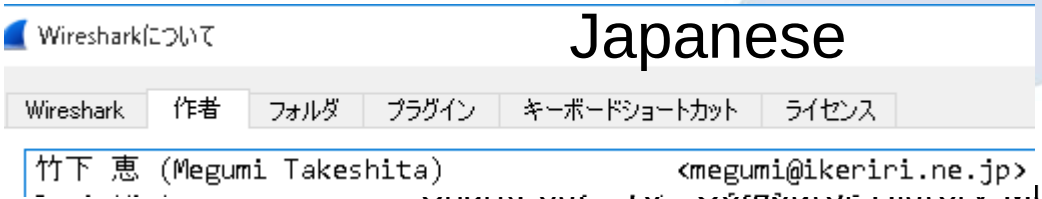
SharkFest '16

Wireshark 2.0 Tips for HTTP 1/2 Analysis: Goodies about New Wireshark and Packet Analysis for HTTP

Megumi Takeshita
Packet Otaku | ikeriri network service co.,ltd

Megumi Takeshita, ikeriri network service a.k.a. packet otaku since first Sharkfest

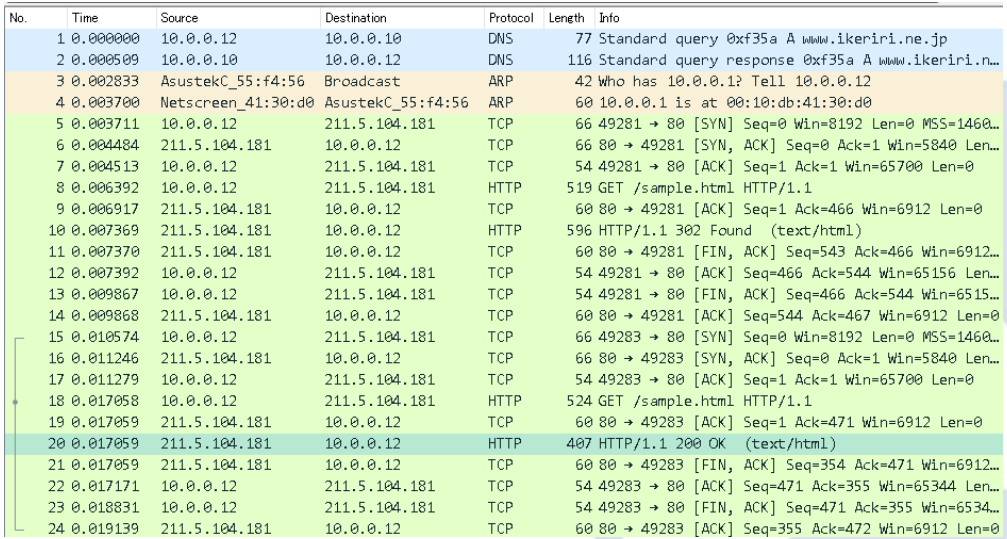
- Founder, ikeriri network service co.,ltd
- Wrote 10+ books of Wireshark and capturing and network analysis.
- Reseller of Riverbed Technology (former CACE technologies) and Metageek, Dualcomm etc. in Japan
- Attending all Sharkfest 9 times
- and translator of QT Wireshark into Japanese



21 Wireshark 2.0 Tips for HTTP 1/2 Analysis: Goodies about New Wireshark and Packet Analysis for HTTP

•This session contains TIPS and TRICKs
for HTTP 1 / 2 using Wireshark 2.0,
and also includes HTTP and HTTP2 analysis
for packet analysis beginners.
•Limited English skills,
so please ask me if you have some question.

sample trace files in the session you can download
Download: http://www.ikeriri.ne.jp/wireshark/traces/

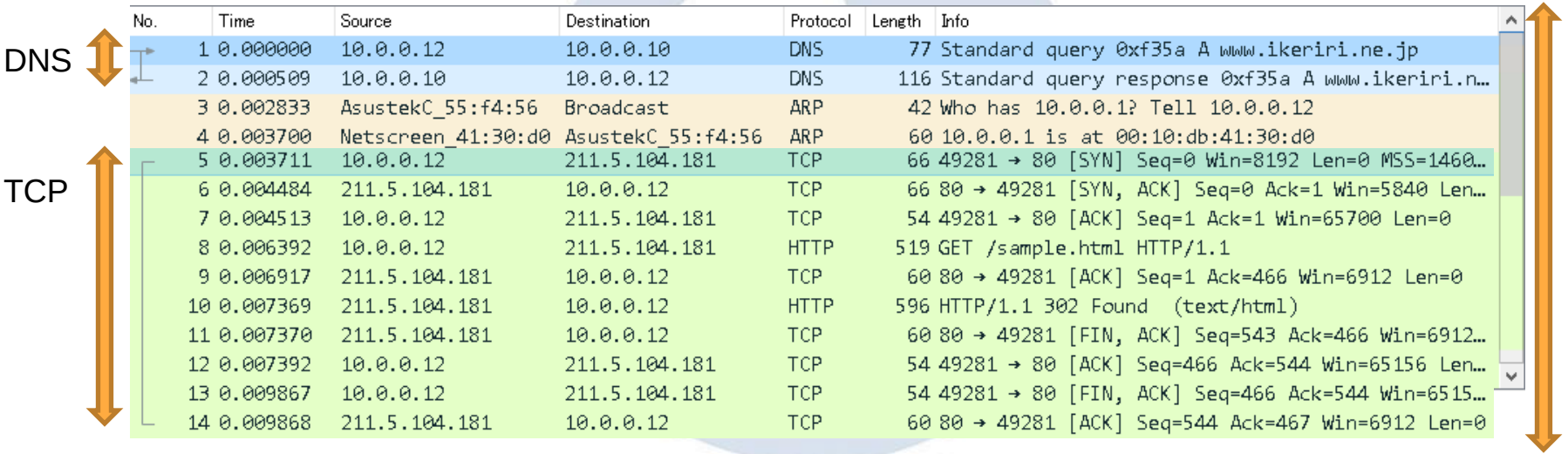


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.12	10.0.0.10	DNS	77	Standard query 0xf35a A www.ikeriri.ne.jp
2	0.000509	10.0.0.10	10.0.0.12	DNS	116	Standard query response 0xf35a A www.ikeriri.n…
3	0.002833	AsustekC_55:f4:56	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.12
4	0.003700	Netscreen_41:30:d0	AsustekC_55:f4:56	ARP	60	10.0.0.1 is at 00:10:db:41:30:d0
5	0.003711	10.0.0.12	211.5.104.181	TCP	66	49281 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460…
6	0.004484	211.5.104.181	10.0.0.12	TCP	66	80 → 49281 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len…
7	0.004513	10.0.0.12	211.5.104.181	TCP	54	49281 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
8	0.006392	10.0.0.12	211.5.104.181	HTTP	519	GET /sample.html HTTP/1.1
9	0.006917	211.5.104.181	10.0.0.12	TCP	60	80 → 49281 [ACK] Seq=1 Ack=466 Win=6912 Len=0
10	0.007369	211.5.104.181	10.0.0.12	HTTP	596	HTTP/1.1 302 Found (text/html)
11	0.007370	211.5.104.181	10.0.0.12	TCP	60	80 → 49281 [FIN, ACK] Seq=543 Ack=466 Win=6912…
12	0.007392	10.0.0.12	211.5.104.181	TCP	54	49281 → 80 [ACK] Seq=466 Ack=544 Win=65156 Len…
13	0.009867	10.0.0.12	211.5.104.181	TCP	54	49281 → 80 [FIN, ACK] Seq=466 Ack=544 Win=6515…
14	0.009868	211.5.104.181	10.0.0.12	TCP	60	80 → 49281 [ACK] Seq=544 Ack=467 Win=6912 Len=0
15	0.010574	10.0.0.12	211.5.104.181	TCP	66	49283 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460…
16	0.011246	211.5.104.181	10.0.0.12	TCP	66	80 → 49283 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len…
17	0.011279	10.0.0.12	211.5.104.181	TCP	54	49283 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
18	0.017058	10.0.0.12	211.5.104.181	HTTP	524	GET /sample.html HTTP/1.1
19	0.017059	211.5.104.181	10.0.0.12	TCP	60	80 → 49283 [ACK] Seq=1 Ack=471 Win=6912 Len=0
20	0.017059	211.5.104.181	10.0.0.12	HTTP	407	HTTP/1.1 200 OK (text/html)
21	0.017059	211.5.104.181	10.0.0.12	TCP	60	80 → 49283 [FIN, ACK] Seq=354 Ack=471 Win=6912…
22	0.017171	10.0.0.12	211.5.104.181	TCP	54	49283 → 80 [ACK] Seq=471 Ack=355 Win=65344 Len…
23	0.018831	10.0.0.12	211.5.104.181	TCP	54	49283 → 80 [FIN, ACK] Seq=471 Ack=355 Win=6534…
24	0.019139	211.5.104.181	10.0.0.12	TCP	60	80 → 49283 [ACK] Seq=355 Ack=472 Win=6912 Len=0

At first, open the homepage.pcap
The trace file is just open the simple website
http://www.ikeriri.ne.jp/sample.html

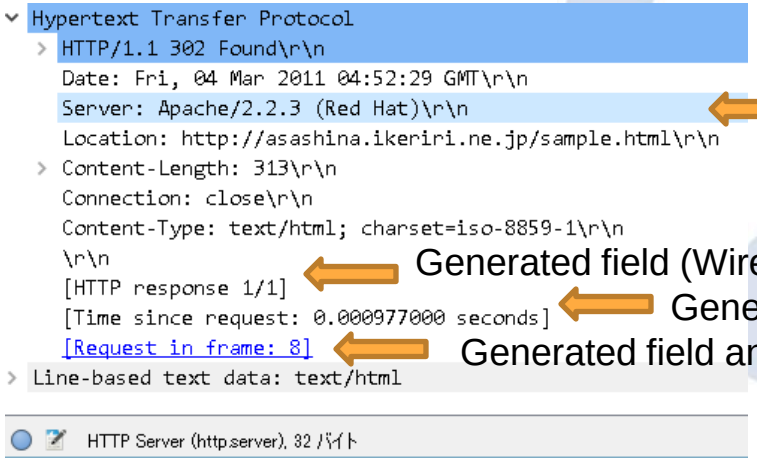
TIPS #1 first, check arrows and colors of the frame and the intelligent scroll bar

- New Wireshark tell you traffic with arrow and color of the scroll bar. It tells us the traffic



TIPS #2 Generated fields and links tell us the important information

- There are two kinds of fields in Wireshark header,
  the actual field like Web Server (http.server) in HTTP header,
  the generated field (easily to find [generated field name]) that Wireshark created for understanding the packet.
  Some generated fields have a link to jump the corresponding frame.



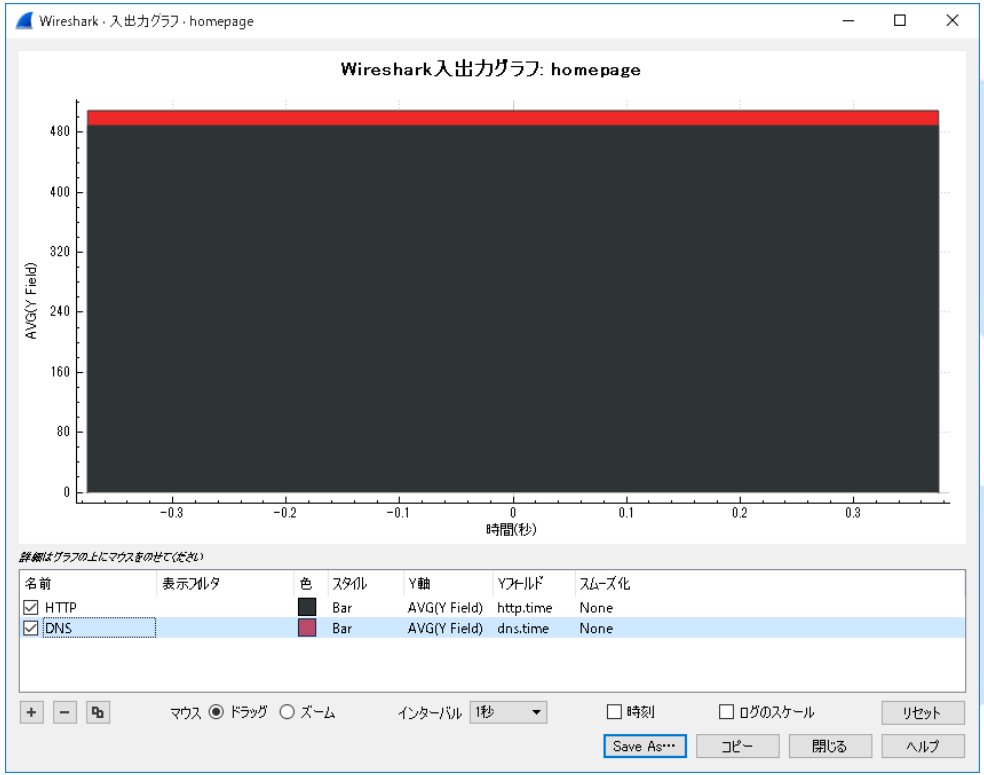
Actual header (HTTP Server field)

Generated field (Wireshark counts response number (http.response_number))

Generated field (Wireshark calculate http response time (http.time))

Generated field and Link (Wireshark set link to the http request frame)

We can use generated fields as actual fields in I/O graph, display filter, etc.



Generated Fields can be used as the index of the I/O Graph, Display filter string and the other of Wireshark.

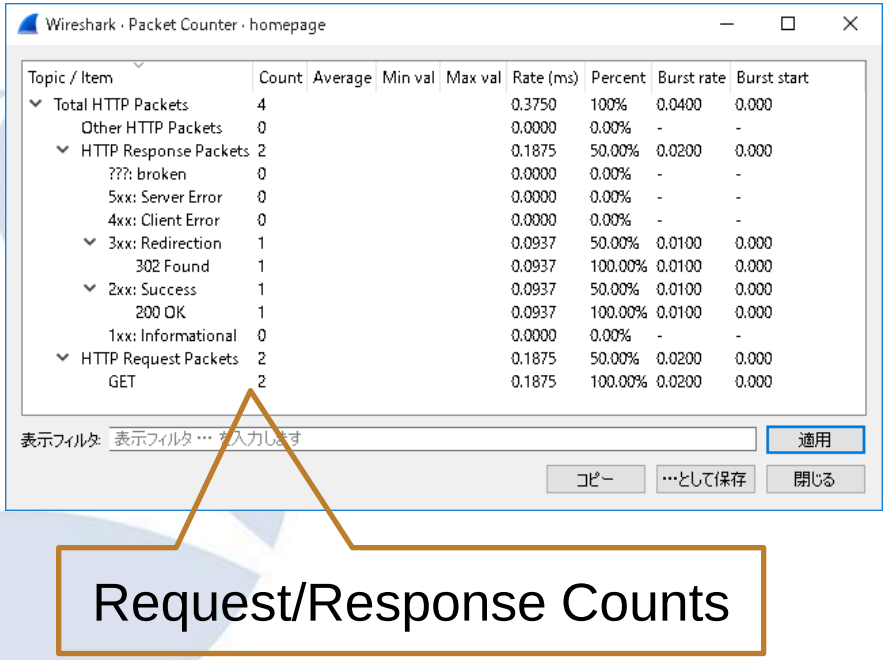
For example, there are two generated fields, http.time (Wireshark calculates time between HTTP request and response) and dns.time (between DNS query and response)

so you can easily compare the web speed (blue bar) versus dns speed (red bar)

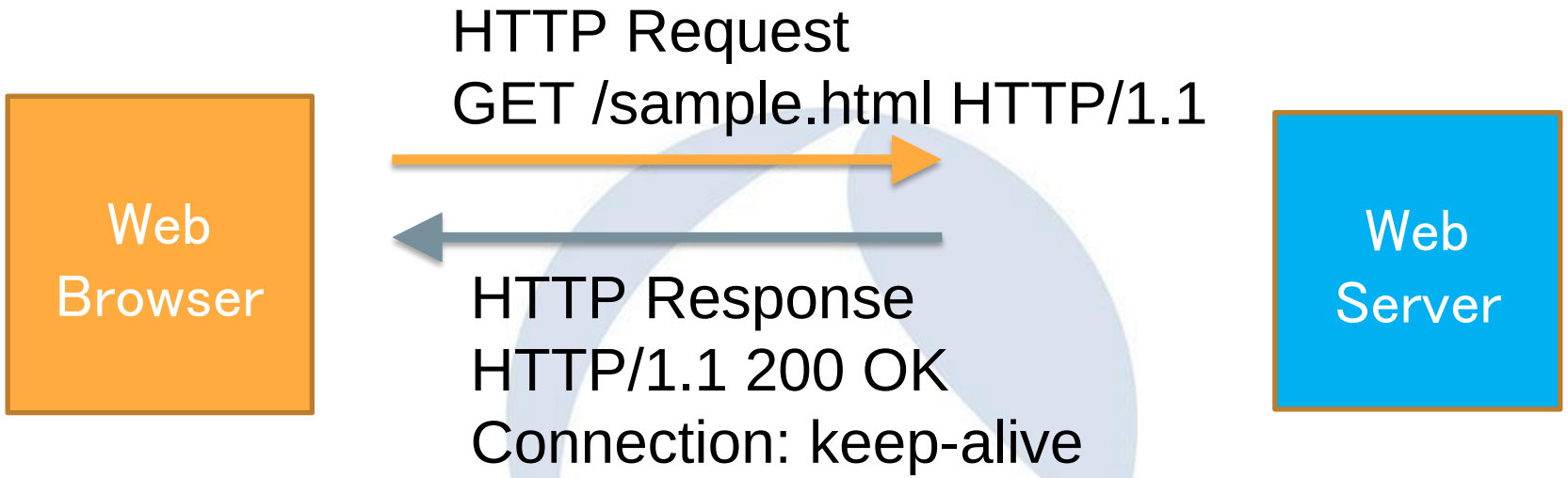
TIPS #3 HTTP statistics tell us the scale and the TURN(LOOP) of the whole Web traffic.

