



Investigation of the HTTP-based large-scale data transmission over a long distance

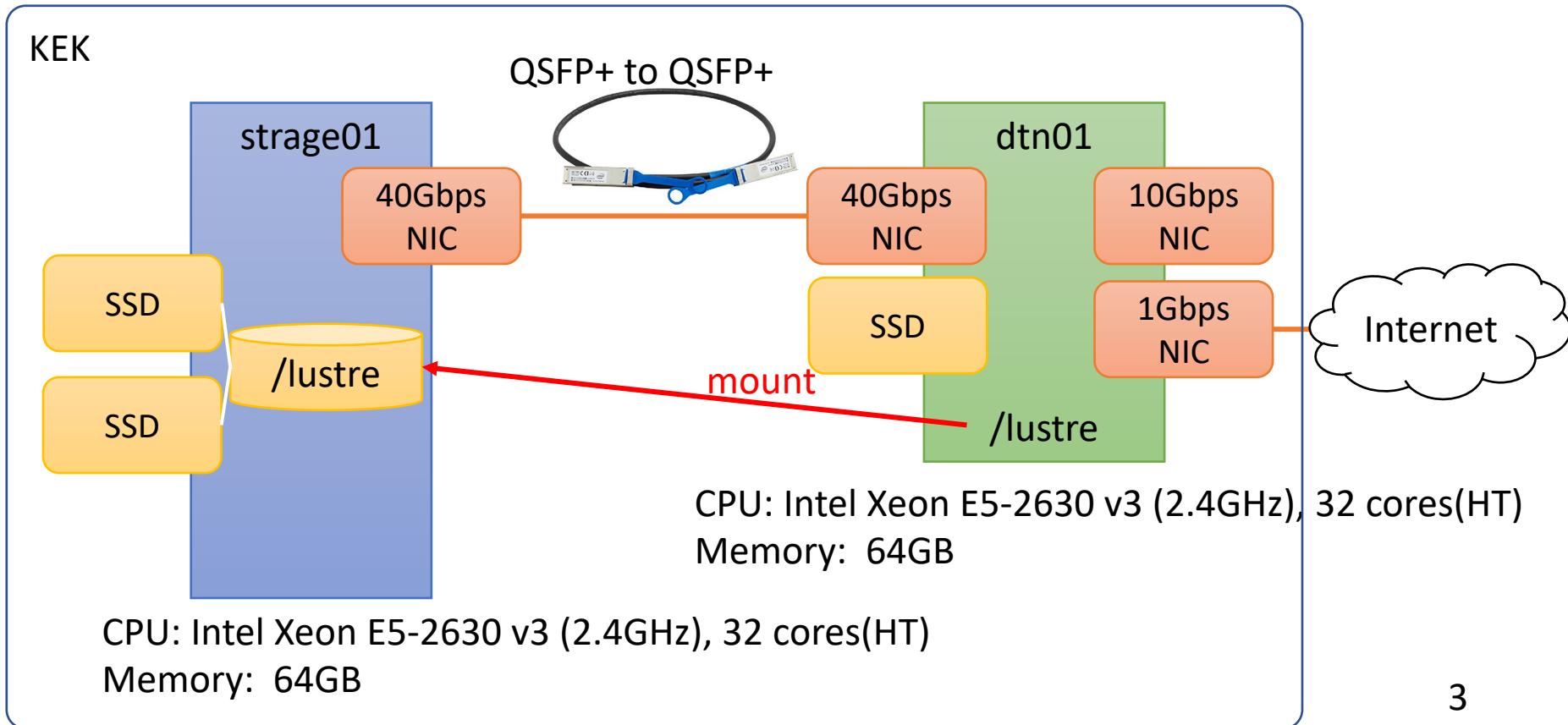
Wataru Takase, Sari Kaneko, Tomoaki Nakamura
Computing Research Center, KEK
2nd December, 2019

Background: Open Source Globus Toolkit Support Ended

- GridFTP based file transfer has been mainly used in High Energy Physics.
- It's implemented in the Globus Toolkit.
- We have started to investigate alternative way of efficient data transfer.
 - Possible candidates:
 - HTTP, XrootD

