

# Chase the latency

## look for the root cause of the latency problems

Megumi Takeshita  
Packet Otaku,  
ikeriri network service

Sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

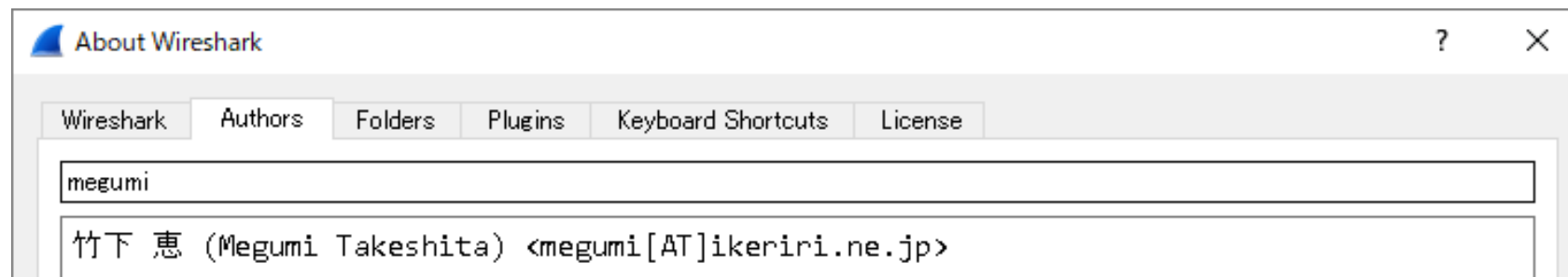
#sf25us



# Megumi Takeshita, Packet Otaku, ikeriri network service



- Worked SE/IS at BayNetwork, Nortel
- Reseller of CACE technologies in 2008
- Founder, ikeriri network service co., Ltd
- Reseller of packet capture / wireless-tools
- Wrote 10+ books about Wireshark in Japanese
- Instruct Wireshark to JSDF etc.
- Lecturer of CHUO University
- One of the contributors to Wireshark
- Translate Wireshark into Japanese



End users and application teams complain to you about the latency, but we want to prove it is not network. How about that? The latency lies everywhere, not only in the network round-trip time.

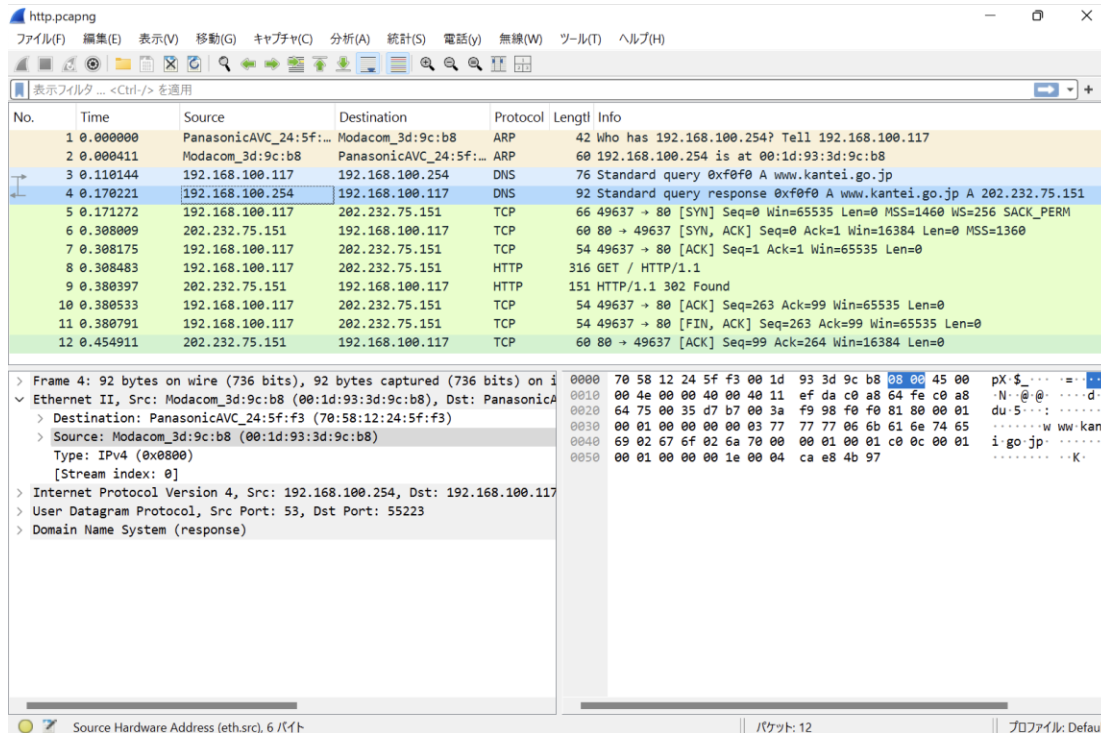
In this session, you can learn how to divide latency properly between the client OS/apps, the server OS/apps and the network side. Including reasonable ways to debug the latency problems. We can find the root cause of the latency with TCP/UDP analysis TIPS and tricks using Wireshark.

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>



# STEP1 Wireshark generated fields calculate the latency

- Let's start with a simple web access trace file. Open http.pcapng in your Wireshark
- Wireshark calculates response time



- DNS request/reply
- TCP SYN/SYN-ACK
- TCP Segment/ACK
- HTTP request/reply
- etc.


sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

- Check #4 and open DNS dissector,  
We can see the Wireshark generated field  
[Time: 0.060077000 seconds]  
***dns.time*** The time between the Query  
and the Response.



```
▼ Domain Name System (response)
  Transaction ID: 0xf0f0
  > Flags: 0x8180 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  > Queries
  > Answers
    [Request In: 3]
    [Time: 0.060077000 seconds]
```

The time means the latency between DNS client, OS resolver, and DNS server including network when we capture at a remote client side

 The time between the Query and the Response (dns.time)

# STEP1 Wireshark generated fields calculate the latency

- Check #6 and open TCP dissector, open Wireshark generated header, [SEQ/ACK analysis], we can find iRTT [iRTT: 0.136903000 seconds]



```
✓ Transmission Control Protocol, Src Port: 80, Dst Port: 49637, S
  Source Port: 80
  Destination Port: 49637
  [Stream index: 0]
  [Stream Packet Number: 2]
  > Options: (4 bytes), Maximum segment size
  > [Timestamps]
  ✓ [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 5]
    [The RTT to ACK the segment was: 0.136737000 seconds]
    [iRTT: 0.136903000 seconds]
```

***tcp.analysis.initial\_rtt***  
means the latency between  
SYN and SYN-ACK segments  
including RTT when we  
capture at a client side

How long it took for the SYN to ACK handshake (iRTT) (tcp.analysis.initial\_rtt)


sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>




# STEP1 Wireshark generated fields calculate the latency

- Check #7 and open TCP dissector, open Wireshark generated header, [SEQ/ACK analysis], we can find ack\_rtt [The RTT to ACK the segment was: 0.000166000]



✓ Transmission Control Protocol, Src Port: 49637, Dst Port: 80  
Source Port: 49637  
Destination Port: 80  
[Stream index: 0]  
  
[Checksum Status: Unverified]  
Urgent Pointer: 0  
> [Timestamps]  
✓ [SEQ/ACK analysis]  
    [This is an ACK to the segment in frame: 6]  
    [The RTT to ACK the segment was: 0.000166000 seconds]  
    [iRTT: 0.136903000 seconds]

*tcp.analysis.ack\_rtt*  
means the latency between  
the segment and related  
ACK (this packet)

 How long time it took to ACK the segment (RTT) (tcp.analysis.ack\_rtt)


# STEP1 Wireshark generated fields calculate the latency

- Check #9 and open the HTTP dissector, open Wireshark generated field, [Time since request: 0.071914000 seconds] ***http.time*** (Time since the request was sent)



```
▼ Hypertext Transfer Protocol
  > HTTP/1.1 302 Found\r\n
    Connection: close\r\n
    Pragma: no-cache\r\n
    cache-control: no-cache\r\n
    Location: /\r\n
    \r\n
    [Request in frame: 8]
    [Time since request: 0.071914000 seconds]
    [Request URI: /]
    [Full request URI: http://www.kantei.go.jp/]
```

http.time means the latency between HTTP request and HTTP response, including RTT when we capture at a client side.