- HTTP statistics contains important information of HTTP trends

- Packet counter shows HTTP packets by the request method and by the response code, and shows subtotal of the each method and code in details. so we can grab the scale and the TURNs of the certain web traffic,

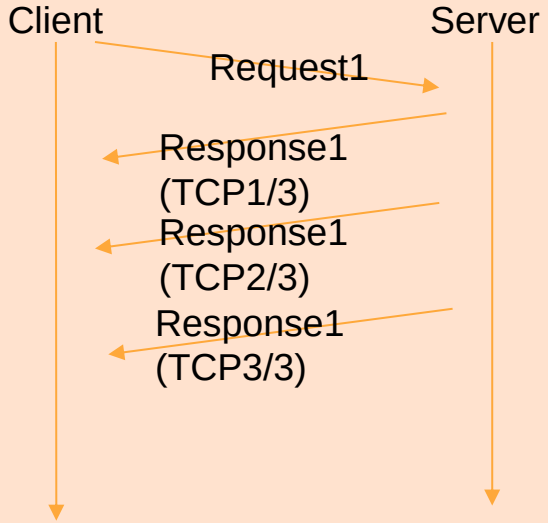
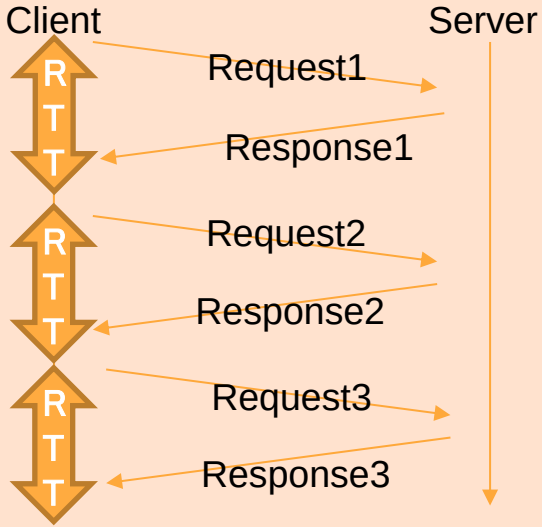


HTTP/1.1　TURN=Request Response Loop

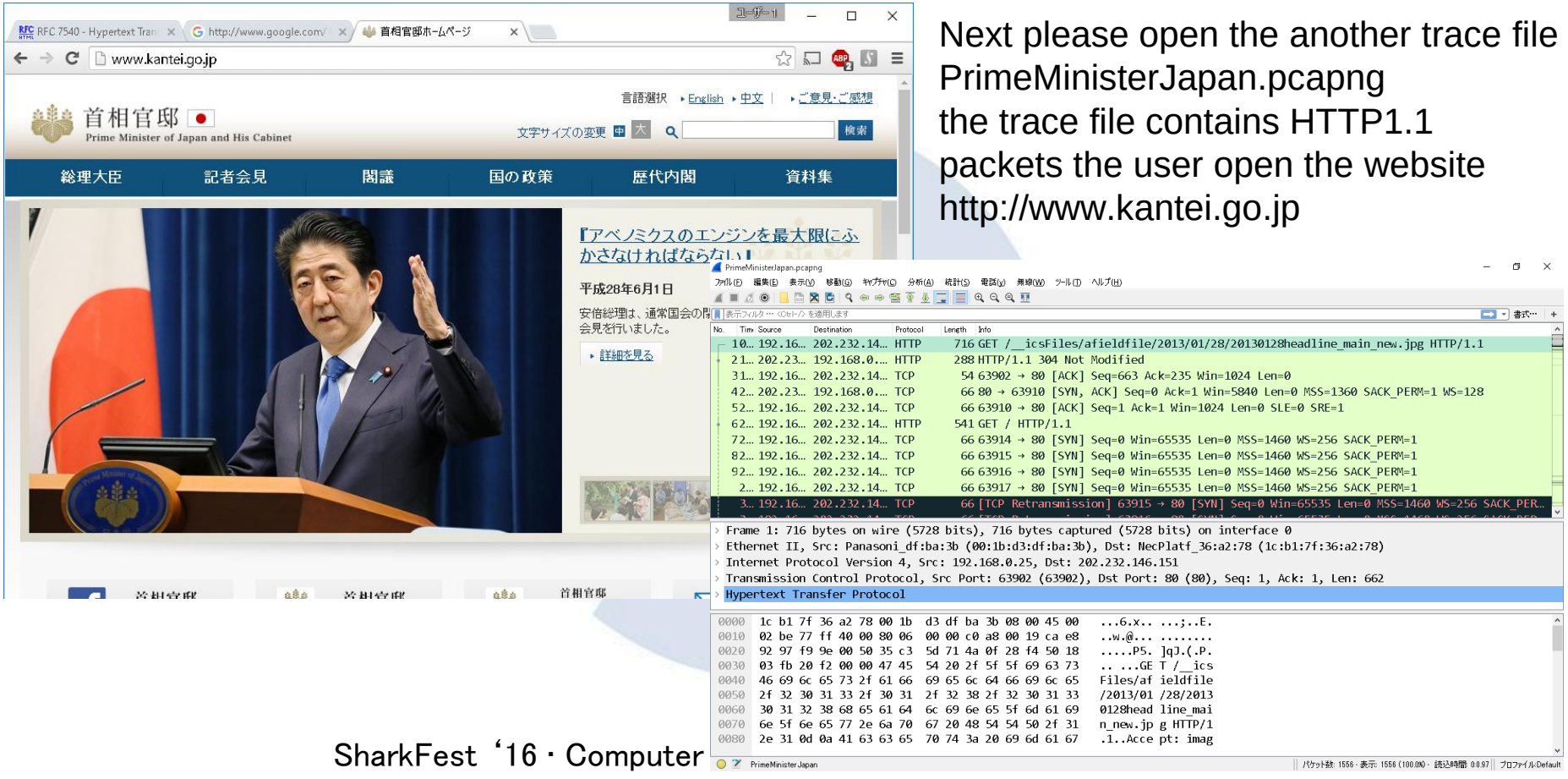


- HTTP/1.1 has a Connection header, Server response with Connection: keep-alive so you can re-use the same connection.

Bulk transfer vs interactive access of webpage

	Bulk transfer webpage	Interactive access webpage
Bulk transfer runs faster than interactive design website. Just think of RTT x TURNs		
	Get one image map file using HTTP 1.1 (1 request	Get some Small icon file (icon1.jpg, icon2.jpg) using HTTP 1.1

Test trace file: PrimeMinisterJapan.pcapng

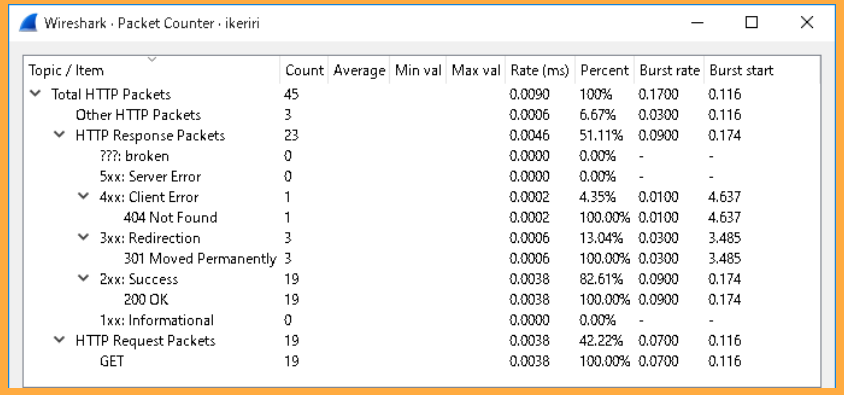


Next please open the another trace file PrimeMinisterJapan.pcapng
the trace file contains HTTP1.1 packets the user open the website http://www.kantei.go.jp

Compare Counts and grab the efficiency and speed and scale of webpage

ikeriri.pcapng
http://www.ikeriri.ne.jp/

Less Requests
Less Response
Less TURNs * RTT
Simple homepage



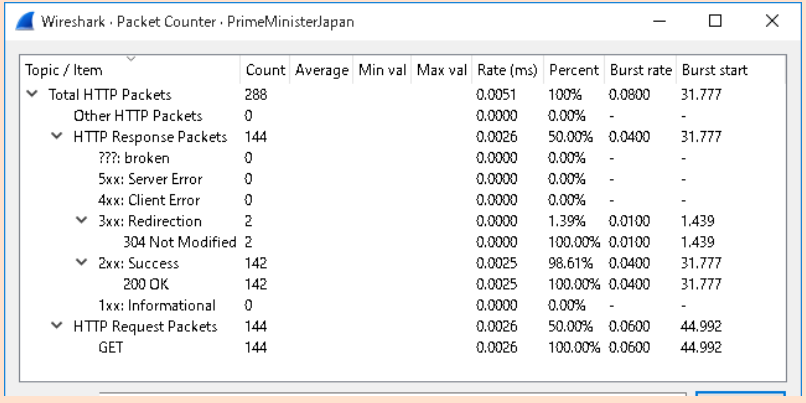
Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Total HTTP Packets	45				0.0090	100%	0.1700	0.116
Other HTTP Packets	3				0.0006	6.67%	0.0300	0.116
HTTP Response Packets	23				0.0046	51.11%	0.0900	0.174
???: broken	0				0.0000	0.00%	-	-
5xx: Server Error	0				0.0000	0.00%	-	-
4xx: Client Error	1				0.0002	4.35%	0.0100	4.637
404 Not Found	1				0.0002	100.00%	0.0100	4.637
3xx: Redirection	3				0.0006	13.04%	0.0300	3.485
301 Moved Permanently	3				0.0006	100.00%	0.0300	3.485
2xx: Success	19				0.0038	82.61%	0.0900	0.174
200 OK	19				0.0038	100.00%	0.0900	0.174
1xx: Informational	0				0.0000	0.00%	-	-
HTTP Request Packets	19				0.0038	42.22%	0.0700	0.116
GET	19				0.0038	100.00%	0.0700	0.116

19 Request GET

23 Response
19 Response 200 OK
1 404 Client Error
3 2xx Redirection

PrimeMinisterJapan.pcapng
http://www.kantei.go.jp/

More Requests
More Response
More TURNs * RTT
Complicated homepage



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Total HTTP Packets	288				0.0051	100%	0.0800	31.777
Other HTTP Packets	0				0.0000	0.00%	-	-
HTTP Response Packets	144				0.0026	50.00%	0.0400	31.777
???: broken	0				0.0000	0.00%	-	-
5xx: Server Error	0				0.0000	0.00%	-	-
4xx: Client Error	0				0.0000	0.00%	-	-
3xx: Redirection	2				0.0000	1.39%	0.0100	1.439
304 Not Modified	2				0.0000	100.00%	0.0100	1.439
2xx: Success	142				0.0025	98.61%	0.0400	31.777
200 OK	142				0.0025	100.00%	0.0400	31.777
1xx: Informational	0				0.0000	0.00%	-	-
HTTP Request Packets	144				0.0026	50.00%	0.0600	44.992
GET	144				0.0026	100.00%	0.0600	44.992

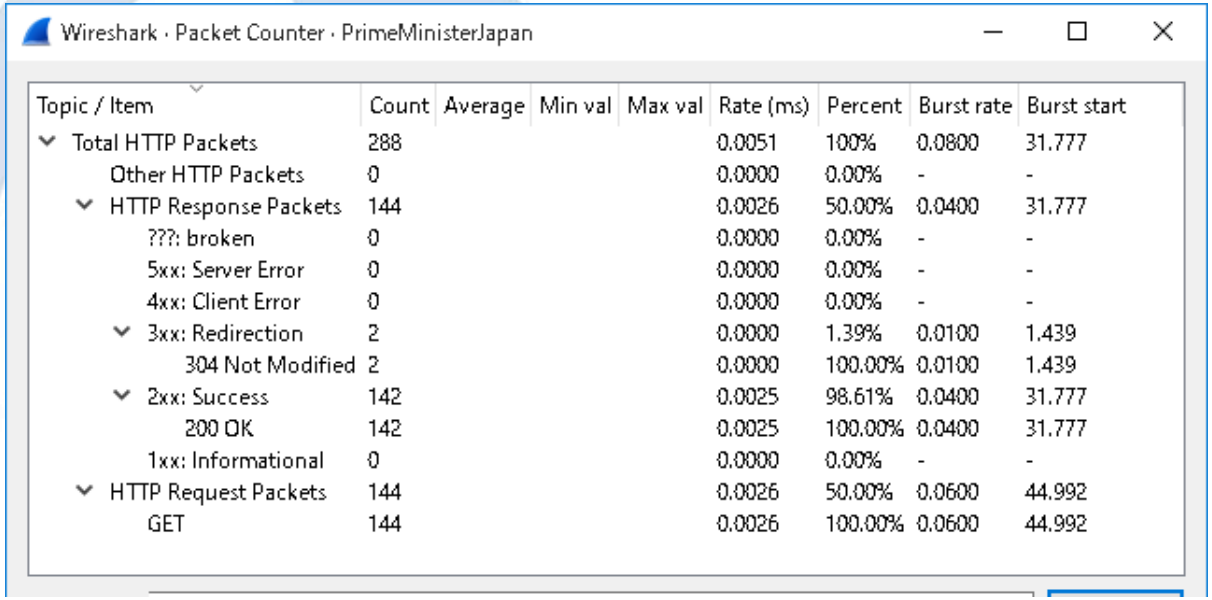
144 Request GET

144 Response
2 Redirection
142 Response 200 OK

Sort the HTTP counts
by Percent tells us the ratio of request/response,
by Rate(ms) tells us the slow point (3xx redirect),
by Burst rate tells us the congestion point of traffic

Burst = the maximum number of packets sent per interval of time

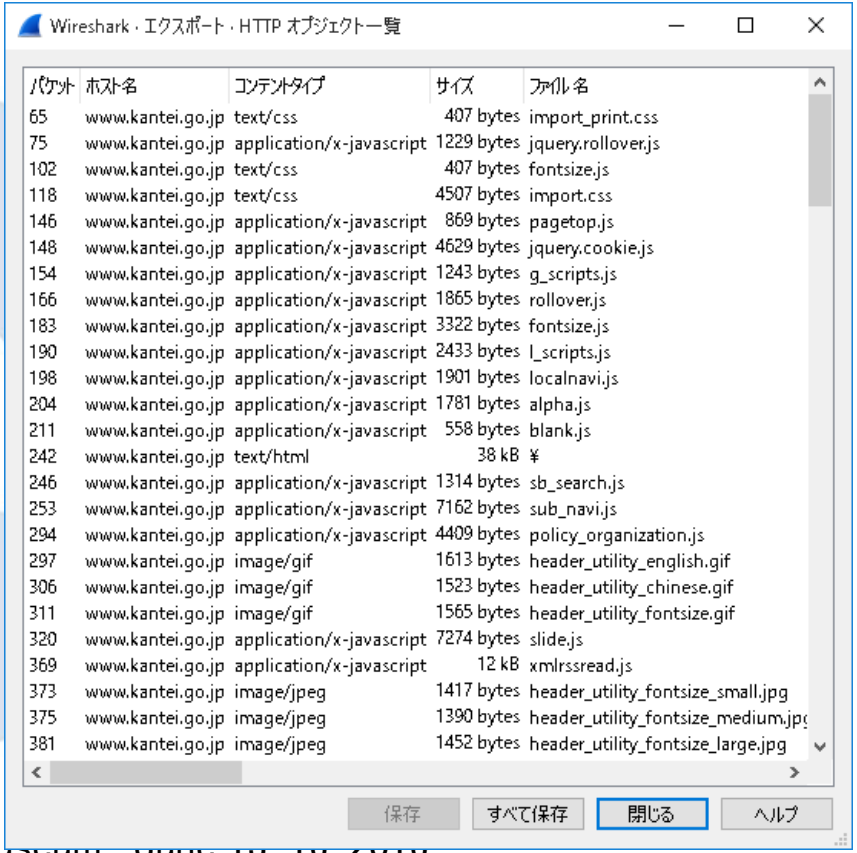
Burst start = the time when the maximum number of packets sent occurred



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Total HTTP Packets	288				0.0051	100%	0.0800	31.777
Other HTTP Packets	0				0.0000	0.00%	-	-
HTTP Response Packets	144				0.0026	50.00%	0.0400	31.777
???: broken	0				0.0000	0.00%	-	-
5xx: Server Error	0				0.0000	0.00%	-	-
4xx: Client Error	0				0.0000	0.00%	-	-
3xx: Redirection	2				0.0000	1.39%	0.0100	1.439
304 Not Modified	2				0.0000	100.00%	0.0100	1.439
2xx: Success	142				0.0025	98.61%	0.0400	31.777
200 OK	142				0.0025	100.00%	0.0400	31.777
1xx: Informational	0				0.0000	0.00%	-	-
HTTP Request Packets	144				0.0026	50.00%	0.0600	44.992
GET	144				0.0026	100.00%	0.0600	44.992

TIPS #4 Export Objects function is useful in retrieving Web contents from packet (not from cache).

