Performance Evaluation of WebSocket Protocol for Implementation of Full-Duplex Web Streams

Oleg Bilovus

Università degli Studi di Salerno

1st Scalability Research Forum

WebSocket

Performance Evaluation of WebSocket Protocol for Implementation of Full-Duplex Web Streams

Oleg Bilovus

Università degli Studi di Salerno

1st Scalability Research Forum

We will talk about WebSockets and compare its performance with TCP Socket. But, before diving into analyzing the performance we need to understand why we needed WebSockets and what they are.

Outline

Outline

Background
HTTP polling
HTTP long polling
Streaming

WebSocket protocol
Definition
Handshake
Frame
API

Performance vs TCP Socket
Performance Evaluation
WebSocket sequence diagram
Network traffic
Data Transfer Time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

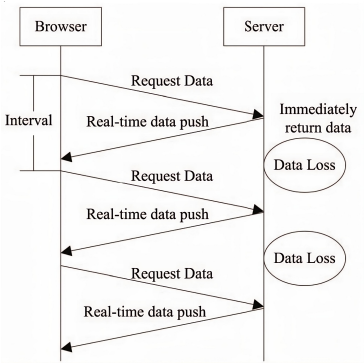
Background

- *Historically*, creating web applications that need bidirectional communication between a client and a server has required an abuse of HTTP to poll the server for updates while sending upstream notifications as distinct HTTP calls.

Bidirectional means the server and the client can send data to each other at any time

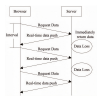
HTTP polling

Check whether the server is changed in a while, thereby performing incremental updates.



HTTP polling

Check whether the server is changed in a while, thereby performing incremental updates.

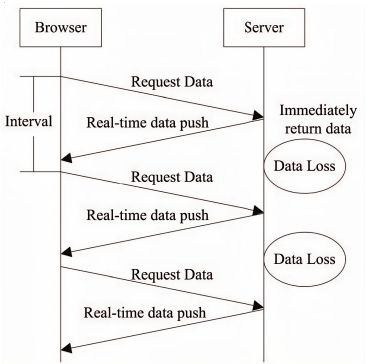


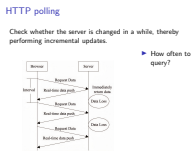
A client can send data and ask for data at the same time. But, if client has no data and server has no data, a request and response will still be generated with all the HTTP headers and thus wasting resources.No real-time data because while the client waits, an event could occur and the client will know about it only when the timeout expires.

HTTP polling

Check whether the server is changed in a while, thereby performing incremental updates.

- How often to query?

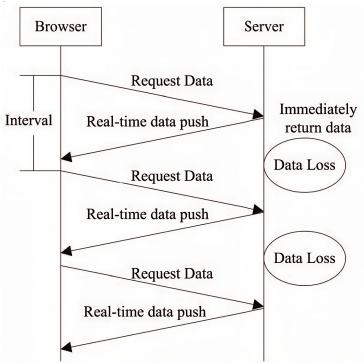




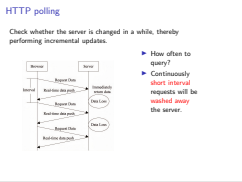
A client can send data and ask for data at the same time. But, if client has no data and server has no data, a request and response will still be generated with all the HTTP headers and thus wasting resources.No real-time data because while the client waits, an event could occur and the client will know about it only when the timeout expires.

HTTP polling

Check whether the server is changed in a while, thereby performing incremental updates.



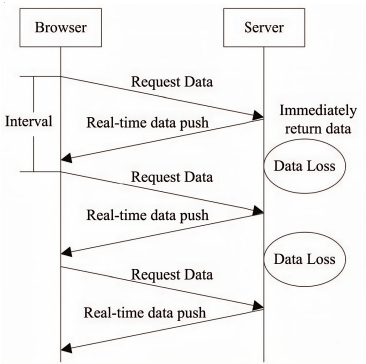
- How often to query?
- Continuously short interval requests will be washed away the server.



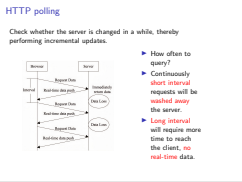
A client can send data and ask for data at the same time. But, if client has no data and server has no data, a request and response will still be generated with all the HTTP headers and thus wasting resources.No real-time data because while the client waits, an event could occur and the client will know about it only when the timeout expires.

HTTP polling

Check whether the server is changed in a while, thereby performing incremental updates.



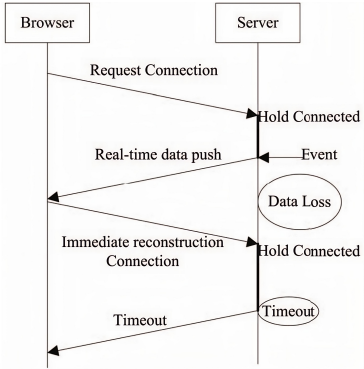
- How often to query?
- Continuously short interval requests will be washed away the server.
- Long interval will require more time to reach the client, no real-time data.



A client can send data and ask for data at the same time. But, if client has no data and server has no data, a request and response will still be generated with all the HTTP headers and thus wasting resources.No real-time data because while the client waits, an event could occur and the client will know about it only when the timeout expires.

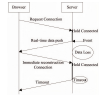
HTTP long polling

When a client sends a data request, the server will block the request until there is data transfer or timeout before returning.



HTTP long polling

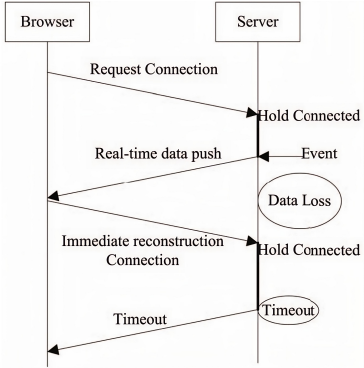
When a client sends a data request, the server will block the request until there is data transfer or timeout before returning.