 Time since the request was sent (http.time)

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>



- Create the table and write out the latencies

Field name	Value (seconds)	Description	Meaning
<b><i>dns.time</i></b>	0.060077000	The time between the Query and the Response.	The latency between the DNS client and the DNS server, including RTT, when we capture at the client side
<b><i>tcp.analysis.initial_rtt</i></b>	<b>0.13690300 highest</b>	The time between SYN and SYN-ACK at the initial phase of 3way handshake	The latency between SYN and SYN-ACK segments, including network, when we capture at the client side
<b><i>tcp.analysis.ack_rtt</i></b>	<b>0.000166000</b>	The time between TCP segment and the related ACK	The latency between the segment and the related ACK (sometimes RTT is not included)
<b><i>http.time</i></b>	0.071914000	The time between HTTP request and HTTP response.	The latency between the HTTP request and the response, including network, when we capture at the client side.



# STEP1 Wireshark generated fields calculate the latency



- Divide the source of the delays.

Field name	Value (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<b><i>dns.time</i></b>	0.060077000	Yes	No	No	Yes	Yes	DNS latency + network
<b><i>tcp.analysis.initial_rtt</i></b>	<b>0.13690300</b>	Yes	No	No	Yes	No	Server OS's initial socket processing time + RTT
<b><i>tcp.analysis.ack_rtt</i></b>	<b>0.000166000</b>	No	Yes	No	No	No	Client OS's socket processing time
<b><i>http.time</i></b>	<b>0.071914000</b>	Yes	No	No	Yes	Yes	HTTP latency + network

In this trace, `tcp.analysis.initial_rtt` is the highest, Server OS initial socket processing time and RTT use the most time, and HTTP response time is higher than DNS.

- How about ARP? Check #2 ARP response  
There is no field about ARP latency.



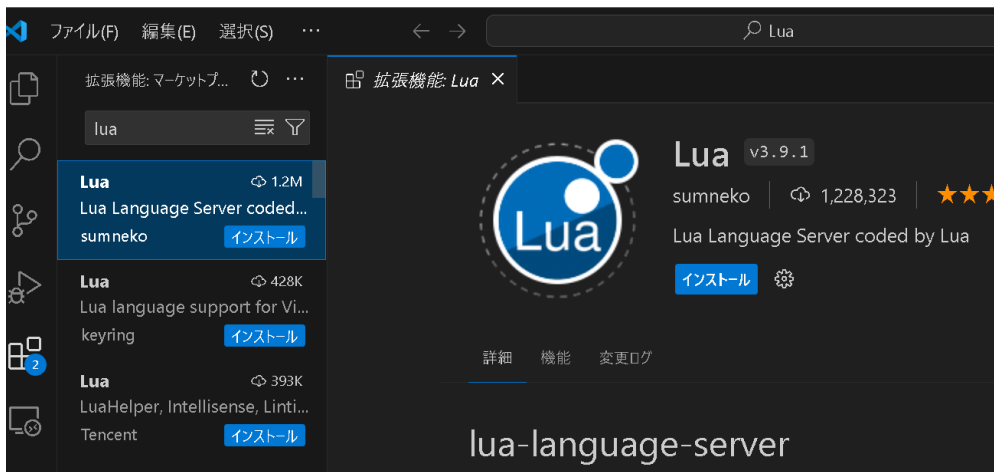
We can see the latency from Frame dissector  
[Time delta from previous displayed frame]  
frame.time\_delta\_displayed field

```
✓ Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: Modacom_3d:9c:b8 (00:1d:93:3d:9c:b8)
  Sender IP address: 192.168.100.254
  Target MAC address: PanasonicAVC_24:5f:f3 (70:58:12:24:5f:f3)
  Target IP address: 192.168.100.117
```

How can we do this?  
Extend your Wireshark  
Create Wireshark Post  
Dissector by Lua

## STEP2 Extend Wireshark fields by Post Dissector

- We can add Post Dissector to extend the current dissection, add additional protocols and fields.
- Dissectors are used for analyzing new protocols, but Post Dissectors are used for additional processing after Wireshark finishes dissections.



Install Visual Studio Code and  
Lua Extensions

Help>About Wireshark>Folder  
Check Personal Lua Plugins

C:\Users\username\AppData\Roaming\Wireshark\plugins

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

## STEP2 Extend Wireshark fields by Post Dissector



```
C: > Users > megumitakeshita > AppData > Roaming > Wireshark > plugins > arp_latency.lua > [e] arp
1  -- Define Post Dissector
2  arp_latency_proto = Proto("arp_latency", "ARP Latency")
3  -- Define additinal field
4  latency_field = ProtoField.double("arp_latency.time", "Time", base.NONE)
5  -- Add additional field
6  arp_latency_proto.fields = { latency_field }
```

- We can refer Wireshark Developer Guide.
- Define Post Dissector ARP Latency (arp\_latency)
- Define a field Time (arp\_latency.time) as double
- Add Time field into ARP Latency dissecotor.



```
-- Get fields for calculation
local arp_opcode_f = Field.new("arp.opcode")
local arp_src_ip_f = Field.new("arp.src.proto_ipv4")
local arp_dst_ip_f = Field.new("arp.dst.proto_ipv4")
local frame_time_f = Field.new("frame.time_epoch")
-- Save fields as table
local arp_requests = {}
```

- Get fields from current ARP, Frame dissectors. (These variables are used for response time calculation in Post Dissector)
- Create tables arp\_requests for saving results





## STEP2 Extend Wireshark fields by Post Dissector

```
function arp_latency_proto.dissector(buffer, pinfo, tree)
    local opcode = arp_opcode_f()
    local src_ip = arp_src_ip_f()
    local dst_ip = arp_dst_ip_f()
    local time_epoch = frame_time_f()

    if not opcode or not src_ip or not dst_ip or not time_epoch then return end

    local opcode_val = opcode.value
    local src = tostring(src_ip)
    local dst = tostring(dst_ip)
    local t = tonumber(tostring(time_epoch))
```



- Get fields and input into local variables
- Check the packet is ARP to check there are fields, opcode\_val (arp.opcode), src (ip.src), dst (ip.dst), and t ( (string) frame.time\_epoch )

## STEP2 Extend Wireshark fields by Post Dissector



```
if opcode_val == 1 then -- ARP Request
    -- save request with SrcIP_DestIP
    arp_requests[dst .. "_" .. src] = { time = t, frame = pinfo.number }
elseif opcode_val == 2 then -- ARP Reply
    local key = src .. "_" .. dst
    local req = arp_requests[key]
    if req then
        local latency = t - req.time
        -- add results as field
        local subtree = tree:add(arp_latency_proto, "ARP Latency")
        subtree:add(latency_field, latency)
        -- Delete used record
        arp_requests[key] = nil
    end
end
end
end
```

- If ARP request, save epoch time and frame number with the key(ip.dst\_ip.src), else if ARP response, save and calculate time and add header, field

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

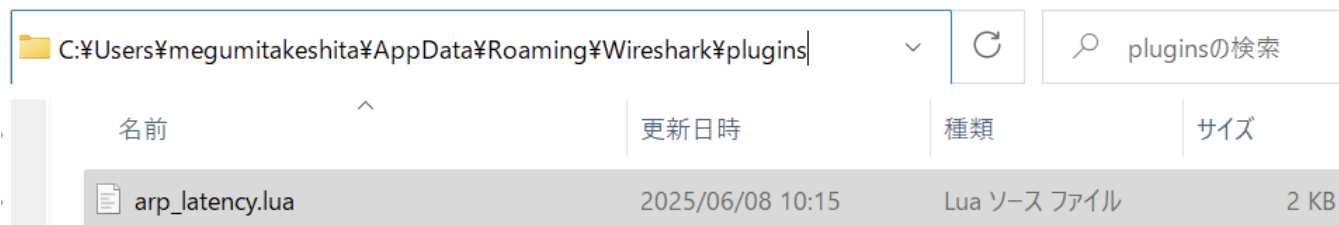
# STEP2 Extend Wireshark fields by Post Dissector