- If you want to get the original web content from the packet, you use “export Objects > HTTP” from file menu.
- Wireshark made lists of web contents with packet number, host name, content type, size and file name.
- You can save them all or each.



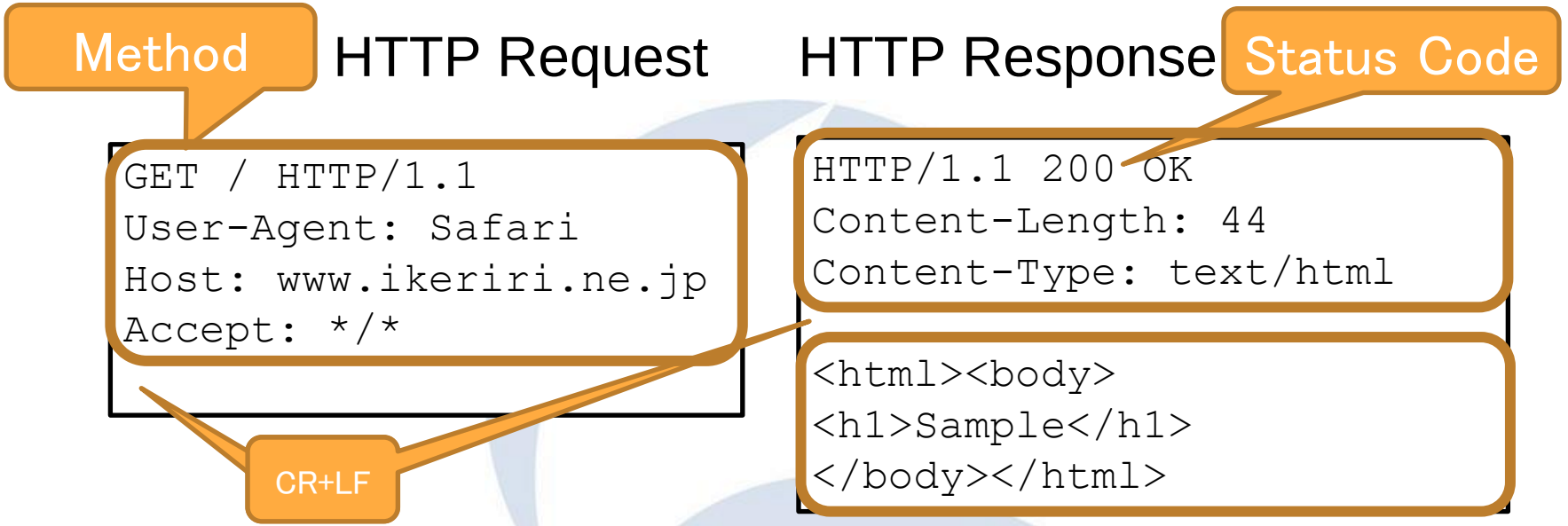
Sort the objects by the size, you know which contents need the transfer time.

- “export Objects > HTTP” from file menu and save all to the temporal folder.
- Sort the objects in explorer by size, and check which one is the biggest.
- Some HTTP uses compression mechanism so sometimes actual traffic is smaller.

TIPS #5 to understand HTTP request and response loop, use the “follow TCP stream”

- “Follow TCP Stream” function sorts TCP stream from both client and server side, so we can look a series of HTTP communication at a glance
- Follow TCP Stream is just a simple.
  select TCP packet (any frame
  you want to look into the socket)
  and right clik then select
  “Follow TCP Stream”

HTTP/1.1 message format

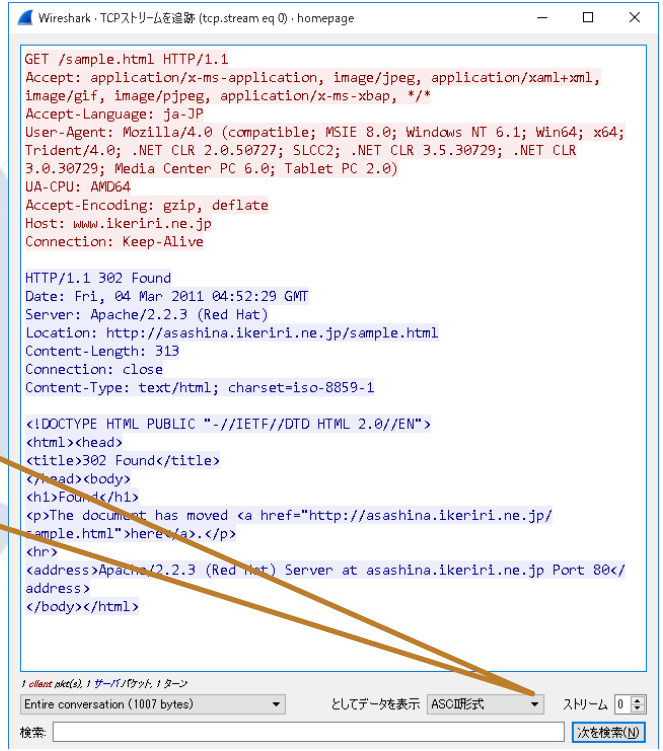


CR/LF separates between HTTP header and body (two CR/LF appears.)

if you failed to select the target stream, no problem, press up and down to select the stream forward and backward in the follow TCP/UDP/SSL screen.

input stream number or press up or down arrow to serlect the stream

Wireshark set the number of stream as generated field, we can handle the stream as tcp.stream, udp.stream, ssl.stream



How to grab Follow TCP Stream

Select HTTP headers between beginning and blank CR/LF

HTTP response usually has the body, select from fisrt blank to the end.

TIPS #6 memorize major request method

Method	Mean
CONNECT	Tunneling the other tcp connection over HTTP
DELETE	Deleting Objects (used by WebDAV)
GET	Give me the content with some order (?para1=val1&para2=val2&…) Not secure than POST because the URI and message are combined and recorded by Referrer header. Usually order message is small and not important.
HEAD	Give me only the HEADER of the content
OPTIONS	Check the method web server accepts.
POST	Sending information with body (secure) Message send as the body part of the request. Post message is secure and able to send much data
PUT	Uploading Objects (used by WebDAV)
TRACE	Check proxy sever by displaying request message, Used with “Max-Forwards” header

You can test the method by telnet client with port 80

```
C:¥WINDOWS¥system32¥cmd.exe - telnet
Microsoft Telnet クライアントへようこそ

エスケープ文字は 'CTRL+]' です

Microsoft Telnet> set localecho
ローカル エコー: オン
Microsoft Telnet> open www.ikeriri.ne.jp 80
```

```
HTTP/1.1 500 URL Rewrite Module Error.
Content-Type: text/html
Server: Microsoft-IIS/7.0
X-Powered-By: ASP.NET
Date: Mon, 13 Jun 2016 06:25:14 GMT
Connection: close
Content-Length: 1193

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Stri
html1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/htm
<title>500 -内部サーバー エラーです。</title>
<style type="text/css">
```

- You can test the methods by telnet with TCP port 80
- Recommend “set localecho” because your typing is directly sent and not displayed. And you cannot modify the typing.
- Blank line needs two Enter key pressing (CR/LF, and blank CR/LF)
- Example: OPTIONS * HTTP/1.0
- Sometimes web server is not accepted because security reason.

TIPS #7 memorize major response code

•Response code is important for the understanding the HTTP response
•Response codes are 3 digits and categorized by the hundred number digit, so you may memorize 5 types of the response.

Code	type	Description
1xx	Information	Temporal information for the client web browser
2xx	Success	Your request is OK. I send the response.
3xx	Redirection	The object was moved., I tell you new address with the location header, so please send the request again with new URI
4xx	Client error	Your error caused from web browser
5xx	Server error	My error caused from web server

Famous response code (1xx)

code	message	description
100	Continue	You can continue to send the rest of the request. (Large data request)
101	S w i t c h i n g Protocols	Web browser send request with “Upgrade:” header Then Web server answers OK to switch protocol. (Used by starting process of HTTP/2.0 connection)

Famous response code (2xx)

code	message	説 明
200	OK	OK, I send the response.
201	Created	I created the content. (send back to PUT method and used by WebDAV)
202	Accepted	Accepted, please continue the rest, (usually accepted PUT method)
203	Non Authoritative Information	I received your request, but I have no authority of the content. (Proxy server says)
204	No Content	I received your request, but there are no content.
205	Reset Content	Please reset the content (used with input form of web page)
206	Partial Content	I send the partial content for your partial get method

Famous response code (3xx)

Code	Message	Description
300	Multiple Content	The contents locates in many other places. Web server says the list of the locations.
301	Moved Permamently	The content was moved and please go to the new URI with Location header
302	Found	The URI is ok but please go to the new address with Location header.
304	Not Modified	The content is not modified so you don't have to get the content, you may use the cache of the web browser.
305	Use Proxy	Please use the specified proxy server.
307	Temporary Redirect	The content was located in another URI temporally, so please go to the new URI with Location header.

Famous response code (4xx)

Code	Message	Description
400	Bad Request	The request message from the web browser is not corrected or wrong, sometimes it occurs when the web server is attacked.
401	Unauthorized	You need the authentication with basic auth or digest auth. web browser show up the authentication screen.
403	Forbidden	Authentication was failed.
404	Not Found	The web page is not found.
405	Method Not Allowed	The method the client browser sent is not permitted.
407	Proxy Authentication Requred	You need proxy server authentication.
411	Length Required	You need to send request with Content-Length header
413	Request Entity Too Large	The size of request is exceeded. It occurs POST message is too large.
414	Request URI Too Long	The URI is too long to accept. It occurs the length of the GET request is oversize

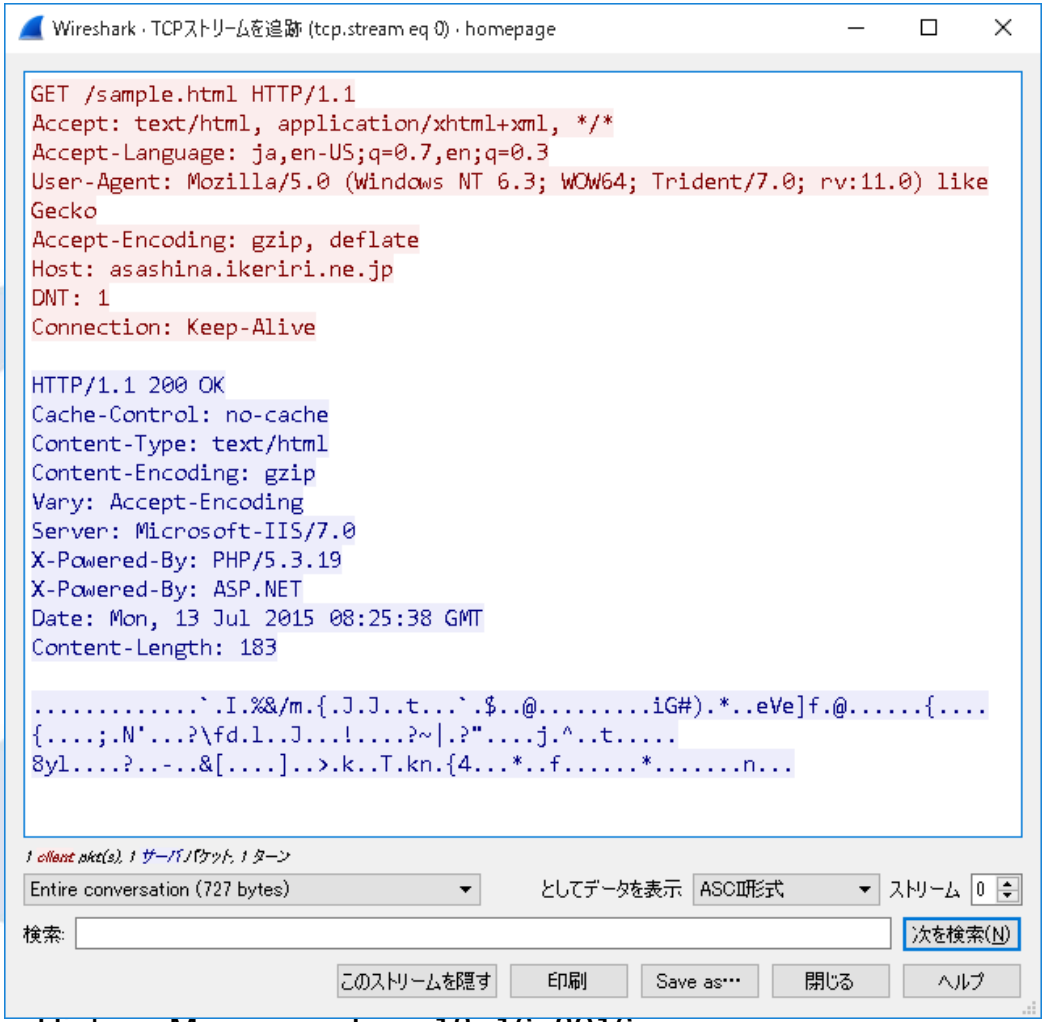
Famous response code (5xx)

Code	message	Description
500	Internal Server Error	Web server cannot respond to the client browser because server side problem (such as Java, CGI, PHP).
501	Not Implemented	The method client sent is not implemented.
502	Bad Gateway	Proxy server receives the error from the origin server or another relay proxy.
503	Service Unavailale	Web server could not process the request message, commonly it happens when web server is heavy load, tons of access or high stress.

