

# Finding Duplications

Duplication tells us the key  
to troubleshoot the problems



#sf24us

**Megumi Takeshita**  
Packet Otaku,  
ikeriri network service

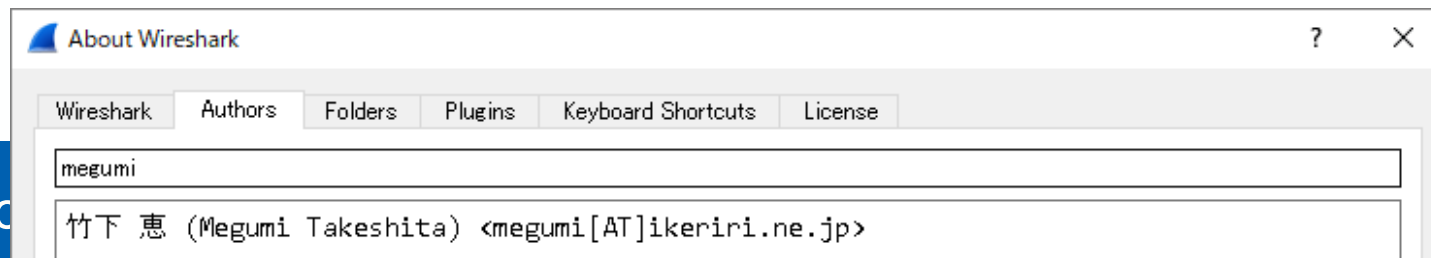
**Sample traces and configurations**

<https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Megumi Takeshita, packet otaku



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



## Session Details

When you troubleshoot or investigate network and security problems, duplications are good indicators to find the clues. We use Wireshark and CLI tools to find, recognize and dissect network/security anomalies to solve the issues. Duplications exist in every layer in a trace file, so we follow each layer to check protocol-specified troubleshooting points.

In this session, you can learn how to find duplications in each layer of a trace file, the meanings implied by duplications in the trace with TIPS and tricks of display filters and major plugins of Wireshark/tshark.

We troubleshoot and investigate the issues using ARP/IP/TCP and major basic protocols. Duplications is one of the best anomalies to understand the packets.

# Duplication of ARP

- Open trace file arp-storm.pcap

[https://wiki.wireshark.org/uploads/\\_\\_moin\\_import\\_\\_/attachments/SampleCaptures/arp-storm.pcap](https://wiki.wireshark.org/uploads/__moin_import__/attachments/SampleCaptures/arp-storm.pcap)

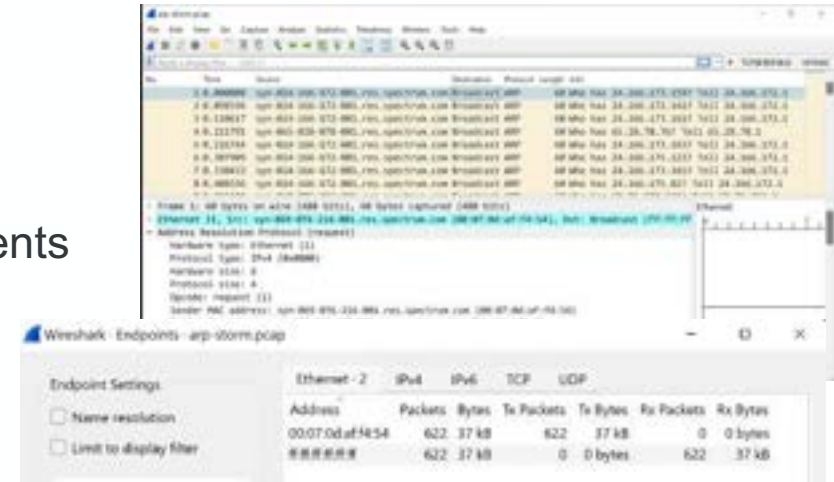
- Statistics>Endpoints>Ethernet

622packets of broadcast from Cisco\_af:f4:54

- Tons of ARP requests use bandwidth and interrupt all nodes in the LAN.

- Misconfiguration of router ( bridge mode?) or ARP poisoning attack from the pwned router?