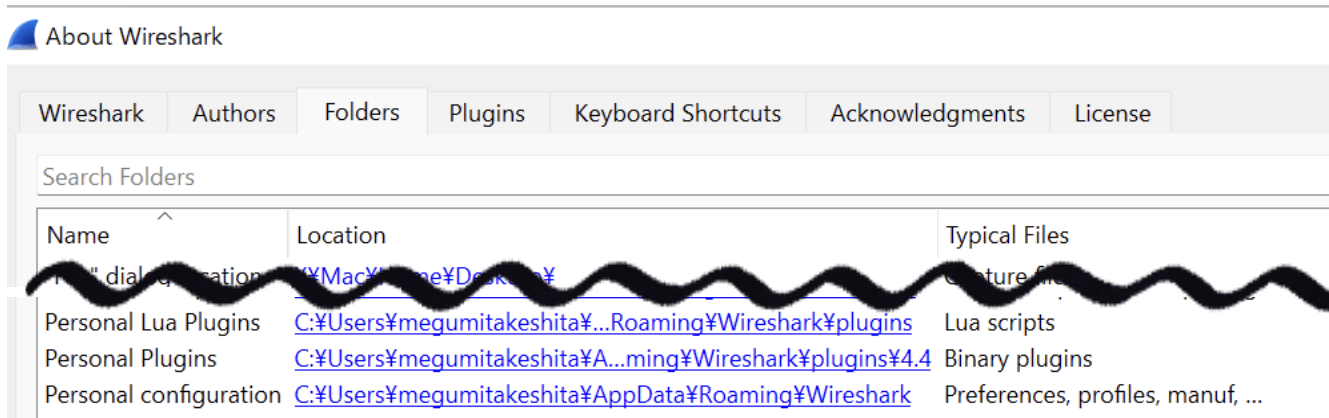
```
register_postdissector(arp_latency_proto)
```



- Add this dissector as PostDissector



- Copy arp\_latency.lua into your personal lua plugins



C:\Users\username\AppData  
\Roaming\Wireshark\plugins

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

- Close Wireshark and open http.pcapng again  
reload trace and check #2 packet detail pane



```
> Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0  
> Ethernet II, Src: Modacom_3d:9c:b8 (00:1d:93:3d:9c:b8), Dst: PanasonicAVC_24:5f:f3 (70:58:12:24:5f:f3)  
▼ Address Resolution Protocol (reply)
```

```
  Hardware type: Ethernet (1)  
  Protocol type: IPv4 (0x0800)  
  Hardware size: 6  
  Protocol size: 4  
  Opcode: reply (2)  
  Sender MAC address: Modacom_3d:9c:b8 (00:1d:93:3d:9c:b8)  
  Sender IP address: 192.168.100.254  
  Target MAC address: PanasonicAVC_24:5f:f3 (70:58:12:24:5f:f3)  
  Target IP address: 192.168.100.117
```

### ▼ ARP Latency

```
  Time: 0.000411033630371094
```

arp_latency.time						
No.	Time	Source	Destination	Protocol	Length	Info
2	0.000411	Modacom_3d:9c:b8	PanasonicAVC_24:5f:f3	ARP	60	192.168.100.254 is at 00:1d:93:3d:9c:b8

PostDissector adds  
ARP Latency header  
and Time field  
0.000411033630371094 seconds

We can filter with  
arp\_latency.time

## STEP2 Extend Wireshark fields by Post Dissector

- Think about the meaning of ARP latency.  
(Note: this trace is captured at the client side)



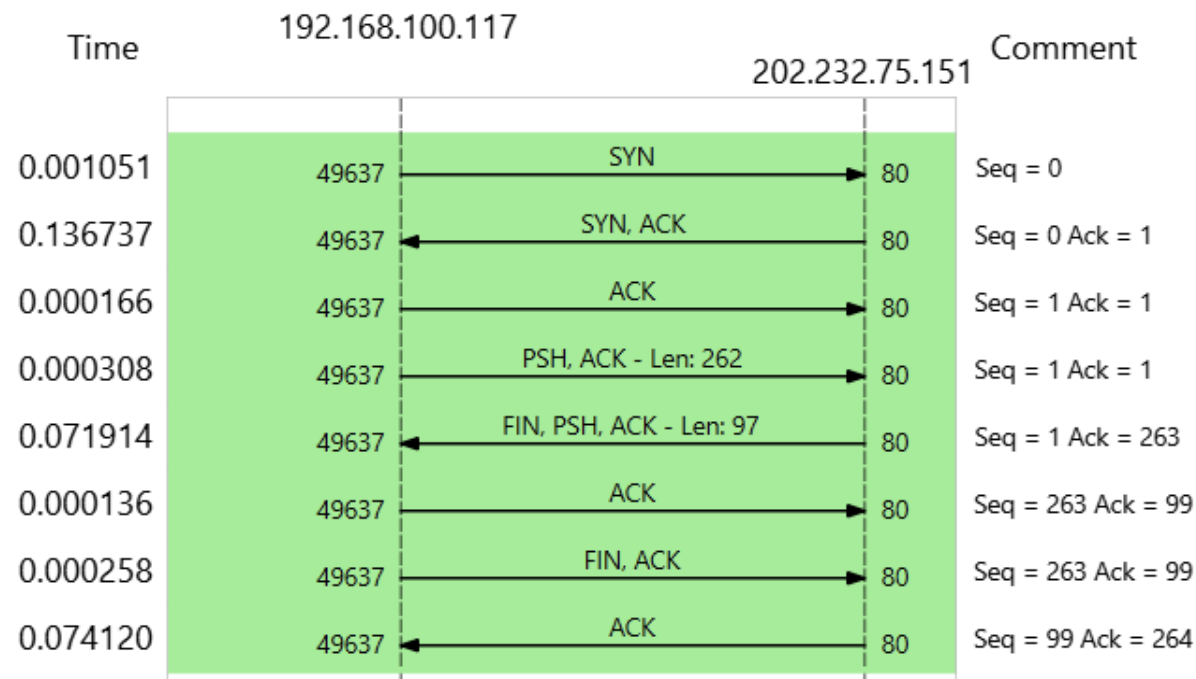
Field name	Value (seconds)	Description	Meaning
<i>arp_latency.time</i>	0.000411033	The time between the ARP request and the ARP response	Client NIC set target IP address (Default GW's IP) into ARP header, sends ARP request with broadcast, Default GW(usually Router) sends back ARP response with GW's MAC address by unicast

- This latency is caused by layer 2, between ClientNIC and default GWs (Router's) interface.

Field name	Value (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<i>arp_latency.time</i>	0.000411033	Yes (LAN)	No (NIC)	No	No (GW)	No	ARP response time is Layer 2 latency between Client NIC and default GW's NIC(Router)

## STEP3 Dividing TCP connection's latency at the client side

- Choose View>Time Display Format, “Seconds Since Previous Displayed Packet”
- Choose Statistics>Flow Graph, select “TCP Flows” from the flow type list box, show TCP Flow Graph.



Let's divide TCP latency  
the interval between..

- 1:SYN<>SYN/ACK
- 2:SYN/ACK<>ACK
- 3:ACK<>First segment
- 4:First segment<>ACK



- Note: This trace was captured at the client side.

Interval of TCP segments	Value (seconds)	Meaning
<b><i>SYN&lt;&gt;SYN/ACK</i></b>	<b>0.136737</b>	The client sends a SYN packet, and then the Server receives a SYN packet via the network. ServerOS creates the Socket, assigns the CPU and memory, and sends back a SYN/ACK with the initial sequence number and ack number.
<b><i>SYN/ACK&lt;&gt;ACK</i></b>	0.000166	ClientOS receive SYN/ACK and sets the ack number with the Server's initial sequence number +1, and creates and sends an ACK
<b><i>ACK&lt;&gt;First segment ACK&lt;&gt;PSH,ACK Len:262</i></b>	0.000308	ClientApps(web browser) receives the socket from ClientOS, creates and sets the value into Application data(HTTP request), and sends data using the socket API.
<b><i>First segment&lt;&gt;ACK PSH,ACK Len:262 &lt;&gt; FIN,PSH,ACK Len:97</i></b>	0.071914	ServerOS receives the First segment from the Client via the network, adds the ACK number by the segment size, sends the data to the ServerApps (web server) when the PSH flag is set. ServerApps create the response message and ServerOS set FIN/PSH/ACK flag and sends the segment via network

What factor causes the time of this latency?