Example1
homepage.pcapng
Access the website
http://www.ikeriri.ne.jp/
sample.html

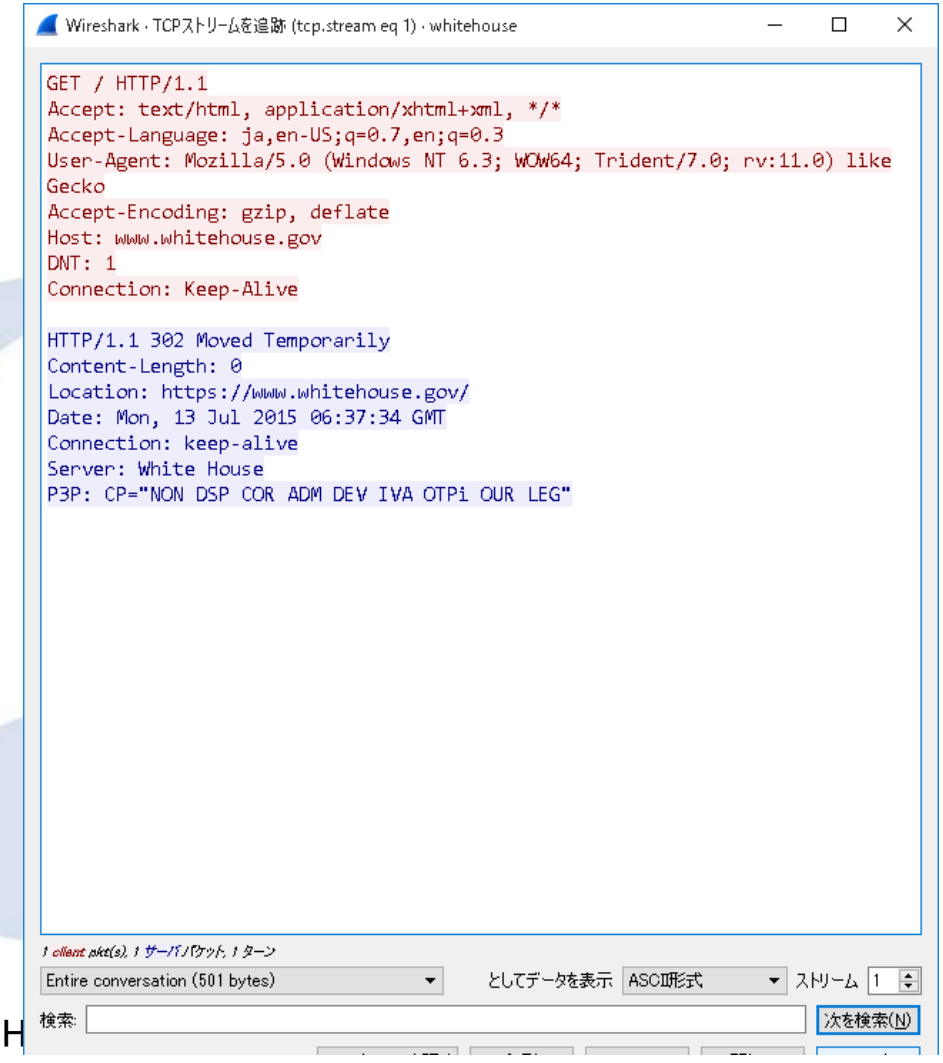
HTTP request
GET /sample.html HTTP/1.1

HTTP response
HTTP/1.1 200 OK



Example2
whitehouse.pcapng
Access the website
http://www.whitehouse.gov/

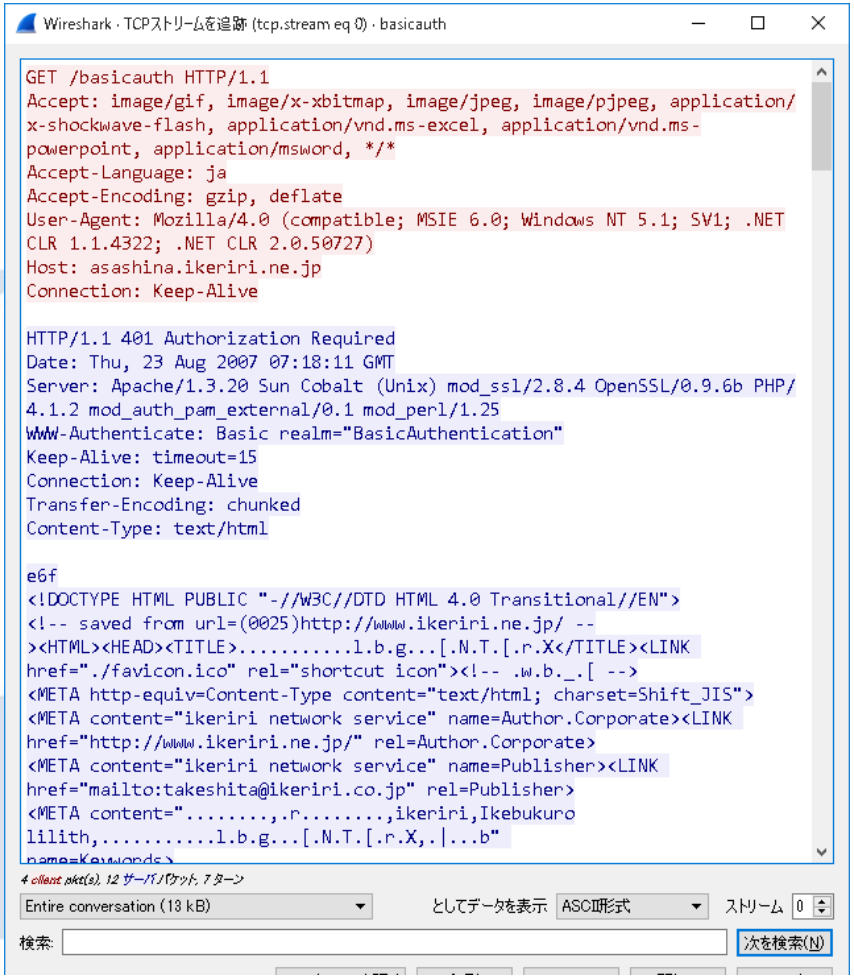
HTTP request
GET / HTTP/1.1
HTTP response
HTTP/1.1 302 Moved Temporally
to change HTTPS



Example3
basicauth.pcapng

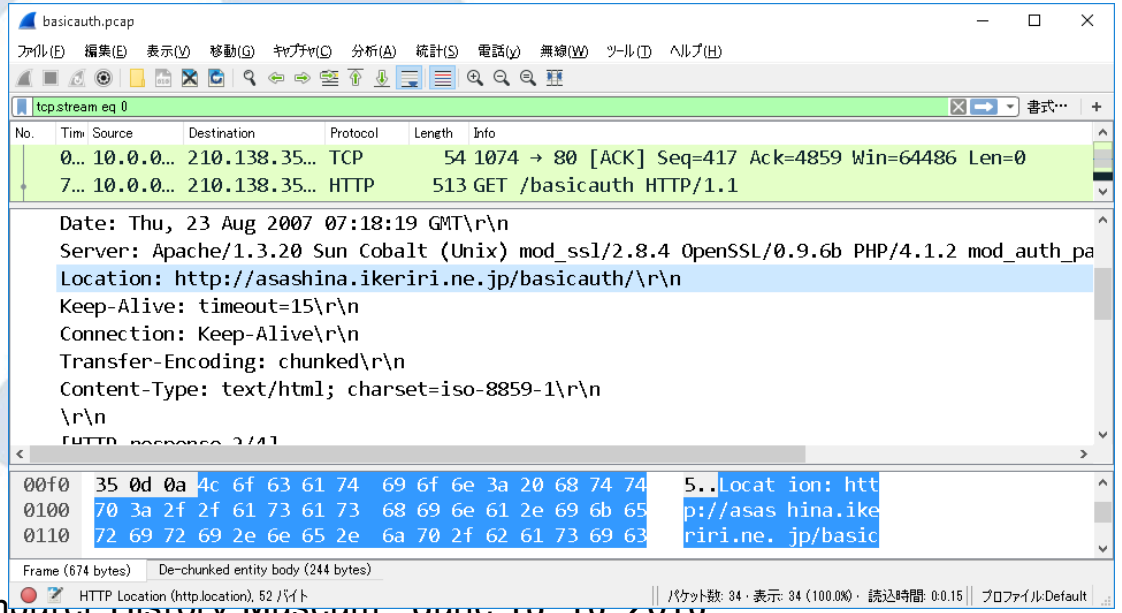
Access the website
http://www.ikeriri.ne.jp/basicauth/

HTTP request/response
(1)GET /basicauth HTTP/1.1
HTTP/1.1 401 Authorization Required
(2)GET /basicauth HTTP/1.1
HTTP/1.1 301 Moved Permanently
(3)GET /basicauth/ HTTP/1.1
HTTP/1.1 200 OK



TIPS #8 tons of HTTP headers so Wireshark helps us

- There are tons of HTTP headers so you cannot memorize them.
- Wireshark helps us understanding HTTP header.
- Select one of HTTP header and look the status bar
- Explanation of the header and display filter will be displayed.
- Some HTTP header
  is used only in request
  (request header)
- Some only in response
  (response header)
- Some is used for end-to-end
  (browser to server)
- Some is used for hop-by-hop
  (proxy to proxy and so on)



Easy to memorize the header symmetric ways

HTTP Request	HTTP Response
User-Agent Web browser information http.user_agent	Server Web server information http.server
Cookie Send the information http.cookie	Set-Cookie Set the information http.set_cookie
Referrer The past URI http.referer	Location The new URI http.location

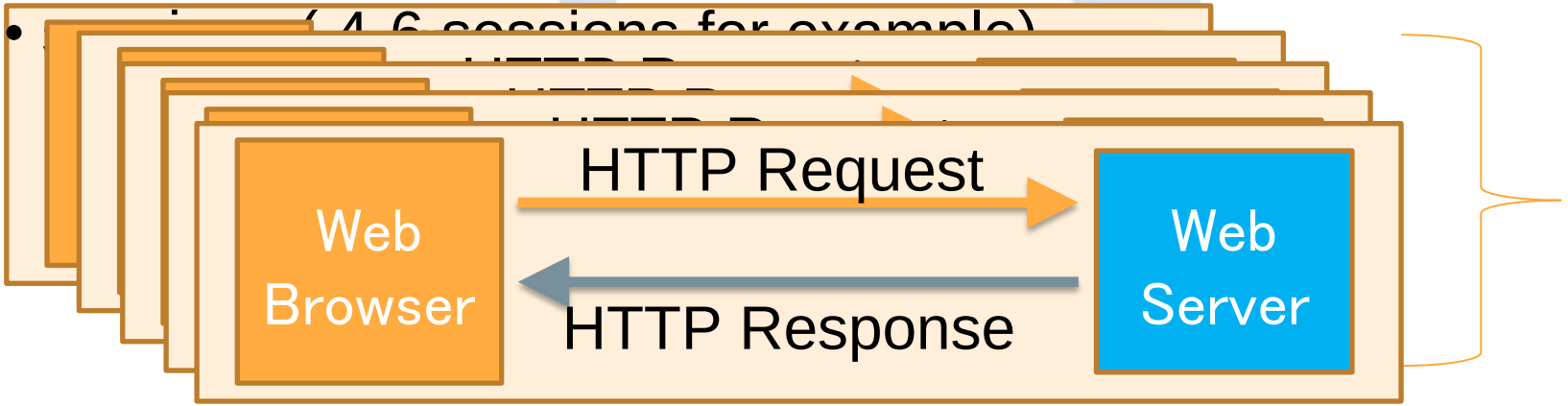
TIPS #9 HTTP/2.0 has already come

•HTTP/1.0 is just designed for simple HTML text and a few graphic in 1990's, HTTP/1.1 added connection features.

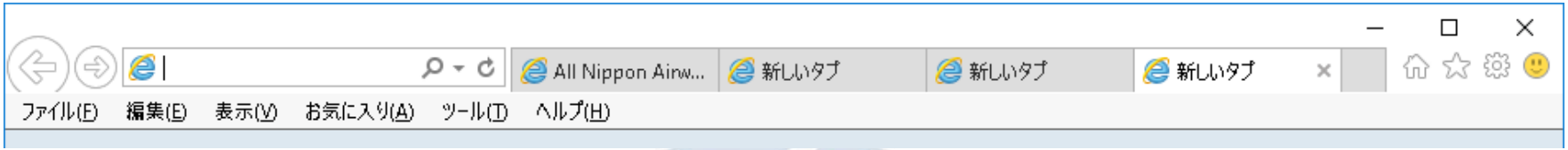
•Now HTTP is so popular, sending rich data and the basis of many services.

HTTP's Request-response based connection is simple but difficult to speed up, so we have to multiplex many TCP sessions = HTTP

•

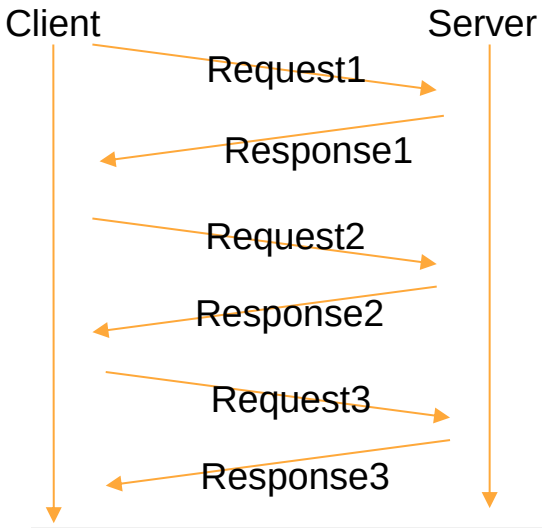


Test downloading 4-6 files at same time using HTTP



- Please click 4-6 files download link using your browser such as IE, Chrome, or Safari.
  (for example http://www.ikeriri.ne.jp/download/wireshark/developer-guide-a4.pdf)
- Web browser stop downloading when 4-6 sessions at the same time.
- Please Check sessions

HTTP/1.1 is difficult to speed up



- HTTP request have to send after previous response has been received.
- Please input display filter in Wireshark “http.next_request_in” (Next request in frame in HTTP request)
- HTTP request is always waiting in one connection. (head line blocking)

http.next_request_in

No.	Tim	Source	Destination	Protocol	Length	Info
10…	192.16…	202.232.14…	HTTP	716	GET /__icsFiles/afieldfile/2013/01/28/20130128
21…	202.23…	192.168.0.…	HTTP	288	HTTP/1.1 304 Not Modified
62…	192.16…	202.232.14…	HTTP	541	GET / HTTP/1.1
5…	192.16…	202.232.14…	HTTP	574	GET /jp/n2-common/css/import.css HTTP/1.1
5…	192.16…	202.232.14…	HTTP	580	GET /jp/n2-common/css/import_print.css HTTP/1.1
5…	192.16…	202.232.14…	HTTP	607	GET /jp/n2-common/js/jquery-1.7.1.min.js HTTP/1

HTTP/1.1 is text based, not efficient protocol

- HTTP is text-based application protocol, easy to read, but not efficient, ambiguous, and redundant
- HTTP messages are clear texts so they uses more data and CPU power for dissection.
- Many connection is separated by each other TCP connection, they work their own TCP rules without HTTP.

```
GET / HTTP/1.1
User-Agent: Safari
Host: www.ikeriri.ne.jp
Accept: */*
```

CR+LF

```
HTTP/1.1 200 OK
Content-Length: 44
Content-Type: text/html
```

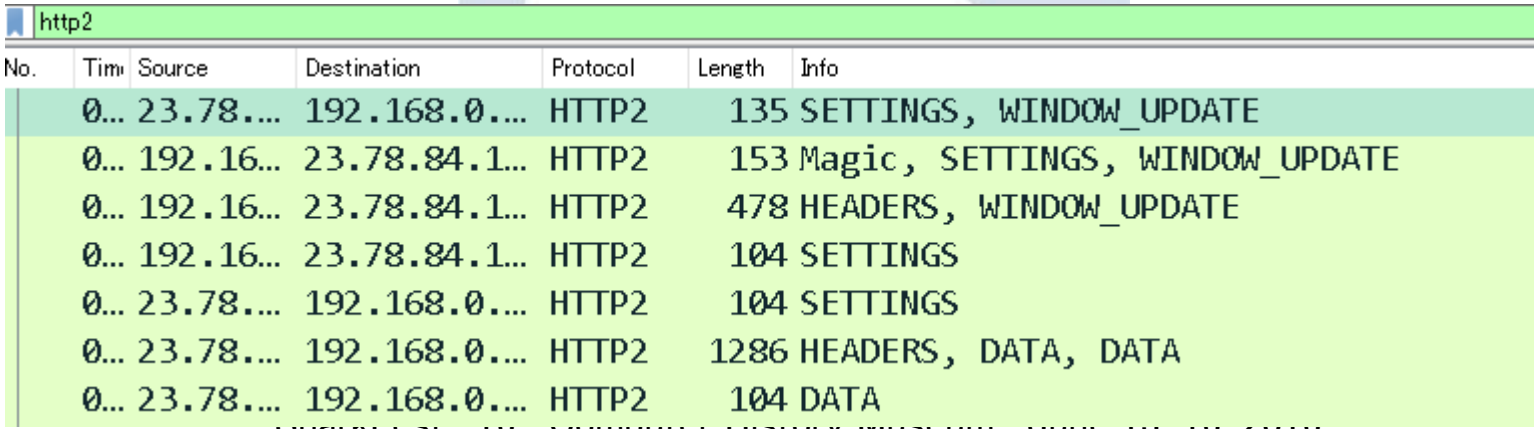
```
<html><body>
<h1>Sample</h1>
</body></html>
```