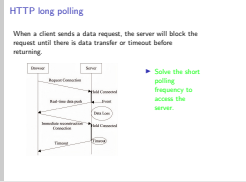
Can hold the connection up to a certain time, after that a timeout is exceeded and need a new connection.No bidirectional because the client may only send data the first time, but then it will only receive until a timeout and another request is made.In the normal polling we could have bidirectional because the interval was shorter.

HTTP long polling

When a client sends a data request, the server will block the request until there is data transfer or timeout before returning.



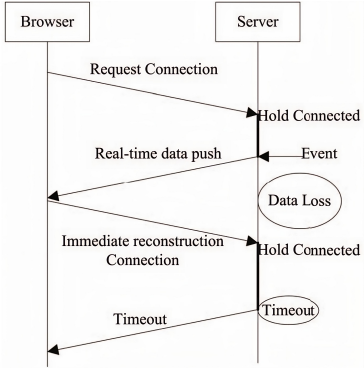
- Solve the short polling frequency to access the server.



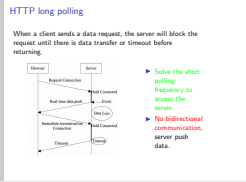
Can hold the connection up to a certain time, after that a timeout is exceeded and need a new connection.No bidirectional because the client may only send data the first time, but then it will only receive until a timeout and another request is made.In the normal polling we could have bidirectional because the interval was shorter.

HTTP long polling

When a client sends a data request, the server will block the request until there is data transfer or timeout before returning.



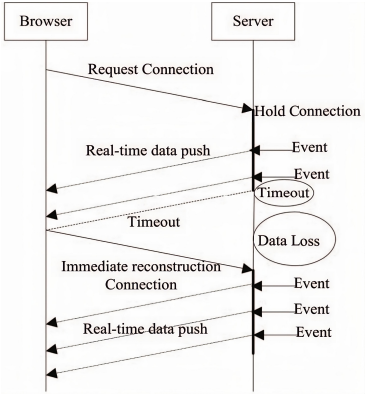
- Solve the short polling frequency to access the server.
- No bidirectional communication, server push data.

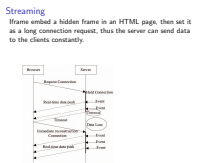


Can hold the connection up to a certain time, after that a timeout is exceeded and need a new connection.No bidirectional because the client may only send data the first time, but then it will only receive until a timeout and another request is made.In the normal polling we could have bidirectional because the interval was shorter.

Streaming

Iframe embed a hidden frame in an HTML page, then set it as a long connection request, thus the server can send data to the clients constantly.

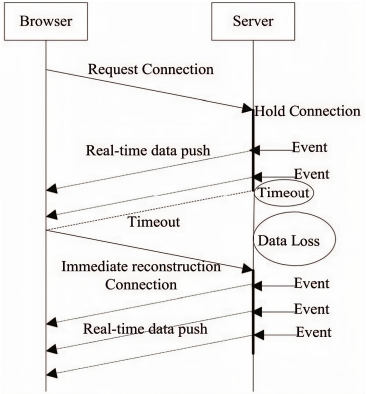




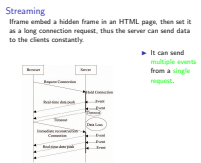
iframe is a html page inside another.Because the server need to keep the connections alive.

Streaming

Iframe embed a hidden frame in an HTML page, then set it as a long connection request, thus the server can send data to the clients constantly.



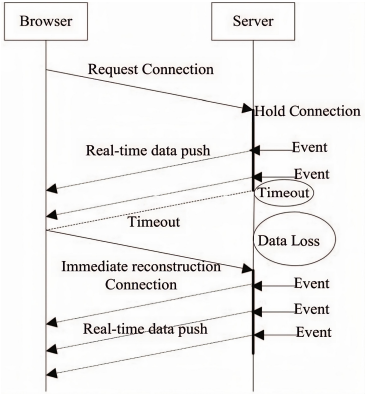
- It can send multiple events from a single request.



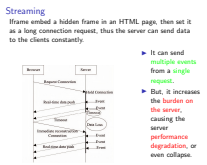
iframe is a html page inside another.Because the server need to keep the connections alive.

Streaming

Iframe embed a hidden frame in an HTML page, then set it as a long connection request, thus the server can send data to the clients constantly.



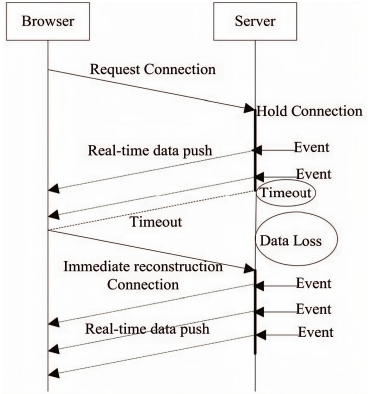
- It can send multiple events from a single request.
- But, it increases the burden on the server, causing the server performance degradation, or even collapse.



iframe is a html page inside another.Because the server need to keep the connections alive.

Streaming

Iframe embed a hidden frame in an HTML page, then set it as a long connection request, thus the server can send data to the clients constantly.



- It can send multiple events from a single request.
- But, it increases the burden on the server, causing the server performance degradation, or even collapse.
- No bidirectional communication.

Streaming
Iframe embed a hidden frame in an HTML page, then set it as a long connection request, thus the server can send data to the clients constantly.
- It can send multiple events from a single request.
- But, it increases the burden on the server, causing the server performance degradation, or even collapse.
- No bidirectional communication.

iframe is a html page inside another.Because the server need to keep the connections alive.