Network, ClientOS, ClientApps(web browser),  
ServerOS or ServerApps(web server)



- Create table and divide the cause of latency



Interval of TCP segments	Value (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<b><i>SYN&lt;&gt;SYN/ACK</i></b>	<b>0.136737</b>	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<b><i>SYN/ACK&lt;&gt;ACK</i></b>	0.000166	No	Yes	No	No	No	Client OS socket processing time
<b><i>ACK&lt;&gt;First segment</i></b> <i>ACK&lt;&gt;PSH,ACK Len:262</i>	0.000308	No	No	Yes	No	No	Client App (browser) request creation time
<b><i>First segment&lt;&gt;ACK</i></b> <i>PSH,ACK Len:262 &lt;&gt;</i> <i>FIN,PSH,ACK Len:97</i>	0.071914	Yes	No	No	Yes	Yes	Network + ServerOS socket processing + ServerApps

**Server OS initial socket processing + network**  
are the highest latencies in this client-side trace,

## STEP1-3 Write up all delay into one table

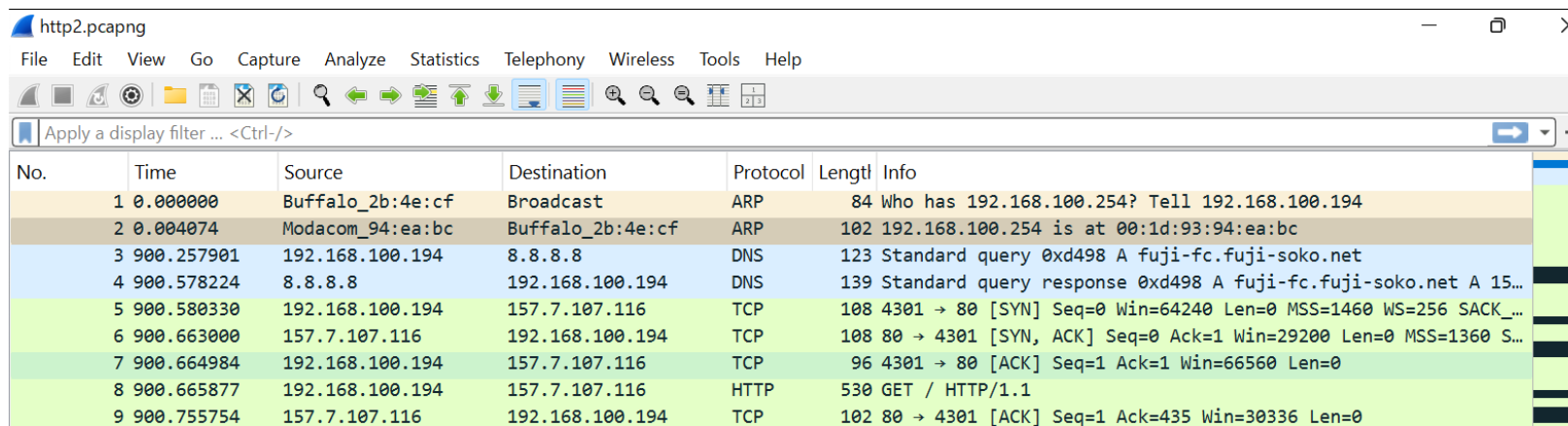


Field name Interval of TCP segs	Value (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<i>arp_latency.time</i>	0.000411033	Yes (LAN)	No (NIC)	No	No (GW)	No	Layer 2 latency between Client NIC and default GW
<i>dns.time</i>	0.060077000	Yes	No	No	Yes	Yes	DNS latency + network
<b><i>SYN&lt;&gt;SYN/ACK</i></b>	<b>0.136737</b>	Yes	No	No	Yes	No	<b>Server OS initial socket processing time</b> + network
<b><i>SYN/ACK&lt;&gt;ACK</i></b>	0.000166	No	Yes	No	No	No	Client OS socket processing time
<i>ACK&lt;&gt;First Segment ACK&lt;&gt;PSH,ACK Len:262</i>	0.000308	No	No	Yes	No	No	Client App (browser) request creation time
<i>First segment&lt;&gt;ACK PSH,ACK Len:262 &lt;&gt; FIN,PSH,ACK Len:97</i>	0.071914	Yes	No	No	Yes	Yes	Network + ServerOS socket processing + ServerApps
<b><i>http.time</i></b>	0.071914000	Yes	No	No	Yes	Yes	HTTP latency + network

**Server OS initial socket processing latency is the highest**

## STEP4 Compare another trace file

- When troubleshooting the root cause of latency, compare two trace files with different settings, conditions, and environments.
- Open http2.pcapng, an HTTP web access trace with different websites, client/server and environment ( IEEE802.11b 5ch wireless trace)



The image shows a Wireshark window titled 'http2.pcapng'. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help) and a toolbar with various icons. Below the toolbar is a display filter bar showing 'Apply a display filter ... <Ctrl-/>'. The main packet list table is displayed with the following data:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Buffalo_2b:4e:cf	Broadcast	ARP	84	Who has 192.168.100.254? Tell 192.168.100.194
2	0.004074	Modacom_94:ea:bc	Buffalo_2b:4e:cf	ARP	102	192.168.100.254 is at 00:1d:93:94:ea:bc
3	900.257901	192.168.100.194	8.8.8.8	DNS	123	Standard query 0xd498 A fuji-fc.fuji-soko.net
4	900.578224	8.8.8.8	192.168.100.194	DNS	139	Standard query response 0xd498 A fuji-fc.fuji-soko.net A 15...
5	900.580330	192.168.100.194	157.7.107.116	TCP	108	4301 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_...
6	900.663000	157.7.107.116	192.168.100.194	TCP	108	80 → 4301 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1360 S...
7	900.664984	192.168.100.194	157.7.107.116	TCP	96	4301 → 80 [ACK] Seq=1 Ack=1 Win=66560 Len=0
8	900.665877	192.168.100.194	157.7.107.116	HTTP	530	GET / HTTP/1.1
9	900.755754	157.7.107.116	192.168.100.194	TCP	102	80 → 4301 [ACK] Seq=1 Ack=435 Win=30336 Len=0

<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>



## STEP4 Compare another trace file

- This trace contains different Layer 2 headers, another client/server, and a website.



http.pcapng

File Edit View Go Capture Analyze Stat

Apply a display filter ... <Ctrl-/>

No.	Time	Source
1	0.000000	PanasonicAVC_24
2	0.000411	Modacom_3d:9c:b8
3	0.001144	102.168.100.115

> Frame 2: 60 bytes on wire (480 bits),  
> Ethernet II, Src: Modacom\_3d:9c:b8 (0  
> Address Resolution Protocol (reply)  
✓ ARP Latency  
Time: 0.000411033630371094

http2.pcapng

File Edit View Go Capture Analyze Stat

Apply a display filter ... <Ctrl-/>


No.	Time	Source	De
1	0.000000	Buffalo_2b:4e:cf	Br
2	0.004074	Modacom_94:ea:bc	Bu

> Frame 2: 102 bytes on wire (816 bits).  
> Radiotap Header v0, Length 20  
> 802.11 radio information  
> IEEE 802.11 Data, Flags: .....F.C  
> Logical-Link Control  
> Address Resolution Protocol (reply)  
✓ ARP Latency  
Time: 0.0040740966796875

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

For example,  
wired trace(http1)  
contains EthernetII,  
wireless trace  
(http2) contains  
Radiotap, 802.11  
radio information,  
IEEE802.11 data  
and LLC header

## STEP4 Compare another trace file

- By comparing two different trace files, we can understand the another aspects of the latency 

Trace file	<i>arp_latency.time</i> (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<i>http.pcapng</i>	0.000411033	Yes (LAN)	No (NIC)	No	No (GW)	No	ARP response time is Layer 2 latency between the Client (Wired) NIC and default GW
<i>http2.pcapng</i>	0.004074096	Yes (WiFi)	No (WLAN)	No	No (GW)	No	ARP response time is Layer 2 latency between the Client (WLAN) NIC and default GW

Full Duplex 100BASE-TX wired EthernetII is about 10 times faster than half duplex IEEE802.11b wireless.