HTTP/1.1 -> AJAX -> SPDY -> HTTP/2.0

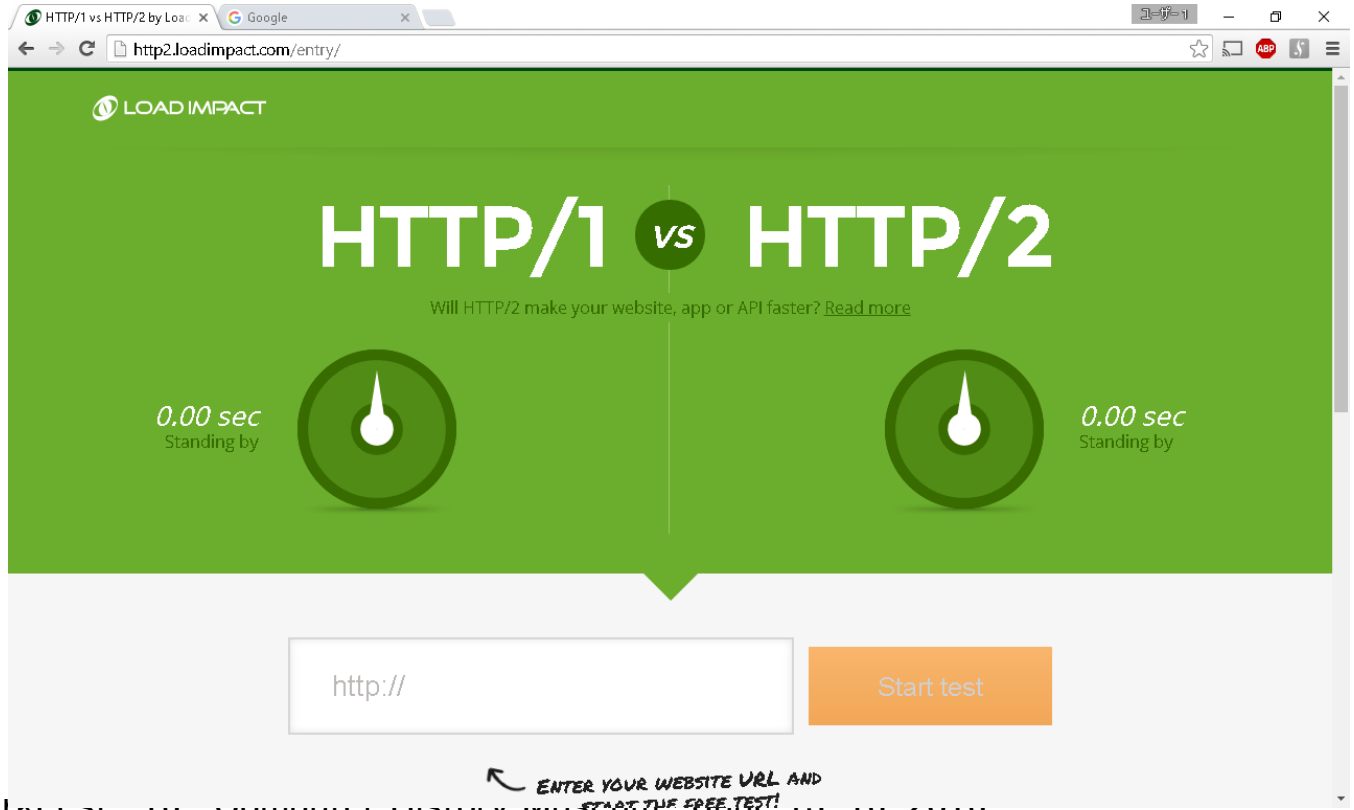
- AJAX（ Asynchronous JavaScript + XML ）is one of good ways to speed up HTTP. AJAX preload another contents using JavaScript, and enrich user experiences.
- Google creates SPDY that extends HTTP/1.1 (SPDY uses same method, response code, header and message of HTTP/1.1)
- SPDY uses binary frame such as lower layer frame, and once some header appears, next time SPDY uses index instead of the header itself, SPDY uses table based code (table-based header compression)
- SPDY uses stream mechanism for multiplexing, there are many HTTP communications in one TCP connection
- SPDY is evolved and known as HTTP/2.0 (RFC7540)

HTTP/2.0 has already come

- In Windows10 age, major Web browser and websites such as google services including Google Map and Gmail, Facebook, Twitter, Yahoo and web services are ready.
- HTTP/2.0 uses with TLS and all traffic is encrypted.
- Please run the Chrome, Microsoft Edge, Safari and capture major webpage, HTTP2 has already been here.

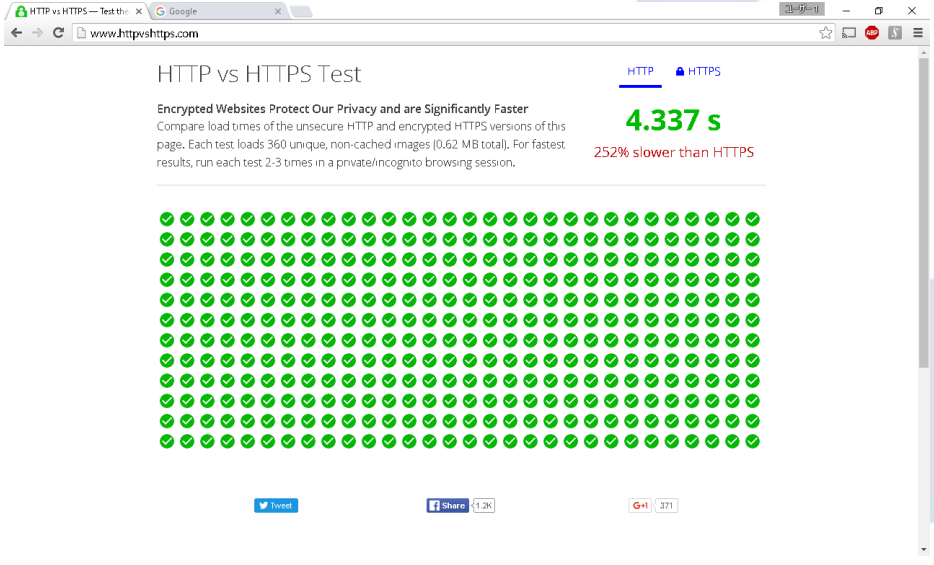


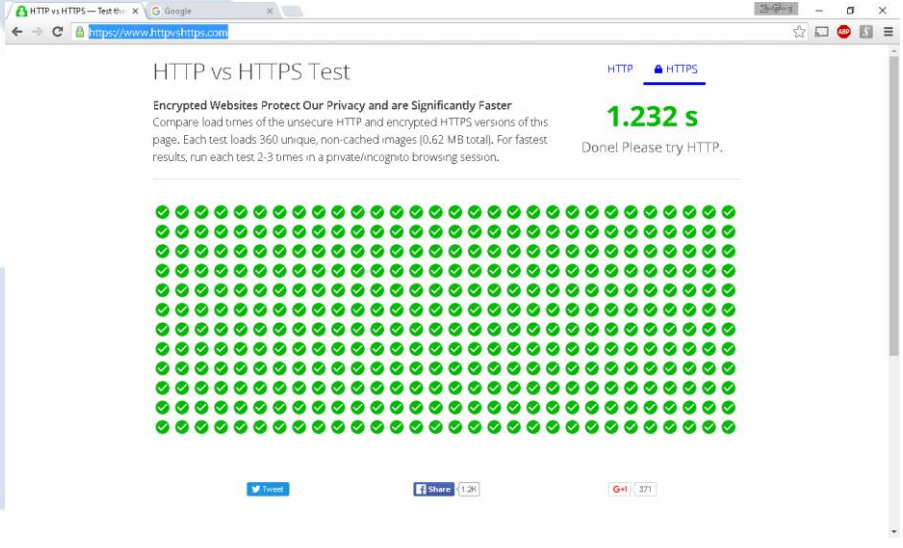
Please open the test website HTTP/2.0 vs HTTP/1.1 http://http2.loadimpact.com/entry/



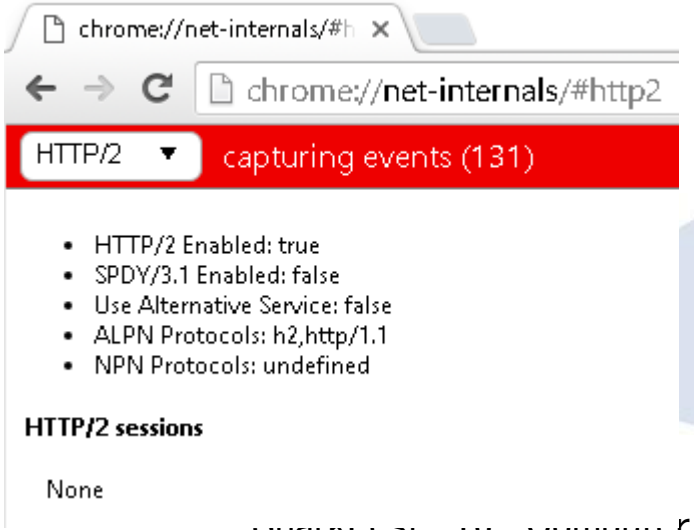
Please open the test website HTTPS vs HTTP https://www.httpvshttps.com/

Compare load times of the unsecure HTTP and encrypted HTTPS versions of this page. Each test loads 360 unique, non-cached images (0.62 MB total). For fastest results, run each test 2-3 times in a private/incognito browsing session.





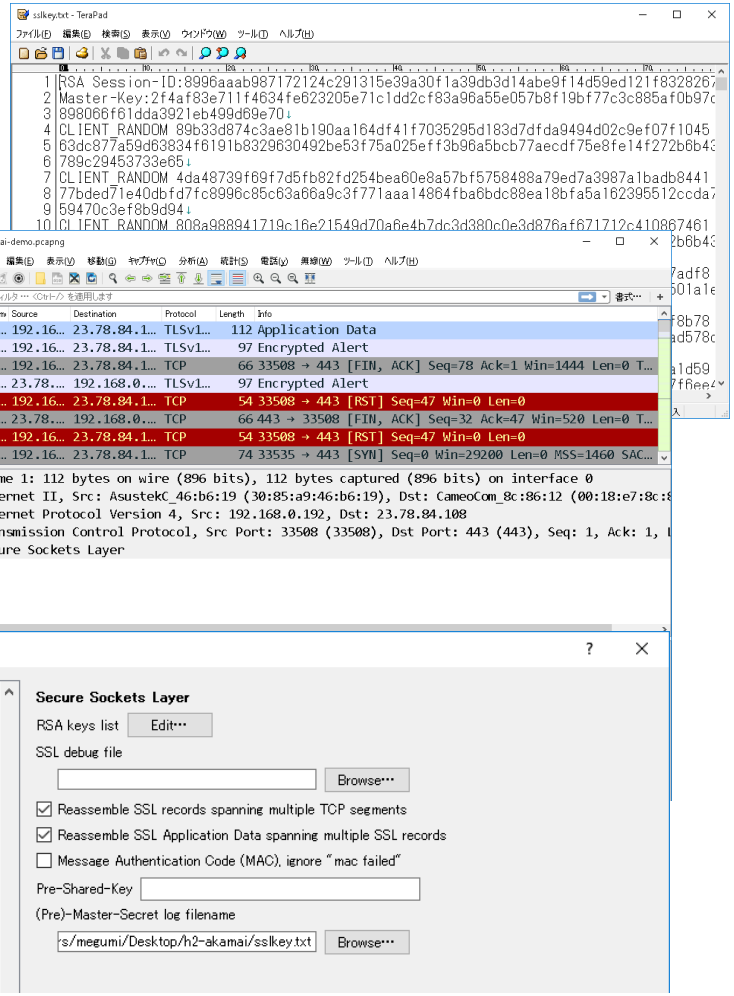
HTTP/2.0 uses SSL/TLS so difficult to decode



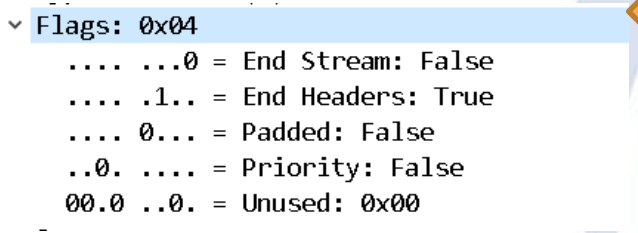
- HTTP/2.0 works on SSL/TLS connection in general.
- Decoding HTTP/2.0 is difficult to read, sometimes you need the proxy in the middle.
- Open the Chrome and type chrome://net-internals/#http2 you can see the HTTP/2 sessions
- Wireshark has the dissector of HTTP2.0 (http2) https://www.wireshark.org/docs/dfref/h/http2

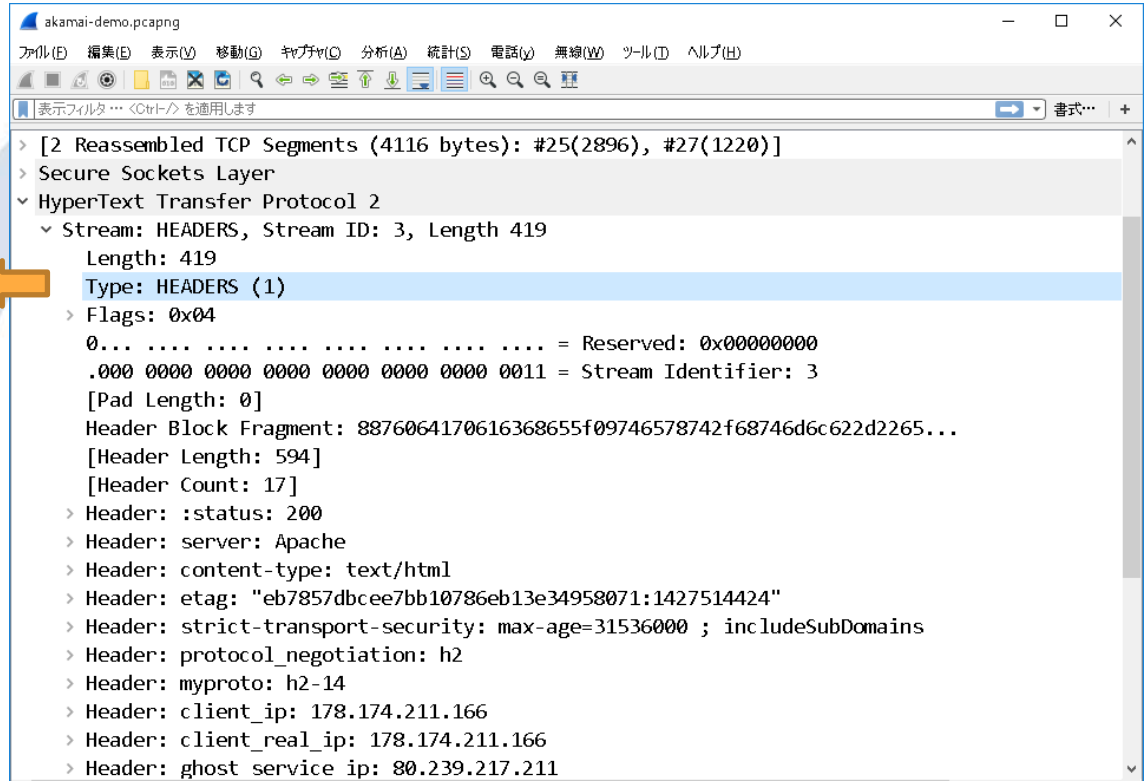
Sample trace file: akamai-demo.pcapng with ssl.key

- HTTP2 packet is always encrypted to SSL, so this time we use the sample of akamai-demo.pcapng with ssl.key file that describes the pre-master secret key with SSL session ID.
- So please open the packet, and select SSL layer and right click, to choose the protocol preferences. And set (pre)-Master-secret log file using Browse button to set ssl.key
- Client web browser: 192.168.0.192
  Web server: 23.78.84.108

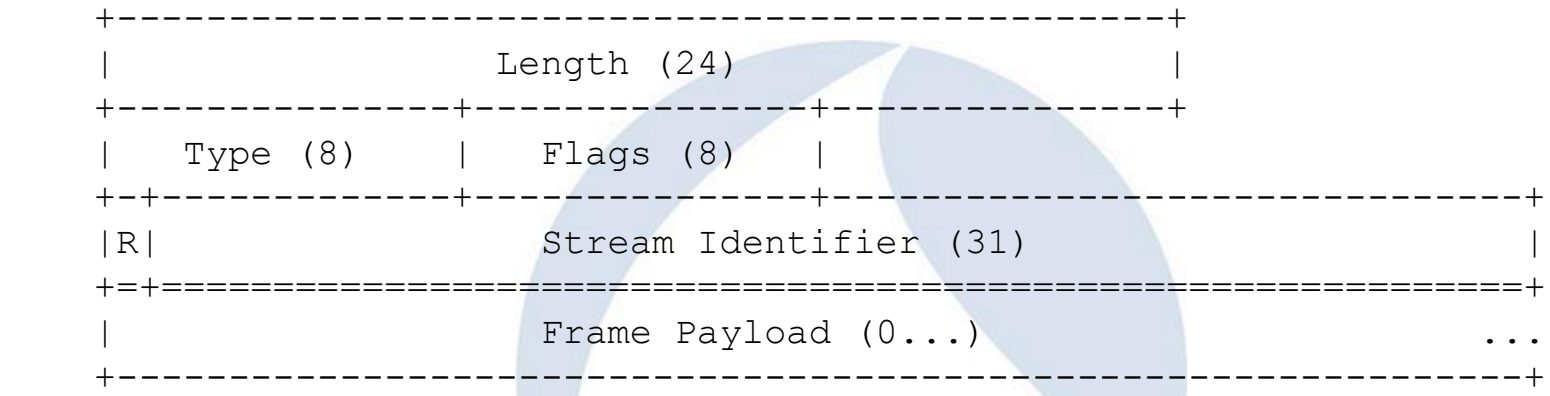


HTTP2.0 binary frame
(fixed header field, size and position of each fields)





HTTP2.0 binary frame （RFC7540)
(fixed header field, size and position of each fields)

