7kSNaFiM1iOQ+Cwg==
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36
▼ Query String Parameters    view source    view URL encoded
```

Figure 1. The measured HTTP headers.

From this we can conclude that one of the Websocket performance factors is possibly the establishment of the initial connection. The HTTP request will need to be parsed by the server which needs some processing power and memory. By looking at the figure above, it was estimated that the request and response together make up about 1KB of data. This was estimated by counting the amount of characters. Each character in a HTTP request or response takes up one byte (8 bits) of memory (Fielding et al. 1999). A small amount of processing power and memory will also be used to set up the TCP socket connection.

It can be concluded that since no more HTTP headers must be sent after establishing the Websocket connection the only overhead information needed to continue the communication will be the overhead data specified by the Websocket protocol itself and is no longer influenced by the HTTP.

4.1.2 Sending data between the client and server

Communication between client and server using the Websocket protocol is full duplex. This means both the client and server have the possibility to simultaneously send data to each other without having to wait for a response or only send one way at a time. (Fette and Melnikov 2011).

Communicating using the Websocket protocol is not purely sending data over a TCP socket. The protocol does add some of its own overhead information as illustrated in the figure below (Figure 2). It is possible to split a message into multiple frames, but not necessary except if the payload size exceeds the maximum value of an unsigned 64-bit value in bytes (Fette and Melnikov 2011).

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-------+-+-------------+-------------------------------+
 |F|R|R|R| opcode|M| Payload len |    Extended payload length    |
 |I|S|S|S|  (4)  |A|     (7)     |             (16/64)           |
 |N|V|V|V|       |S|             |   (if payload len==126/127)   |
 | |1|2|3|       |K|             |                               |
 +-+-+-+-+-------+-+-------------+ - - - - - - - - - - - - - - - +
 |     Extended payload length continued, if payload len == 127  |
 + - - - - - - - - - - - - - - - +-------------------------------+
 |                               |Masking-key, if MASK set to 1  |
 +-------------------------------+-------------------------------+
 | Masking-key (continued)       |          Payload Data         |
 +-------------------------------- - - - - - - - - - - - - - - - +
 :                     Payload Data continued ...                :
 + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
 |                     Payload Data continued ...                |
 +---------------------------------------------------------------+
```

Figure 2. Websocket protocol header. Source: (Fette and Melnikov 2011)

Not all headers are always in use. “Payload len” can indicate the payload length in bytes and then the next two payload length headers are not used. However, if the payload is larger than 125 bytes the “Payload len” will be set to 126 and the “Extended payload length” will be used to indicate the payload length in bytes. If the payload is too big to be indicated by the 2 bytes of the “Extended payload length”; “Payload len” will be set to 127 and both the “Extended payload length” and “Extended payload length continued” fields will be used to indicate payload length in bytes. The “Masking-key” field will only be used if the “Mask” header is set which it should be for all client to server messages. (Fette and Melnikov 2011).

This means that for messages smaller than 126 bytes there will be a 2-byte overhead for server to client communication per message sent. When the payload size increases, the overhead information increases with it in steps. For messages between 126 bytes to about 65.5 kilobytes (2^{16} bytes) you would add another 2 bytes to the overhead data. If the payload is even bigger (between 2^{16} and 2^{64} bytes) another 6 bytes are added to the overhead data. To summarize, the overhead data for each message will be between 2 and 10 bytes depending on payload size.

Data masking
Because of security reasons the client must mask the payload when sending messages to the server using the Websocket protocol (Fette and Melnikov 2011).

In other words, this means that the server does not have to mask the data when sending messages to the client, which is the focus of this study. Data masking will because of this not be considered a performance factor for Websockets in this study.

4.2 XHR-Polling

XHR-Polling is built on HTTP and as such uses a request-response design. They can send a request to the server and the server can respond to this request. This means the client will have to keep sending requests to the server to get updates without the user reloading the webpage. (Fielding and Reschke 2014).

4.2.1 The connection

HTTP over the internet is most commonly achieved using a TCP connection that defaults to port 80. Assuming a situation with a simple request, where the client sends a request to which the server sends a response, a single TCP connection will be opened to send the request and will stay open until the response have been received by the client. (Fielding and Reschke 2014)

Persistent connection
To reduce the performance, hit of opening and closing TCP connections for every request there is a feature in HTTP called "Persistent connection". This feature allows multiple requests to be sent over the same TCP connection instead of closing the connection after the first response is sent. Both the client and the server can choose to close the connection. The specification does not say how long this connection stays open (Fielding and Reschke 2014). This means that the duration of this connection depends on both the client and server.

Client side
On the client side the duration of the persistent connection differs between browsers. For example; Internet explorer defaults to 60 seconds (Microsoft 2016), while Mozilla browsers like Firefox defaults to 115 seconds (MozillaZine 2011).

Server side
On the server side, it depends on what web server software is used. Apache which is one of the most commonly used web server defaults to keeping the persistent connection open for about five to fifteen seconds depending on which version of the apache software is used (Apache 2017a; 2017b; 2017c; 2017d).

Pipelining
When using persistent connection, it is possible for the client to use something called pipelining. Pipelining allows the client to send multiple requests to the server without having to wait for a response to the first request before sending the next one. However, it does require the server to respond to the requests in the same order as they came in. (Fielding and Reschke 2014)
Currently pipelining is not supported by all browsers because of bugs caused by misbehaving servers. Mozilla browsers and Google chrome are two examples of browsers that do not support pipelining by default. (MozillaZine 2012; MacVittie 2016).

Illustration
The figures below (Figure 3) visually explain how in theory the persistent connection and pipelining features could improve the performance of the HTTP protocol.

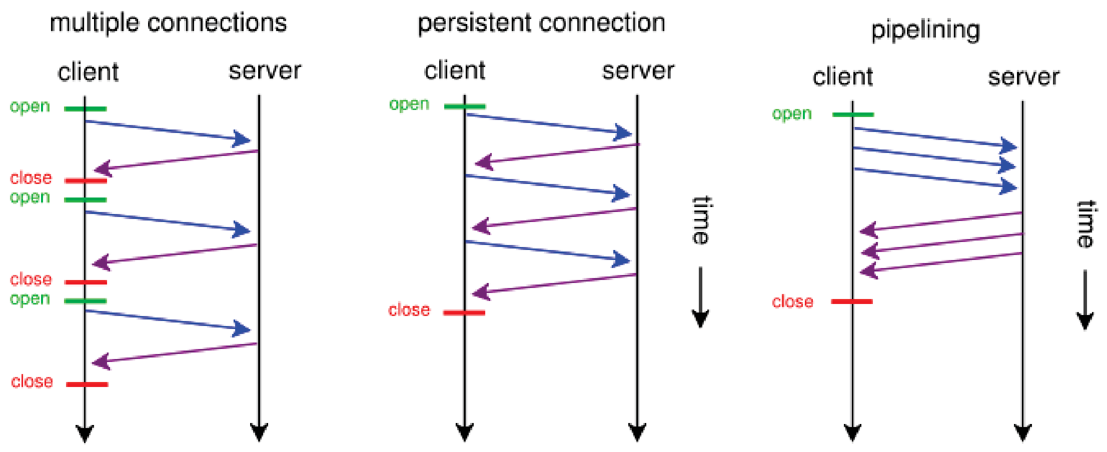


Figure 3. HTTP pipelining. Source: (MacVittie 2016)

Pipelining was concluded to be more useful for situations where many different resources is to be retrieved from the server at the same time, images for example, and not for retrieving messages from the server like a chat application would do.

4.2.2 The HTTP request

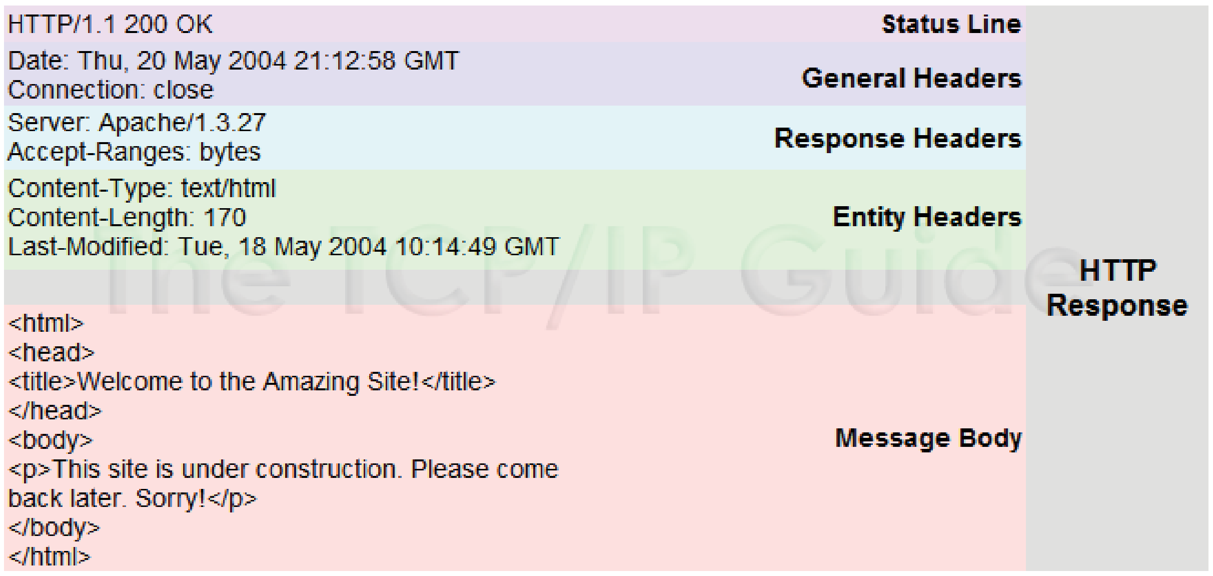
A HTTP request consists of overhead data and the actual message. The size of a HTTP request header can vary from about 200 to over 2000 bytes where “typical header sizes of 700-800 bytes is common” (Chromium 2017b). This means that the server will need to process about 700 to 800 bytes of overhead for each message sent from the client.

Processing the request on the server
When the request is delivered to the server it will need to be parsed by some HTTP parsing software. There are some different steps that the request will need to go through while being processed on the server (Oracle 2010):

- Authorization translation
  If the requested resource needs authorization the server would have to verify any authorization information sent in the request
- Name translation
  The URI in the request needs to be translated into a local file system path to figure out which resource on the web server the request is intended for
- Path checking
  The local file system path is checked for validity and what the access privileges are for this path/resource
- Log the request
  The request is logged to a file on the hard drive.

4.2.3 The HTTP response

The HTTP response also consists of overhead information and payload. The figure below (Figure 4) illustrates what a http response can look like:



```
HTTP/1.1 200 OK
Date: Thu, 20 May 2004 21:12:58 GMT
Connection: close
Server: Apache/1.3.27
Accept-Ranges: bytes
Content-Type: text/html
Content-Length: 170
Last-Modified: Tue, 18 May 2004 10:14:49 GMT

<html>
<head>
<title>Welcome to the Amazing Site!</title>
</head>
<body>
<p>This site is under construction. Please come
back later. Sorry!</p>
</body>
</html>
```

Figure 4. The HTTP response overhead. Source: (Kozierok 2005)

Counting the characters shows that the overhead for this response is about 200 characters/bytes. This means that the server will need to generate that much data for each message and send it to the client which will require some system resources to do.

4.3 HTTP Long polling

HTTP Long polling is an attempt to minimize latency and use of processing and network resources compared to XHR polling by only responding to the HTTP request once an event, status, or timeout occurs. On receiving the response, the client would send a new request either immediately or after a delay within an acceptable latency period. The main difference from XHR-polling is that when using HTTP Long polling there is no need for the client to keep sending frequent request to find out if the server has any new updates. A single long poll can be split into four parts (Loreto et al. 2011):

1. The client makes an initial request and then waits for a response.
2. The server defers its response until an update is available or until a status or timeout has occurred.
3. When an update is available, the server sends a complete response to the client.
4. The client typically sends a new long poll request, either immediately upon receiving a response or after a pause to allow an acceptable latency period.

Since a timeout will force a new request cycle without any useful information being exchanged, it is desired to keep the timeout value as high as possible to minimize the number of timeouts. A timeout can occur either by the client browser terminating the connection because no response arrives from the server, however it is more common for the web server or any intermediates (like a proxy) to initiate the timeout by sending a HTTP 408 or 504 response because they usually have a lower default timeout setting. A long poll will stay alive generally up to 120 seconds but 30 seconds is a safer value to avoid web servers or intermediates to timeout the long poll (Loreto et al. 2011).

This means that at best there will be one complete HTTP request response cycle per server update. If the server message frequency is less than the environment timeout settings long polling will become slightly less efficient because of the "unnecessary" timeouts and resending of requests.

This technology is in many ways like XHR polling. Instead of the client sending for example ten requests where only the last one contains useful information for the client, the client sends one request and the server does not respond to it before it has anything new for the client. This leads to fewer messages being sent between server and client compared to XHR polling depending on how often the server has new data for the client and what the polling frequencies are. If the XHR polling frequency is the same as the server message frequency, XHR polling and Long polling should in theory have approximately the same performance impact.