## STEP4 Compare another trace file



- How about DNS response, check dns.time of both trace files and write out

### Domain Name System (response)

```
Transaction ID: 0xf0f0
> Flags: 0x8180 Standard query
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
> Queries
> Answers
[Request In: 3]
[Time: 0.060077000 seconds]
```

### Domain Name System (response)

```
Transaction ID: 0xd498
> Flags: 0x8180 Standard query
Questions: 1
Answer RRs: 1
Authority RRs: 0
Additional RRs: 0
> Queries
> Answers
[Request In: 3]
[Time: 0.320323000 seconds]
```

Both traces use the same DNS server 8.8.8.8, so the **network environment** mainly causes the difference

Trace file	dns.time (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<i>http.pcapng</i>	0.060077000	Yes	No	No	Yes	Yes	DNS latency + network
<i>http2.pcapng</i>	0.320323000	Yes	No	No	Yes	Yes	DNS latency + network

## STEP4 Compare another trace file



- Next, compare HTTP response time, check http.time of both trace files and write out

```
Frame 9: 151 bytes on wire (1208 bits), 151 byte captured on interface  
Ethernet II, Src: Modacom_3d:9c:b8 (00:1d:93:3d:9c:b8), Dst: 08:00:27:00:00:00  
Internet Protocol Version 4, Src: 202.232.75.15, Dst: 157.7.107.116  
Transmission Control Protocol, Src Port: 80, Dst Port: 80, Seq: 10390, Len: 10390  
Hypertext Transfer Protocol
```

```
> HTTP/1.1 302 Found\r\n  
Connection: close\r\n  
Pragma: no-cache\r\n  
cache-control: no-cache\r\n  
Location: /\r\n  
\r\n  
[Request in frame: 8]
```

```
[Time since request: 0.071914000 seconds]  
[Request URI: /]  
[Full request URI: http://www.kantei.go.jp/]
```

http2.pcapng  
contains  
10037 tcp  
bytes in flight

```
Frame 20: 966 bytes on wire (7728 bits), 966 bytes captured on interface  
Radiotap Header v0, Length 20  
802.11 radio information  
IEEE 802.11 Data, Flags: .....F.C  
Logical-Link Control  
Internet Protocol Version 4, Src: 157.7.107.116, Dst: 202.232.75.15  
Transmission Control Protocol, Src Port: 80, Dst Port: 80, Seq: 10390, Len: 10390  
[8 Reassembled TCP Segments (10390 bytes): #10(1360), #11(1360), #12(1360), #13(1360), #14(1360), #15(1360), #16(1360), #17(1360)]  
Hypertext Transfer Protocol
```

```
> HTTP/1.1 200 OK\r\n  
Date: Fri, 13 Jul 2018 05:55:19 GMT\r\n  
Content-Type: text/html; charset=UTF-8\r\n  
> Content-Length: 10037\r\n  
Connection: keep-alive\r\n  
X-Powered-By: PHP/5.3.29\r\n  
Link: <http://fuji-fc.fuji-soko.net/wp-json/>; rel="alternate"; type="application/json"; title="Fuji FC" \r\n  
Vary: Accept-Encoding\r\n  
Content-Encoding: gzip\r\n  
Server: Apache\r\n  
\r\n
```

```
[Request in frame: 8]  
[Time since request: 1.655489000 seconds]  
[Request URI: /]  
[Full request URI: http://fuji-fc.fuji-soko.net/]  
Content-encoded entity body (gzip): 10037 bytes -> 10037 bytes  
File Data: 43463 bytes
```

Line-based text data: text/html (711 lines) ns are

<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

[www.kantei.go.jp](http://www.kantei.go.jp)  
responds just one  
302 Found response.  
[fuji-fc.fuji-soko.net](http://fuji-fc.fuji-soko.net)  
responds 200 OK,  
full contents of  
body(text/html) by 8  
Reassembled TCP  
segments (8xRTT)

## STEP4 Compare another trace file

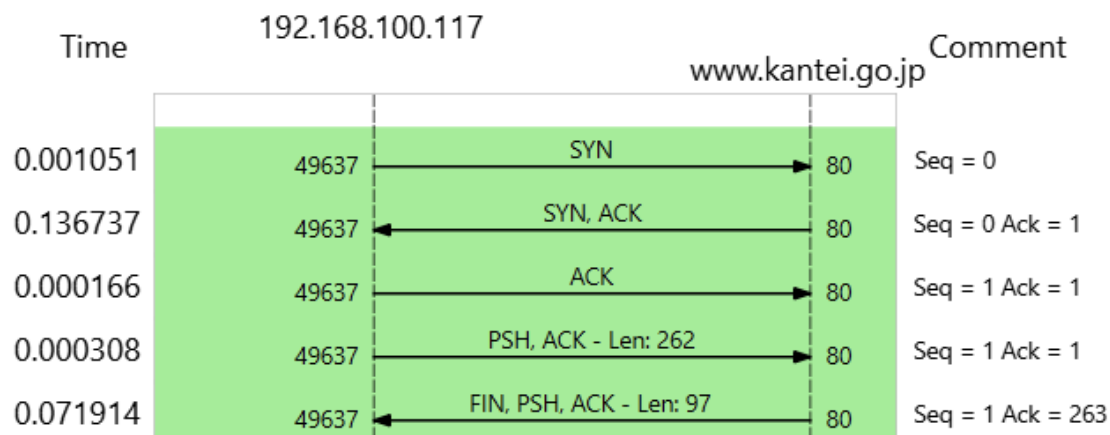
Trace file	http.time (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
<i>http.pcapng</i> <i>www.kantei.go.jp</i>	<b>0.071914000</b>	Yes	No	No	Yes	Yes	HTTP latency + network 302 found by 1 segment.
<i>http2.pcapng</i> <i>fuji-fc.fuji-soko.net</i>	<b>1.655489000</b>	Yes	No	No	Yes	Yes	HTTP latency + network 200 OK by 8 segments

- Compare both HTTP response time,
- A wired trace is about 23 times faster than a wireless one. A wired connection contains just one turn TCP segment, with a 97-byte payload. In contrast, a wireless connection contains eight TCP segments, with 10,037 bytes of data.



# STEP4 Compare another trace file

Wireshark · Flow · http.pcapng



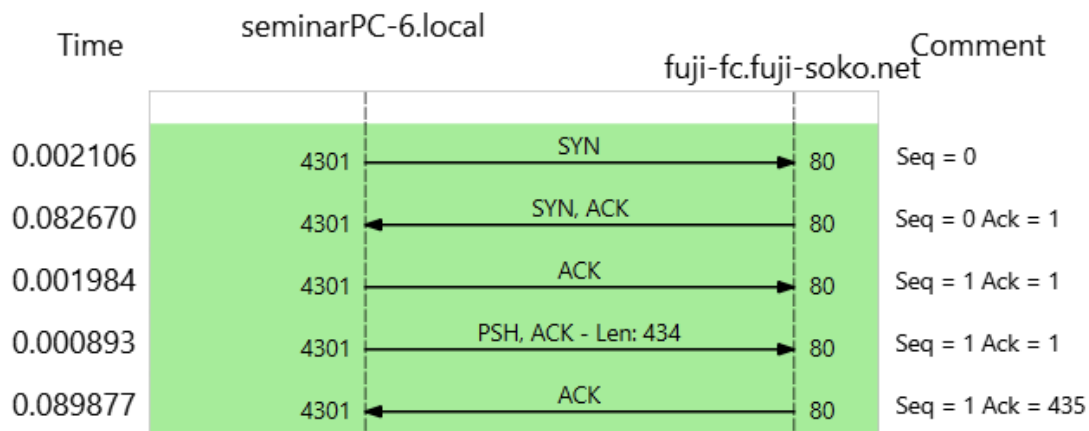
Then compare TCP latency  
Change Time Display  
Format, Name Resolution,  
Statistics>Flow Graph to  
create TCP flow graph

Interval of TCP segments	http.pcapng Wired	network	Client OS	Client App	Server OS	Server App	Meaning
<b><i>SYN&lt;&gt;SYN/ACK</i></b>	0.136737	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<b><i>SYN/ACK&lt;&gt;ACK</i></b>	0.000166	No	Yes	No	No	No	Client OS socket processing time
<b><i>ACK&lt;&gt;1st segment (PSH/ACK)</i></b>	0.000308	No	No	Yes	No	No	Client App (browser) request creation time
<b><i>1st segment(PSH/ACK) &lt;&gt;2<sup>nd</sup> segment(PSH/ACK)</i></b>	0.071914	Yes	No	No	Yes	Yes	ServerOS socket processing time + ServerApp(web server) response creation time (304 Found) + network

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

# STEP4 Compare another trace file

Wireshark · Flow · http2.pcapng



In http2.pcapng, the server sends back just an ACK after the HTTP request

There are 8 TCP segment in HTTP response later

Interval of TCP segments	http2.pcapng Wireless	network	Client OS	Client App	Server OS	Server App	Meaning
<b>SYN&lt;&gt;SYN/ACK</b>	0.082670	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<b>SYN/ACK&lt;&gt;ACK</b>	0.001984	No	Yes	No	No	No	Client OS socket processing time
<b>ACK&lt;&gt;1st segment (PSH/ACK)</b>	0.000893	No	No	Yes	No	No	Client App (browser) request creation time
<b>1st segment(PSH/ACK) &lt;&gt;ACK</b>	0.089877	Yes	No	No	Yes	No	ServerOS socket processing time + network (No ServerApp in this trace)