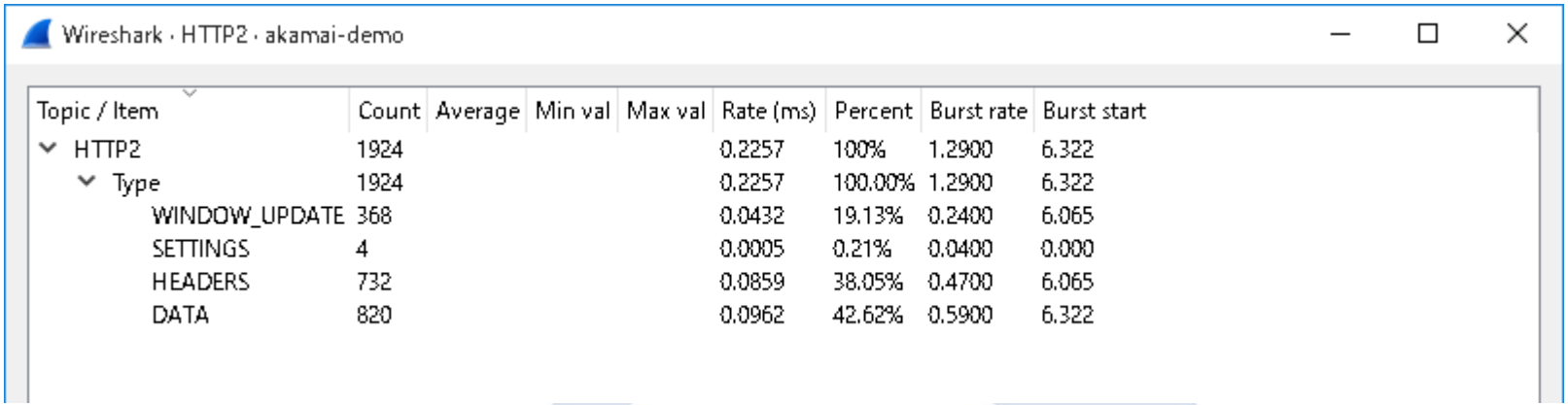


maximum payload 16383 octets
Length : Payload Size
Type   : Type of Frame
Flags : for example END_STREAM END_HEADERS
Stream Identifier : 0-Management Odd-Client Even-Server
 (each connection) incremental use the id

Major HTTP/2.0 frame type (http2.type)

Type	Description	http2.type
DATA	HTTP/2.0 data	0x0
HEADERS	HTTP/2.0 header	0x1
PRIORITY	Stream priority	0x2
RST_STREAM	Reset stream	0x3
SETTINGS	Connection Setting information	0x4
PUSH_PROMISE	Server push	0x5
PING	Ping	0x6
GOAWAY	Finish connection	0x7
WINDOW_UPDATE	Update window (receive buffer)	0x8
CONTINUATION	Continue information	0x9

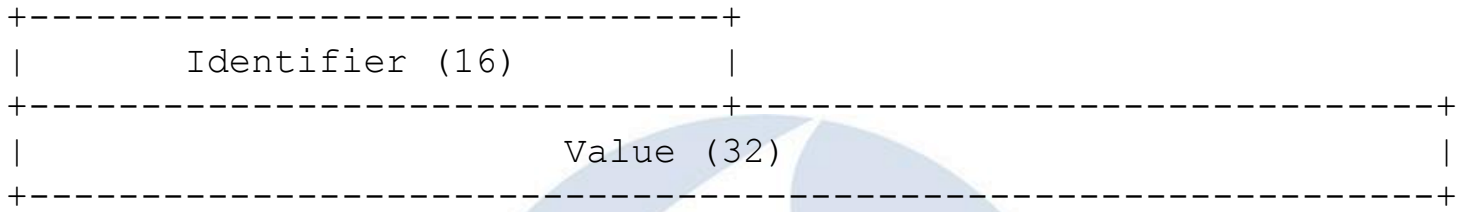
HTTP/2.0 Statistics



Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
HTTP2	1924				0.2257	100%	1.2900	6.322
Type	1924				0.2257	100.00%	1.2900	6.322
WINDOW_UPDATE	368				0.0432	19.13%	0.2400	6.065
SETTINGS	4				0.0005	0.21%	0.0400	0.000
HEADERS	732				0.0859	38.05%	0.4700	6.065
DATA	820				0.0962	42.62%	0.5900	6.322

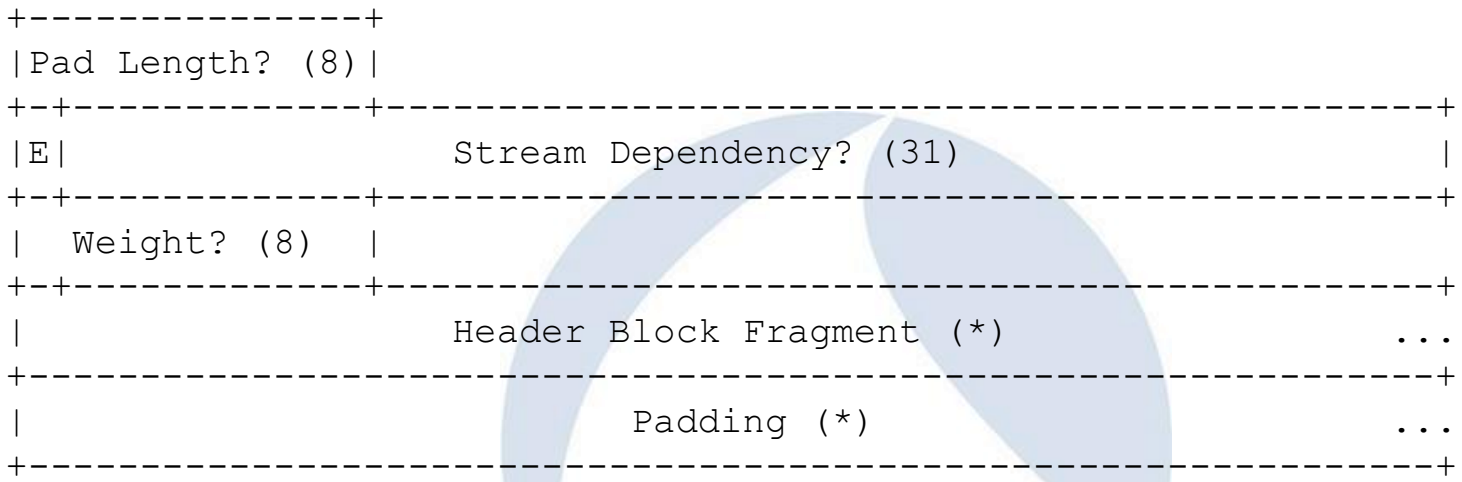
Wireshark has statistic feature of HTTP2
Select Statistics -> HTTP2
Wireshark collect all HTTP2 frames and divided by each Type of the frames, and list up with count, Rate(ms), Burst rate, and Burst start time

SETTING frame （RFC7540)



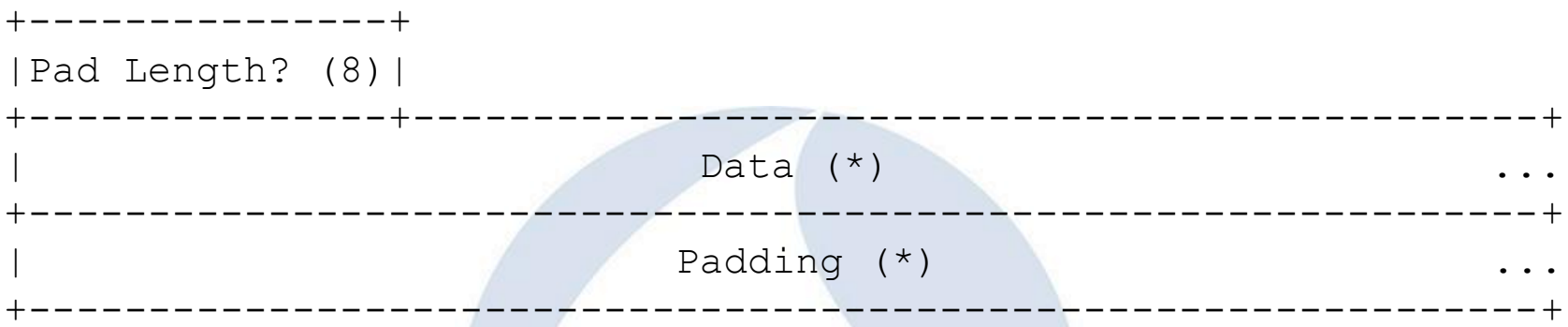
SETTING is used in setting up the HTTP 2 connection
 by both client and server
Identifier is Stream no and is set as 0
SETTING frame needs ACK
To reply the setting frame, use ACK flag
And setting connection information such as
 SETTINGS_HEADER_TABLE_SIZE
 SETTINGS_ENABLE_PUSH
 SETTINGS_MAX_CONCURRENT_STREAMS
 SETTINGS_INITIAL_WINDOW_SIZE
 SETTINGS_COMPRESS_DATA

HEADER frame （RFC7540)



HEADER frame is used to tell HTTP2 Header information.
Stream Dependency and weight is used to assign stream index and priority.
Header Block Fragment contains HTTP2 header
HTTP2 Header is table based compressed (HPACK) and set as index no.

DATA frame （RFC7540)

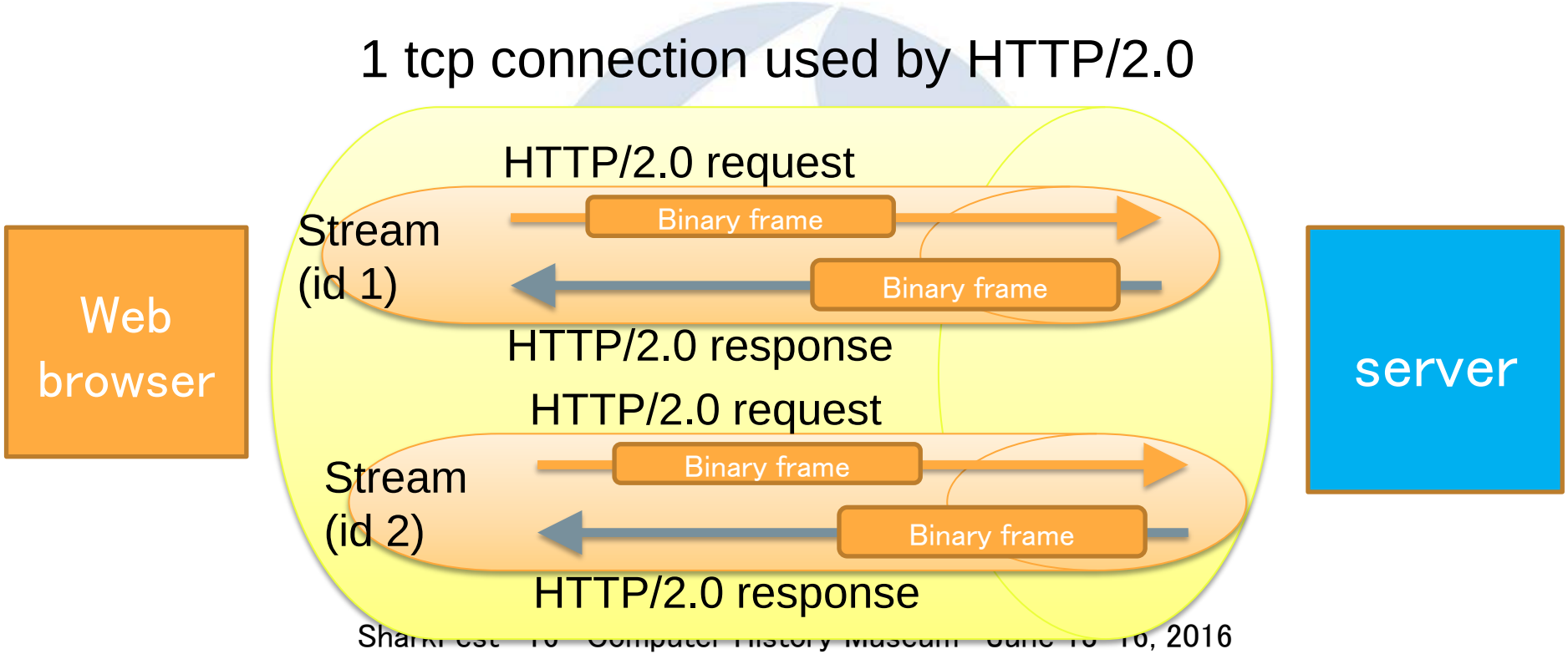


DATA frame contains actual HTTP2 data (HTTP2 body)
Actual HTTP data stores in Data field (http2.data.data)

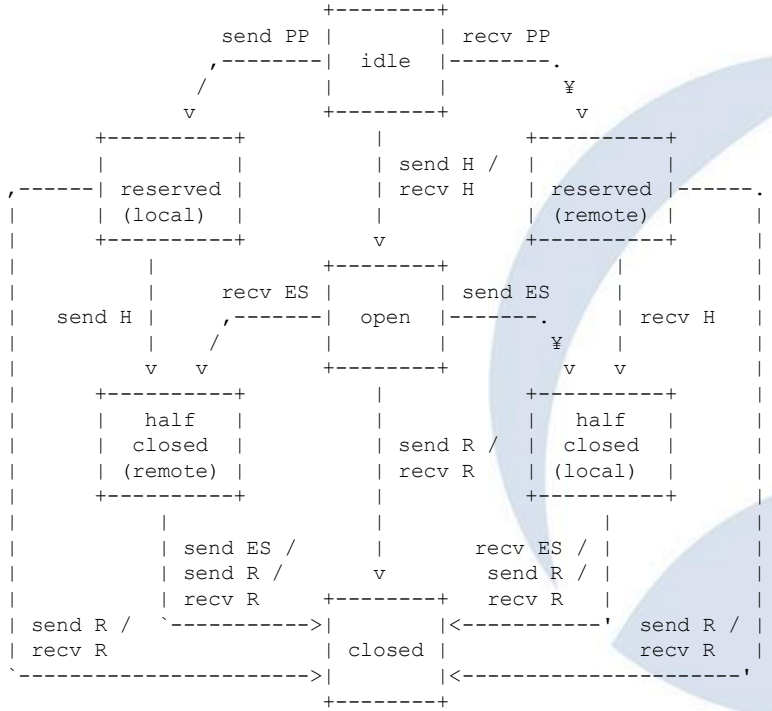
RST_Stream frame（RFC7540)

RST_STREAM frame is used when the user want to stop and reset http2 stream immediately.
RST_STREAM frame type is 3
Error code contains the reason of resetting stream

HTTP/2.0 manages communications using Stream mechanism
HTTP/2.0 uses 1 tcp connection and many Stream (virtual connection channel) that has id and priority

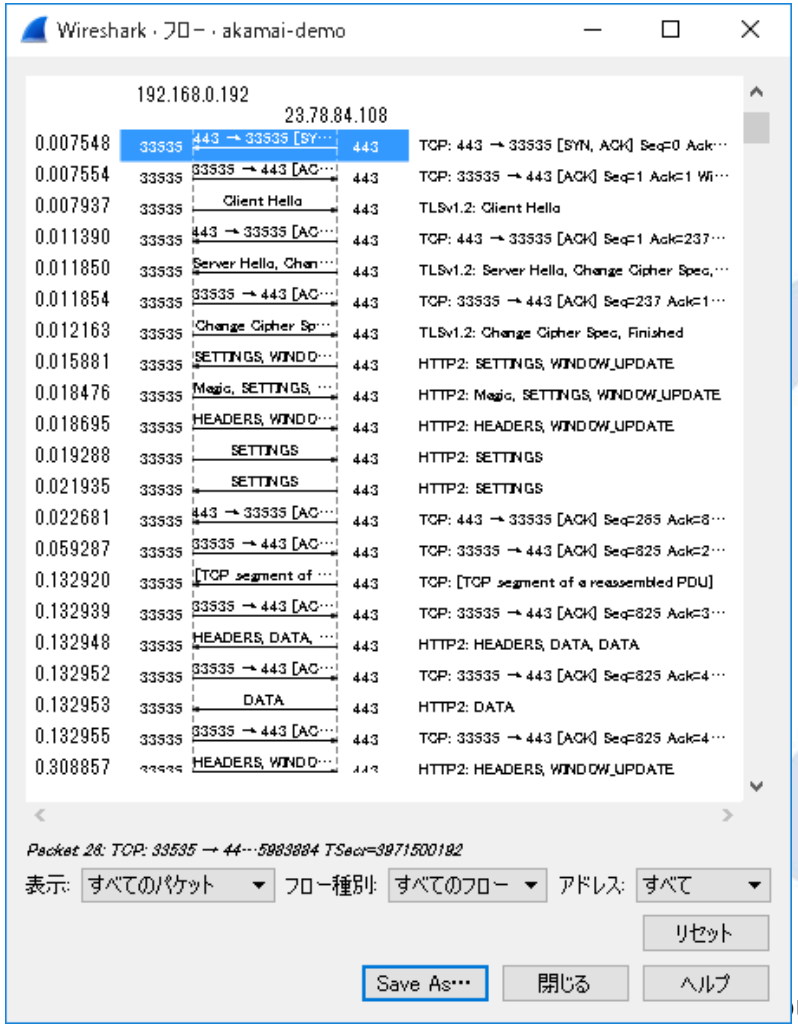


Stream based connection management of HTTP/2.0 = TCP based connection management of HTTP/1.1



send: endpoint sends this frame recv: endpoint receives this frame
H: HEADERS frame
PP: PUSH_PROMISE frame
ES: END_STREAM flag
R: RST_STREAM frame