Outline

Outline
Background
HTTP polling
HTTP long polling
Streaming
WebSocket protocol
Definition
Handshake
Frame
API
Performance vs TCP Socket
Performance Evaluation
WebSocket sequence diagram
Network traffic
Data Transfer Time

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

RFC 6455
Keywords
The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

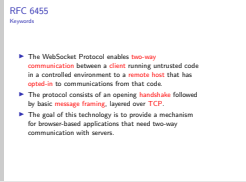
RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

Keywords

- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.
- The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers.

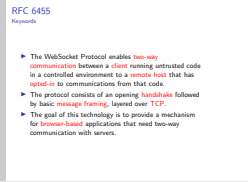

RFC 6455
Keywords
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.
- The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers.

opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

RFC 6455

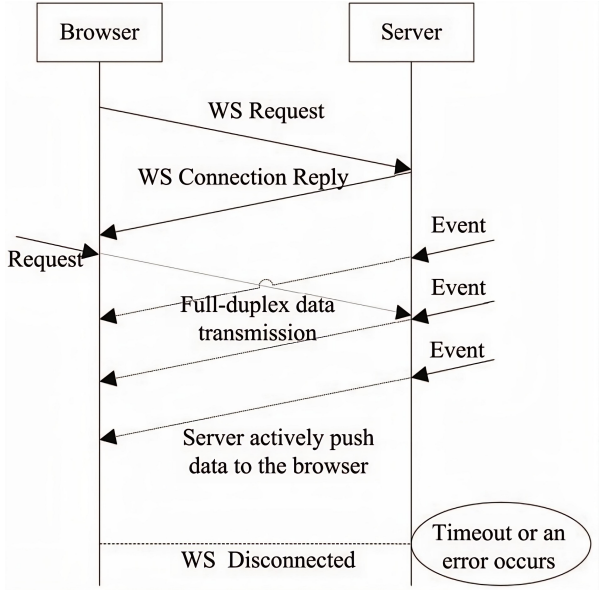
Keywords

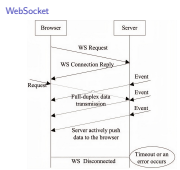
- The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.
- The protocol consists of an opening handshake followed by basic message framing, layered over TCP.
- The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers.



opted-in is important because with polling any HTTP server would accept it, but here additional steps are needed.Handshake means client and server have to agree that they can both use the protocol and the server has to prove it.Message framing because we do not want to send every time the headers.TCP means it is reliable, no messages will be lost.

WebSocket





There is the initial handshake, after that, client and server can send and receive data at any moment without further interaction.There is no timeout. If it disconnects, it is because of an error and to establish the connection, the handshake has to be done again.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.

With the Upgrade Response, the server proves that it can communicate with WebSockets.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.
- To establish a session, client sends a WebSocket Upgrade Request to the server, upon which server responds with a WebSocket Upgrade Response.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.
- To establish a session, client sends a WebSocket Upgrade Request to the server, upon which server responds with a WebSocket Upgrade Response.

With the Upgrade Response, the server proves that it can communicate with WebSockets.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.
- To establish a session, client sends a WebSocket Upgrade Request to the server, upon which server responds with a WebSocket Upgrade Response.
- From this point forward, the client and server can send data back and forth in asynchronous full-duplex mode.

Handshake

- For WebSocket-based communication, a WebSocket session should be established first.
- To establish a session, client sends a WebSocket Upgrade Request to the server, upon which server responds with a WebSocket Upgrade Response.
- From this point forward, the client and server can send data back and forth in asynchronous full-duplex mode.

With the Upgrade Response, the server proves that it can communicate with WebSockets.

WebSocket Upgrade Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```


WebSocket Upgrade Request

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13

Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

- HTTP GET request.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```


WebSocket Upgrade Request

- HTTP GET request.

GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13

Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

- HTTP GET request.
- URI to identify endpoint.

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```

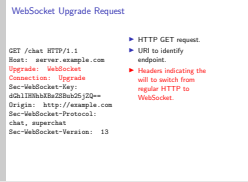

WebSocket Upgrade Request
HTTP GET request.
URI to identify endpoint.
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13

Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```

- HTTP GET request.
- URI to identify endpoint.
- Headers indicating the will to switch from regular HTTP to WebSocket.

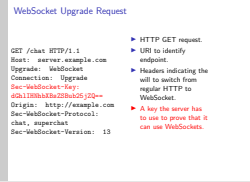


Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```

- HTTP GET request.
- URI to identify endpoint.
- Headers indicating the will to switch from regular HTTP to WebSocket.
- A key the server has to use to prove that it can use WebSockets.

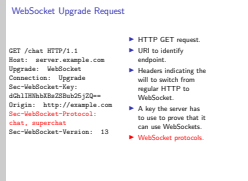


Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```

- HTTP GET request.
- URI to identify endpoint.
- Headers indicating the will to switch from regular HTTP to WebSocket.
- A key the server has to use to prove that it can use WebSockets.
- WebSocket protocols.

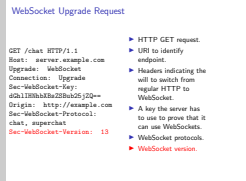


Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Request

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Key:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol:
chat, superchat
Sec-WebSocket-Version: 13
```

- HTTP GET request.
- URI to identify endpoint.
- Headers indicating the will to switch from regular HTTP to WebSocket.
- A key the server has to use to prove that it can use WebSockets.
- WebSocket protocols.
- WebSocket version.



Different URI can be used to identify different endpoints. A URI can be regular HTTP, another can be WebSocket.

WebSocket Upgrade Response

```
HTTP/1.1 101 Switching
protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat
```


WebSocket Upgrade Response

HTTP/1.1 101 Switching
protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat

There is a specific algorithm to generate this Header from a key.

WebSocket Upgrade Response