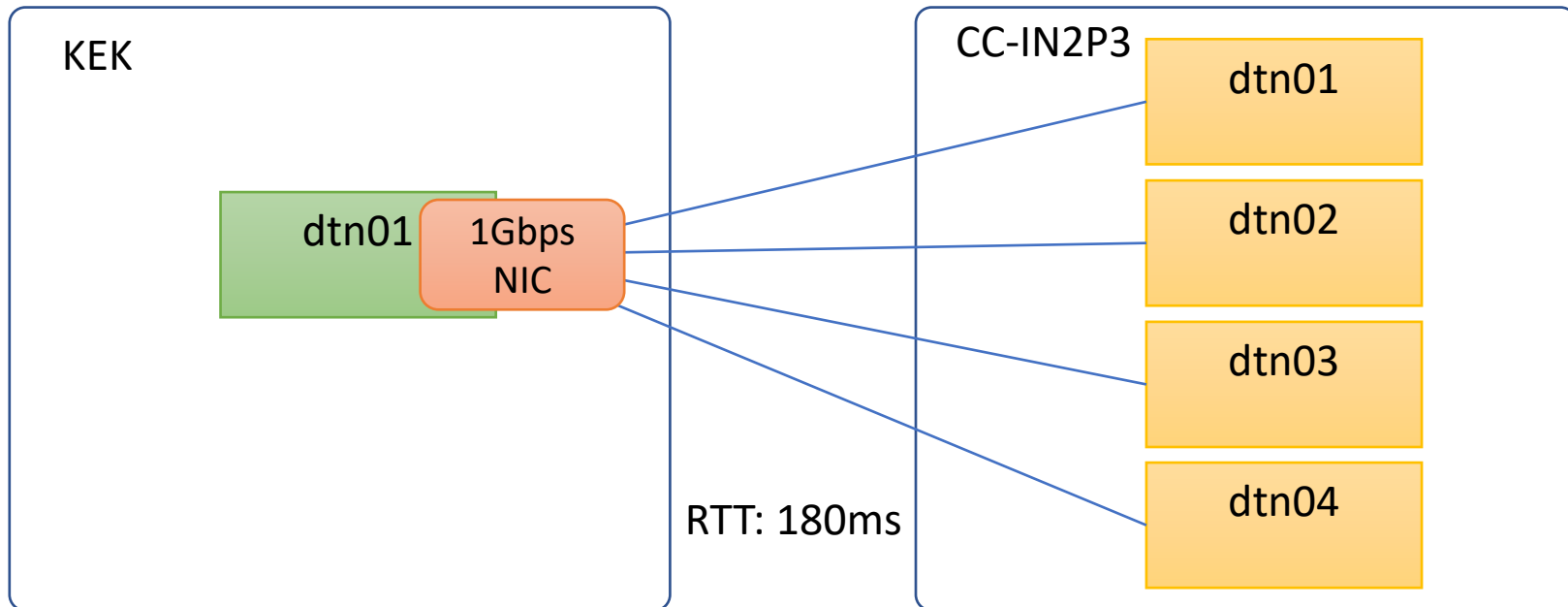
First Test Environment

- Prepare a data transfer node and a backend storage node.
- Use Lustre as a shared file system on top of two SSDs and QSFP+ interconnect
- Internet bandwidth is limited to 1Gbps due to the test environment



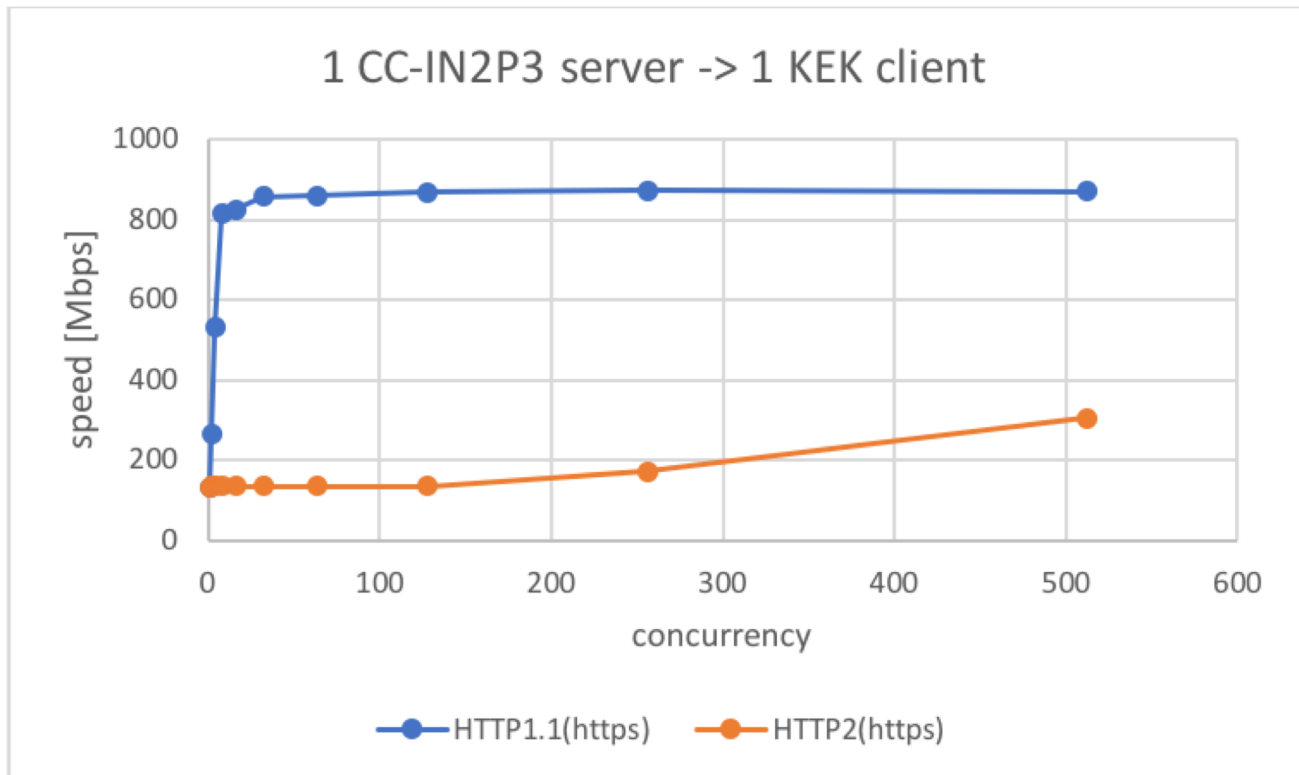
HTTP-based File Transfer Test Between CC-IN2P3 and KEK

- Use **chasqui** developed by Fabio.
 - HTTP-based memory-to-memory file transfer tool.
 - It supports HTTP/1.1 and HTTP/2 connection over TLS.
- There are 4 DTN (Data Transfer Node) in CC-IN2P3 and there is 1 DTN in KEK.
- There is a 'chasqui server' daemon running on each machine and listening to port 20000.
- There is a 'chasqui client' daemon running on each machine and listening to port 20100.
- Download data from the chasqui server(s) by HTTP GET requests.