4.4 Server sent events (SSE)

4.4.1 Establishing the connection

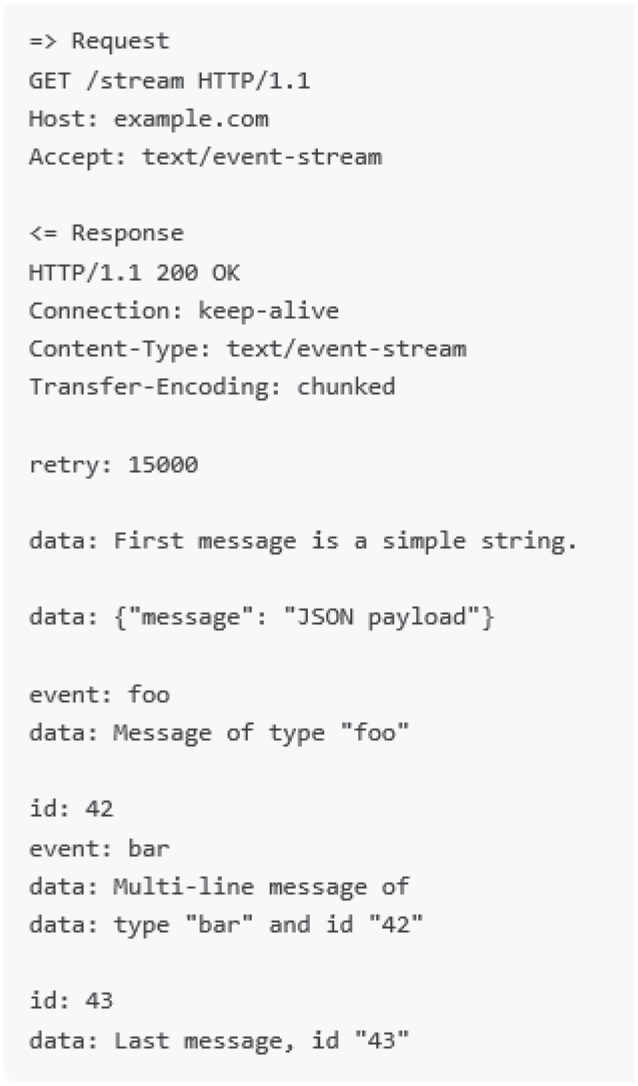
The communication between server and client is initiated by the client setting up a TCP connection and sending a HTTP request to the server. After parsing the request, the server sends a HTTP response to the client. The HTTP response will have the *"Content-Type"* header set to *"text/event-stream"* to inform the client that the server sent events technology will be used. All content will then be done over this connection (Grigorik 2013).

4.4.2 Pushing data from server to client

In SSE, the response is not considered complete after the first payload has been received. After receiving the headers, the client expects a chunked response where every chunk represents a message from the server. The chunks are sent to the client as they are generated on the server, not all together at once, and separated by an empty line (Grigorik 2013).

A SSE chunk is structured into fields. There are four field types defined in the SSE standard; "data", "id", "event" and "retry". The data field contains the information the server is sending to the client. The other fields are optional. The id field is used by the client to let the server know which was the last message received by the client in case the connection was dropped. The event field allows the client to set up listener functions to listen to specific types of events. Lastly, the retry field is used for the server to tell the client how long to wait before trying to reestablish the connection in case it gets dropped (Grigorik 2013).

The following figure (Figure 5) shows what a server sent event stream can look like. Note that the request and response lines are added for explanatory purposes only and is not included in the actual data stream.



```
=> Request
GET /stream HTTP/1.1
Host: example.com
Accept: text/event-stream

<= Response
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: text/event-stream
Transfer-Encoding: chunked

retry: 15000

data: First message is a simple string.

data: {"message": "JSON payload"}

event: foo
data: Message of type "foo"

id: 42
event: bar
data: Multi-line message of
data: type "bar" and id "42"

id: 43
data: Last message, id "43"
```

Figure 5. The SSE protocol. Source: (Grigorik 2013)

The first four lines of the response in this case are the response HTTP headers telling the client that the response will be using the server side events protocol. The retry line is optional. It tells the client how long to wait before trying to reestablish the connection if it is dropped. Web browsers generally defaults to 2-3 seconds. This is followed by five chunks/messages with different fields (Grigorik 2013).

The white space character after the colon between the field and value is optional. It is not necessary for the server to send this to the client. A line starting with a colon character is defined as a comment and will be ignored by the client (WHATWG n.d). This means that SSE needs a set of HTTP request and response headers when initializing the connection. After that the overhead for the data sent is at minimum 7 bytes per message (data field plus two new line characters). It can grow indefinitely depending on how many lines of data is sent. Making use of more fields for the messages such as event or id will also increase the overhead by 4-11 bytes if counting fiend names and newlines while omitting unnecessary whitespaces. If using all available overhead fields per message (excluding retry since it is optional and only should be sent once) the total overhead per message would be 18 bytes if not using more than one data field line.

In case of intermediates, browsers or servers timing out the connection because of inactivity in case the message frequency from the server is low, it is possible to send a colon plus two new line characters (three bytes) as a keep alive to prevent timeouts.

4.5 Performance factor identification

This section contains an analysis of the factors that were found when summarizing the technology specifications. All information about the technologies in this section are retrieved from the specification summary done above. In the end of this section there is a table where the identified performance factors are summarized and linked to their respective technologies.

4.5.1 TCP Connections

All technologies are using TCP as transport protocol. XHR polling and Long polling may use the same TCP connection using the HTTP persistent connection feature if new requests are being sent before the persistent connection times out, otherwise a new TCP connection is used for the next message. The persistent connection timeout is most likely initiated by the web server that usually keeps it open for about 5-15 seconds compared to a web browser that waits about 4 times as long or more before timing out the connection.

Server sent events and Websockets will set up one TCP connection which will stay open and all communication will be done over this connection.

The exact resource usage of a TCP connection is hard to determine. The default settings for the TCP write buffer on the host computer used for the performance tests in this study defaults to 16384 bytes and the read buffer to 87380 bytes. The min value of the buffers is both set to 4096 bytes and the max values are set to 126976 bytes. The buffer size for a TCP connection will change dynamically between these limits depending on how much data is handled by the connection and available server memory.

The overhead data used by TCP is not considered as a performance factor in this study. All technologies compared will be affected by TCP overhead and therefore it does not have a big effect on the comparison done in this study. XHR polling and long polling are likely affected slightly more in percentage than Websockets and SSE because the overhead data used for those technologies are larger, however the HTTP header sizes cannot be precisely determined and varies between different implementations. Calculating the difference in TCP headers would depend on the http header sizes but also connection stability, payload size etc.

Although the exact resource usage is hard to determine it is clear that a TCP connection will use system resources. Opening and closing TCP many connections should therefore be more performance heavy than only using one. TCP connections are used by all technologies in this study and every connection will use server resources. This means that the amount of TCP connections per client needed will likely be a performance factor.

4.5.2 HTTP Request

All technologies are client initiated by setting up a TCP connection to the server and sending a HTTP request. XHR Polling and Long polling need to send at least one request for every message from the server while Server sent events and Websockets only require one request when establishing the connection.

The HTTP request headers are expected to be around 200 bytes in a good case and because of that must use server bandwidth and be stored somewhere in the memory for processing.

HTTP request processing
All HTTP requests sent to the server needs to be parsed and go through steps like Authorization, Name translation, Path checking before forwarding the request to the target web application on the server. This is taking up system resources.

4.5.3 Generating overhead data for server to client messages

HTTP Response
The HTTP response headers are expected to be at least around 200 bytes in size and some system resources and bandwidth will be needed to generate and send this for each message.

Other overhead
Websockets need between 2 and 10 bytes depending on how big the payload is and SSE uses between 7 and 18 bytes depending on how many fields are used. Some system resources and bandwidth will be needed to generate and send this overhead data.

4.5.4 Request - response architecture VS Server push architecture

Both XHR Polling and Long polling requires a request from the client to the server for each message the server wants to send to the client while Websockets and SSE allows the server to push information to the client at any time. Being able to push at any time reduces the servers need for incoming network bandwidth, cpu power and memory usage since it does not need to handle any inbound requests and is therefore more performance efficient.

4.5.5 Identified performance factors per technology

	XHR polling	**Long polling**	**Server sent events**	**Websockets**
TCP connection	One per client if persistent connections used and the polling interval is lower than the connection timeout settings	One per client if persistent connections are enabled.	One per client	One per client
Parsing and processing HTTP requests **(Request - response architecture VS Server push architecture)**	One per poll. Poll frequencies are set on the client and does not equal the server message frequency. This can result in many polls for one message (A request from client to server is needed for every message)	One per poll. The poll frequency is the same as the server message frequency which means one poll per message unless the server message frequency is low enough to timeout the connection in which case a new poll needs to be sent. (A request from client to server is needed for every message)	Only needed once when the client is connecting. (Server can send messages to the client without requiring the client to request it for every message)	Only needed once when the client is connecting. (Server can send messages to the client without requiring the client to request it for every message)
Generating overhead data for server to client messages	One HTTP header set per poll	One HTTP header set per poll	Only needed once when the client is connecting	Only needed once when the client is connecting
Overhead data	Http headers for both request and response per poll. *~200 bytes inbound and outbound* in a good case	Http headers for both request and response per poll. *~200 bytes inbound and outbound* in a good case	7-18 bytes outbound	2-10 bytes outbound depending on payload size

5 PERFORMANCE TESTS

This section is divided into four parts. The first part is describing what performance aspects were measured during the tests. The second part is describing the setup used for the tests. The third part describes what tests are done and the final part contains the hypothesis, result and analysis for each test.

5.1 Performance metrics used

Research articles on which metrics to measure when performance testing this kind of communication technologies were not found. Other articles such as the ones written by Meier et al. (2007) and Majumdar et al. (2007) were found which mentions how to test web services/applications. The key metrics used to measure performance mentioned in those articles are the response time, throughput (such as messages per second or bytes per second), resource utilization (CPU, memory, disk input and output, network input and output), maximum user load and business-related metrics.

The mentioned articles are describing how to performance test web services / applications. This means that the output from those tests should include performance results of the communication technology used combined with the web application logic executed on the server. Only the communication part is of interest in this study. Because the server software is implemented specifically for this study it was possible to remove the impact of the web application logic. This was done by making the application logic as simple and small as possible and equal for all technology server implementations. This leaves only the performance impact of the communication technology. Using the same performance metrics as previously mentioned should therefore allow for measuring the performance of the studied technologies.

Not all metrics mentioned by those articles were found to be useful in this study. The response time cannot be compared between the technologies since only half of them includes sending a request and waiting for a response. The other half keeps the communication one way which means there are no response times to measure. Business related metrics are also not considered in this study since the focus of this study is on the technical performance of the technologies. Disk input and output were also not measured because writing to the disk is not part of what the studied technologies are doing. The maximum possible user load was not measured as not enough hardware resources were available to the authors during this study to enable testing this. The tests were instead using a static number of clients.

As concluded in the performance factor analysis in section (4.5) it is possible that the amount of TCP connections used can be a performance factor, which is why that was also decided to be measured.

To summarize, the metrics used in this study are:

- CPU utilization
- Memory utilization
- Inbound and outbound network traffic
- Sent messages per second
- Amount of TCP connections used.

5.2 The test setup

5.2.1 The server host computer

The experiments were conducted on an Ubuntu (version 16.04.2 desktop amd64) environment inside a Virtual Machine with Oracles Virtual Box (version 5.1.18 r 114002 (Qt5.6.2), *https://www.virtualbox.org/*). The Virtual Machine was allocated with 8192 MB of memory and a VirtualBox Disk Image (VDI) as storage with a fixed size of 100 GB.

The computer hosting the Virtual Machine had access to 16 GB of memory, an Intel i5 4670K CPU 3.40 GHz processor, a Samsung SSD 840 EVO 250 GB (232 GB) as primary storage and a TOSHIBA DT01ACA200 (1863GB) as secondary hard drive, running Windows 10 x64 installed on the SSD.

Docker (version 1.12.16, build 78d1802, https://www.docker.com/) was used to run instances of the servers and the performance measuring tools. Docker is used to run and manage applications in isolated containers. Unlike a Virtual Machine, containers don't need a full operative system for it to run, only libraries and settings.

5.2.2 Implementation of the servers for each technology

Each server resides in their own Node.js based Docker container. The goal for the implementation of each server is to keep it as simple as possible. All servers send the same payload, which is the actual intended message of the transmitted data. The payload consists of a JSON-string containing an incremental ID and a static short message. The payload is generated with the help of a recursive timer. The frequency of the timer will vary depending on the test that is to be performed. When the timer has reached the set frequency, a payload is generated and sent to the clients. When a new payload has been generated, only the ID is updated while the short message is always the same.

The client web page is not served from the servers hosting each technology. A HTTP security feature is in place for web servers and browsers that prevents requests from web pages not served by the webserver. To make the tests work, all servers include the *"Access-Control-Allow-Origin"* HTTP header to disable this security feature for the tests to work.