```
HTTP/1.1 101 Switching
protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat
```

- Server confirms it supports WebSocket.

WebSocket Upgrade Response

HTTP/1.1 101 Switching protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat

- Server confirms it supports WebSocket.

There is a specific algorithm to generate this Header from a key.

WebSocket Upgrade Response

```
HTTP/1.1 101 Switching
protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept:
dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat
```

- Server confirms it supports WebSocket.
- Server proves that it can use WebSocket. Client checks it.

WebSocket Upgrade Response

HTTP/1.1 101 Switching protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat

- Server confirms it supports WebSocket.
- Server proves that it can use WebSocket. Client checks it.

There is a specific algorithm to generate this Header from a key.

WebSocket Upgrade Response

```
HTTP/1.1 101 Switching
protocols
Upgrade:  WebSocket
Connection:  Upgrade
Sec-WebSocket-Accept:
dGhlIHNhbXBsZSBub25jZQ==
Origin:  http://example.com
Sec-WebSocket-Protocol:  chat
```

- Server confirms it supports WebSocket.
- Server proves that it can use WebSocket. Client checks it.
- Server tells which protocol it supports.

WebSocket Upgrade Response

HTTP/1.1 101 Switching protocols
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat

- Server confirms it supports WebSocket.
- Server proves that it can use WebSocket. Client checks it.
- Server tells which protocol it supports.

There is a specific algorithm to generate this Header from a key.

WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.

WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.

WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.
- The added overhead to the payload data is minimal because it does not send all the HTTP headers for each frame.

WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.
- The added overhead to the payload data is minimal because it does not send all the HTTP headers for each frame.

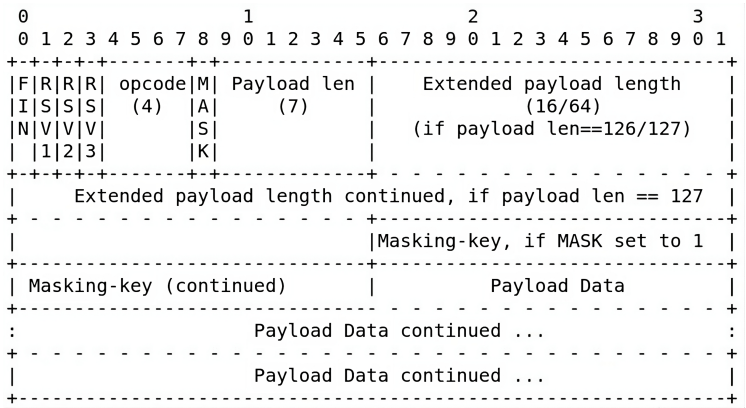
WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.
- The added overhead to the payload data is minimal because it does not send all the HTTP headers for each frame.
- Each frame adds at least 2 bytes of overhead to the payload data. Depending on the length of the payload data and the direction of the communication, the length of the overhead may increase up to 14 bytes.

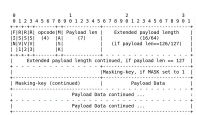
WebSocket Frame

- After the handshake is successful, client and server can communicate in full-duplex by using frames.
- The added overhead to the payload data is minimal because it does not send all the HTTP headers for each frame.
- Each frame adds at least 2 bytes of overhead to the payload data. Depending on the length of the payload data and the direction of the communication, the length of the overhead may increase up to 14 bytes.

WebSocket Frame Structure



WebSocket Frame Structure



We will not go into the details because it is out of the scope of this presentation and, as mentioned earlier, the added overhead to the payload data is minimal.

WebSocket API

The API is defined by its states of readiness, responses to a networking or messaging event.

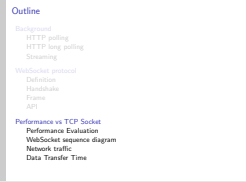
Callback	Description
`onopen`	invoked when WebSocket `session is established`, signalizes that the protocol is ready to transfer payload data
`onerror`	invoked whenever an `error occurs`
`onclose`	invoked when one of the peers has `terminated the session`
`onmessage`	invoked when an `incoming message` from another peer has arrived

WebSocket API

The API is defined by its states of readiness, responses to a networking or messaging event.

Callback	Description
onopen	invoked when WebSocket session is established, signalizes that the protocol is ready to transfer payload data
onerror	invoked whenever an error occurs
onclose	invoked when one of the peers has terminated the session
onmessage	invoked when an incoming message from another peer has arrived