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

## STEP4 Compare another trace file

- Then, try dividing the TCP delays of both traces



Interval of TCP segments	http.pcapng Wired	http2.pcapn g Wireless	network	Client OS	Client App	Server OS	Server App	Meaning
<i>SYN&lt;&gt;SYN/ACK</i>	0.136737	0.082670	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<i>SYN/ACK&lt;&gt;ACK</i>	0.000166	0.001984	No	Yes	No	No	No	Client OS socket processing time
<i>ACK&lt;&gt;1st segment (PSH/ACK)</i>	0.000308	0.000893	No	No	Yes	No	No	Client App (browser) request creation time
<i>http.pcapng 1st segment(PSH/ACK) &lt;&gt;2<sup>nd</sup> segment(PSH/ACK) http2.pcapng 1st segment(PSH/ACK) &lt;&gt;ACK</i>	0.071914	0.089877	Yes	No	No	Yes	wired No wireless Yes	http.pcapng: ServerOS socket processing time + ServerApp(web server) response creation + network http2.pcapng: ServerOS socket processing time + network

- Pick up each latency in the TCP connection in both trace files and write out the table.



# STEP4 Compare another trace file



Interval of TCP segments	http.pcapng Wired	http2.pcapng Wireless	network	Client OS	Client App	Server OS	Server App	Meaning
<b>SYN&lt;&gt;SYN/ACK</b>	<b>0.136737</b>	0.082670	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<b>SYN/ACK&lt;&gt;ACK</b>	0.000166	0.001984	No	Yes	No	No	No	Client OS socket processing time
<b>ACK&lt;&gt;1<sup>st</sup> segment</b>	0.000308	0.000893	No	No	Yes	No	No	Client App (browser) request creation time
<i>http.pcapng</i> 1st segment(PSH/ACK) <>2 <sup>nd</sup> segment(PSH/ACK) <i>http2.pcapng</i> 1st segment(PSH/ACK) <>ACK	0.071914	0.089877	Yes	No	No	Yes	wired Yes wireless No	http.pcapng Server OS socket processing time + Server App response + network http2.pcapng Or Server OS socket processing time + network

www.kantei.go.jp (Wired) SYN<>SYN/ACK interval time takes more time than fuji-fc.fuji-soko.net (Wireless)

The Prime Minister's office website may use CDN (Contents Delivery Network) So it takes the biggest.

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

# STEP4 Compare another trace file



Field name Interval of TCP segments	http.pcapng www.kantei.go.jp Wired network	http2.pcapng fuji-fc.fuji- soko.net WiFi	network	Client OS	Client App	Server OS	Server App	Meaning
<i>arp_latency.time</i>	0.000411033	<b>0.004074096</b>	Yes (LAN)	No (NIC)	No	No (GW)	No	Layer 2 latency between Client NIC and defaultGW
<i>dns.time</i>	0.060077000	<b>0.320323000</b>	Yes	No	No	Yes	Yes	DNS latency + network
<i>SYN&lt;&gt;SYN/ACK</i>	<b>0.136737</b>	0.082670	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<i>SYN/ACK&lt;&gt;ACK</i>	0.000166	<b>0.001984</b>	No	Yes	No	No	No	Client OS socket processing time
<i>ACK&lt;&gt;1<sup>st</sup> segment</i>	0.000308	<b>0.000893</b>	No	No	Yes	No	No	Client App (browser) request creation time
<i>http.pcapng 1st segment &lt;&gt; 2nd segment(PSH/ACK) http2.pcapng 1st segment(PSH/ACK) &lt;&gt;ACK</i>	0.071914	<b>0.089877</b>	Yes	No	No	Yes	http.pcapng Yes http2.pcapng No	http.pcapng Server OS socket processing time + Server App response + network http2.pcapng Or Server OS socket processing time + network
<i>http.time</i>	0.071914000	<b>1.655489000</b>	Yes	No	No	Yes	Yes	HTTP latency + network

• We can find another aspect of the latencies.

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

- We may encounter a stuck when we capture at just one location for troubleshooting.
- Packet-based troubleshooting is a black box test; we need to find the cause from the outside traces.
- This time we use simple ftp connection trace files, “clientside.pcapng” and “serverside.pcapng”
- Compare different point traces of the same traffic. ClientIP(10.211.55.4), ServerIP(192.168.43.134)  
The client is Windows 11 on a Parallels Desktop, and the server is macOS 15.5, on a MacBook.



# STEP5 Compare different point of capturing

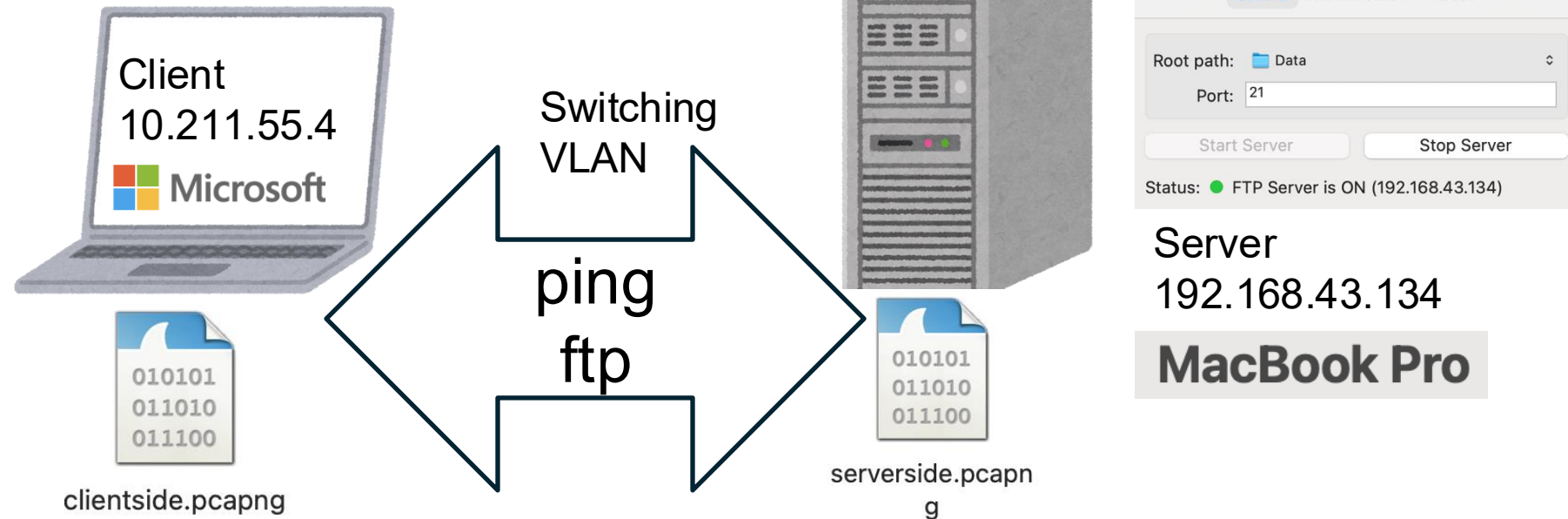

```
FLARE-VM 2025/06/09 7:37:41.91
C:\Users\megumitakeshita>ping 192.168.43.134

192.168.43.134 に ping を送信しています 32 バイトのデータ:
192.168.43.134 からの応答: バイト数 =32 時間 =2ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 =3ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 <1ms TTL=128

192.168.43.134 の ping 統計:
    パケット数: 送信 = 4、受信 = 4、損失 = 0 (0% の損失)、
    ラウンドトリップの概算時間 (ミリ秒):
        最小 = 0ms、最大 = 3ms、平均 = 1ms

FLARE-VM 2025/06/09 7:37:41.91
C:\Users\megumitakeshita>ftp 192.168.43.134
192.168.43.134 に接続しました。
220 Browser Ftp Server.
530 Please login with USER and PASS.
ユーザー (192.168.43.134:(none)): ikeriri
331 Password required for this user.
パスワード:

230 User Logged In.
ftp> bye
221 Goodbye.
```



- We use a virtual LAN network environment on one PC. So there is no router. IP TTL, IPID, IP header checksum and client TCP port are not changed in both sides trace, though the capture location is different

sample traces and configurations are  
<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>



# STEP5 Compare different point of capturing



- Let's start with Layer2, check #2 ARP packets

clientside.pcapng

File Edit View Go Capture Analyze Sta

Apply a display filter ... <Ctrl-/>

No.	Time	Source
1	0.000000	Parallels_99:30...
2	0.000351	Parallels_00:00...