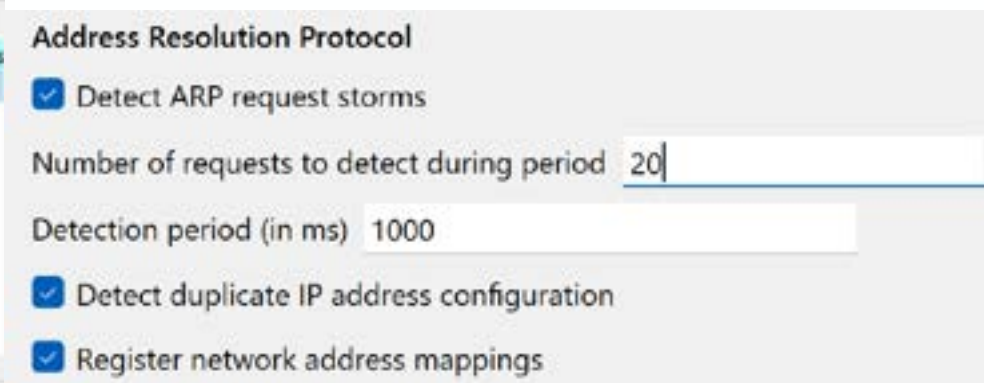
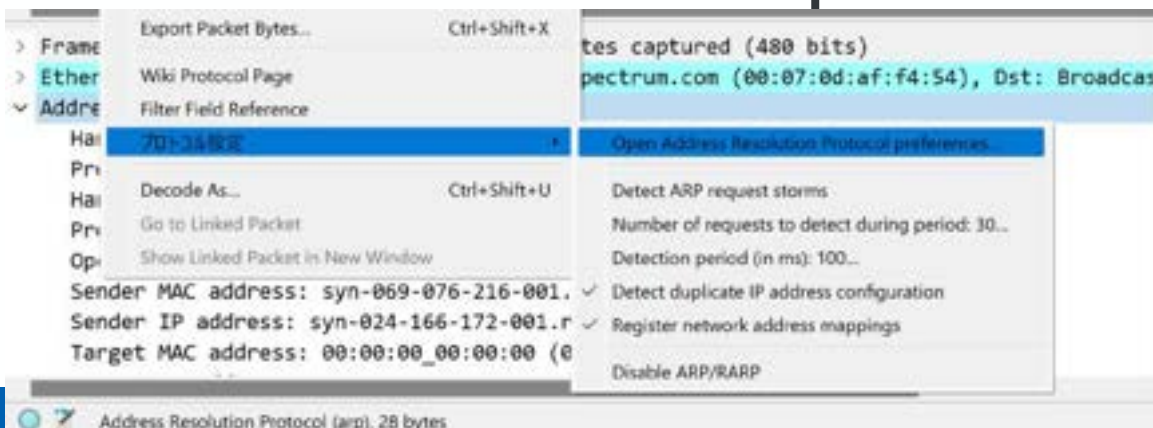
Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>4</sup>



# Using Expert Info to find ARP storms



- How about finding duplication,
- Choose ARP in the packet detail pane, right-click Protocol configuration > Open ARP preferences
- Check "Detect ARP request storms", set Number of requests as 20 and period as 1000



Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>5</sup>

# Using Expert Info to find ARP storms



- Click blue signal at left side of bottom

Packet	Summary	Group	Protocol	Count
37	ARP packet storm detected	Sequence	ARP/NAAP	
42	Who has 24.146.174.207? [Seq 45,26,92.1]	Sequence	ARP/NAAP	
43	Who has 45.26.76.142? [Seq 45,26,76.1]	Sequence	ARP/NAAP	
84	Who has 24.146.174.207? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
122	Who has 45.26.94.131? [Seq 45,26,92.1]	Sequence	ARP/NAAP	
143	Who has 24.146.174.197? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
302	Who has 69.23.182.253? [Seq 69,23,182.1]	Sequence	ARP/NAAP	
323	Who has 24.146.174.167? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
360	Who has 24.146.175.233? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
391	Who has 69.23.182.253? [Seq 69,23,182.1]	Sequence	ARP/NAAP	
329	Who has 24.146.175.121? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
341	Who has 69.23.182.253? [Seq 69,23,182.1]	Sequence	ARP/NAAP	
362	Who has 45.26.92.190? [Seq 45,26,92.1]	Sequence	ARP/NAAP	
452	Who has 24.146.173.90? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
473	Who has 69.76.221.207? [Seq 69,76,216.1]	Sequence	ARP/NAAP	
119	Who has 69.76.221.107? [Seq 69,76,216.1]	Sequence	ARP/NAAP	
156	Who has 24.146.174.207? [Seq 24.146.172.1]	Sequence	ARP/NAAP	
163	Who has 24.146.175.291? [Seq 24.146.172.1]	Sequence	ARP/NAAP	

```
ARP packet storm detected (20 packets in < 1000 ms)
[Expert Info (Note/Sequence): ARP packet storm detected (20 packets in < 1000 ms)]
[ARP packet storm detected (20 packets in < 1000 ms)]
[Severity level: Note]
[Group: Sequence]
```