Outline



We will now try to evaluate the WebSocket performance compared to the raw TCP Socket.

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic

Performace Evaluation
- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time

Performace Evaluation
- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time
- Network traffic is *evaluated analytically* using the protocol specifications.

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time
- Network traffic is *evaluated analytically* using the protocol specifications.

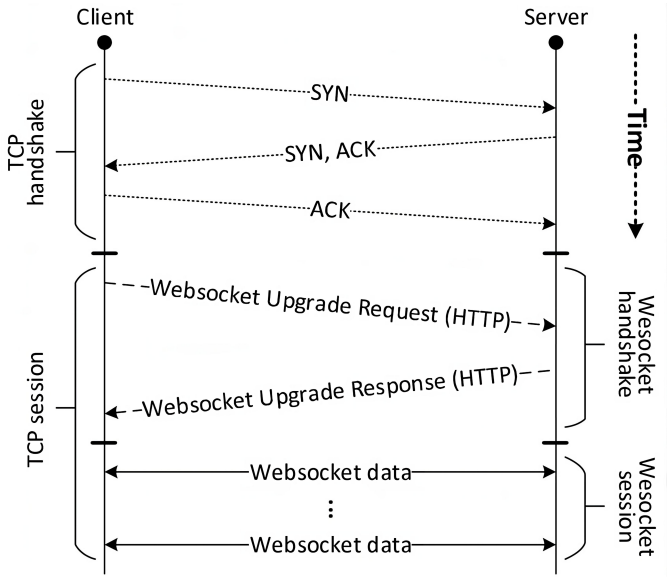
Performace Evaluation

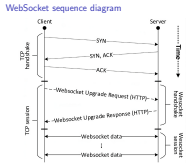
- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time
- Network traffic is *evaluated analytically* using the protocol specifications.
- Data transfer time is *evaluated experimentally* in a laboratory test bed.

Performace Evaluation

- Performance evaluation of the WebSocket and the TCP Socket protocol consists of:
  - Network traffic
  - Data transfer time
- Network traffic is *evaluated analytically* using the protocol specifications.
- Data transfer time is *evaluated experimentally* in a laboratory test bed.

WebSocket sequence diagram





WebSocket stay on top of TCP which means it will always add more overhead than the raw TCP Socket, but WebSocket is easier to use in a web environment.

Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet*, *IP and TCP header*.

Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet*, *IP and TCP header*.

Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:

Analytical Evaluation of Network Traffic
- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:

Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:
  - Handshake

Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:
  - Handshake

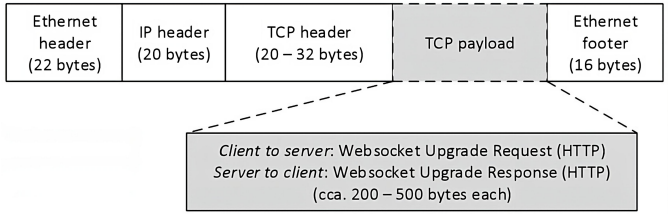
Analytical Evaluation of Network Traffic

- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:
  - Handshake
  - Frame header for each frame

Analytical Evaluation of Network Traffic

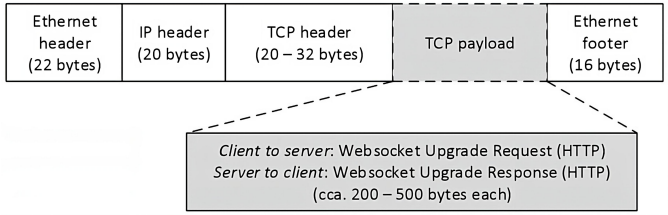
- Both protocols will have the lower level protocols fields overhead such as *Ethernet, IP and TCP header*.
- For this reason, the analysis consider only the overhead the WebSocket incurs:
  - Handshake
  - Frame header for each frame

Handshake overhead



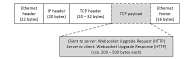
Handshake overhead

Handshake overhead



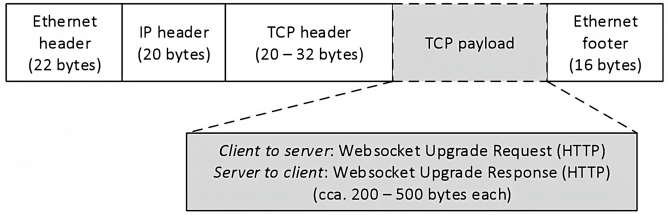
- The overhead is fixed in length and typically counts few hundreds of bytes.

Handshake overhead

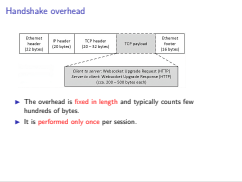


- The overhead is fixed in length and typically counts few hundreds of bytes.

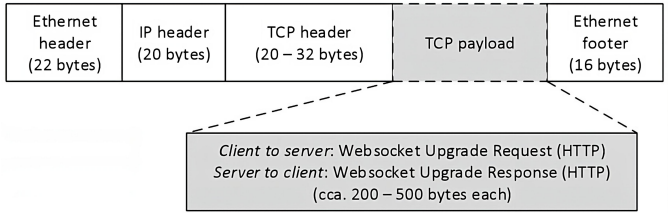
Handshake overhead



- The overhead is fixed in length and typically counts few hundreds of bytes.
- It is performed only once per session.



Handshake overhead

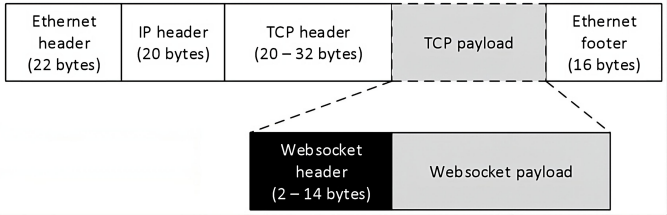


- The overhead is fixed in length and typically counts few hundreds of bytes.
- It is performed only once per session.
- Its significance decreases with the increasing number of frames sent over the same session. Thus, the evaluation is focused on long-running sessions.

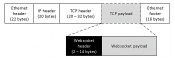
Handshake overhead

- The overhead is fixed in length and typically counts few hundreds of bytes.
- It is performed only once per session.
- Its significance decreases with the increasing number of frames sent over the same session. Thus, the evaluation is focused on long-running sessions.

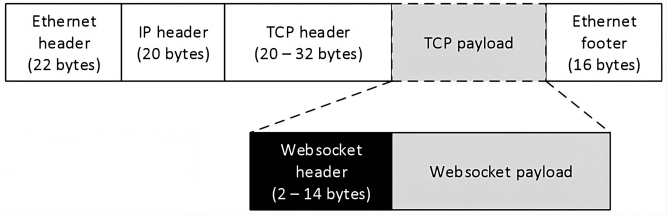
Frame overhead



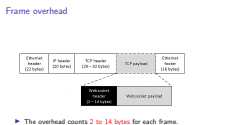
Frame overhead



Frame overhead



- The overhead counts 2 to 14 bytes for each frame.



Results

- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.

Results
- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.

Results

- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
- With WebSocket, the TCP Payload consists of both data and Frame header.

Results
- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