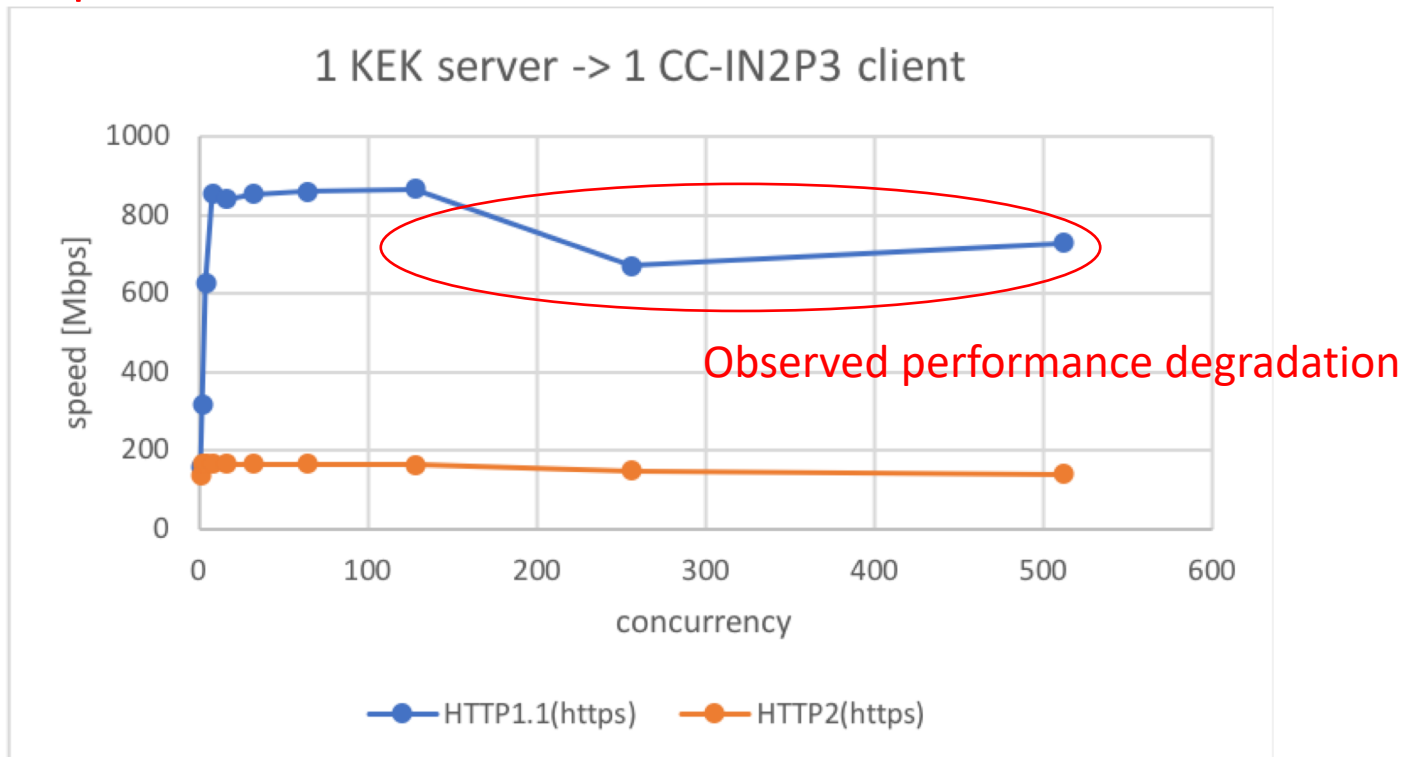
Chasqui Results: Download from CC-IN2P3 to KEK

- HTTP/1.1 downloading speed reached almost 1Gbps with 8 parallel connections from single server.
- HTTP/1.1 performance is much better than HTTP/2.



Chasqui Results: Download from KEK to CC-IN2P3

- HTTP/1.1 downloading speed reached almost 1Gbps with 8 parallel connections from single server.
- We have observed HTTP/1.1 performance degradation with higher parallelism.
- HTTP/1.1 performance is much better than HTTP/2.