> Frame 2: 60 bytes on wire (480 bits),  
> Ethernet II, Src: Parallels\_00:00:18  
> Address Resolution Protocol (reply)  
✓ ARP Latency  
Time: 0.000351190567016602

serverside.pcapng

File Edit View Go Capture Analyze Sta

Apply a display filter ... <Ctrl-/>

No.	Time	Source
1	0.000000	Parallels_99:30...
2	0.000216	Parallels_00:00...

> Frame 2: 42 bytes on wire (336 bits),  
> Ethernet II, Src: Parallels\_00:00:18  
> Address Resolution Protocol (reply)  
✓ ARP Latency  
Time: 0.000216007232666016

Client-side trace contains layer two network latency  
Server-side trace only contains ARP processing.  
So, layer two network latency is  
 $0.000351190567016602 - 0.000216007232666016 = 0.00013518$   
small values in a virtual network

Trace file	arp_latency.time (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
clientside.pcapng	0.000351190567016602	Yes	No	No	No	No	ARP between Client NIC and default GW includes network
serverside.pcapng	0.000216007232666016	No	No	No	No	No	Server NIC's arp processing time without network



# STEP5 Compare different point of capturing



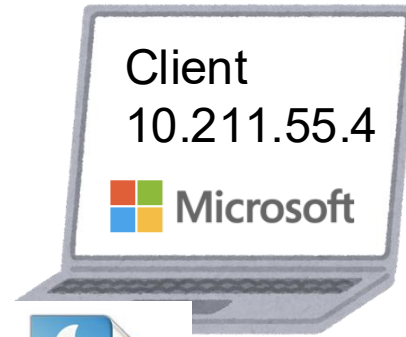
- Next Layer3, check #4 ICMP packet of both side.

```
C:\Users\megumitakeshita>ping 192.168.43.134

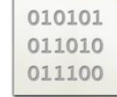
192.168.43.134 に ping を送信しています 32 バイトのデータ:
192.168.43.134 からの応答: バイト数 =32 時間 =2ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 <1ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 =3ms TTL=128
192.168.43.134 からの応答: バイト数 =32 時間 <1ms TTL=128

192.168.43.134 の ping 統計:
    パケット数: 送信 = 4、受信 = 4、損失 = 0 (0% の損失)、
    ラウンドトリップの概算時間 (ミリ秒):
        最小 = 0ms、最大 = 3ms、平均 = 1ms

FLARE-VM 2025/06/09 7:37:41.91
C:\Users\megumitakeshita>ftp 192.168.43.134
192.168.43.134 に接続しました。
220 Browser Ftp Server.
530 Please login with USER and PASS.
ユーザー (192.168.43.134:(none)): ikeriri
331 Password required for this user.
パスワード:
230 User Logged In.
ftp> bye
221 Goodbye.
```



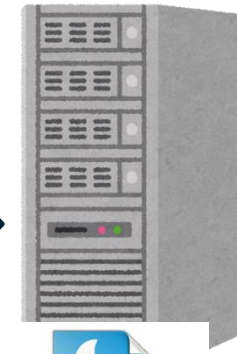
clientside.pcapng



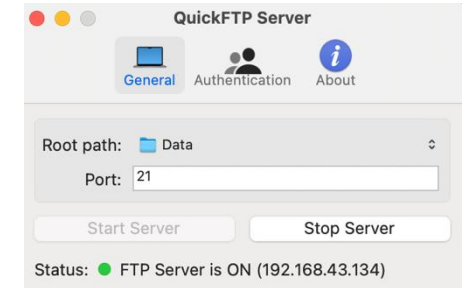
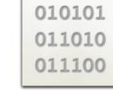
Switching  
VLAN

ping

*icmp.resptime*



serverside.pcapng



Server  
192.168.43.134

MacBook Pro

No.	Time	Source	De
→ 3	0.000130	10.211.55.4	192.168.43.134
← 4	0.001332	192.168.43.134	10.211.55.4

> Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: Parallels_00:00:18 (08:00:00:00:00:18), Dst: 10.211.55.4
> Internet Protocol Version 4, Src: 192.168.43.134, Dst: 10.211.55.4
> Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x54c7 [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 148 (0x0094)
Sequence Number (LE): 37888 (0x9400)
[Request frame: 3]
[Response time: 1.332 ms]
> Data (32 bytes)

icmp.resptime is Wireshark  
Generated field to calculate  
the ICMP response time  
Client-side trace contains  
Round Trip Time

sample traces and configurations are

<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

No.	Time	Source	De
→ 3	0.000016	10.211.55.4	192.168.43.134
← 4	0.001531	192.168.43.134	10.211.55.4

> Frame 4: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
> Ethernet II, Src: Parallels_00:00:18 (08:00:00:00:00:18), Dst: 10.211.55.4
> Internet Protocol Version 4, Src: 192.168.43.134, Dst: 10.211.55.4
> Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x54c7 [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 148 (0x0094)
Sequence Number (LE): 37888 (0x9400)
[Request frame: 3]
[Response time: 1.531 ms]
> Data (32 bytes)



# STEP5 Compare different point of capturing



## Compare both sides of the ICMP latency

Check icmp.resptime and  
Calculate network latency as  
 $1.531 - 1.332 = 0.199\text{ms}$   
(so small latency because of  
virtual environments)

No.	Time	Source	De
→ 3	0.000016	10.211.55.4	192.168.43.134
← 4	0.001531	192.168.43.134	10.211.55.4

> Frame 4: 74 bytes on wire (592 bits), 74 captured (592 bytes) on interface 0
> Ethernet II, Src: Parallels_00:00:18 (08:00:00:00:18:00), Dst: 10.211.55.4 (08:00:00:00:18:00)
> Internet Protocol Version 4, Src: 192.168.43.134, Dst: 10.211.55.4
✓ Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x54c7 [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 148 (0x0094)
Sequence Number (LE): 37888 (0x9400)
[Request frame: 3]
[Response time: 1.531 ms]
> Data (32 bytes)

No.	Time	Source	De
→ 3	0.000130	10.211.55.4	192.168.43.134
← 4	0.001332	192.168.43.134	10.211.55.4

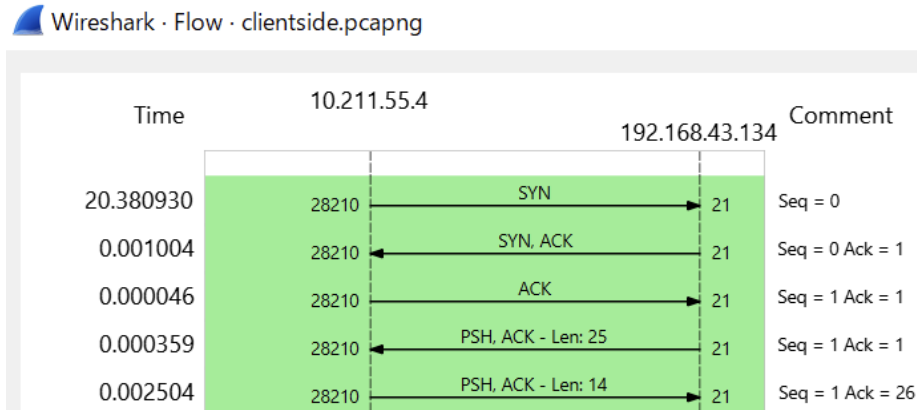
> Frame 4: 74 bytes on wire (592 bits), 74 captured (592 bytes) on interface 0
> Ethernet II, Src: Parallels_00:00:18 (08:00:00:00:18:00), Dst: 10.211.55.4 (08:00:00:00:18:00)
> Internet Protocol Version 4, Src: 192.168.43.134, Dst: 10.211.55.4
✓ Internet Control Message Protocol
Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x54c7 [correct]
[Checksum Status: Good]
Identifier (BE): 1 (0x0001)
Identifier (LE): 256 (0x0100)
Sequence Number (BE): 148 (0x0094)
Sequence Number (LE): 37888 (0x9400)
[Request frame: 3]
[Response time: 1.332 ms]
> Data (32 bytes)

Trace file	icmp.resptime (seconds)	network	Client OS	Client App	Server OS	Server App	Meaning
clientside.pcapng	0.001531	Yes	No	No	Yes	No	RTT + Server OS processing
serverside.pcapng	0.001332	No	No	No	Yes	No	Server OS IP stack receives ICMP then create echo reply and send back to the Client.