- With WebSocket, the TCP Payload consists of both data and Frame header.

Results

- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
- With WebSocket, the TCP Payload consists of both data and Frame header.
- This relation can be written as:

$$P_{TCP} = data \tag{1}$$

$$P_{WS} = data + H \tag{2}$$

where:

P = payload
$data$ = data to send
H = length of frame's header

Results
When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
With WebSocket, the TCP Payload consists of both data and Frame header.
This relation can be written as:
$P_{TCP} = data$ (1)
$P_{WS} = data + H$ (2)
where:
P = payload
$data$ = data to send
H = length of frame's header

Results

- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
- With WebSocket, the TCP Payload consists of both data and Frame header.
- This relation can be written as:

$$P_{TCP} = data \tag{1}$$

$$P_{WS} = data + H \tag{2}$$

where:

P = payload
$data$ = data to send
H = length of frame's header

- We can now define the network traffic overhead O_P a WebSocket has over a TCP Socket:

$$O_P = \frac{P_{WS} - P_{TCP}}{P_{TCP}} \cdot 100\% \tag{3}$$

Results
- When the data are transferred with TCP Socket, they are directly embedded as TCP Payload.
- With WebSocket, the TCP Payload consists of both data and Frame header.
- This relation can be written as:

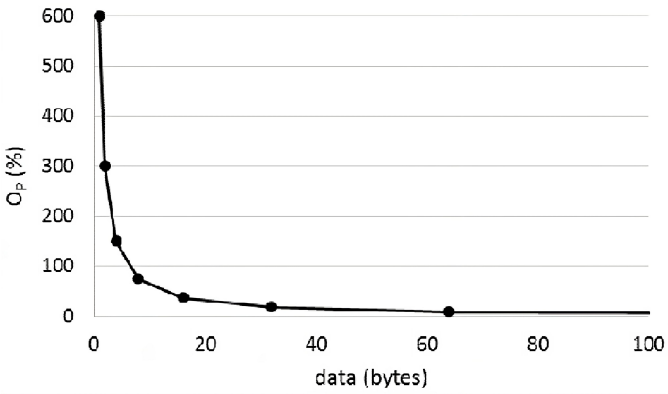
$P_{TCP} = data$ (1)
$P_{WS} = data + H$ (2)

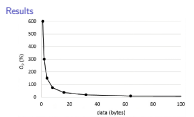
where:
P = payload
$data$ = data to send
H = length of frame's header

- We can now define the network traffic overhead O_P a WebSocket has over a TCP Socket:

$O_P = \frac{P_{WS} - P_{TCP}}{P_{TCP}} \cdot 100\%$ (3)

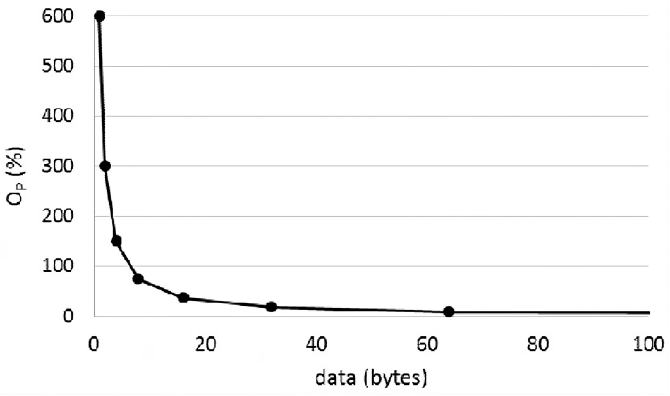
Results



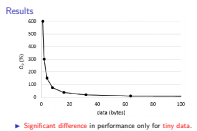


Because if I want to send 1 byte, with WebSocket it will could add 14 bytes of frame header.The difference is smaller and smaller because 14 bytes of overhead on a 1 KB data is nothing.The difference is almost identical.

Results

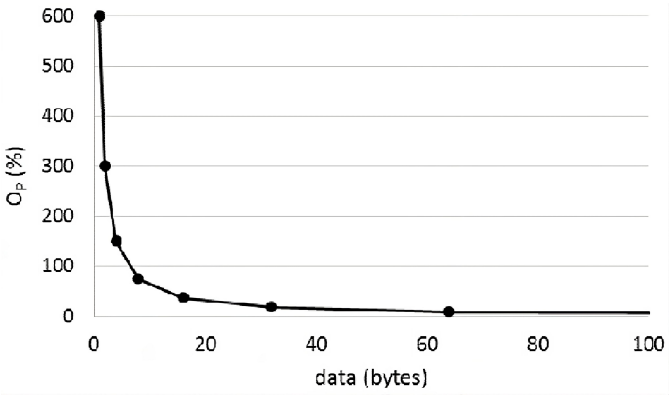


- Significant difference in performance only for tiny data.

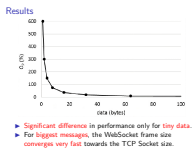


Because if I want to send 1 byte, with WebSocket it will could add 14 bytes of frame header.The difference is smaller and smaller because 14 bytes of overhead on a 1 KB data is nothing.The difference is almost identical.

Results

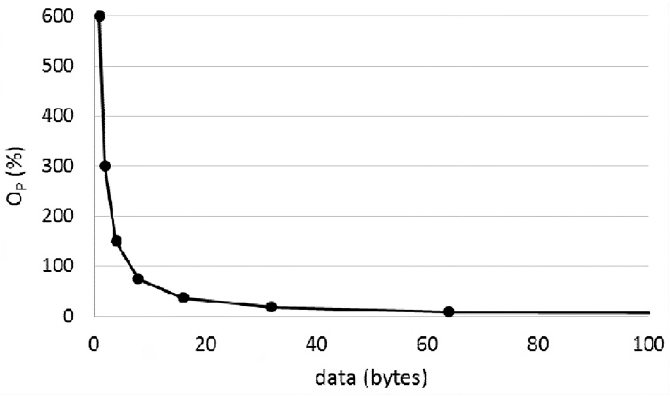


- Significant difference in performance only for tiny data.
- For biggest messages, the WebSocket frame size converges very fast towards the TCP Socket size.

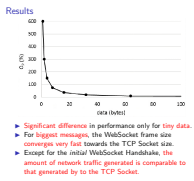


Because if I want to send 1 byte, with WebSocket it will could add 14 bytes of frame header.The difference is smaller and smaller because 14 bytes of overhead on a 1 KB data is nothing.The difference is almost identical.

Results



- Significant difference in performance only for tiny data.
- For biggest messages, the WebSocket frame size converges very fast towards the TCP Socket size.
- Except for the *initial* WebSocket Handshake, the amount of network traffic generated is comparable to that generated by to the TCP Socket.



Because if I want to send 1 byte, with WebSocket it will could add 14 bytes of frame header.The difference is smaller and smaller because 14 bytes of overhead on a 1 KB data is nothing.The difference is almost identical.

Experimental Evaluation of Data Transfer Time

- Two host machines.

Experimental Evaluation of Data Transfer Time

- Two host machines.

Experimental Evaluation of Data Transfer Time

- Two host machines.
- One playing the role of the server.

Experimental Evaluation of Data Transfer Time

- Two host machines.
- One playing the role of the server.

Experimental Evaluation of Data Transfer Time

- Two host machines.
- One playing the role of the server.
- While the other being a client.

Experimental Evaluation of Data Transfer Time

- Two host machines.
- One playing the role of the server.
- While the other being a client.

Environment configuration

	Client	**Server**