- Expert Info tells us “ARP packet storm detected”  
Open the tree and click and check each packet.
- We find an additional header about “ARP packet storm detected (20 packets in < 1000 ms)”

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>6</sup>

# Wireshark ARP storms settings



We need to check the preferences of setting, open the “preferences” text file in your profile of personal configuration.

(C:\Users\[username]\AppData\Roaming\Wireshark\profiles\[your profile]\preferences

```
# Attempt to detect excessive rate of ARP requests
# TRUE or FALSE (case-insensitive)
arp.detect_request_storms: TRUE

# Number of requests needed within period to indicate a storm
# A decimal number
arp.detect_storm_number_of_packets: 20
|
# Period in milliseconds during which a packet storm may be detected
# A decimal number
arp.detect_storm_period: 1000
```

Search text  
by “arp” to find  
ARP settings  
of Wireshark

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>7</sup>

# Find ARP storms by tshark



- We need to override tshark settings

```
tshark -r arp-storm.pcap -O arp ←
```

Show ARP  
packet detail #sf24us

```
-Y arp.packet-storm-detected ← Display filter
```

```
-o "arp.detect_request_storms:TRUE" ← preferences  
without space
```

```
-o "arp.detect_storm_number_of_packets:20"
```

```
-o "arp.detect_storm_period: 1000"
```

- If you just need the frame number and source mac address, add "-T fields -e frame.number -e eth.src"

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>8</sup>



# Find duplication by tshark



#sf24us

```
C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r arp-storm.pcap -Y arp.packet-storm-detected -O arp -o "arp.detect_request_storms:TRUE" -o "arp.detect_storm_number_of_packets:20" -o "arp.detect_storm_period:1000"
Frame 21: 60 bytes on wire (480 bits): 60 bytes captured (480 bits) on interface 0
Ethernet II, Src: syn-069-076-216-001.res.spectrum.com (00:07:0d:af:f4:54), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
  Hardware type: Ethernet (I)
  Protocol type: IPv4 (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (I)
  Sender MAC address: syn-024-166-172-001.res.spectrum.com (00:07:0d:af:f4:54)
  Sender IP address: syn-024-166-172-001.res.spectrum.com (24.166.172.1)
  Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Target IP address: syn-024-166-174-207.res.spectrum.com (24.166.174.207)
  ARP packet storm detected (20 packets in < 1000 ms)
  [Expert info (Note/Sequence): ARP packet storm detected (20 packets in < 1000 ms)]
  [ARP packet storm detected (20 packets in < 1000 ms)]
  [Severity level: Note]
  [Group: Sequence]
```

```
tshark -r arp-storm.pcap -O arp
-Y arp.packet-storm-detected
-o "arp.detect_request_storms:TRUE"
-o "arp.detect_storm_number_of_packets:20"
-o "arp.detect_storm_period: 1000"
```

```
C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r arp-storm.pcap -Y arp.packet-storm-detected -O arp -o "arp.detect_request_storms:TRUE" -o "arp.detect_storm_number_of_packets:20" -o "arp.detect_storm_period:1000" -T fields -e frame.number -e eth.src
21      00:07:0d:af:f4:54
42      00:07:0d:af:f4:54
63      00:07:0d:af:f4:54
84      00:07:0d:af:f4:54
122     00:07:0d:af:f4:54
143     00:07:0d:af:f4:54
202     00:07:0d:af:f4:54
223     00:07:0d:af:f4:54
260     00:07:0d:af:f4:54
281     00:07:0d:af:f4:54
320     00:07:0d:af:f4:54
341     00:07:0d:af:f4:54
362     00:07:0d:af:f4:54
452     00:07:0d:af:f4:54
473     00:07:0d:af:f4:54
519     00:07:0d:af:f4:54
556     00:07:0d:af:f4:54
593     00:07:0d:af:f4:54
```

```
tshark -r arp-storm.pcap -O arp
-Y arp.packet-storm-detected
-o "arp.detect_request_storms:TRUE"
-o "arp.detect_storm_number_of_packets:20"
-o "arp.detect_storm_period: 1000"
-T fields -e frame.number -e eth.src
```