Implementing the XHR server
The XHR-server uses the node *http* module to listen to incoming requests. When the server receives a request from a client, it expects the request to contain an ID, which should represent the ID of the last payload that the client previously has received and should be located in the url query of the request. If the ID contained in the request is lesser than the server's current payload ID, the server will send the current payload back to the client, otherwise it will send an empty JSON-string. This check is important to avoid sending the same information multiple times to the client. There is no need to spend resources on sending the whole payload if the client is already up to date.

Implementing the Websocket server
For the Websocket server the node *ws* module is used for listening to incoming requests. The *ws* module will handle all the requests. When a new payload has been generated, the server sends it to all the clients.

Implementing the Server Sent Events server
The node *http* module was used to set up the server and to listen for requests. In order to be able to continuously send data to the client and make sure that the request is not intercepted and cached by a proxy, the following headers needed to be set:

- Content-Type: text/event-stream
- Cache-Control: no-cache

All response objects received from request callback is saved to a list. When the connection is closed, the response object is removed from the list. When a new payload is generated, it's streamed to all the response objects.

Implementing the Long Polling server
The node *http* module was used to set up the server and to listen for requests. All response objects received from request callback is saved to a list. When a new payload is generated, it's sent to all the response objects and the connection is closed. When the payload has been sent to everyone, the list of response objects is cleared.

5.2.3 Implementation of the clients

The client consists of a single HTML-page driven by JavaScript. The client is running one instance each of XHR-polling, Long polling, Websockets and Server Sent Events at the same time. The instances are connecting to their respective server. All instances output the status of their connection for validation purposes. The clients were not run on the same computer as the servers.

XHR polling
The XHR client polls the XHR server with a constant frequency. The frequency can be changed as desired.

Websockets
The Websocket client initializes a connection to the server and starts listening for messages.

Server Sent Events
The EventSource JavaScript API was used to establish a connection with the SSE-server.

Long Polling
On startup, the long polling client sent a http request to the long polling server. When a response is received or a timeout occurs a new request is immediately sent to the server.

Mozilla Firefox
A tool was needed to connect the client to the servers, keep the connection alive and execute JavaScript. Ideally, the clients would run on multiple machines. No such tool was found except for web-browsers, so it was decided to run the clients in multiple web-browser tabs. Firefox was specifically used because it exposed settings which were necessary to run multiple instances of the client on the same machine. The settings altered was:

- network.http.max-connections
    - default: 256
- network.http.max-persistent-connections-per-server
    - default: 6
- dom.min_timeout_value
    - default 1000

The first two settings (network.http.max-connections and network.http.max-persistent-connections-per-server) was necessary for all the tabs to be able to connect to the servers. Each client uses four different connections, so if 100 clients is to be running, the previously mentioned settings would be needed to be set to at least 400.

Since the XHR-client uses a recursive timer to know when to poll the server, the third setting (dom.min_timeout_value) was needed. In some tests, the XHR-client was required to poll at a frequency of 500 ms. But the default minimum timeout value is set to 1000, which means it needs to be set to a maximum of 500.

5.2.4 Measurement tools

A tool developed by Google called cAdvisor (https://github.com/google/cadvisor) was used to measure the server's CPU, memory and network usage. cAdvisor collects, aggregates, processes and exports information about running Docker containers. The data collected by cAdvisor is stored in an InfluxDB database. (https://github.com/influxdata/influxdb). Grafana (https://grafana.com/) was used to inspect the data and create graphs to visually present the data.

Grafana
The CPU usage, memory usage and sent and received network traffic is visually presented by graphs drawn using Grafana. Each graph includes usage data for all servers, XHR, Websocket, Server Sent Events and Long Polling.

Memory usage
To measure memory usage, within Grafana metrics, *memory_usage* is selected. cAdvisor is collecting memory usage in bytes but Docker stats displays memory usage as MiB (Mebibyte). The memory usage is therefore divided by 1024^2 to match Docker stats.

CPU usage
To measure CPU usage, within Grafana metrics, *cpu_usage_total* is selected. cAdvisor is collecting the amount of CPU usage in nanoseconds since start up but Docker stats is displaying the current CPU usage as a percentage of the total CPU capacity per second. The total CPU usage is transformed by the derivative of one second to get the rate of change of the accumulated CPU usage. The CPU usage is then divided by a billion to get the usage in seconds and multiplied by 100 to get an integer percentage value. This value is not normalized by the number of cores. By dividing the value by the amount of CPU cores we get a value between 0 and 100 percent.

Network data
To measure network data, within Grafana metrics, *rx_bytes* was selected to show received bytes and *tx_bytes* was selected for transmitted bytes. The measurement tool used is collecting the total amount of sent or received bytes from the point where the server is started. This data is recalculated into rate of change per second in bytes.

cAdvisor versions
To be able to match the data cAdvisor collects with Docker stats, two different cAdvisor versions are needed. Older versions of cAdvisor does not collect network data that matches Docker stats and newer version does not collect memory and CPU usage that matches Docker stats. For measuring memory and CPU usage in this study, version 0.22.0 and later of cAdvisor was used. For measuring network traffic, version 0.23.0 and older of cAdvisor was used.

Validating the measurements
Two other tools were used to validate that the information collected by cAdvisor was correct. By comparing that the collected data matches with what is displayed when running the stats command in the Docker system.

Another tool that was used to validate the results is Datadog (https://www.datadoghq.com) Datadog had a minimum sampling of one data point every 15 seconds which is why this tool was not used as a main measurement tool and instead only to validate the results collected by cAdvisor.

5.3 Payload

The payload sent with each message from the server to the client was the following JSON string: {"id": 0, "msg": "Message"}

The id value was generated with each new message by incrementing the previous value by one. This means that after 9, 99, 9999 and so on messages the payload would increase by one byte. The resulting payload is about 30 bytes in total.

5.4 Overhead sizes

5.4.1 Http overhead

Using the built-in Firefox network supervising tool on the client the response and request HTTP header sizes could be measured.

	Request	Response
XHR	281 bytes	191 bytes
Long polling	273 bytes	191 bytes

Table showing the measured HTTP request and response header sizes.

5.4.2 Server sent events overhead

8 bytes overhead was sent per SSE message during the performance tests in this study.

5.4.3 Websocket overhead

The Websocket overhead size should be 2 bytes because messages are sent from server to client and the payload is less than 126 bytes. This is based on the technology analysis in section 4.5.

5.5 Idle test

To be able to see the difference between the servers being idle and serving the clients a performance test was made on the servers with no clients connected.

The graphs in the figure below illustrates the server resources of each server when idling. All servers are newly created with requests sent to them.

The most notable about the server's idle state is the difference in memory usage between the Websocket server and the rest. A possible explanation for this is that the Websocket server is running the server on the Node.js *ws*-module while the other servers are running on the Node.js *http*-module.

There are no notable differences between the server's other than the Websocket memory usage.

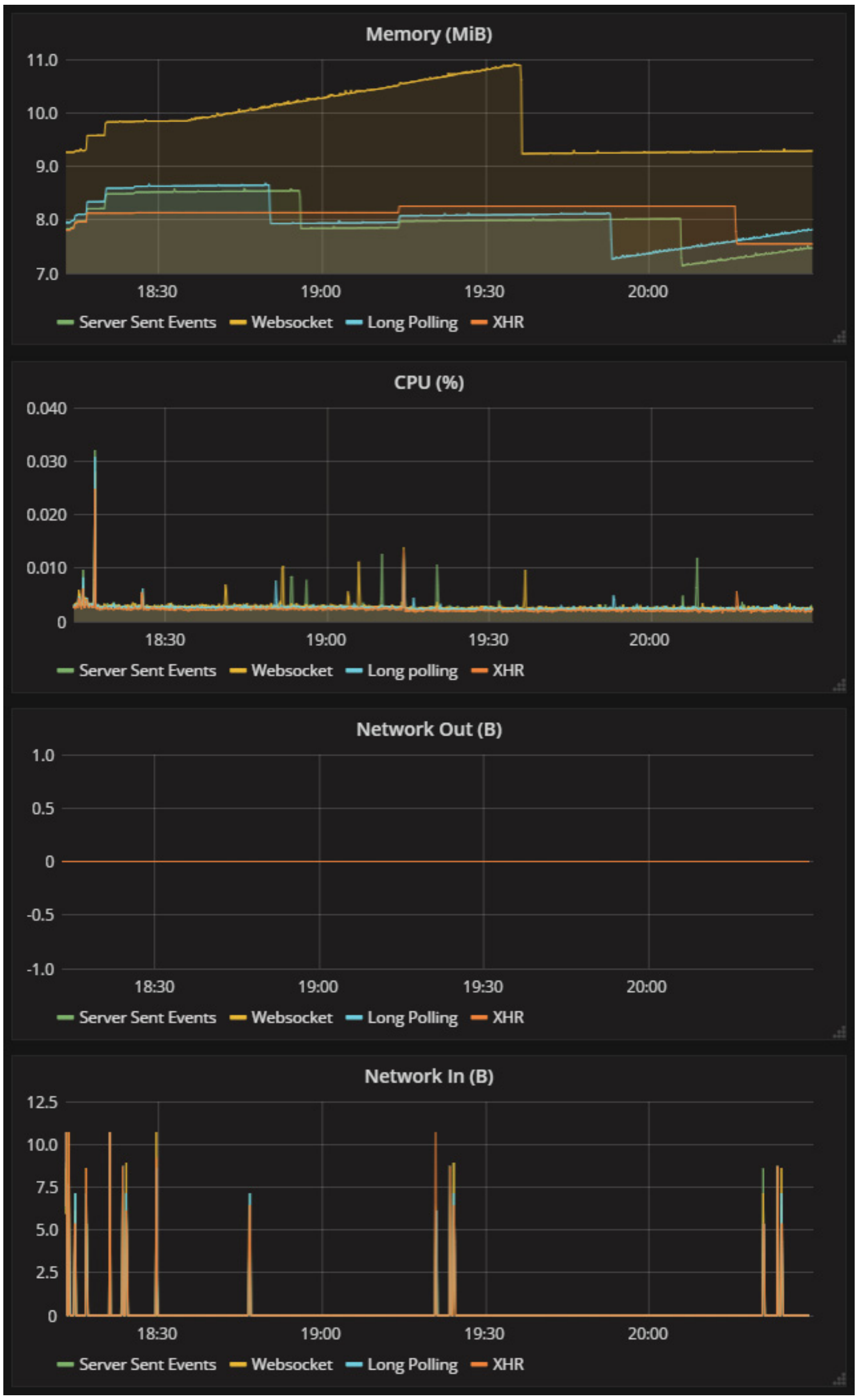


Figure 6. Idle test result

5.6 Test case design

Three performance tests were made in this study to compare the performance of the technologies. Each test was done twice, with and without garbage collection requests. Each test was done over a long time (between 40-60 minutes) to enable more accurate average calculation on the results. 100 simultaneous clients were used in this study. The goal was to use as many clients as possible and it got limited to 100 due to hardware limitations.

5.6.1 Description of performance test 1

- 100 simultaneous clients
- 500 ms server message interval
- 500 ms XHR poll interval

This test was done to compare the technologies under similar conditions. XHR-polling was polling at the same rate as the other technologies were receiving their updates. This was done to have a fair comparison between the technologies where all of them have the same task: send updates to the client from the server every 500 ms.

The 500 ms interval was chosen to keep the update interval under 1 second. This was done to simulate a system that would according to Nielsen (2010) be perceived as very responsive by a user.

5.6.2 Description of performance test 2

- 100 simultaneous clients
- 2000 ms server message interval
- 500 ms XHR poll interval

Frequent server updates were desired in order to stress the servers to see clear differences between the results but still with less frequency than the last test. 2000 ms was chosen as the interval value to balance between fast updates and pauses between messages. The XHR poll frequency was set to twice per second to make sure updates were delivered within one second after the message was generated on the server. This was done to simulate a system that would according to Nielsen (2010) be perceived as very responsive by a user.

5.6.3 Description of performance test 3

- 100 simultaneous clients
- 2000 ms server message interval
- 2000 ms XHR poll interval

Because of the unexpected results regarding XHR Polling from the previous tests it was decided to do another test. The idea was that a lower XHR polling frequency than what was set in the previous tests could perhaps reduce the stress on the clients and avoid triggering the unexpected result that were seen (possibly because of malfunctioning connection optimization discussed in the analysis of test 2 (section 5.7.3)). The point of this test is to see if the results for XHR are predictable when the XHR polling technology is not influenced by the potentially misbehaving clients.

5.7 Performance test 1

- 100 simultaneous clients
- 500 ms server message interval
- 500 ms XHR poll interval

5.7.1 Hypothesis of performance test 1

Memory usage of performance test 1

Because of Long polling being essentially the same technology as XHR polling except for Long polling waiting with responding until it has something new for the client, it was expected that this test would result in those two technologies using the same amount of memory since the server message interval and XHR-poll interval was set to the same value.

Websockets and SSE are expected to use less memory than Long polling and XHR polling because they do not need to parse requests from the clients.

CPU usage of performance test 1

Because of the server message interval and XHR-poll interval is set to the same value both XHR polling and Long polling is expected to use the same amount of CPU.

Server sent events and Websockets were expected to have about the same CPU usage as each other since no requests needs to be handled and the overhead data difference is just a few bytes. The CPU usage was expected to be very low for these technologies as the only work done is to send about 30 bytes of data two times every second to 100 clients.