Hardware	CPU: AMD Turion II P520 RAM: 6 GB	CPU: AMD Athlon X2 5000 RAM: 5 GB
OS	Windows 8 64-bit	Windows 8 64-bit
Network	1000BASE-T (Gigabit Ethernet, **host machines directly connected** using UTP Cat5 Ethernet cable)	
TCP implementation	*java.net.Socket* (Java JDK 1.7)	*java.net.Socket* (Java JDK 1.7)
WebSocket implementation	*websocket.***client** (Jetty 9.1.0)	*websocket.***servlet** (Jetty 9.1.0)

Environment configuration

	Client	Server
Hardware	CPU: AMD Turion II P520 RAM: 6 GB	CPU: AMD Athlon X2 5000 RAM: 5 GB
OS	Windows 8 64-bit	Windows 8 64-bit
Network	1000BASE-T (Gigabit Ethernet, **host machines directly connected** using UTP Cat5 Ethernet cable)	
TCP implementation	*java.net.Socket* (Java JDK 1.7)	*java.net.Socket* (Java JDK 1.7)
WebSocket implementation	*websocket.***client** (Jetty 9.1.0)	*websocket.***servlet** (Jetty 9.1.0)

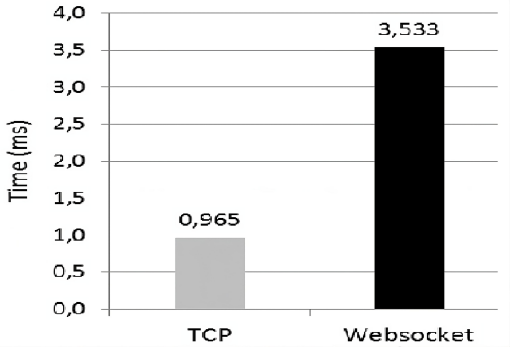
TCP Connection and WebSocket session time

In the first experiment, it is measured the time required for client and server to establish a TCP connection and WebSocket session.

TCP Connection and WebSocket session time
In the first experiment, it is measured the time required for client and server to establish a TCP connection and WebSocket session.

TCP Connection and WebSocket session time

In the first experiment, it is measured the time required for client and server to establish a TCP connection and WebSocket session.



- WebSocket session lasts 3.7 times longer than establishing a TCP connection.

TCP Connection and WebSocket session time

In the first experiment, it is measured the time required for client and server to establish a TCP connection and WebSocket session.

WebSocket session lasts 3.7 times longer than establishing a TCP connection.

TCP Connection and WebSocket session time

- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.

TCP Connection and WebSocket session time

- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.

We have already seen this previously.

TCP Connection and WebSocket session time

- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.
- WebSocket sits on top of TCP and uses HTTP for the handshake.

TCP Connection and WebSocket session time

- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.
- WebSocket sits on top of TCP and uses HTTP for the handshake.

We have already seen this previously.

TCP Connection and WebSocket session time

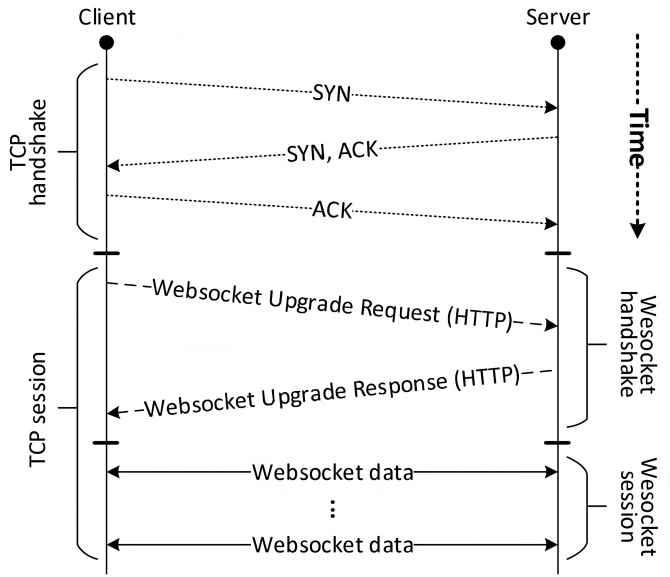
- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.
- WebSocket sits on top of TCP and uses HTTP for the handshake.
- Which means it first has to establish a TCP connection, allocate the resource for HTTP and then can establish a WebSocket handshake.

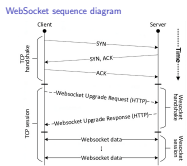
TCP Connection and WebSocket session time

- The reason for such slow performance of the WebSocket is the fact that the protocol is not a *transport protocol*.
- WebSocket sits on top of TCP and uses HTTP for the handshake.
- Which means it first has to establish a TCP connection, allocate the resource for HTTP and then can establish a WebSocket handshake.

We have already seen this previously.

WebSocket sequence diagram





WebSocket stay on top of TCP which means it will always add more overhead than the raw TCP Socket, but WebSocket is easier to use in a web environment.

Data transfer time after connection

- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.

Data transfer time after connection
- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.

Data transfer time after connection

- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.

Data transfer time after connection
- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.

Data transfer time after connection

- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.
- The Server echoes the same data back to the client.

Data transfer time after connection
- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.
- The Server echoes the same data back to the client.

Data transfer time after connection

- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.