- **Stream mechanism is very similar to TCP**
- **HTTP2 Stream have state chart diagrams like TCP**
- **Start from idle, connect in open state, ES(End_Stream) and end in closed state**

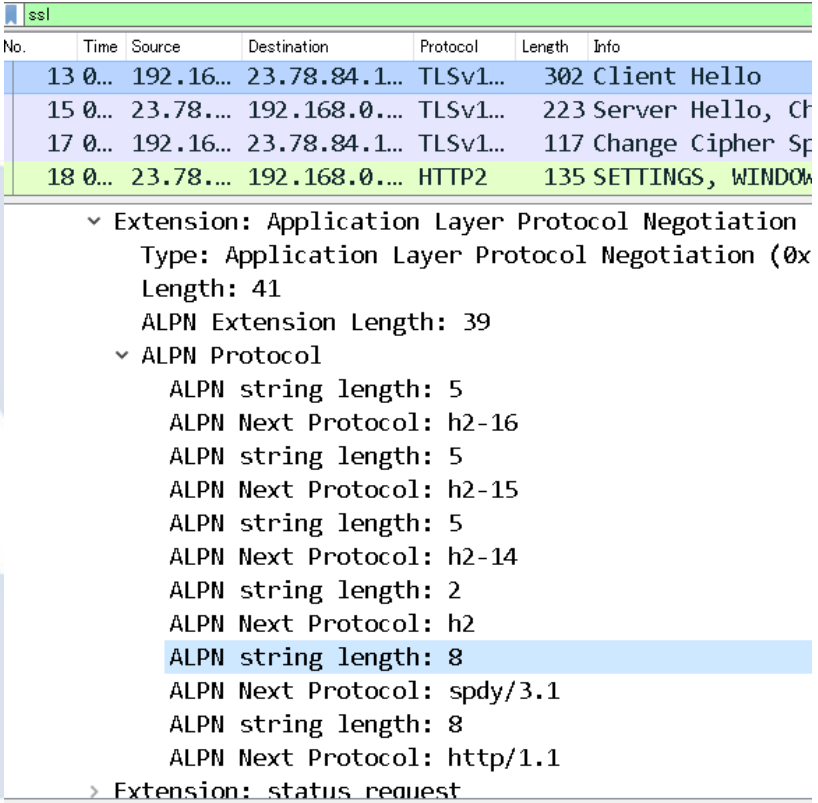


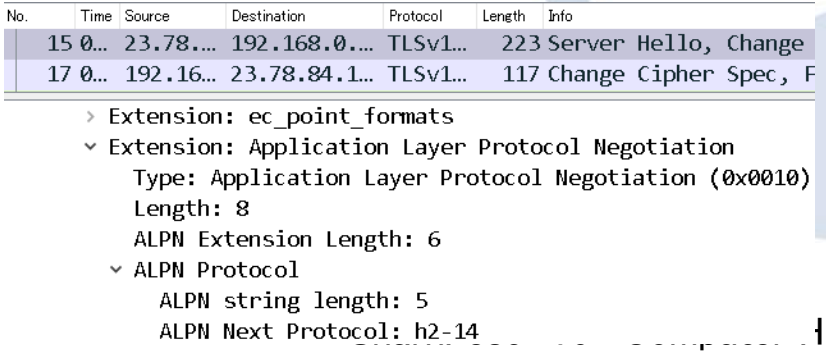
Flows of HTTP2 connection

i. Connect TLS connection between client and server (with ALPN to determine protocol and version of HTTP2)
ii. Server sends SETTINGS and WINDOW_UPDATE frame
iii. Client sends Magic, SETTINGS, WINDOW_UPDATE
iv. Client sends HEADERS, WINDOW_UPDATE
v. Client sends SETTINGS
vi. Server sends SETTINGS
vii. Server sends HEADER, DATA, DATA

(i) ALPN Application Layer Protocol Negotiation

1. The user use the URL as https://... and start up TLS connection
2. When client sends “Client Hello” in TLS connection, client sends ALPN information with the list of protocols client want to use
3. Server respond with “Server Hello” with ALPN information server determined to use HTTP2 draft14





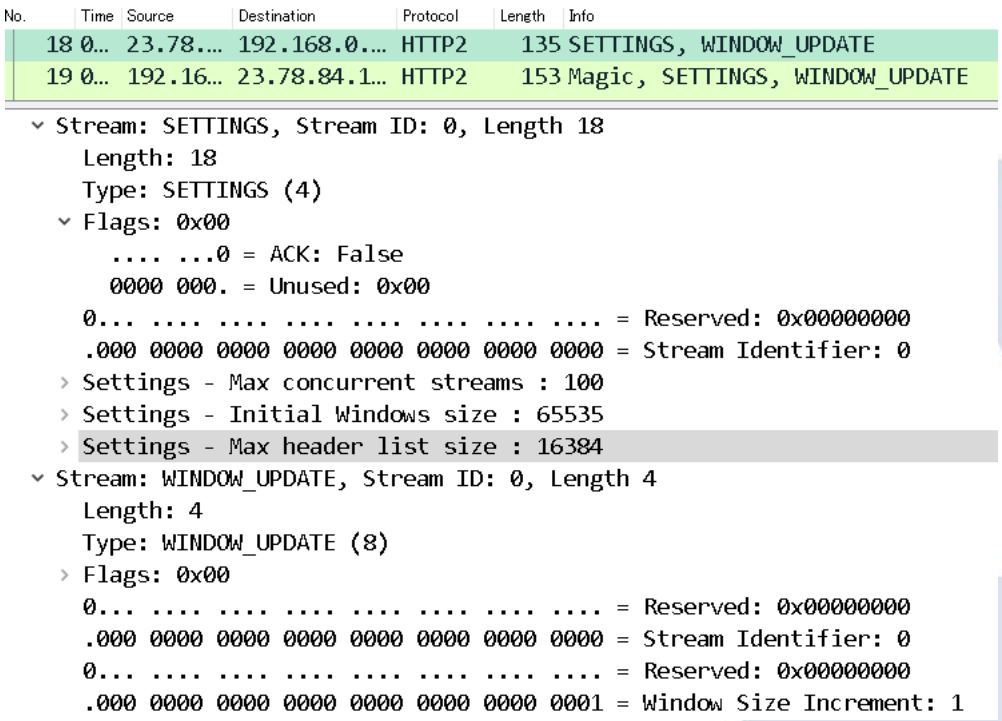
Another way to start up HTTP2 connection
HTTP Upgrade (RFC7540)

1. The user the URL as “http://...” and start up HTTP/1.1 connection
2. HTTP request contains “Upgrade” and “HTTP2-Settings” header

   GET / HTTP/1.1
   Host: server.example.com
   Connection: Upgrade, HTTP2-Settings
   Upgrade: h2c
   HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
3. Server respond with 101 status code

   HTTP/1.1 101 Switching Protocols
   Connection: Upgrade
   Upgrade: h2c
   [HTTP/2 connection ...

(ii) Server sends SETTINGS and WINDOW_UPDATE frame



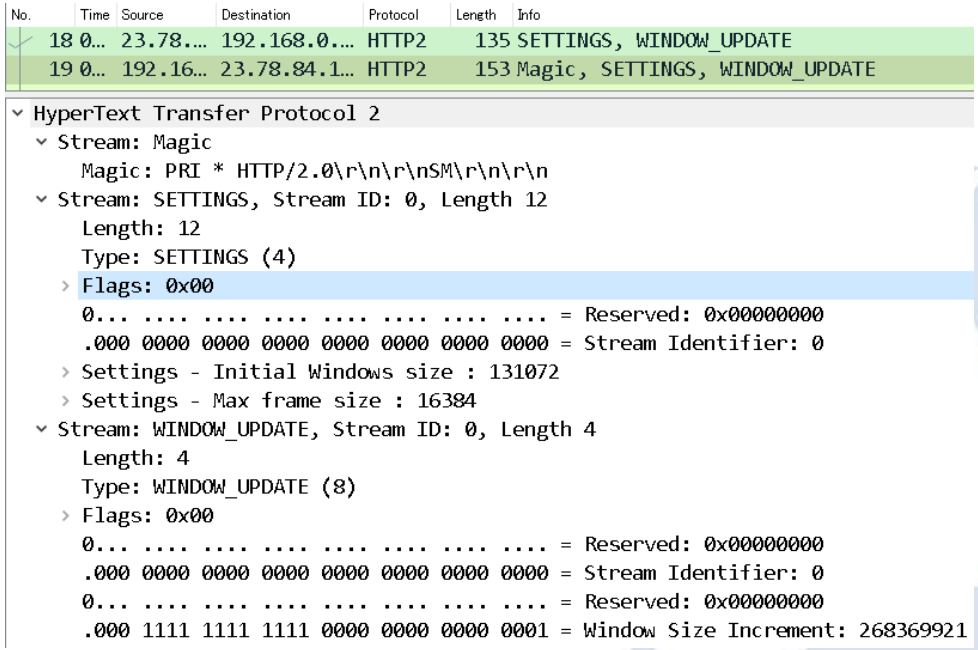
After finishing TLS connection with ALPN to determine protocol and version of HTTP2

Server sends SETTINGS and WINDOW_UPDATE frame

SETTING frame contains
- Max concurrent stream
- Initial Windows size
- Max header list size

WINDOW_UPDATE frame contains
- Window Size Increment

(iii) Client sends Magic, SETTINGS, WINDOW_UPDATE frame



No.	Time	Source	Destination	Protocol	Length	Info
18	0…	23.78.…	192.168.0.…	HTTP2	135	SETTINGS, WINDOW_UPDATE
19	0…	192.16…	23.78.84.1…	HTTP2	153	Magic, SETTINGS, WINDOW_UPDATE

```
HyperText Transfer Protocol 2
  Stream: Magic
    Magic: PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
  Stream: SETTINGS, Stream ID: 0, Length 12
    Length: 12
    Type: SETTINGS (4)
    Flags: 0x00
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
    Settings - Initial Windows size : 131072
    Settings - Max frame size : 16384
  Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
    Length: 4
    Type: WINDOW_UPDATE (8)
    Flags: 0x00
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
    0... .... .... .... .... .... .... .... = Reserved: 0x00000000
    .000 1111 1111 1111 0000 0000 0000 0001 = Window Size Increment: 268369921
```

Client sends Magic, SETTINGS, WINDOW_UPDATE frame

Magic frame contains

“PRI * HTTP/2.0 CR+LF

CR+LF

SM

CR+LF, CR+LF

SETTING frame contains

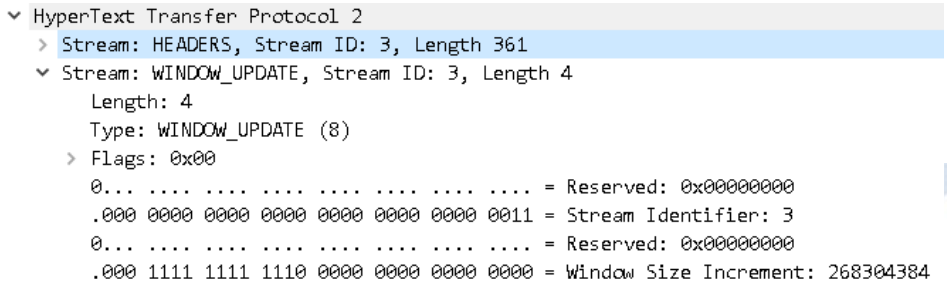
Initial Windows size

Max frame size

WINDOW_UPDATE frame contains

Window Size Increment

(iv) Client sends HEADERS, WINDOW_UPDATE frame

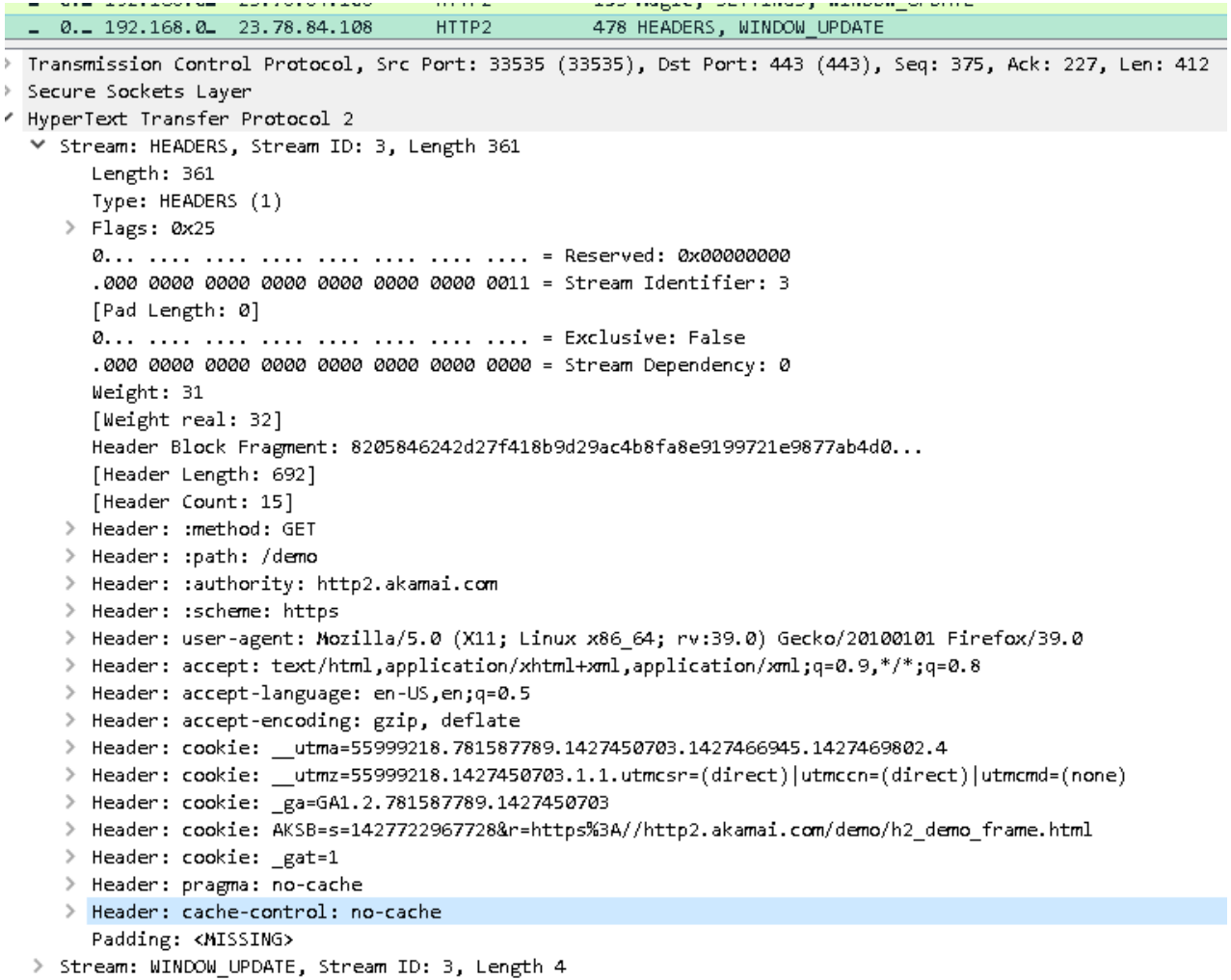


Client sends HEADERS, WINDOW_UPDATE frame

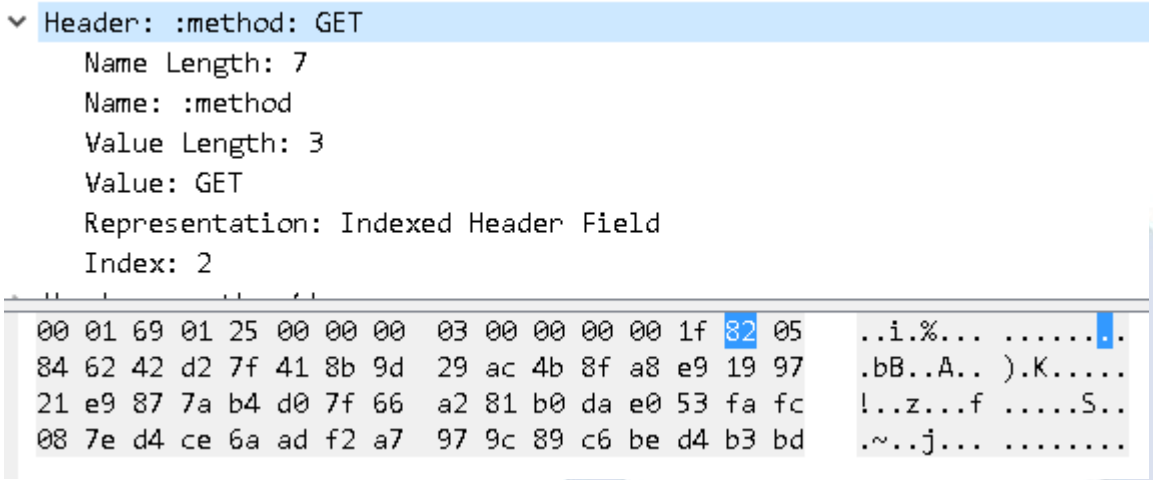
HEADERS frame contains HTTP2.0 header information
WINDOW_UPDATE frame contains Window Size Increment