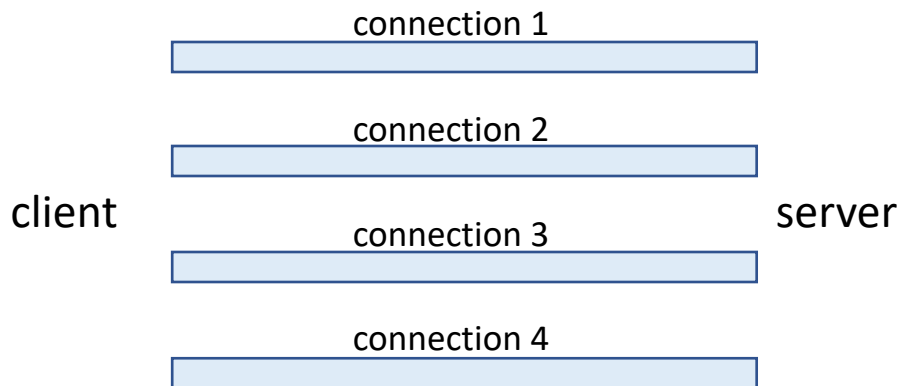


Comparison of the way of Parallel Access from the same client

- Ex. 4 parallel accesses from the same client

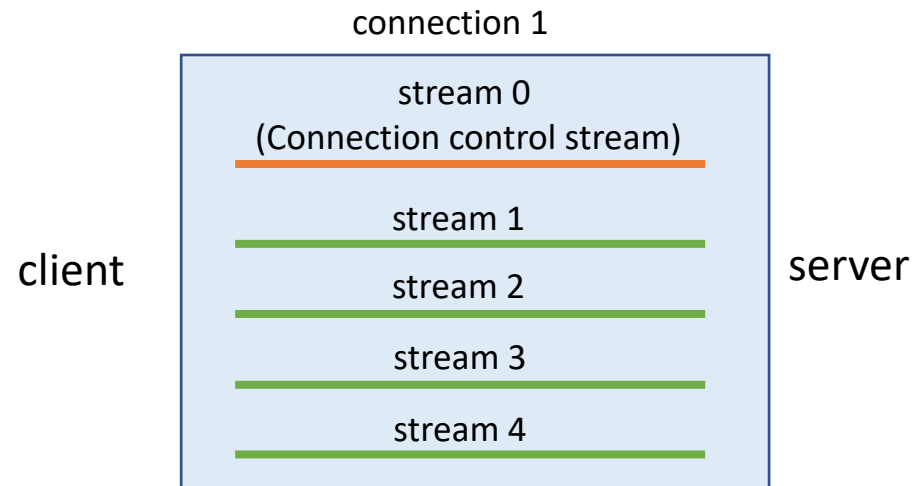
HTTP/1.1

- Establish 4 connections, and each has its own TCP window size.



HTTP/2

- Establish only 1 connection, and it has 1 TCP window size.
- Data transfer by using 4 streams in the connection.



Single TCP connection limits the data transfer speed in HTTP/2?

Dump HTTP/2 Connection: Small Frame Size

chasqui_http2_plain_1_both

http2

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000768	192.168.10.218	192.168.10.217	HTTP2	130	Magic, SETTINGS[0], WINDOW_UPDATE[0]
6	0.000950	192.168.10.218	192.168.10.217	HTTP2	141	HEADERS[3]: GET /file?id=file-1&size=106453301
8	0.002881	192.168.10.217	192.168.10.218	HTTP2	99	SETTINGS[0]
10	0.002962	192.168.10.217	192.168.10.218	HTTP2	88	SETTINGS[0], WINDOW_UPDATE[0]
12	0.003064	192.168.10.218	192.168.10.217	HTTP2	75	SETTINGS[0]
13	0.003321	192.168.10.217	192.168.10.218	HTTP2	138	HEADERS[3]: 200 OK
18	0.003802	192.168.10.217	192.168.10.218	HTTP2	11650	DATA[3] [TCP segment of a reassembled PDU]
22	0.003883	192.168.10.217	192.168.10.218	HTTP2	14546	DATA[3] [TCP segment of a reassembled PDU]
24	0.003994	192.168.10.217	192.168.10.218	HTTP2	14546	DATA[3] [TCP segment of a reassembled PDU]
25	0.004028	192.168.10.217	192.168.10.218	HTTP2	14546	DATA[3] [TCP segment of a reassembled PDU]
26	0.004049	192.168.10.217	192.168.10.218	HTTP2	27578	DATA[3], DATA[3]
30	0.004246	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE[3]
31	0.004276	192.168.10.217	192.168.10.218	HTTP2	29026	DATA[3] [TCP segment of a reassembled PDU]
32	0.004310	192.168.10.217	192.168.10.218	HTTP2	20179	DATA[3], DATA[3]
33	0.004347	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE[3]
34	0.004396	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE[3]
35	0.004421	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE[3]

Frame 26: 27578 bytes on wire (220624 bits), 27578 bytes captured (220624 bits)

Ethernet II, Src: RealtekU_72:ff:2c (52:54:00:72:ff:2c), Dst: RealtekU_b6:8d:68 (52:54:00:b6:8d:68)

Internet Protocol Version 4, Src: 192.168.10.217, Dst: 192.168.10.218

Transmission Control Protocol, Src Port: 5678, Dst Port: 34782, Seq: 71080, Ack: 149, Len: 27512

[2 Reassembled TCP Segments (16393 bytes): #25(5380), #26(11013)]

[Frame: 25, payload: 0-5379 (5380 bytes)]

[Frame: 26, payload: 5380-16392 (11013 bytes)]

[Segment count: 2]

[Reassembled TCP length: 16393]

[Reassembled TCP Data: 004000000000000034ca14d91d7954b05db1d17d8d9c74f...]