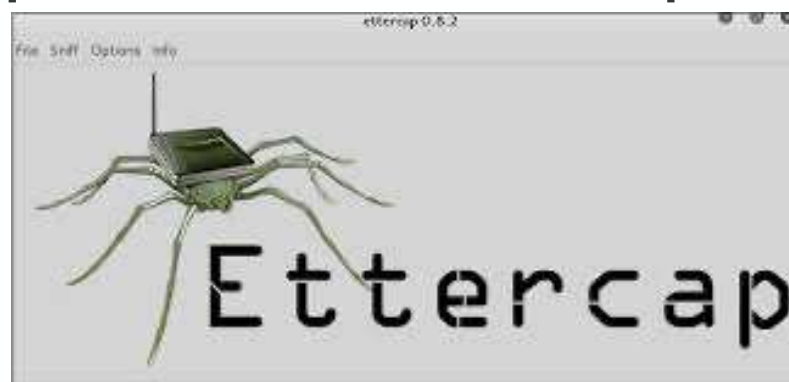
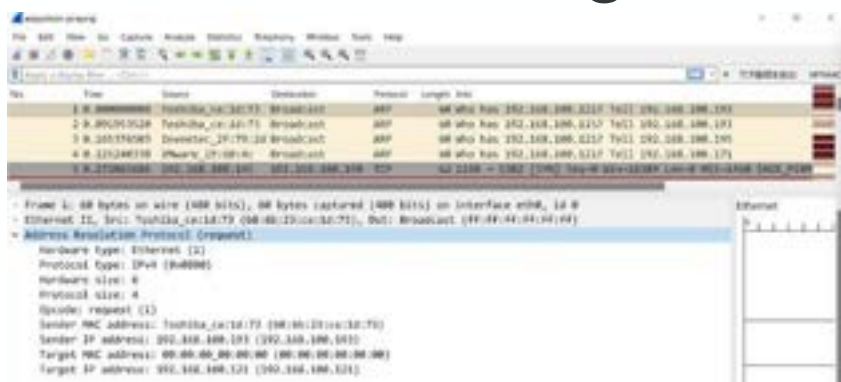
Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>9</sup>

# Excessive ARP request tells us



#sf24us

1. Misconfiguration of network devices, such as the bridge mode of the router?
  2. MITM attacks such as ARP poisoning.
- Open arppoison.pcapng. This trace file is captured by KaliLinux, using Ettercap to execute ARP poisoning.



Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>10</sup>

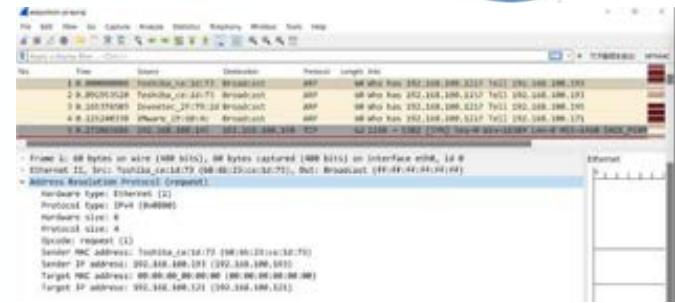
# ARP poisoning attack by Ettercap



- Check Expert Info to find the ARP request storm.