- The Server echoes the same data back to the client.
- It is possible to define the relative time overhead O_T a WebSocket incurs over TCP as:

$$O_T = \frac{T_{WS} - T_{TCP}}{T_{TCP}} \cdot 100\% \qquad (4)$$

where:

T = time to transfer data

Data transfer time after connection

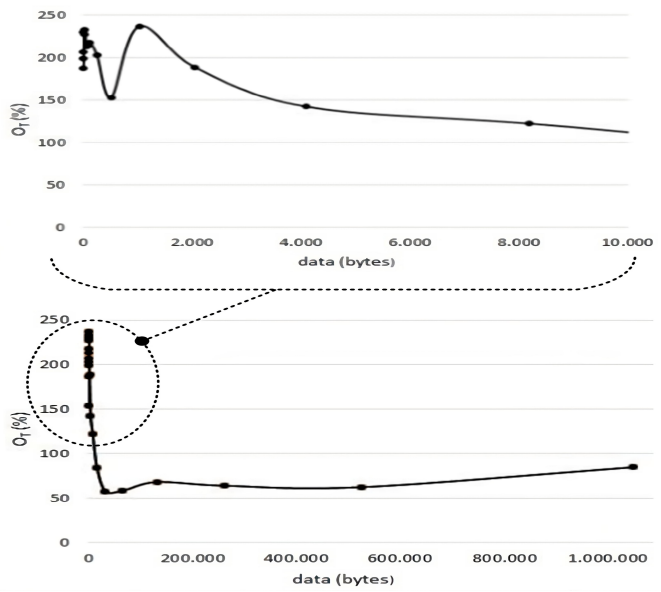
- In the second experiment, it is measured the data transfer time after the TCP connection and WebSocket session have been established.
- The Client generates a given amount of data and sends them to the server.
- The Server echoes the same data back to the client.
- It is possible to define the relative time overhead O_T a WebSocket incurs over TCP as:

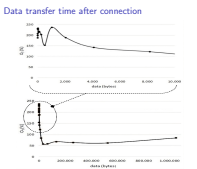
$$O_T = \frac{T_{WS} - T_{TCP}}{T_{TCP}} \cdot 100\% \qquad (4)$$

where:

T = time to transfer data

Data transfer time after connection





Data transfer time after connection

- The WebSocket performs slower than the TCP.

Data transfer time after connection

- The WebSocket performs slower than the TCP.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.
- The overhead of WebSocket fluctuates between 150% and 250% for small messages.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.
- The overhead of WebSocket fluctuates between 150% and 250% for small messages.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.
- The overhead of WebSocket fluctuates between 150% and 250% for small messages.
- For biggest messages, the overhead is more stable at 60-70%.

Data transfer time after connection

- The WebSocket performs slower than the TCP.
- The performance drop is more significant for small messages.
- The overhead of WebSocket fluctuates between 150% and 250% for small messages.
- For biggest messages, the overhead is more stable at 60-70%.

Why WebSocket performs slower?

- WebSocket sits on top of TCP.

Why WebSocket performs slower?
- WebSocket sits on top of TCP.

Why WebSocket performs slower?

- WebSocket sits on top of TCP.
- WebSocket uses event-driven callback-based API to deliver data, which requires additional application data handling.

Why WebSocket performs slower?

- WebSocket sits on top of TCP.
- WebSocket uses event-driven callback-based API to deliver data, which requires additional application data handling.

Why WebSocket performs slower?

- WebSocket sits on top of TCP.
- WebSocket uses event-driven callback-based API to deliver data, which requires additional application data handling.
- In 2014, the WebSocket protocol emerged just a few years before with little production systems deployed so far. While TCP has been used in production for decades are has highly optimized libraries.

Why WebSocket performs slower?

- WebSocket sits on top of TCP.
- WebSocket uses event-driven callback-based API to deliver data, which requires additional application data handling.
- In 2014, the WebSocket protocol emerged just a few years before with little production systems deployed so far. While TCP has been used in production for decades are has highly optimized libraries.

References

- Alexey Melnikov and Ian Fette, *The WebSocket Protocol*, RFC 6455, December 2011.
- D. Skvorc, M. Horvat, and S. Srbljic, *Performance evaluation of websocket protocol for implementation of full-duplex web streams*, 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014, pp. 1003–1008.
- Lijing Zhang and Xiaoxiao Shen, *Research and development of real-time monitoring system based on websocket technology*, Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), 2013, pp. 1955–1958.

References

Alexey Melnikov and Ian Fette, *The WebSocket Protocol*, RFC 6455, December 2011.

D. Skvorc, M. Horvat, and S. Srbljic, *Performance evaluation of websocket protocol for implementation of full-duplex web streams*, 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014, pp. 1003–1008.

Lijing Zhang and Xiaoxiao Shen, *Research and development of real-time monitoring system based on websocket technology*, Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC), 2013, pp. 1955–1958.