HyperText Transfer Protocol 2

Stream: DATA, Stream ID: 3, Length 16384 (partial entity body)

HyperText Transfer Protocol 2

Stream: DATA, Stream ID: 3, Length 16384 (partial entity body)

0000 00 40 00 00 00 00 00 00 03 4c a1 4d 91 d7 95 4b @.....L.M...K

0010 05 db 1d 17 d8 d9 c7 4f 35 48 4d a7 95 b8 e2 910 5HM.....

Frame (27578 bytes) Reassembled TCP (16393 bytes)

TCP Segments (tcp.segments), 16393 bytes

Packets: 85538 · Displayed: 68799 (80.4%) Profile: Default

- Chunk data to 16384 byte, and send it.
- What about increasing the value ?

HTTP/2 Frame Size: 16384 Bytes

- The value is defined by `SETTINGS_MAX_FRAME_SIZE`:
 - <https://http2.github.io/http2-spec/#SettingValues>
 - The default value is 16484 bytes.
- HTTP/2 client of Go doesn't set the above value, so the default value is used.
- I modified the client to set the value:
 - <https://github.com/wtakase/net/commit/a9a97aa674f4b9705ad4775e5be164ab4070fd05>

Changed from [golang/net/http2/transport.go](https://golang.org/pkg/net/http2/transport.go)

```
140 - func configureTransport(t1 *http.Transport) (*Transport, error) {
202 + func configureTransport(t1 *http.Transport, fct *FlowControlTransport) (*Transport, error) {
141 203     connPool := new(clientConnPool)
142 204     t2 := &Transport{
143 -         ConnPool: noDialClientConnPool{connPool},
144 -         t1:      t1,
205 +         ConnPool:      noDialClientConnPool{connPool},
206 +         MaxFrameSize:   fct.MaxFrameSize,
207 +         TransportDefaultConnFlow: fct.TransportDefaultConnFlow,
208 +         TransportDefaultStreamFlow: fct.TransportDefaultStreamFlow,
209 +         TransportDefaultStreamMinRefresh: fct.TransportDefaultStreamMinRefresh,
210 +         t1:             t1,
145 211     }
146 212     connPool.t = t2

606 672 func (t *Transport) newClientConn(c net.Conn, singleUse bool) (*ClientConn, error) {
607 673     cc := &ClientConn{
608 674         t:      t,
609 675         tconn:  c,
610 676         readerDone: make(chan struct{}),
611 677         nextStreamID: 1,
612 -         maxFrameSize: 16 << 10, // spec default
678 +         maxFrameSize: t.maxFrameSize(), // spec default 16384
613 679         initialWindowSize: 65535, // spec default
614 680         maxConcurrentStreams: 1000, // "infinite", per spec. 1000 seems good enough.
615 681         peerMaxHeaderListSize: 0xffffffffffffffff, // "infinite", per spec. Use 2^64-1 instead.

@@ -652,16 +718,17 @@ func (t *Transport) newClientConn(c net.Conn, singleUse bool) (*ClientConn, erro
652 718
653 719     initialSettings := []Setting{
654 720         {ID: SettingEnablePush, Val: 0},
655 -         {ID: SettingInitialWindowSize, Val: transportDefaultStreamFlow},
721 +         {ID: SettingInitialWindowSize, Val: t.transportDefaultStreamFlow()},
722 +         {ID: SettingMaxFrameSize, Val: t.maxFrameSize()},
656 723     }
```

Dump HTTP/2 Connection: Too many Window Update Notification

The image shows a Wireshark packet capture of an HTTP/2 connection. The main pane displays a list of packets. Packet 33 is highlighted in blue and is a WINDOW_UPDATE [3] of 79 bytes. A red box highlights packets 33 through 40, all of which are WINDOW_UPDATE [3] notifications. A red arrow points from the text below to the 'WINDOW_UPDATE [3]' entry in packet 33.

No.	Time	Source	Destination	Protocol	Length	Info
13	0.003321	192.168.10.217	192.168.10.218	HTTP2	138	HEADERS [3]: 200 OK
18	0.003802	192.168.10.217	192.168.10.218	HTTP2	11650	DATA [3] [TCP segment of a reassembled PDU]
22	0.003883	192.168.10.217	192.168.10.218	HTTP2	14546	DATA [3] [TCP segment of a reassembled PDU]
24	0.003994	192.168.10.217	192.168.10.218	HTTP2	14546	DATA [3] [TCP segment of a reassembled PDU]
25	0.004028	192.168.10.217	192.168.10.218	HTTP2	14546	DATA [3] [TCP segment of a reassembled PDU]
26	0.004049	192.168.10.217	192.168.10.218	HTTP2	27578	DATA [3], DATA [3]
30	0.004246	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
31	0.004276	192.168.10.217	192.168.10.218	HTTP2	29026	DATA [3] [TCP segment of a reassembled PDU]
32	0.004310	192.168.10.217	192.168.10.218	HTTP2	20179	DATA [3], DATA [3]
33	0.004347	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
34	0.004396	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
35	0.004421	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
36	0.004434	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
37	0.004457	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
38	0.004482	192.168.10.217	192.168.10.218	HTTP2	16459	DATA [3]
39	0.004522	192.168.10.218	192.168.10.217	HTTP2	79	WINDOW_UPDATE [3]
40	0.004536	192.168.10.217	192.168.10.218	HTTP2	16459	DATA [3]