XHR and Long polling are expected to use more CPU power than Server sent events and Websockets. This is because SSE and Websockets does not have to process any requests from the clients which XHR and Long polling must do.

Network traffic of performance test 1

Sent network traffic

The network traffic sent for XHR polling and Long polling were expected to be approximately equal due to the server message rate being the same as the XHR polling rate.

The network data for SSE was expected to be slightly higher than Websockets due to 6 bytes more overhead per message.

XHR polling and Long polling were expected to use more network than SSE and Websockets because of 191 bytes overhead compared to the 8 bytes of SSE overhead and 2 bytes of Websocket overhead.

Received network traffic

The network data received for XHR polling and Long polling is estimated to be higher than the network data sent but still equal because of the request sizes being is larger than the response headers used in the tests.

SSE and Websockets were expected to be low as the only expected inbound data from the client is from TCP overhead acknowledgements that is sent back from the client when sending messages to the client.

TCP connections of performance test 1

It was estimated that 100 TCP connections would be in use for each technology. This was based on the specifications stating that Websockets and SSE technologies keeps one connection open per client. The message and polling frequencies set for this test should allow XHR and Long polling clients to make use of the HTTP persistent connection feature and therefore also only use one connection open per client.

Message frequency of performance test 1

The message interval for all servers and the XHR client poll interval was set to 500 ms. Because of this, a message frequency of 2 messages per second is expected to be sent per client. 100 clients should therefore result in 200 messages sent per second.

5.7.2 Result for performance test 1

Memory usage without garbage collecting requests for performance test 1

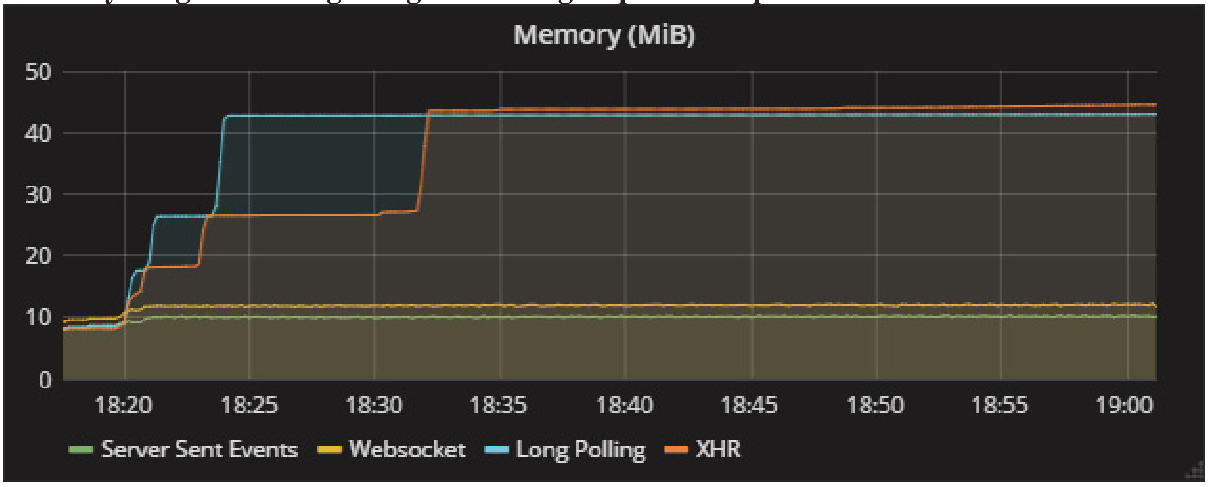


Figure 7. Performance test 1 memory usage result

Values

- XHR: Memory increased in four steps where it stayed stable for a while. The steps were averaging at about 13.7, 18.2, 26.6 and 43.9 MiB. (Idle levels at 8.14 MiB)
- Long Polling: Memory increased in four steps. The steps were averaging at about 9, 18.2, 26.6 and 43.9 MiB. (Idle levels at 8.14 MiB)
- Server Sent Events: Stable average of 10.11 MiB (Idle levels at 8.33 MiB)
- Websocket: Stable average of 11.90 MiB (Idle levels at 9.83 MiB)

Memory usage with garbage collecting requests for performance test 1

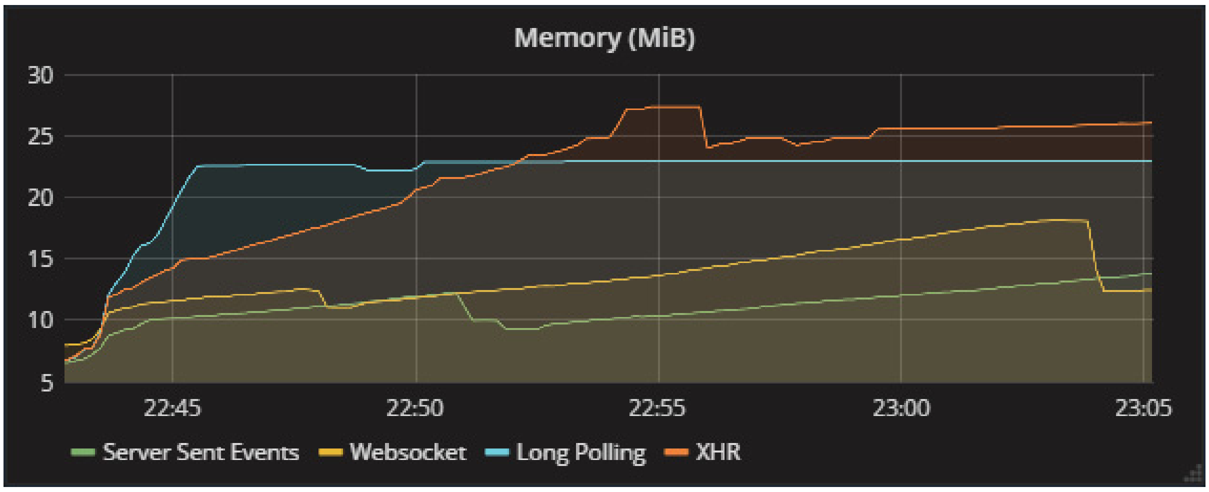


Figure 8. Performance test 1 memory usage with garbage collection requests result

Values

- XHR: Fairly constantly increasing up to 27 MiB, drops down to 24 MiB and increases again to 26 MiB where test was terminated
- Long Polling: Rapidly increasing up to 23 MiB where it stabilizes.
- Server Sent Events: Increasing to 12 MiB, drops to 10 MiB, shortly stabilized, drops to 9 MiB, increasing until the test was terminated
- Websocket: Increasing to 12.5 MiB, drops to 11 MiB, increases to 18 MiB, drops to 12.5 MiB, stabilized until the test was terminated

CPU usage for performance test 1

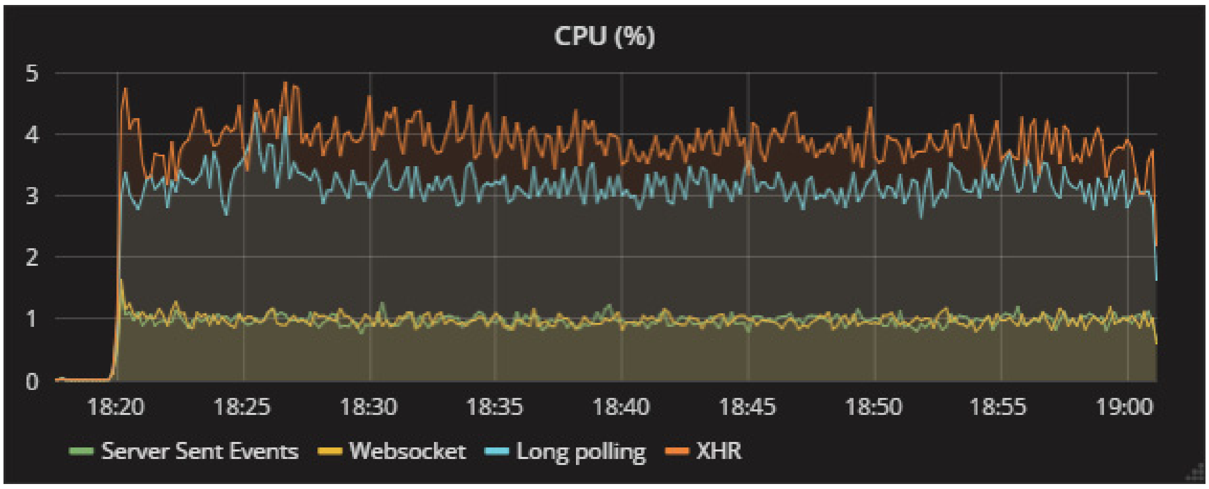


Figure 9. Performance test 1 CPU usage result

Values

Calculating the average of the CPU usage results in the following numbers:

- XHR: 4.23%
- Long Polling: 3.43%
- Server Sent Events: 0.99%
- Websocket: 0.91%

Network traffic for performance test 1

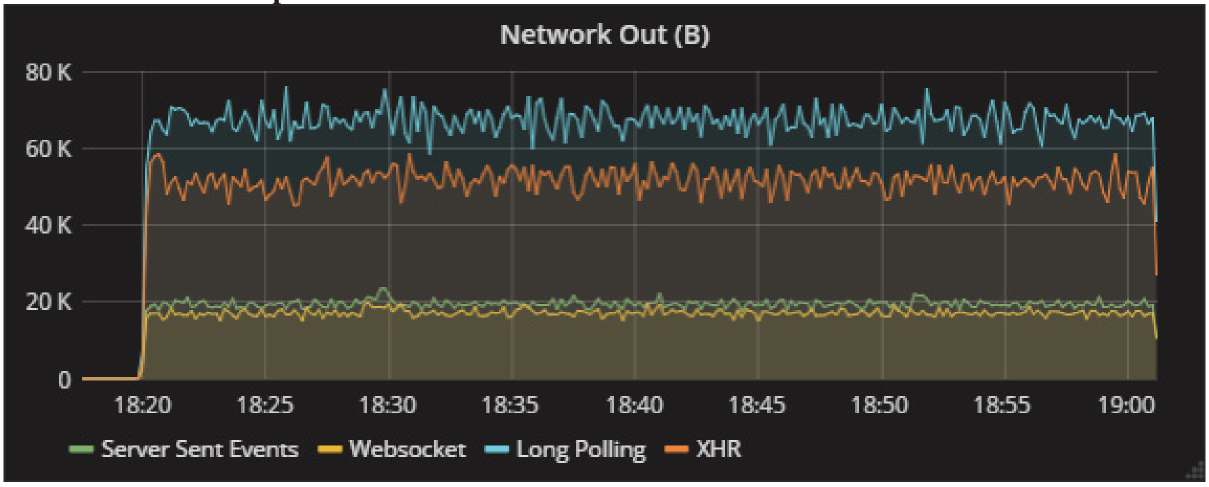


Figure 10. Performance test 1 sent network traffic result

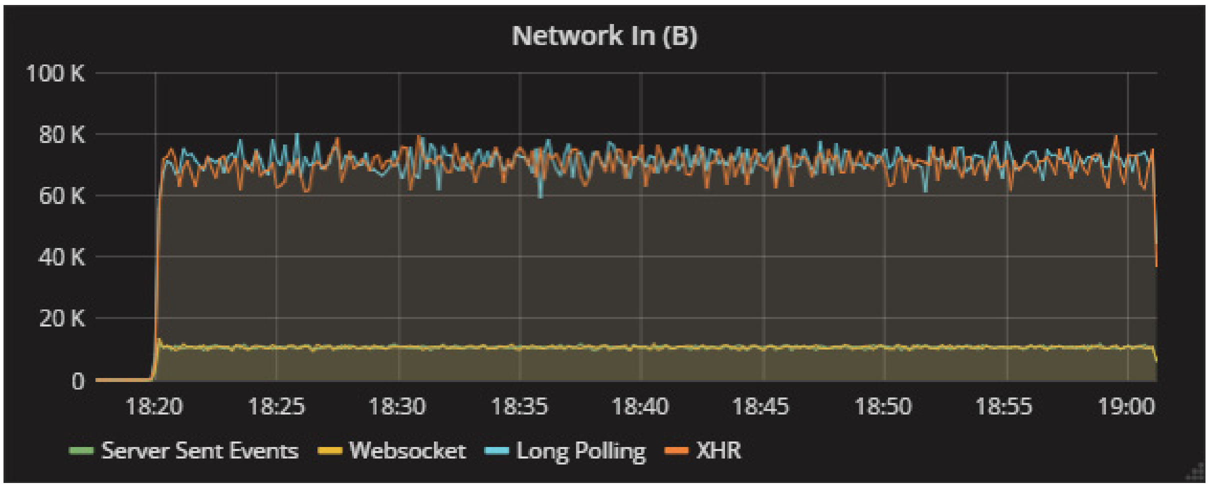


Figure 11. Performance test 1 received network traffic result

Values

- Sent
    - XHR: 51.7 kilobytes per second
    - Long Polling: 67.3 kilobytes per second
    - Server Sent Events: 19.5 kilobytes per second
    - Websocket: 17.1 kilobytes per second
- Received
    - XHR: 70.1: kilobytes per second
    - Long Polling: 71.3 kilobytes per second
    - Server Sent Events: 10.7 kilobytes per second
    - Websocket: 10.7 kilobytes per second

TCP Connections for performance test 1

- XHR: TCP connections increased in several steps over time. 22 connections were used at the start of the test and 289 simultaneous open connections were used at the end of the test
- Long Polling: 100 simultaneous open connections
- Server Sent Events: 100 simultaneous open connections
- Websocket: 100 simultaneous open connections

Message frequency for performance test 1

- XHR: 160 messages sent per second
- Long Polling: 200 messages sent per second
- Server Sent Events: 200 messages sent per second
- Websocket: 200 messages sent per second

5.7.3 Analysis for performance test 1

Memory usage for performance test 1

SSE was using approximately the same amount of memory as Websockets which was expected. The difference in memory usage between SSE and Websockets remained the same as when the servers were idling. This suggests that the technologies require the same amount of memory.