## STEP5 Compare different point of capturing



- Then look for TCP connection latency. Change Time Display Format and Create TCP Flow Graph
- In clientside.pcapng, Server sends banner with PSH/ACK segment after handshake, so ClientApp is the main cause of the latency



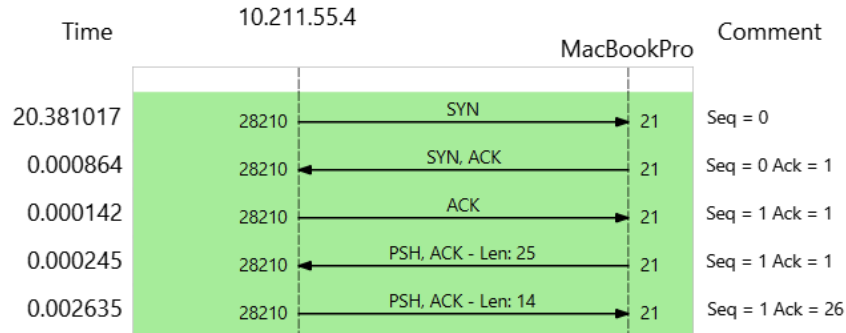
Interval of TCP segments	clientside	network	Client OS	Client App	Server OS	Server App	Meaning
<i>SYN&lt;&gt;SYN/ACK</i>	0.001004	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<i>SYN/ACK&lt;&gt;ACK</i>	<b>0.000046</b>	No	Yes	No	No	No	Client OS (Win11) socket processing time
<i>ACK&lt;&gt;1<sup>st</sup>segment(PSH/ACK Len:25)(from Server with banner)</i>	0.000359	Yes	No	No	Yes	Yes	Server OS processes the socket, and ServerApp create banner, then serverOS send PSH/ACK tcp segment via network
<i>1<sup>st</sup>segment(PSH/ACK Len:25) &lt;&gt; 2<sup>nd</sup> segment(PSH/ACK Len:14)</i>	<b>0.002504 worst!!</b>	No	Yes	Yes	No	No	Client OS processes the socket, and ClientApp create the FTP command, then Client OS create PSH/ACK tcp segment

# STEP5 Compare different point of capturing

Create the tables for both client-side/server-side.

In serverside.pcapng, Server sends PUSH/ACK TCP segment contains FTP server banner message after 3-way handshake (0.000245sec)

Wireshark · Flow · serverside.pcapng



Interval of TCP segments	serverside	network	Client OS	Client App	Server OS	Server App	Meaning
<i>SYN&lt;&gt;SYN/ACK</i>	<b>0.000864</b>	No	No	No	Yes	No	Server OS initial socket processing time
<i>SYN/ACK&lt;&gt;ACK</i>	0.000142	Yes	Yes	No	No	No	Client OS (Win11) socket processing time + network ( in this case, so small latency)
<i>ACK&lt;&gt;1<sup>st</sup>segment(PSH/ACK) (from Server with banner)</i>	0.000245	No	No	No	Yes	Yes	Server OS processes the socket, and ServerApp create a banner, then ServerOS send PSH/ACK tcp segment via network
<i>1<sup>st</sup>segment(PSH/ACK) &lt;&gt; 2<sup>nd</sup> segment(PSH/ACK)</i>	<b>0.002635 worst</b>	Yes	Yes	Yes	No	No	Client OS processes the socket, and ClientApp create the FTP command, then Client OS create and sends PSH/ACK tcp segment via the network

# STEP5 Compare different point of capturing



Interval of TCP segments	clientside	network	Client OS	Client App	Server OS	Server App	Meaning
<i>SYN&lt;&gt;SYN/ACK</i>	0.001004	Yes	No	No	Yes	No	Server OS initial socket processing time + network
<i>SYN/ACK&lt;&gt;ACK</i>	<b>0.000046</b>	No	Yes	No	No	No	Client OS (Win11) socket processing time
<i>ACK&lt;&gt;1<sup>st</sup>segment(PSH/ACK)</i> <i>(from Server with banner)</i>	0.000359	Yes	No	No	Yes	Yes	Server OS processes the socket, and ServerApp create banner, then serverOS send PSH/ACK tcp segment via network
<i>1<sup>st</sup>segment(PSH/ACK) &lt;&gt;</i> <i>2<sup>nd</sup> segment(PSH/ACK)</i>	<b>0.002504</b> <b>worst!!</b>	No	Yes	Yes	No	No	Client OS processes the socket, and ClientApp create the FTP command, then Client OS create PSH/ACK tcp segment

Interval of TCP segments	serverside	network	Client OS	Client App	Server OS	Server App	Meaning
<i>SYN&lt;&gt;SYN/ACK</i>	<b>0.000864</b>	No	No	No	Yes	No	Server OS initial socket processing time
<i>SYN/ACK&lt;&gt;ACK</i>	0.000142	Yes	Yes	No	No	No	Client OS (Win11) socket processing time + network ( in this case, so small latency)
<i>ACK&lt;&gt;1<sup>st</sup>segment(PSH/ACK)</i> <i>(from Server with banner)</i>	0.000245	No	No	No	Yes	Yes	Server OS processes the socket, and ServerApp create a banner, then ServerOS send PSH/ACK tcp segment via network
<i>1<sup>st</sup>segment(PSH/ACK) &lt;&gt;</i> <i>2<sup>nd</sup> segment(PSH/ACK)</i>	<b>0.002635</b> <b>worst</b>	Yes	Yes	Yes	No	No	Client OS processes the socket, and ClientApp create the FTP command, then Client OS create and sends PSH/ACK tcp segment via the network

## STEP5 Compare different point of capturing



- Pick up the latency values from both traces.

Cause of latency	Calculation	Latency (seconds)
Client App (FTP Client)+ClientOS	In clientside.pcapng, 1 <sup>st</sup> segment(PSH/ACK) <> 2 <sup>nd</sup> segment(PSH/ACK) <b>0.002504 seconds ( including few ClientOS processing time)</b>	<b>0.002504</b>
Client OS (Windows11)	In clientside.pcapng, SYN/ACK<>ACK interval <b>0.000046 seconds</b>	<b>0.000046</b>
Network (Virtual network)	In serverside.pcapng, 1 <sup>st</sup> segment(PSH/ACK) <> 2 <sup>nd</sup> segment(PSH/ACK) <b>0.002635 seconds minus</b> In clientside.pcapng, 1 <sup>st</sup> segment(PSH/ACK) <> 2 <sup>nd</sup> segment(PSH/ACK) <b>0.002504 seconds = 0.000131 seconds</b>	<b>0.000131</b> (Virtual network)
Server OS (macos15.5)	In serverside.pcapng SYN<>SYN/ACK interval <b>0.000864 seconds</b>	<b>0.000864</b>
Server App (FTP Server)	In serverside.pcapng, ACK<>1 <sup>st</sup> segment(PSH/ACK) <b>0.000245 seconds (including few ServerOS processing time)</b>	<b>0.000245</b>

- The root cause of latency is ClientApp, FTP Client, not Network!! Tell this to the development team!!



- Collect each layer's latency
- Layer 2: Arp response time (`arp_latency.time`)
- Layer 3: Network RTT (`icmp.resptime`)
- Layer 4: Use Wireshark Generated fields such as `tcp.analysis.initial_rtt`, `tcp.analysis.ack_rtt`  
Create a TCP flow graph of each segment interval.  
Then think the meaning of each latencies
- Layer 5-7: Use Wireshark Generated fields such as `http.time`, `dns.time`, `smb2.time`, `nbns.time`, `radius.time`, `dcerpc.time`, and so on.





- Write out each latency into the table, Consider the meanings of each duration. Then assign each latency to ClientApp, ClientOS, NW, ServerOS and ServerApp
- If you're stuck with the problems, we need to collect more information. Compare another trace with different settings, client/server and network.
- Comparing both client-side and server-side traces in the same connection is very useful in TCP latency analysis.

# USE WIRESHARK

## Thank you for watching.

Please complete the survey and send feedback

<https://conference.wireshark.org/sharkfest-25-us-2024/talk/UPKZY8/feedback>

Sample trace files and Wireshark profiles are

<https://www.ikeriri.ne.jp/sharkfest/ikeriri25.zip>