▼ Frame 33: 79 bytes on wire (632 bits), 79 bytes captured (632 bits)
▶ Ethernet II, Src: RealtekU_b6:8d:68 (52:54:00:b6:8d:68), Dst: RealtekU_72:ff:2c (52:54:00:72:ff:2c)
▶ Internet Protocol Version 4, Src: 192.168.10.218, Dst: 192.168.10.217
▶ Transmission Control Protocol, Src Port: 34782, Dst Port: 5678, Seq: 162, Ack: 147665, Len: 13
▼ HyperText Transfer Protocol 2
 ▼ Stream: WINDOW_UPDATE, Stream ID: 3, Length 4
 Length: 4
 Type: WINDOW_UPDATE (8)
 Flags: 0x00
 ▶ 0... .. = Reserved: 0x0
 .000 0000 0000 0000 0000 0000 0000 0011 = Stream Identifier: 3
 0... .. = Reserved: 0x0
 .000 0000 0000 0000 0010 0000 0000 0000 = Window Size Increment: 8192

- Many window size notification
- What about decrease the interval?

HTTP/2 Window Size Notification Interval

- HTTP/2 client of Go notifies window size when it receives more than 4KB of data.
- I modified the client to change the value:
 - <https://github.com/wtakase/net/commit/a9a97aa674f4b9705ad4775e5be164ab4070fd05>

Changed from [golang/net/http2/transport.go](https://golang.org/pkg/net/http2/transport.go)

```
48 48 // transportDefaultStreamMinRefresh is the minimum number of bytes we'll send
49 49 // a stream-level WINDOW_UPDATE for at a time.
50 50 transportDefaultStreamMinRefresh = 4 << 10
```

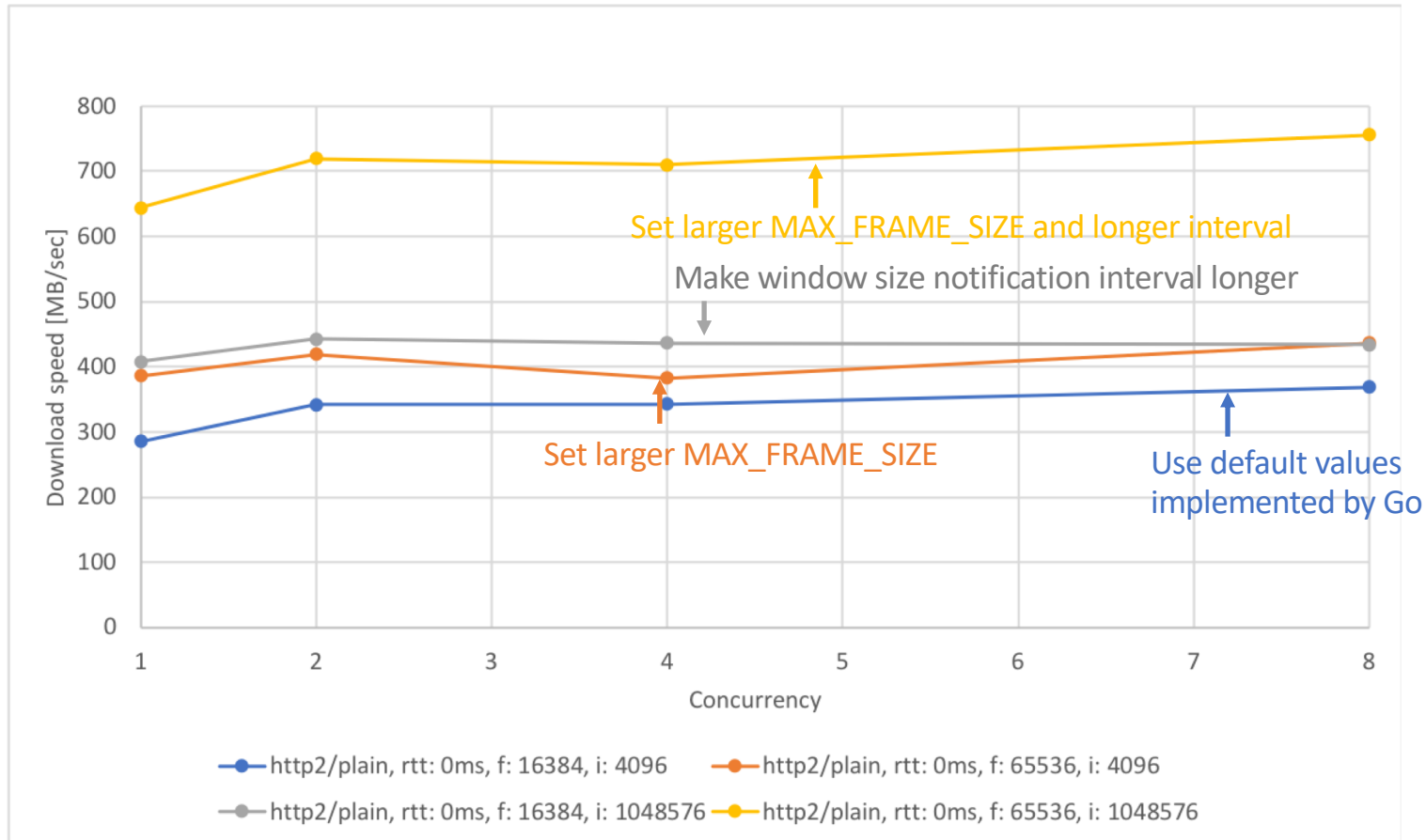
```
200 200
140 - func configureTransport(t1 *http.Transport) (*Transport, error) {
202 + func configureTransport(t1 *http.Transport, fct *FlowControlTransport) (*Transport, error) {
141 203     connPool := new(clientConnPool)
142 204     t2 := &Transport{
143 -         ConnPool: noDialClientConnPool{connPool},
144 -         t1:        t1,
205 +         ConnPool:    noDialClientConnPool{connPool},
206 +         MaxFrameSize: fct.MaxFrameSize,
207 +         TransportDefaultConnFlow: fct.TransportDefaultConnFlow,
208 +         TransportDefaultStreamFlow: fct.TransportDefaultStreamFlow,
209 +         TransportDefaultStreamMinRefresh: fct.TransportDefaultStreamMinRefresh,
210 +         t1:          t1,
145 211     }
146 212     connPool.t = t2
```