XHR polling was measured to use a lot more memory than Long polling when not requesting the garbage collector to run. The test where the garbage collector was requested to be called resulted in a significantly different graph. XHR polling first seemed to use less memory than Long polling but after increasing for a while it stabilized slightly above Long polling. The difference seen in these two tests makes a detailed analysis of the difference between these two technologies difficult. XHR polling using more memory than Long polling could be explained by memory being used by the extra TCP connections noted in the TCP connection analysis in this section.

What can be concluded is that XHR and Long polling is using more memory than SSE and Websockets which was anticipated in the hypothesis.

CPU usage for performance test 1

SSE and Websockets uses about the same amount of CPU power. This was expected in the hypothesis due to similar work being done. XHR and Long polling were using more CPU power than SSE and Websockets as expected.

The message frequency for XHR polling was measured to be 20% lower than for Long polling however this is not represented in the CPU usage graph. A possible reason for that it could not be seen is that the XHR server could be using CPU power to open and close TCP connections which makes up for the 20% message frequency loss. This is further discussed in the analysis of the message frequency in this secction. Because of this uncertainty it is not possible to analyze the CPU difference between XHR polling and Long polling.

Network traffic for performance test 1

Sent

The XHR polling sent about 24% less network traffic than Long polling although they were expected to be equal. This difference could possibly be explained by the lower message frequency mentioned in the analysis of the message frequency in this section.

SSE was sending slightly more traffic than Websockets as expected. Likely due to the overhead data per message being slightly larger for SSE than for Websockets in this test as mentioned in the hypothesis.

XHR and Long polling technologies require the server to send more network traffic than SSE and Websockets which is also expected due to the several times larger overhead size and TCP overhead sent when receiving requests from the clients.

Received
The SSE and Websocket traffic is equal as expected. The clients are not sending any data to the servers and the small amount of network traffic seen here is likely due to TCP overhead being sent back from the clients as they receive the messages.

The message frequency for XHR polling was measured to be lower than what was expected in the hypothesis. The received traffic for XHR polling should because of this also be lower due to less requests being sent to the server from the client. This is not seen in the results. With the information, available it is difficult to do an exact analysis to find out the cause for this. One explanation for this difference could be that extra inbound network traffic is being used to open and close TCP connections which is not done by Long polling. This is further discussed in the analysis of the TCP connections in this section. This network traffic used for opening and closing TCP connections could possibly make up for the lower message frequency and explain the results seen in this test.

XHR polling and Long polling clearly use more network traffic than SSE and Websockets as anticipated in the hypothesis.

TCP connections for performance test 1
SSE, Websockets and Long polling seemed to behave as expected by keeping 100 simultaneously open TCP connections.

The amount of TCP connections used by XHR polling was not stable and used sometimes more and sometimes less connections than the expected amount of 100. One possible explanation for the unexpected XHR polling connection usage could be how Firefox, which the clients are executed by, optimizes the use of connections between many tabs connected to the same host. It is possible that all connections are shared and accessible between all tabs because of this potential optimization. All tabs do not connect to the server at the exact same time. This means the tabs are likely sending their requests spread out over the 500 ms window instead of all sending at the same time. This would allow Firefox to keep less TCP connections open than the number of clients. An advanced configuration was done in Firefox for these tests that heavily increased the allowed limit of simultaneous connections to the same host. This was done to enable more clients to run at the same time in Firefox to stress the servers more and by that get clear results. Changing this setting could possibly lead to the TCP connection optimization malfunctioning, especially since the clients are sending requests at a high frequency (2 per second), which could be the reason why more than 100 simultaneous TCP connections are measured. Establishing and closing more TCP connections than expected could also increase the network traffic used which could explain the network traffic differences noted in the analysis of the TCP connections in this section. It could also explain XHR polling’s extra memory usage that was noticed in the memory analysis in this section.

Message frequency for performance test 1
The message frequency for Long polling, Server sent events and Websockets matched the hypothesis of 200 messages sent per second.

XHR polling did not match the hypothesis and resulted in a 20% lower frequency than expected. A possible explanation for the lower XHR polling frequency is that the XHR clients were not able to keep polling the server at a constant rate of 2 requests per second. The client waiting longer in between requests due to reasons like browser optimization for inactive tabs or the CPU being busy with something else or inaccurate timers et cetera could explain the lower message frequency.

This raises the question why Long polling which also must send requests does not show the same results. The Long polling client sends requests as soon as a response is received from the server. Because of this it should not matter if the Long polling client is running slightly slower than expected if the request is sent before the next batch of messages are sent from the server. In other words, initiating the client requests by sending responses from the server will compensate for a slow running client if the client is fast enough to send within the server's message interval window. A 20% lower frequency, as seen in XHR polling, means the clients were polling on average 100 ms (500 ms * 1.2 - 500 ms) slower than expected. Assuming Long polling would suffer from the same delay it would get compensated by the server because the delay is still shorter than the 500ms interval between server messages.

If the XHR polling client sends requests slower it does not get compensated for which is probably why this difference is seen only in XHR polling and not Long polling despite their similarities.

5.8 Performance test 2

- 100 simultaneous clients
- 2000 ms server message interval
- 500 ms XHR poll interval

5.8.1 Hypothesis for performance test 2

Memory usage for performance test 2

XHR polling memory usage was expected to give the same results as in test 1 (section 5.7.2) because the poll rate was set to the same value.

Long polling memory usage was expected to be lower than in test 1 (section 5.7.2). Four times longer server message interval should result in four times less frequent requests for the Long polling server to handle. It was still expected for Long polling to use more memory than SSE and Websockets because it needs to process requests from the clients.

SSE and Websockets were expected to have the same result as in test 1 (section 5.7.2). Sending messages with a higher frequency is not anticipated to require more memory because the memory is already allocated from the first message and should be able to be re used for further messages.

CPU usage for performance test 2

The XHR polling server’s CPU usage is expected to be the highest of the technologies with about the same results as in test 1 (section 5.7.2).

Long polling was expected use the second highest amount of CPU of the technologies with results around four times less than what was seen in test 1 (section 5.7.2).

SSE and Websockets are expected to use around four times less CPU power than what was seen in test one (SSE at 0.99% and Websocket at 0.91%) because of the four times lesser message frequency. Both technologies are expected to use the same amount of CPU as each other.

Network traffic for performance test 2

Sent

The network traffic sent for XHR polling was expected to be larger than for Long polling due to the XHR polling rate being four times more frequent than the Long polling.

Long polling was expected to send more network traffic than SSE and Websockets because of 191 bytes overhead compared to the 8 bytes of SSE overhead and 2 bytes of Websocket overhead. The network data for SSE was expected to be slightly higher than Websockets due to the 6 bytes more overhead per message.

Received

The received network traffic for XHR polling is expected to be about four times larger than for Long polling due to the XHR polling frequency being four times higher than the server message frequency.

Received network traffic for SSE and Websockets were expected to be low and equal as the inbound data from the client is expected to only consist of TCP overhead data.

TCP connections for performance test 2

The TCP connections for XHR polling was estimated to stay the same as was measured in the previous test due to the XHR polling rate being the same.

The other technologies are also expected to be the same as in the previous test. Less frequent messages should result in less frequent communication over the TCP connections while the amount of TCP connections stays the same.

Message frequency for performance test 2

Because the XHR polling interval is set to the same value as in test 1 (section 5.7), the same result is expected for XHR polling as what was measured in that test.

The server message interval was set to 2000 ms which should result in all servers except XHR polling to send 0.5 messages per second. With 100 clients, this should result in 50 messages sent per second (0.5 * 100).

5.8.2 Result for performance test 2

Memory usage without manually calling garbage collection requests

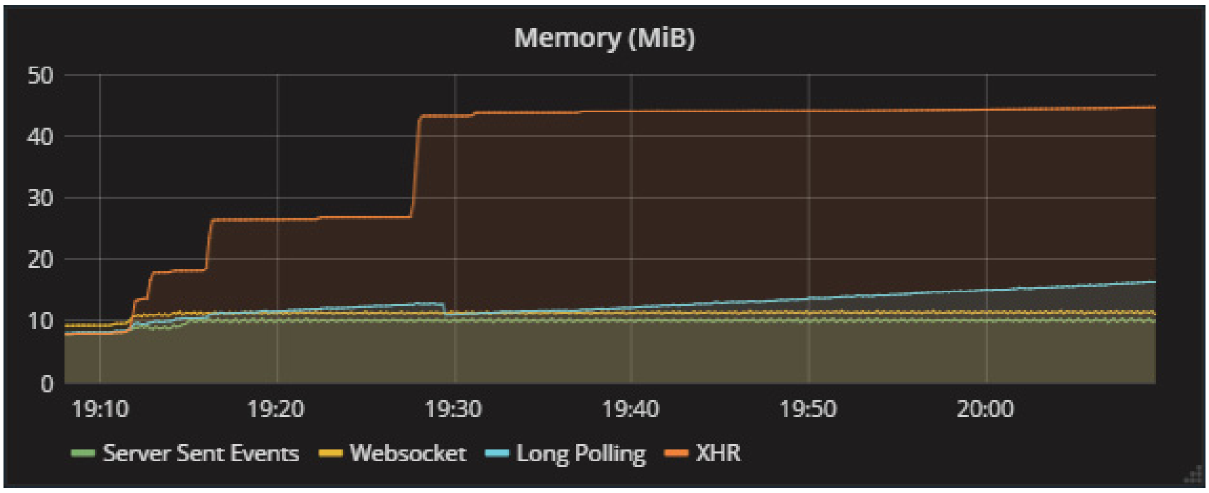


Figure 12. Performance test 2 memory usage result

Values

- XHR: Memory increasing in four steps averaging at 13.50, 17.98, 26.62 and 44.02 MiB. (Idle levels at 7.94 MiB)
- Long Polling: Average of 12.63 MiB (Idle levels at 8.09 MiB)
- Server Sent Events: Average of 9.88MiB (Idle levels at 7.94 MiB)
- Websocket: Average of 11.23MiB (Idle levels at 9.28 MiB)

Memory usage while manually calling garbage collection requests for performance test 2

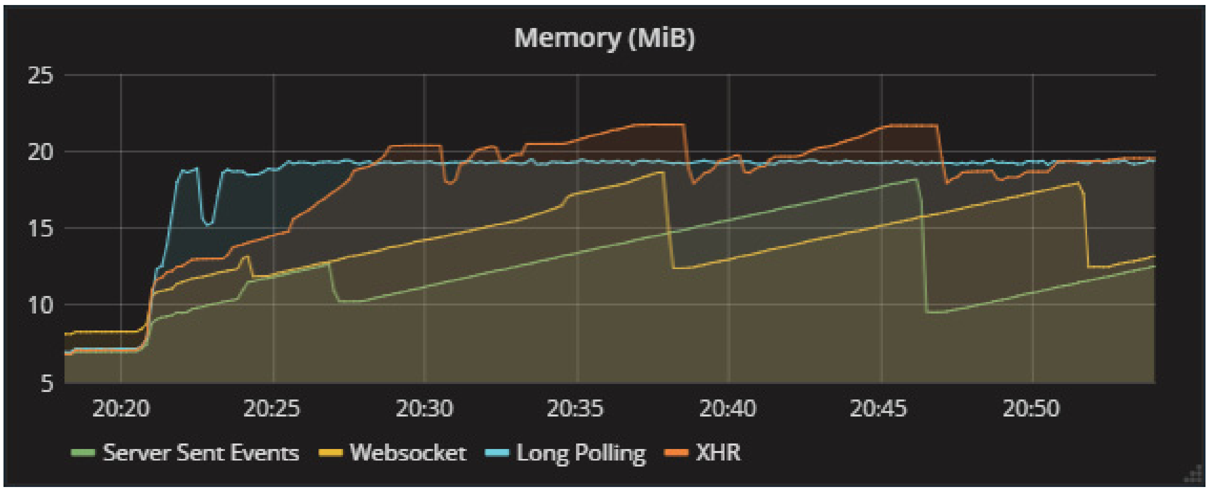


Figure 13. Performance test 2 memory usage with garbage collection requests result

Values

- XHR: Increasing to about 21 MiB and then dropping down to about 18 MiB in cycles
- Long Polling: Rapidly increasing to about 19 MiB, dropped down to about 15 MiB and then again rapidly increasing to 19 MiB where it stabilized
- Server Sent Events: Increasing to about 12.7 MiB and then dropping to 10.2 MiB followed by increasing again to 18.2 MiB. After reaching 18.2 MiB it dropped to 9.6MiB and kept increasing until the test was terminated.