Wireshark · Expert Information · arppoison.pcapng

Severity	Summary	Group	Protocol	Count
> Warning	Duplicate IP address configured	Sequence	ARP/RARP	6
> Warning	Connection reset (RST)	Sequence	TCP	150
▼ Note	ARP packet storm detected	Sequence	ARP/RARP	2
369	Who has 192.168.100.121? Tell 192.168.100.118	Sequence	ARP/RARP	
445	Who has 192.168.100.121? Tell 192.168.100.106	Sequence	ARP/RARP	
> Note	A new tcp-session is started with the same ports as an earlier session in t...	Sequence	TCP	133
> Note	Padding identification may be inaccurate and impact trailer dissector	Protocol	Ethertype	84
> Chat	Connection establish request (SYN)	Sequence	TCP	150



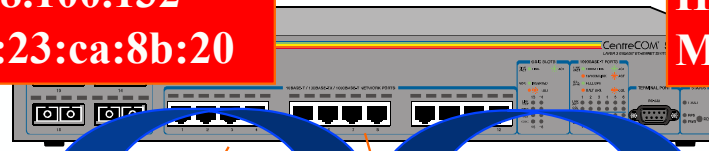
- There are two counts of ARP storm as well as Duplicate IP address configured events.
- IP address configuration mistake or IP spoofing attack by Man In The Middle?

# ARP poisoning attack by Ettercap



**Spoofed ARP broadcast**  
IP address : 192.168.100.132  
MAC address: b8:6b:23:ca:8b:20

**Spoofed ARP broadcast**  
IP address : 192.168.100.104  
MAC address: b8:6b:23:ca:8b:20



**Actual Address configuration**

IP address: 192.168.100.104

MAC address: b8:6b:23:4c:2e:1f



**Actual Address configuration**

IP address: 192.168.100.132

MAC address: 00:0c:29:f7:2e:fc

**ATTACKER KaliLinux**

IP address : 192.168.100.198

MAC address: b8:6b:23:ca:8b:20

■ 192.168.100.104 is used by both b8:6b:23:4c:2e:1f and b8:6b:23:ca:8b:20

■ 192.168.100.132 is used by both 00:0c:29:f7:2e:fc and b8:6b:23:ca:8b:20

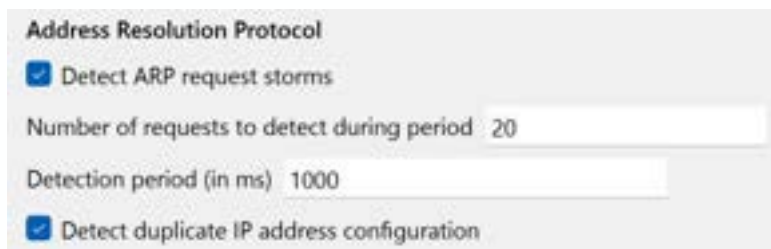
Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Find duplication of IP address



#sf24us

- Wireshark finds duplication of IP address by ARP/RARP dissector by default
- Choose ARP in the packet detail pane, right-click Protocol configuration > Open ARP preference
- Check ON: Detect duplicate IP address configuration
- Also check Warning of Duplicate IP address configured

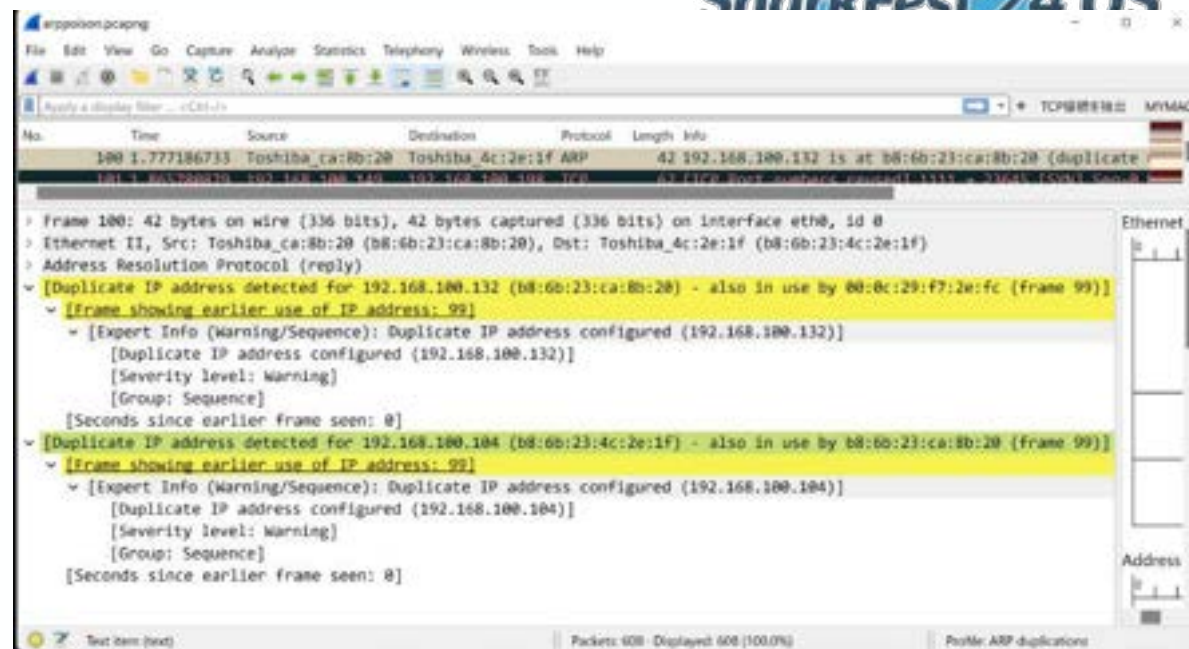
A screenshot of the Wireshark packet list pane showing a warning for duplicate IP address configuration. The warning is expanded to show five packets (100, 100, 233, 233, 426) that all contain the same IP address (192.168.100.132) and MAC address (b8:6b:23:ca:8b:20), which is a duplicate of the IP address 192.168.100.104 that was already detected.