Modification of chasqui

- Uses the modified http2 GO package.
- Supports the following command-line options:
 - Frame size
 - Window size notification interval
- Supports plain-text connection
 - HTTP/1.1 and HTTP/2
- <https://github.com/wtakase/chasqui>

Test Plain-text HTTP/2 Data Transfer on LAN

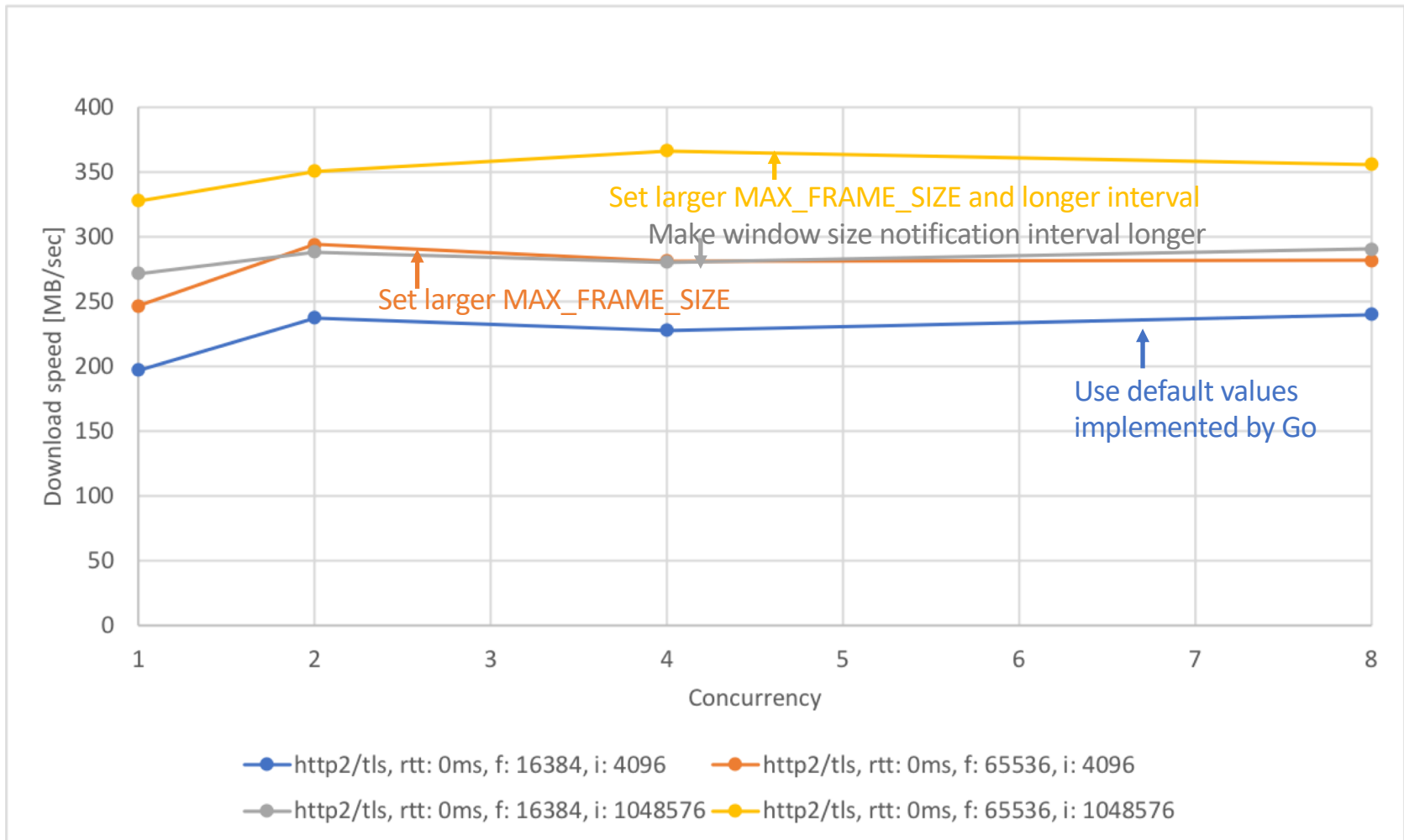
- Prepare 2 DTNs in KEK and connect them directly with QSFP+(40Gbps) and the RTT is 0ms.