- Websocket: Increasing to about 12.7 MiB and then dropping to 10.2 MiB followed by increasing again to 18.2 MiB. After reaching 18.2 MiB it dropped to 9.6MiB and kept increasing until the test was terminated.

CPU usage for performance test 2

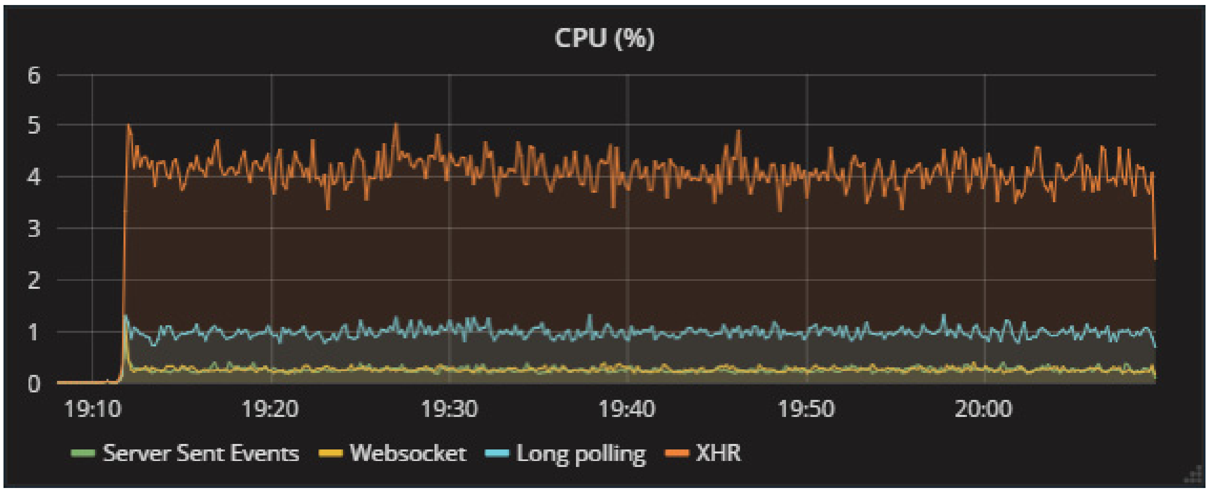


Figure 14. Performance test 2 CPU usage result

Values

- XHR: 4.11%
- Long Polling: 0.98%
- Server Sent Events: 0.25%
- Websocket: 0.26%

Network traffic for performance test 2

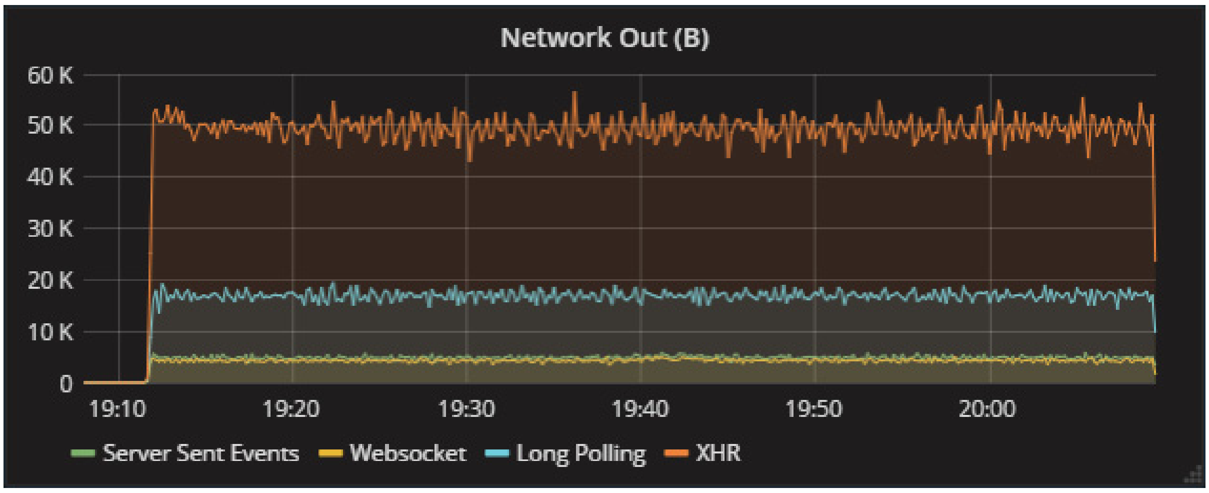


Figure 15. Performance test 2 sent network traffic result

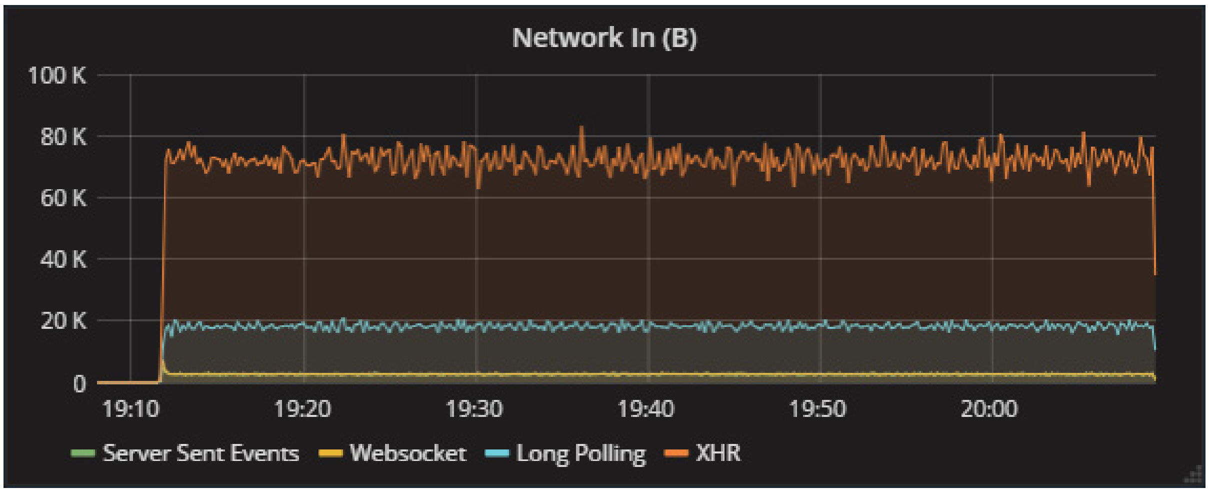


Figure 16. Performance test 2 received network traffic result

Values

- Sent
  - XHR: average of 49.5 kilobytes per second
  - Long Polling: 16.9 kilobytes per second
  - Server Sent Events: 4.9 kilobytes per second
  - Websocket: 4.3 kilobytes per second
- Received
  - XHR: 72.2: kilobytes per second
  - Long Polling: 18.3 kilobytes per second
  - Server Sent Events: 2.7 kilobytes per second
  - Websocket: 2.7 kilobytes per second

TCP connections for performance test 2

Values

- XHR: TCP connections increased in several steps over time. 24 connections were used at the start of the test and 289 simultaneous open connections were used at the end of the test
- Long Polling: 100 open connections
- Server Sent Events: 100 open connections
- Websocket: 100 open connections

Message frequency for performance test 2

Values

- XHR: 160 messages per second
- Long Polling: 50 messages per second
- Server Sent Events: 50 messages per second
- Websocket: 50 messages per second

5.8.3 Analysis for performance test 2

Memory usage for performance test 2
As expected Long polling was using more memory than SSE and Websockets while XHR polling was the most memory heavy. SSE and Websockets used a similar amount of memory as expected.

The XHR polling memory usage was like the results from the previous test. This was also anticipated in the hypothesis. The memory usage is high and an analysis of this is done in the in the memory usage analysis of the first test (section 5.7.3).

CPU usage for performance test 2
As expected in the hypothesis, the CPU usage of the SSE, Websockets and Long polling servers are about four times smaller than in the previous test.

The XHR polling server's CPU usage was as expected similar to the results in test 1 (4.23% in the previous test and 4.11% for this test) with a small difference of 0.12%. The difference could possibly be due to the XHR polling not needing to send the whole payload with every message as it did in the previous test. In this test three out of four outbound messages from the server were expected to be without payload since the request frequency was four times higher than the payload generation frequency.

The CPU usage was as expected the heaviest for XHR polling, followed by Long polling, SSE and Websockets.

Network traffic for performance test 2
Sent
The network traffic for XHR polling is more than in Long polling. Long polling is sending more traffic than SSE and Websockets. SSE is sending slightly more traffic than Websockets which is probably due to the 6 bytes larger overhead size. This means that the result is matching the hypothesis.

Received
The received network traffic for XHR polling is about four times larger than for Long polling as expected. Received network traffic for SSE and Websockets were equal and low as expected. This result is matching the hypothesis.

TCP connections for performance test 2
All results show similar values as in the previous test. This was anticipated in the hypothesis.

XHR polling was, as in the previous test, using a high amount of TCP connections. An analysis of XHR polling's high TCP connection usage can be found in the TCP connection analysis of the previous test (section 5.7.3)

Message frequency for performance test 2
All message frequency results match the hypothesis. XHR polling is still affected by the 20% lower frequency discussed in the message frequency analysis of the previous test (section 5.7.3).

5.9 Performance test 3

- 100 simultaneous clients
- 2000 ms server message interval
- 2000 ms XHR poll interval

5.9.1 Hypothesis for performance test 3

Since these tests are running at the same server message frequency as the second test, it was expected that Long polling, SSE and Websocket will have the same results in all areas as what was measured in the second test (section 5.8).

XHR polling is expected to have the same performance as Long polling in all areas except for TCP connections. Around 22 simultaneous TCP connections should be seen if the theories are correct about Firefox using a shared connection pool between tabs and that using a lower polling frequency should not trigger the possible malfunction mentioned in section 5.7.3. The XHR and Long polling servers should be receiving requests and sending responses at the same frequency since both the XHR poll interval and the server message interval was set to the same value.

5.9.2 Result for performance test 3

Memory usage without manually calling garbage collection requests for performance test 3

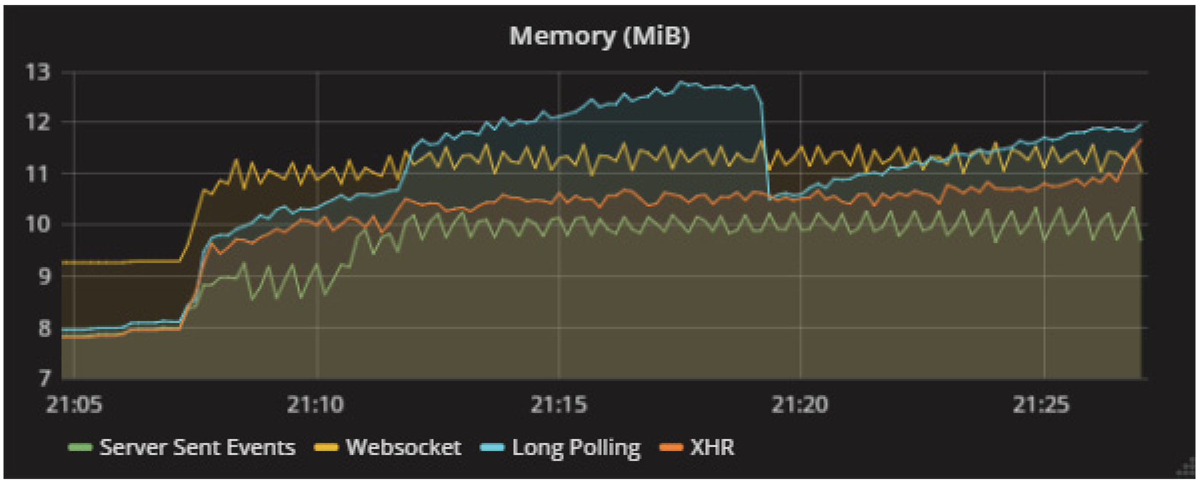


Figure 17. Performance test 3 memory usage result

Values

- XHR: From about 10 MiB, slowly but steadily increasing up to 12 MiB until the test was terminated
- Long Polling: From about 10 MiB, increasing up to 12.5 MiB, dropping down to 10 MiB and then increasing to about 12 MiB until the test was terminated
- Server Sent Events: At the beginning of the test, very frequently going between 9.50 MiB and 8.50 MiB, increasing and then very frequently going between 10 MiB and 9 MiB
- Websocket: Very frequently going between 12 MiB and 11 MiB