Packet	Summary	Group	Protocol	Count
Warning Duplicate IP address configured				
100	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	5
100	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	
233	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	
233	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	
426	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	
426	192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate use of 192.168.100.104 detected!)	Sequence	ARP/RARP	

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Find duplication of IP address

- Check frame #100
- 192.168.100.104
- 192.168.100.132
- are used by two different MAC address, actual PC (blue) and MITM(red)



```
100 1.777186733 Toshiba_ca:8b:20 Toshiba_4c:2e:1f ARP 42 192.168.100.132 is at b8:6b:23:ca:8b:20 (duplicate)
101 1.865900070 192.168.100.104 192.168.100.104 TCP 60 4192 8000 65535 65535 192.168.100.104 192.168.100.104

> Frame 100: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0
> Ethernet II, Src: Toshiba_ca:8b:20 (b8:6b:23:ca:8b:20), Dst: Toshiba_4c:2e:1f (b8:6b:23:4c:2e:1f)
> Address Resolution Protocol (reply)
  [Duplicate IP address detected for 192.168.100.132 (b8:6b:23:ca:8b:20) - also in use by 00:0c:29:f7:2e:fc (frame 99)]
  [Frame showing earlier use of IP address: 99]
    [Expert Info (Warning/Sequence): Duplicate IP address configured (192.168.100.132)]
    [Duplicate IP address configured (192.168.100.132)]
    [Severity level: Warning]
    [Group: Sequence]
    [Seconds since earlier frame seen: 0]
  [Duplicate IP address detected for 192.168.100.104 (b8:6b:23:4c:2e:1f) - also in use by b8:6b:23:ca:8b:20 (frame 99)]
  [Frame showing earlier use of IP address: 99]
    [Expert Info (Warning/Sequence): Duplicate IP address configured (192.168.100.104)]
    [Duplicate IP address configured (192.168.100.104)]
    [Severity level: Warning]
    [Group: Sequence]
    [Seconds since earlier frame seen: 0]
```