RTT: 0ms, Measuring times: 1, Duration: 60 sec

```
$ chasqui driver -clients caddy02:9443 -servers caddy01:5678 -duration 60s -concurrency X ⇒  
-max-frame-size Y -win-update-interval Z -plain-http
```

Test TLS HTTP/2 Data Transfer on LAN

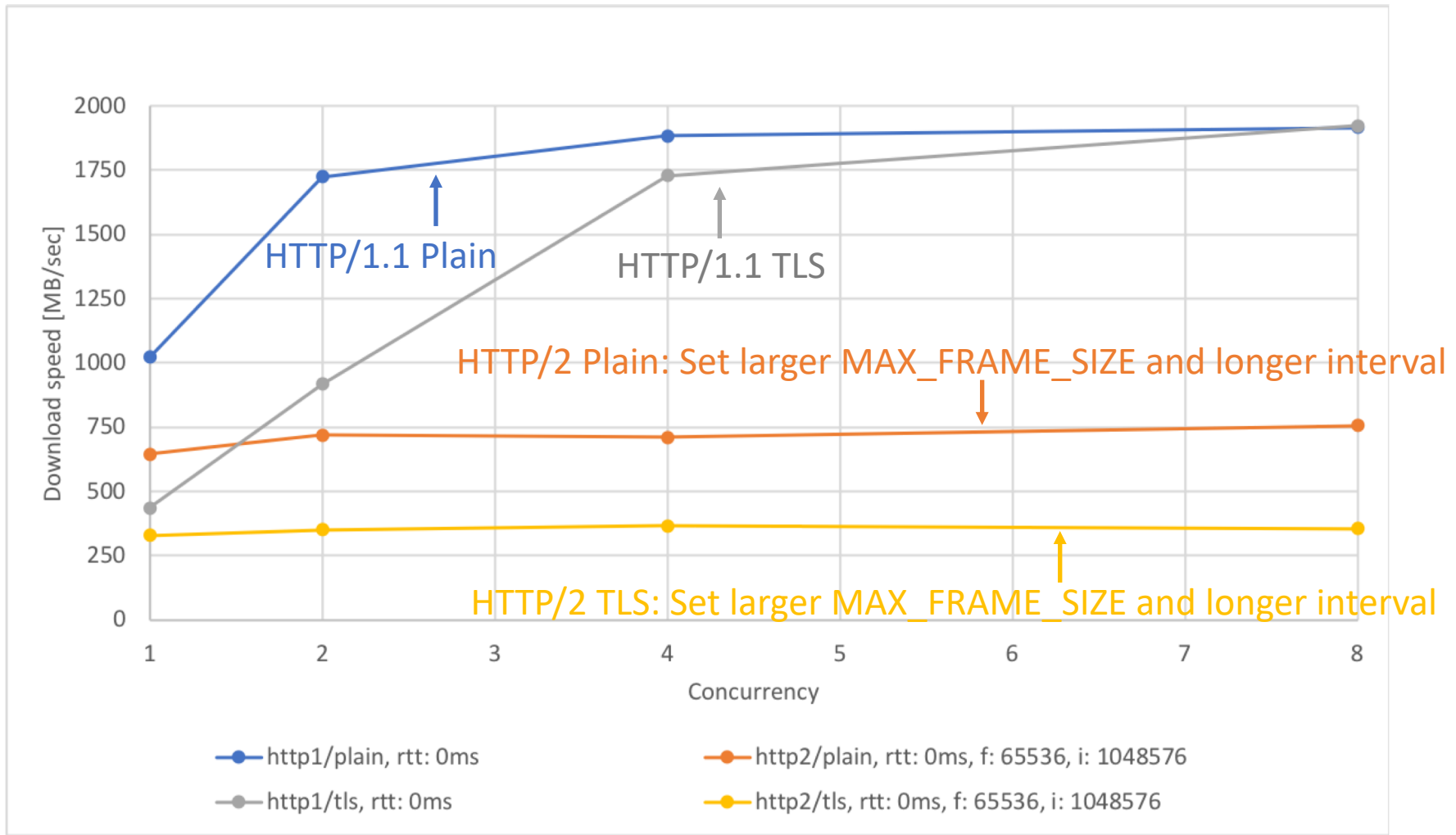


RTT: 0ms, Measuring times: 1, Duration: 60 sec

```
$ chasqui driver -clients caddy02:9443 -servers caddy01:5678 -duration 60s -concurrency X ⇒  
-max-frame-size Y -win-update-interval Z
```


Comparison of HTTP/1.1 and improved HTTP/2

- Still HTTP/1.1 got better results.
 - Single TCP connection limits the data transfer speed in HTTP/2?
- Download performances over TLS are slower than the plain-text because of encryption/decryption overhead.



Next Step: Test with 10 Gbps network: Current Status

- CC-IN2P3:
 - The DTNs have already been connected to 10 Gbps network.
 - The firewall has been configured to allow connection from the DTN in KEK.
- KEK:
 - The DTN have already been connected to 10 Gbps network.
 - The firewall was configured to allow connection from the DTNs in CC-IN2P3 on 28th November.

Summary and Future work

- We have started data transfer test between CC-IN2P3 and KEK.
- We got the first result of memory-to-memory data transfer over 1Gbps.
 - HTTP/1.1 got better performance.
- We investigated the cause of HTTP/2 performance degradation.
 - Frame size, interval of window size notification
- Future work:
 - Test with multi-servers and multi-clients
 - Test with CC-IN2P3 over 10Gbps network
 - Test with modified chasqui
 - Test with larger Kernel TCP buffer sizes
 - Disk-to-disk data transfer test