Memory usage with garbage collecting requests for performance test 3

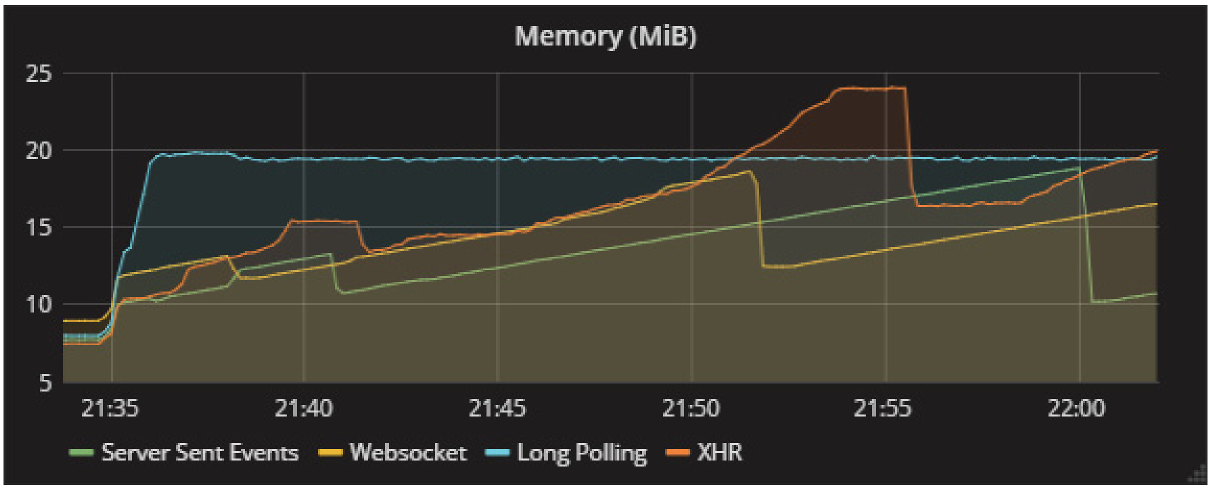


Figure 18. Performance test 3 memory usage with garbage collection requests result

Values

- XHR: Increasing up to 15 MiB and stabilized shortly, dropped down to 13 MiB, increased up to 24 MiB and stabilized shortly, dropped to 16 MiB and increased until the test was terminated
- Long Polling: Rapidly increasing up to 14 MiB and staying there until the test was terminated
- Server Sent Events: Increasing up to 13 MiB, dropping to 10 MiB, increasing to 19 MiB, dropping to 10 MiB, increasing until the test was terminated
- Websocket: Increasing to about 13 MiB and then dropping to 12 MiB followed by increasing again to 19 MiB. After reaching 12 MiB it dropped to 16 MiB and kept increasing until the test was terminated.

CPU usage for performance test 3

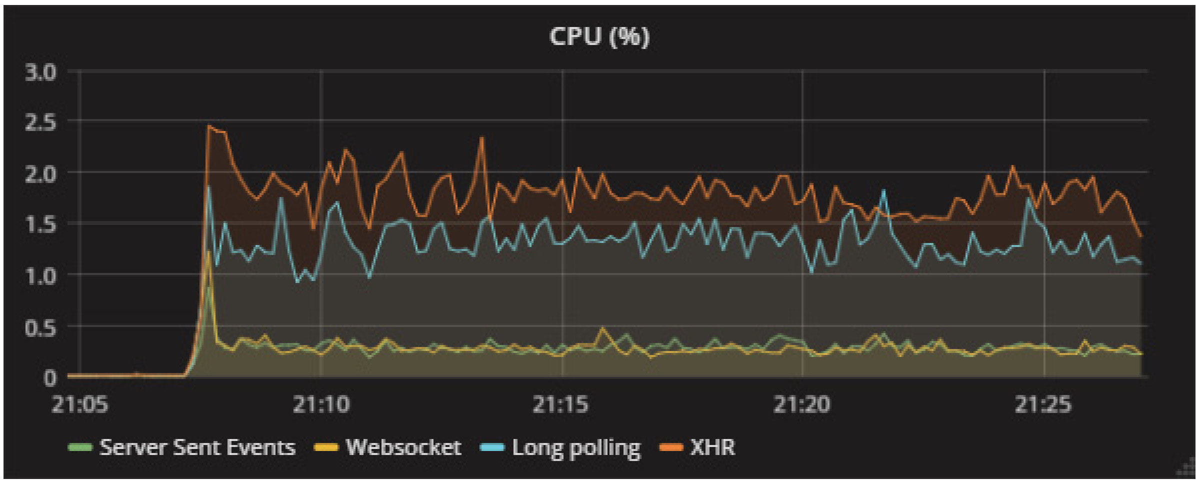


Figure 19. Performance test 3 CPU usage result

Values

- XHR: Averaging at about 1.8%
- Long Polling: Averaging at about 1.3%
- Server Sent Events: Averaging at about 0.3%
- Websocket: Averaging at about 0.3%

Network traffic for performance test 3

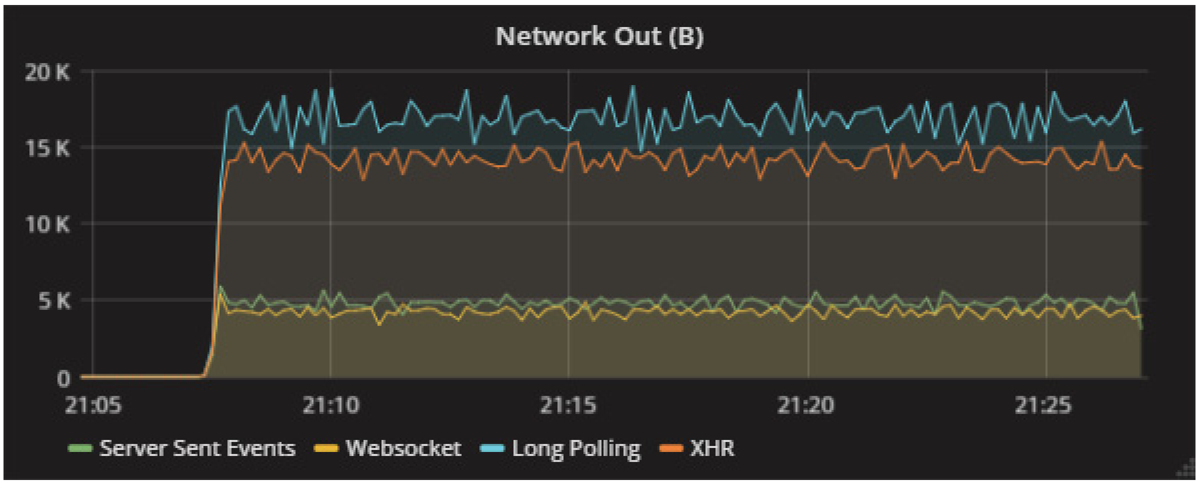


Figure 20. Performance test 3 sent network traffic result

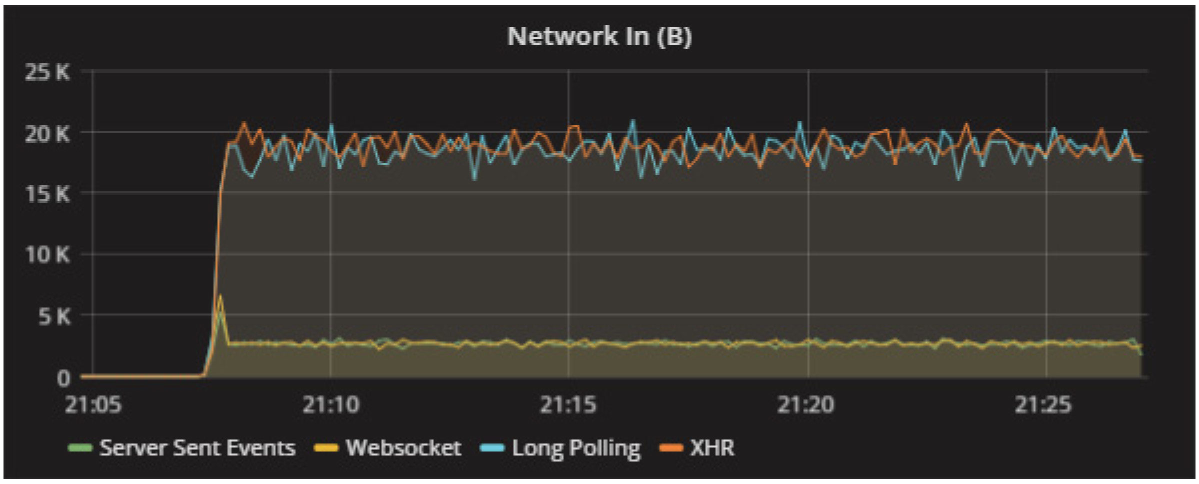


Figure 21. Performance test 3 received network traffic result

Values

- Sent
  - XHR: Averaging at about 14.2 kilobytes per second
  - Long Polling: Averaging at about 16.9 kilobytes per second
  - Server Sent Events: Averaging at about 4.8 kilobytes per second
  - Websocket: Averaging at about 4.2 kilobytes per second
- Received
  - XHR: Averaging at about 18.9 kilobytes per second
  - Long Polling: Averaging at about 18.5 kilobytes per second
  - Server Sent Events: Averaging at about 2.7 kilobytes per second
  - Websocket: Averaging at about 2.7 kilobytes per second

TCP connections for performance test 3

- XHR: 21 open connections
- Long Polling: 100 open connections
- Server Sent Events: 100 open connections
- Websocket: 100 open connections

Message frequency

- XHR: 50 messages per second
- Long Polling: 50 messages per second
- Server Sent Events: 50 messages per second
- Websocket: 50 messages per second

5.9.3 Analysis for performance test 3

Memory usage for performance test 3

The memory usages for all technologies except XHR polling are like the previous test as anticipated in the hypothesis.

Differences can be seen in the memory usage results between the test where memory usage when requesting the garbage collector to run and the test where the garbage collector was not requested to run. Because of this it is not possible to determine if XHR polling is using the same amount of memory as Long polling. A drop can be seen in XHR polling's memory usage in test when the garbage collector is not requested. XHR polling and Long polling are measured to be using the same amount of memory at the lowest point of this drop where the memory stops decreasing. The drop could be caused by the garbage collector making a run. The fact that XHR polling and Long polling end up at the same value after this drop suggests that they were using approximately the same amount of memory.

CPU usage for performance test 3

XHR polling is using 0.5% percentage points more CPU power than Long polling. An explanation for this difference could be that the XHR polling server must process an ID in the request sent from the client. This is not something the Long polling server has to do. This was something done in the tests in this study in order to enable the XHR polling server to respond with empty payloads if the XHR poll interval would be set to a higher value than the server message interval.

The CPU usages measured for all technologies except XHR polling are similar to the previous test as anticipated in the hypothesis.

Network traffic for performance test 3

Sent

All technologies except for XHR polling are sending the same amount of traffic as in the previous test as anticipated in the hypothesis.

The XHR polling clients is sending slightly less traffic than the Long polling clients, which was not expected. One possible explanation is that if the XHR clients poll interval is slightly inaccurate it might sometimes send a request before a new message has been generated on the server and sometimes after, still maintaining the same poll frequency average. This could result in some messages being sent to the XHR clients with no payload and could explain the slightly lower outbound traffic.

Received

All technologies except XHR polling are receiving the same amount of network traffic as in the previous test as was anticipated in the hypothesis.

XHR polling was as expected receiving the same amount of network traffic as Long polling.

TCP connections for performance test 3

All technologies except for XHR polling are using 100 simultaneous connections except for XHR polling which is using 21 (about 22 was expected). This supports but does not confirm the theory that Firefox is using a shared connection pool between tabs and that lowering the stress on the clients could prevent it malfunctioning. The results are matching the hypothesis.

Message frequency for performance test 3
All technologies sent 50 messages per second which was anticipated in the hypothesis.

6 ANALYSIS

This section contains two parts. First an analysis of the information gathered during the study. Information gathered from the performance tests are compared with information retrieved by summarizing and analyzing the specifications for each technology. The second part contains answers to the research questions based on the analysis done in the first part.

6.1 Analysis

6.1.1 Comparison of the performance tests results with the performance impacting factors identified in the specification summary analysis

The technology specification analysis (section 4) identified the following performance impacting factors; the TCP connections, the HTTP requests (request response architecture VS server push architecture), generating overhead data for server to client messages and the size of the overhead data.

The results from the performance tests supports the validity of the performance impacting factors identified in the technology specification analysis for Websockets and Server sent events. This can be concluded by the fact that the result of the performance tests was in all cases matching the hypothesis. All hypotheses were based on what factors were identified in the technology specification analysis.

The results from performance testing XHR polling and Long polling did not always match the hypothesis. The differences between the performance test results and the hypothesis could possibly be explained by the clients used in the test being inaccurate and malfunctioning but it was not confirmed. This means that there might exist some performance factors that were not identified in the technology specification analysis regarding XHR polling and Long polling.

6.1.2 Unexpected XHR polling performance results when using high message frequencies.

Unexpected results were seen for XHR polling in the first two performance tests where the poll frequency was set to 2 requests per second. XHR polling showed a 20% lower message frequency than what was expected. The amount of TCP connections also varied from 22 simultaneous connections up to 289 which was not expected in the hypothesis based on the analysis of the XHR polling specification. A possible explanation for this behavior is discussed in the analysis of the first test (section 5.7.3). The summary of this explanation is that Firefox might be sharing connections between tabs for optimization reasons. That combined with a high poll frequency could make Firefox malfunction with a 20% lower message frequency and an unexpected unstable amount of TCP connections being used.