HEADERS frame

- method
  HTTP method
- path
  Path of the object
- authority
  Host of the server
- scheme
  HTTP / HTTPS
- user-agent:
  browser information

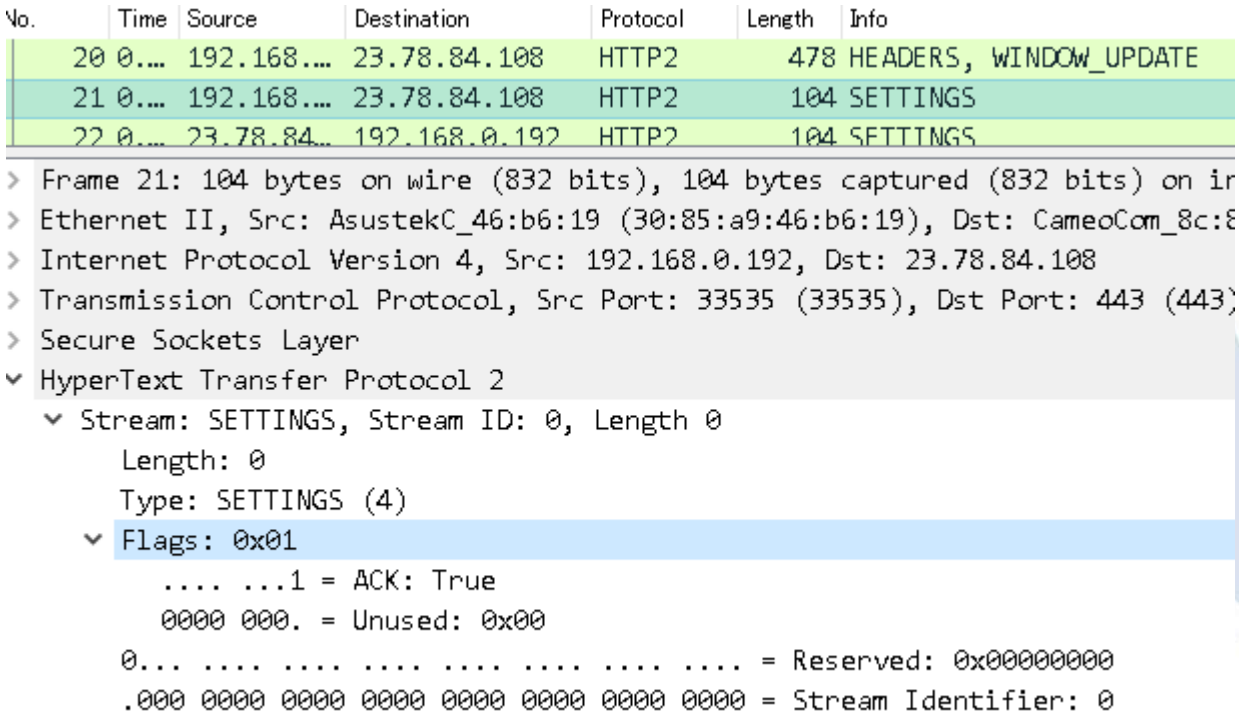


HPACK (table based Header compression)



- HTTP/2.0 doesn't use the String, but just send the value.
- HPACK uses Huffman encoding,
- HPACK uses static table index that defined common headers.
- HPACK uses header table index that used for history of sent header/value.
- HPACK uses reference set that used for sending the difference from last header.

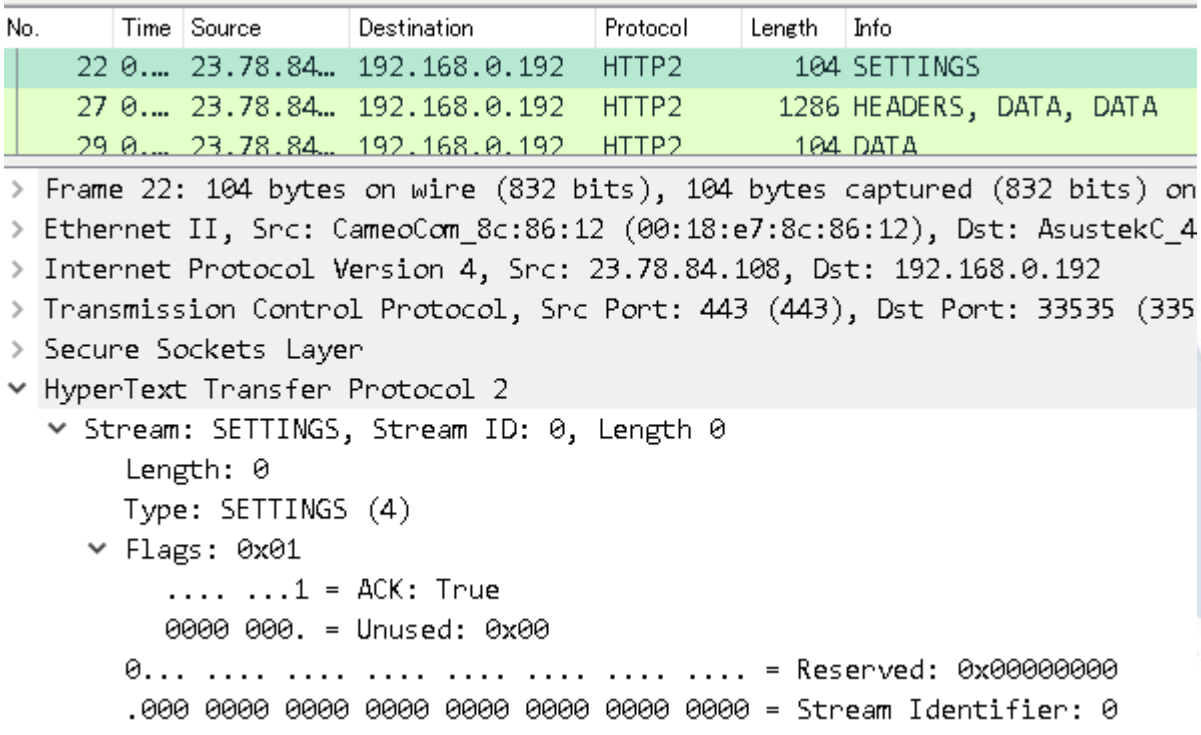
(v) Client sends SETTINGS frame



Client sends SETTINGS frame

SETTINGS frame contains ACK to the server's SETTINGS frame

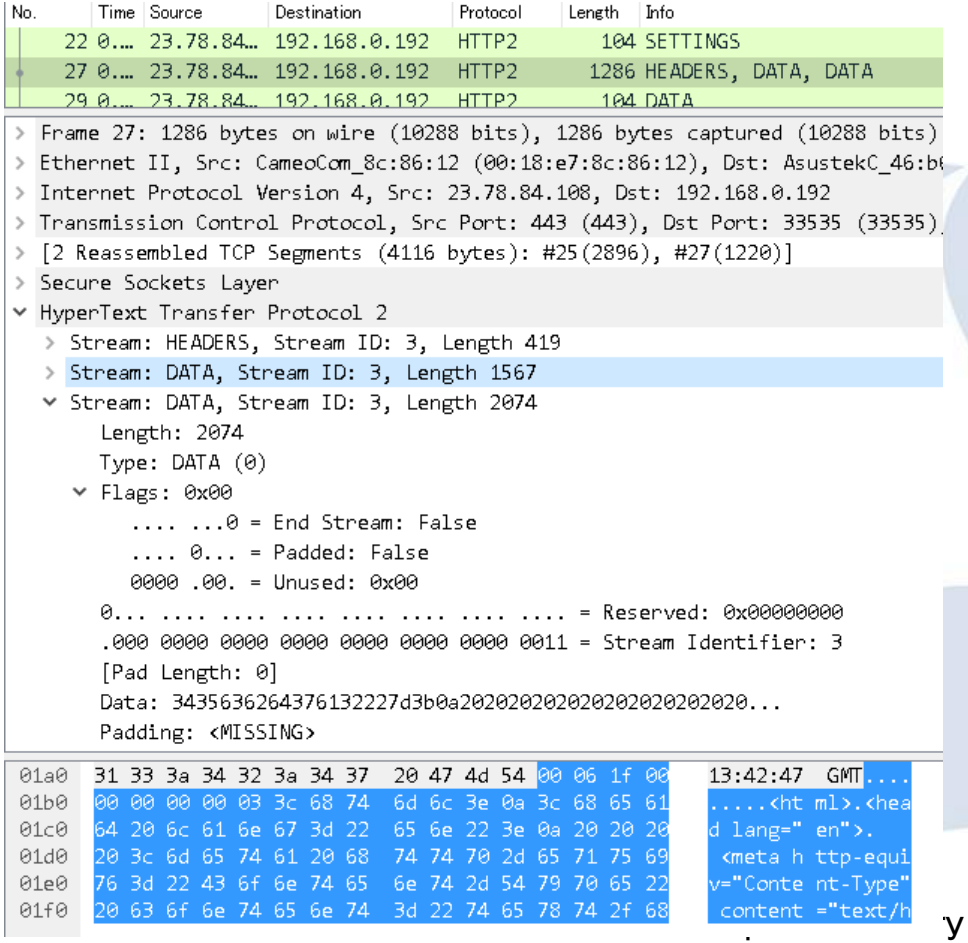
(vi) Server sends SETTINGS frame



Server sends SETTINGS frame

SETTINGS frame contains ACK to the client's SETTINGS frame

(vii) Server sends HEADER, DATA, DATA



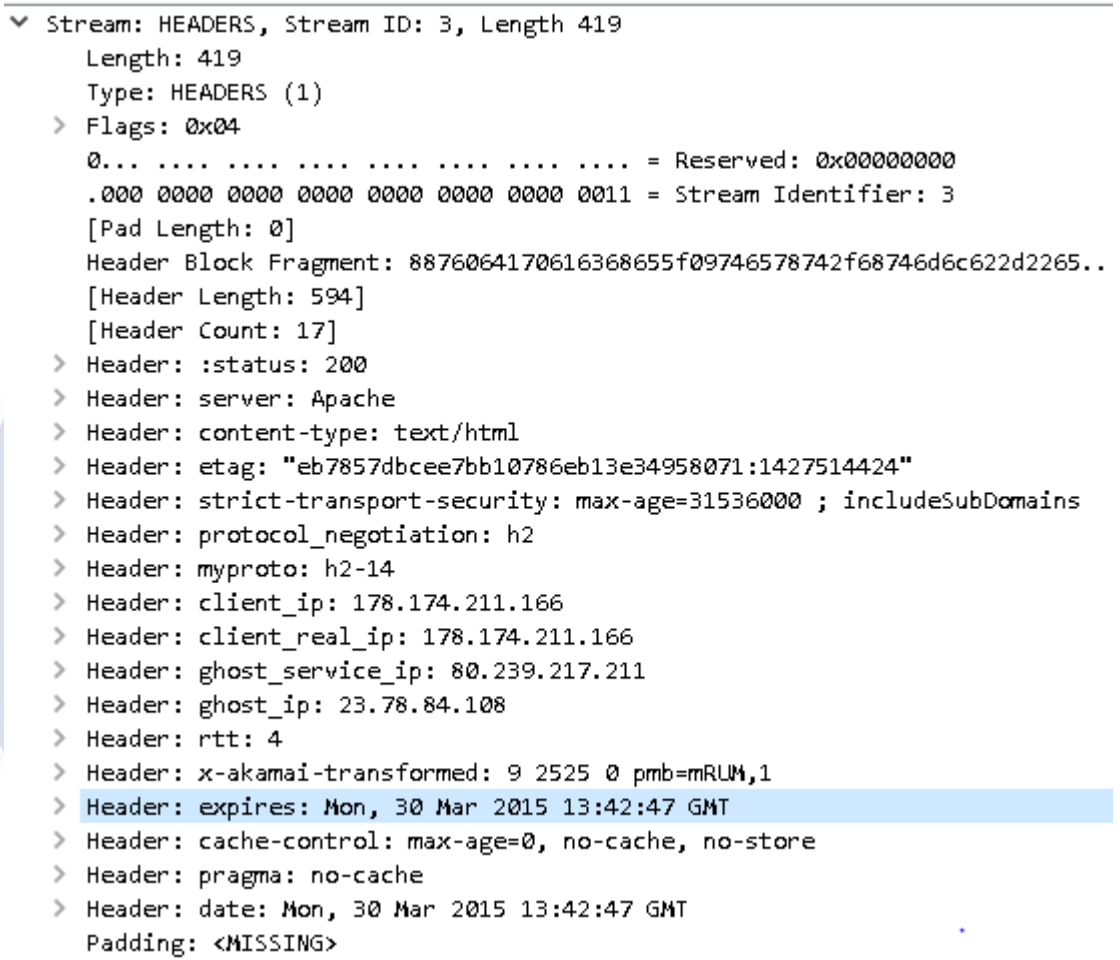
Server sends HEADER, DATA, DATA Frame

HEADERS frame contains HTTP2.0 header information

DATA frame contains HTTP2.0 DATA (body)

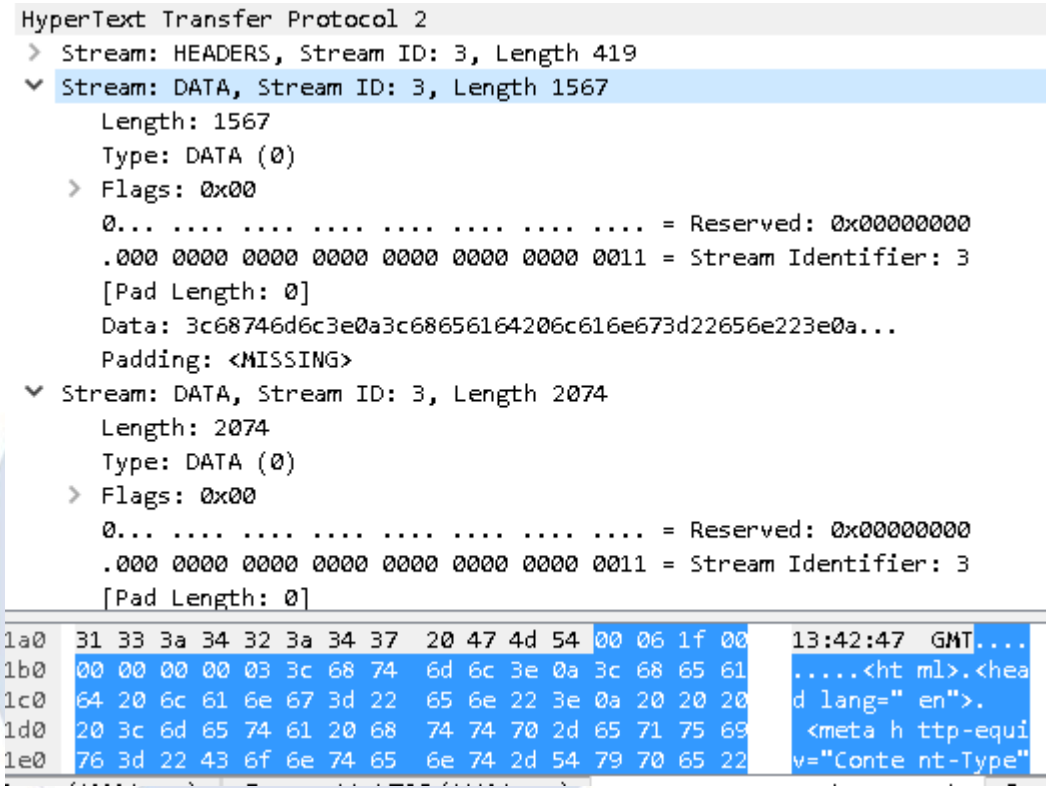
HTTP/2.0 response HEADERS

- :status
  Status code
- server
  web server
- content-type
  document type



HTTP/2.0 response DATA

Actual body part of HTTP/2.0 stores in Data field of DATA frame (http2.data.data)



TIPS #10 USE Wireshark for future protocols.

- HTTP/2.0 also has Server Push feature.
  Client does not have to send request
- HTTP/3.0 is now developing.
- Google creates QUIC (Quic UDP Internet Connections)
  QUIC behaves as TCP/TLS over UDP layer
  QUIC is used in Google services now, and Wireshark also
- Oh all protocols are invisible and
  we are going to the dark age of bender specific unknown protocols by giant venders…
- Don’t worry, Wireshark decodes everything.
- Wireshark is the light for future protocols to the future.

Thank you very much for your listening

Use Wireshark for ever !

Thank you !
どうもありがとうございます！