A third test was done to investigate if a lower polling frequency would prevent the malfunction from occurring and support the theory about Firefox's possible connection sharing optimization. The results from this test supported the hypothesis however the theory could not be confirmed. Because the hypothesis was only supported, and not confirmed, it is possible that a performance factor not found when analysing the XHR polling specifications. This possible unidentified performance factor could be the reason causing the unexpected results.

6.1.3 Performance difference between the technologies

The results from the performance tests showed similar results for Websockets and Server sent events. The results were also similar for XHR polling and Long polling. A clear difference could be seen in CPU, memory usage and network traffic between these two pairs of technologies. This is not unexpected since the factors identified by the technology specification analysis in section 4 are very similar for the paired-up technologies. Both the hypothesis based on the technology specification analysis and the analysis of the performance test results conclude that Websockets and Server sent events are the most performance efficient technologies

Performance difference between Websockets and Server sent events
The performance difference identified between Websockets and SSE is too small to be analyzed. The differences could be a result of measurement inaccuracies. In theory, the overhead data would not increase with payload size for SSE while it should do that for Websockets. Smaller payloads however allow Websockets to send slightly less overhead than SSE. The difference is only a few bytes so a very high client amount would be needed for this to possibly be measured and compared. These differences are situational and it is concluded that identifying one of them as more performance efficient than the other is not doable.

Performance difference between XHR polling and Long polling
The performance difference between these two technologies is dependent on two settings. The performance impact for XHR polling is depending on the poll rate. For Long polling, it is dependent on how fast the server is generating messages. The resulting performance was anticipated and tested to be very similar for XHR and Long polling if these two settings are set to the same value. Which of these two technologies are the best performance wise is therefore depending on what setting the context they are used in requires.

The measured sent network traffic in test 3 was found to be slightly lower for XHR polling than for Long polling which was not expected in the hypothesis based on the technology specification analysis. An explanation for this could be timing inaccuracies in the clients which is further explained in section 5.9.3 This possible explanation is not confirmed which potentially means that a performance factor was missed when analyzing the technology specifications for either XHR polling, Long polling or both.

6.1.4 Inconsistent memory usage results

The memory usage was tested twice in every test. Once where the garbage collector was requested to be executed at the same frequency as the server messages, and one where no requests were made to the garbage collector which leaves complete garbage collector control to Node.js. The results seen when frequently requesting the garbage collector did not result in a memory drop of the same frequencies as the garbage collector was requested. This suggests that the garbage collector was not running with the same frequency as what was requested. This makes an analysis of the difference in memory usage between the technologies and how the garbage collector influences this result difficult.

6.1.5 The impact of payload size

The performance tests in this study focused on small messages being sent with high frequencies. Increasing the payload of these messages would increase the resources used by the server. The payload is not part of the communication technology which means that it does not change the performance of the technologies even though more system resources are used on the server. It does however make the performance effect due to the choice of technology smaller in percentage. For example, the performance tests in this study used an 8-byte overhead for SSE and a 464-byte overhead for Long polling (request + response). This was sent with a 30-byte payload. This would make the total message sizes being 38 and 494 bytes for SSE and Long polling respectively. This results in Long polling sending 1300% more network traffic per message (TCP overhead data is not considered). If the payload would have been 5000 bytes instead the messages would be 5008 and 5494 respectively which results in Long polling sending about 9.7% more network traffic per message. This shows that the performance difference between the technologies depend on the payload size. Smaller payloads lead to a bigger percentage difference in performance.

6.2 Research question answers

RQ1: Which of the compared technologies is the highest performing for server to client communication?

Websockets and Server sent events were found to be performing better than the other studied technologies under the experimental conditions. The performance test results were very similar for Websockets and Server sent events and was not able to be clearly measured. The differences discussed between the technologies in section 6.1.3 were concluded to be situational and one could not be said to be better performing than the other.

RQ2: Which factors, when considering server to client communication, of the compared technologies have an impact on server performance?

The following factors of the compared technologies were concluded to have an impact on the server performance in section.

- The amount of TCP connections needed
- Receiving and processing requests on the server. This is only affecting the XHR polling and Long polling technologies.
- Generating overhead data for server to client messages
- The overhead data

It could not be concluded whether these are the only performance impacting factors or if there are more factors that impacts the performance as also discussed in section 6.1.1.

7 CONCLUSION

The results of this study found that the choice of technology does matter when it comes to web server performance. How much this matters depends on how big payload sizes are being used in the web applications. Smaller payloads were found to result in the technology choice having a larger performance impact.

Websockets and Server sent events were in the experiments measured to have very similar performance and they were measured to be the most performance efficient of the compared technologies in the experimental conditions used in this study. This suggests that to increase server performance and reduce hardware costs these technologies would be better than the other technologies compared.

Factors impacting the server performance were found to be the amount of connections used, if requests had to be received and processed on the server and how much overhead data is needed. It was not determined if these are the only performance impacting factors or if there are more factors impacting the servers that were not found in this study.

Websockets and Server sent events were found to be performing better than the other studied technologies under the experimental conditions. The performance differences found between Websockets and Server sent events were concluded to be very small and situational, which is why the answer was determined to be that both are the highest performing of the studied technologies.

Software projects can be very different with different requirements. It is therefore hard to say which technology to use in which situation as there are so many possible situations. The decision on which technology to use is ultimately up to the developers. However, one thing to consider is that the high performing technologies (SSE and Websockets) require an open connection between the server and the client at all times, which may not be desirable in all situations.

8 VALIDITY THREATS

8.1 Extensibility of the results

The results gathered in this study are valid for the conditions used in the experiments. It does not guarantee that the results in this study will be valid for other conditions in more varied scenarios. The use cases used in the performance tests did all have high message frequencies. However only 100 simultaneous clients were used which is low amount for a big web site. the performance tests did only require the servers to use at max about 4% of the available CPU power. A more extensive performance test where more server resources would be required could possibly have resulted in more information affecting the result of this study.

8.2 Unknown garbage collector behavior, network connection and inaccurate clients affecting test results

Since the server software we used is based on Node.js it was not possible to have full control over how memory is allocated. Separate memory tests were made where more frequent requests to run the Node.js built in garbage collector was made. The results from those tests gave different results but it was not possible to force the garbage collector to run at a specific moment.

Using another server software that allows for manual memory allocation might give different memory usage results that are easier to analyze. Perhaps more differences could be found between the technologies if the memory usage was more accurately measured and not influenced by how the garbage collector works.

The client and the server was communicating over the internet during the performance tests done in this study. This means that the results could possibly have been affected by an unstable connection between the client and server. An unstable connection would likely lead to more TCP overhead data being sent over the network and more CPU power being used to handle the communication.

It was found during the study that all the clients running on the same machine could possibly affect the result of the performance tests through inaccuracies in poll frequencies and TCP connection managing. Using a botnet or similar tool could possibly result in different performance test results.

8.3 Servers or clients not following the standard

The technology specification analysis in this report was mostly based on standards determined by the Internet Engineering Task Force (IETF) for retrieving information about how the technologies work. There is nothing stating that browsers or web servers must follow the IETF standard and all manufacturers can differ from this standard.

It was deemed unlikely that any potential differences between web servers or browsers would be large enough to alter the result of this study. The data collected from the performance tests done in this study is conforming with the results from the technology specification analysis. This is an indication of that the technologies used for the tests in this study is following the IETF standard.

If implementations differ a lot from the IETF standard the question arises if that implementation still falls under the same name or if it would be considered a new technology.

9 Future work

This section contains a list of possible future research related to this study that can be useful to have knowledge about when deciding what technology to use when building a web application.

9.1 Load balancing

Large web sites with huge amounts of clients may be hosted on multiple servers that work together to handle the huge number of clients a popular website can have. The result from this thesis is not enough to answer which technology is most efficient in a multi-server setup. More research needs to be done that considers how easily the technologies can be used together with load balancing to spread the clients over many servers and what the performance effects of this is.

9.2 HTTP over TLS

Another thing to research could be considering HTTP over TLS (HTTPS) and how this affects the studied technologies. The performance impact on encrypting and decrypting messages using TLS would likely influence the performance of technologies. Considering that XHR-polling, Long polling and SSE use HTTP for every message they are likely affected by this. The effect on Websockets is hard to tell without more information.

9.3 HTTP/2

This study used the HTTP version 1.1 when comparing the technologies. How the newer HTTP/2 protocol affects the technologies compared in this study is something to consider in the future as more and more websites start using the new HTTP version. It is likely that a new version of HTTP would affect the performances of the studied technologies as all of them make use of HTTP in some way to communicate between the server and client.

9.4 More powerful and varied performance tests

More varied performance tests considering more situations than what was used in this study could possibly yield more results. In this study 100 simultaneous clients were used to performance test the compared technologies due to hardware limitations available to the authors. This was enough to see clear differences between the technologies but did not result in a very high load of the server where CPU usage was at maximum averaging at about 4% due to our tests. Maybe tests that challenges the servers more can result in new information.

REFERENCES

Alvestrand, H. 2004. "A Mission Statement for the IETF". BCP 95. RFC 3935. DOI 10.17487/RFC3935. http://www.rfc-editor.org/info/rfc3935.

Apache software foundation. 2017a. "Apache HTTP Server Version 1.3." Accessed April 27. https://httpd.apache.org/docs/1.3/mod/core.html.

Apache software foundation. 2017b. "Core - Apache HTTP Server." Accessed April 27. https://httpd.apache.org/docs/2.0/mod/core.html.

Apache software foundation. 2017c. "Core - Apache HTTP Server Version 2.2." Accessed April 27. https://httpd.apache.org/docs/2.2/mod/core.html.

Apache software foundation. 2017d. "Core - Apache HTTP Server Version 2.4." Accessed April 27. https://httpd.apache.org/docs/2.4/mod/core.html.

Chromium. 2017a. "HTTP Pipelining - The Chromium Projects." Accessed May 21. https://www.chromium.org/developers/design-documents/network-stack/http-pipelining.

Chromium. 2017b. "SPDY: An Experimental Protocol for a Faster Web - The Chromium Projects." Accessed April 21. http://dev.chromium.org/spdy/spdy-whitepaper.

Fette, I., and A. Melnikov. 2011. "The WebSocket Protocol". RFC 6455. DOI 10.17487/RFC6455. http://www.rfc-editor.org/info/rfc6455.

Fielding, R., Ed., and J. Reschke, Ed. 2014 "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing". RFC 7230. DOI 10.17487/RFC7230. http://www.rfc-editor.org/info/rfc7230.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee. 199 "Hypertext Transfer Protocol -- HTTP/1.1". RFC 2616. DOI 10.17487/RFC2616. http://www.rfc-editor.org/info/rfc2616.

Grigorik, Ilya. 2013. "Browser APIs and Protocols: Server-Sent Events (SSE) - High Performance Browser Networking (O'Reilly)." *High Performance Browser Networking*. Accessed May 17 https://hpbn.co/server-sent-events-sse/.

Kozierok, Charles M. 2005. "The TCP/IP Guide - HTTP Response Message Format." Accessed April 27 http://www.tcpipguide.com/free/t_HTTPResponseMessageFormat.htm.

Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins. 2011. "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP". RFC 6202. DOI 10.17487/RFC6202. http://www.rfc-editor.org/info/rfc6202.

MacVittie, L. 2016. "HTTP Pipelining: A security risk without real performance benefits". Accessed April 27. https://devcentral.f5.com/articles/http-pipelining-a-security-risk-without-real-performance-benefits.

Majumdar, B., Mysore, U., Sahoo, L., Saxena, S. "Load Testing Web Services". 2007. Accessed May 24. http://soa.sys-con.com/node/284564

Microsoft. 2016. "How to Change the Default Keep-Alive Time-out Value in Internet Explorer." https://support.microsoft.com/en-us/help/813827/how-to-change-the-default-keep-alive-time-out-value-in-internet-explorer.

Meier, J.D., Farre, C., Bansode, P., Barber, S., Rea, D. 2007. Chapter 17 “Performance Testing Guidance for Web Applications”. Microsoft Corporation. Accessed May 24. https://msdn.microsoft.com/en-us/library/bb924372.aspx

MozillaZine. 2011. "Network.http.keep-Alive.timeout - MozillaZine Knowledge Base." http://kb.mozillazine.org/Network.http.keep-alive.timeout.

MozillaZine. 2012. "Network.http.pipelining - MozillaZine Knowledge Base." http://kb.mozillazine.org/Network.http.pipelining.

Nielsen, Jakob. 2010. *Usability Engineering*. Nachdr. Amsterdam: Kaufmann.

Oracle. 2010. "How the Server Handles Requests from Clients (Sun Java System Web Server 6.1 SP11 NSAPI Programmer’s Guide)." https://docs.oracle.com/cd/E19857-01/820-7655/abvah/index.html.

MDN. 2017. "Writing WebSocket Servers." *Mozilla Developer Network*. Accessed April 13. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers.

WHATWG. n.d. "Living Standard." Web Hypertext Application Technology Working Group. Accessed May 17. https ://html.spec.whatwg.org/multipage/comms.html.