192.168.100.104 is used by both **b8:6b:23:4c:2e:1f** and **b8:6b:23:ca:8b:20**

192.168.100.132 is used by both **00:0c:29:f7:2e:fc** and **b8:6b:23:ca:8b:20**

# Find IP address duplications by tshark

SharkFest'24 US  
June 15-20 - Fairfax, VA

- You can find IP address duplications

```
tshark -r arppoison.pcapng -O arp ← Show ARP packet detail #sf24us  
-Y arp.duplicate-address-frame ← Display filter
```

```
C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r arppoison.pcapng -O arp -Y arp.duplicate-address-frame  
Frame 100: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0  
Ethernet II, Src: Toshiba_ca:8b:20 (b8:6b:23:ca:8b:20), Dst: Toshiba_4c:2e:1f (b8:6b:23:4c:2e:1f)  
Address Resolution Protocol (reply)  
  Hardware type: Ethernet (1)  
  Protocol type: IPv4 (0x0800)  
  Hardware size: 6  
  Protocol size: 4  
  Opcode: reply (2)  
  Sender MAC address: Toshiba_ca:8b:20 (b8:6b:23:ca:8b:20)  
  Sender IP address: 192.168.100.132 (192.168.100.132)  
  Target MAC address: Toshiba_4c:2e:1f (b8:6b:23:4c:2e:1f)  
  Target IP address: 192.168.100.104 (192.168.100.104)  
[Duplicate IP address detected for 192.168.100.132 (b8:6b:23:ca:8b:20) - also in use by 00:0c:29:f7:2e:fc (frame 99)]  
[Duplicate IP address detected for 192.168.100.104 (b8:6b:23:4c:2e:1f) - also in use by b8:6b:23:ca:8b:20 (frame 99)]
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>15</sup>

# Find IP address duplications by tshark

SharkFest'24 US  
June 15-20 - Fairfax, VA

#sf24us

- We need the set of spoofed and actual sets of IP/MAC address from the trace file

1: Source and Target IP/MAC address sets sent by ARP request and reply message

```
tshark -r arppoison.pcapng -T fields -e arp.src.proto_ipv4 -e arp.src.hw_mac
```

```
tshark -r arppoison.pcapng -T fields -e arp.dst.proto_ipv4 -e arp.dst.hw_mac
```

2: Source and Destination MAC/IP sets from actual header

```
tshark -r arppoison.pcapng -T fields -e ip.src -e eth.src -Y !arp
```

```
tshark -r arppoison.pcapng -T fields -e ip.dst -e eth.dst -Y !arp
```

← Without

ARP packet

3: Merge and pick up duplications



# Find IP address duplications by tshark



## Create list, merge, sort, pick up duplication

```
tshark -r arppoison.pcapng -T fields -e arp.src.proto_ipv4 -e arp.src.hw_mac >> list.txt #sf24us
```

```
tshark -r arppoison.pcapng -T fields -e arp.dst.proto_ipv4 -e arp.dst.hw_mac >> list.txt
```

```
tshark -r arppoison.pcapng -Y !arp -T fields -e ip.src -e eth.src >> list.txt
```

```
tshark -r arppoison.pcapng -Y !arp -T fields -e ip.dst -e eth.dst >> list.txt
```

```
cat list.txt | sort | uniq | grep -v 00:00:00:00:00:00 >>dup.txt
```

```
C:\Users\keegumitakeshita\Desktop\finding_duplications>tshark -r arppoison.pcapng -T fields -e arp.src.proto_ipv4 -e arp.src.hw_mac >> list.txt
C:\Users\keegumitakeshita\Desktop\finding_duplications>tshark -r arppoison.pcapng -T fields -e arp.dst.proto_ipv4 -e arp.dst.hw_mac >> list.txt
C:\Users\keegumitakeshita\Desktop\finding_duplications>tshark -r arppoison.pcapng -Y !arp -T fields -e ip.src -e eth.src >> list.txt
C:\Users\keegumitakeshita\Desktop\finding_duplications>tshark -r arppoison.pcapng -Y !arp -T fields -e ip.dst -e eth.dst >> list.txt
C:\Users\keegumitakeshita\Desktop\finding_duplications>cat list.txt | sort | uniq | grep -v 00:00:00:00:00:00
```

```
192.168.100.104 b8:6b:23:4c:2e:1f
192.168.100.104 b8:6b:23:ca:8b:20
192.168.100.106 b8:6b:23:16:1b:73
192.168.100.110 00:8c:fa:2f:7a:42
192.168.100.113 44:d4:e0:ce:22:09
192.168.100.118 40:40:a7:53:d6:0a
192.168.100.120 00:0c:29:6e:5f:e8
192.168.100.132 00:0c:29:f7:2e:fc
192.168.100.132 b8:6b:23:ca:8b:20
192.168.100.139 00:0c:29:d5:68:3e
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>17</sup>

# Find IP address duplications by tshark

SharkFest'24 US  
June 15-20 - Fairfax, VA

Finally, pick up duplicate IP address entries by Powershell

```
Get-Content dup.txt | Group-Object -Property { $_.Split()[0] } | Where-Object  
{ $_.Count -gt 1 } | ForEach-Object { $_.Group | Select-Object -Unique }
```

#sf24us

or

```
awk '{if (ip[$1] && ip[$1] != $2) {print $0; print $1, ip[$1]}  
else ip[$1]=$2}' < dup.txt
```

```
PS C:\Users\megumitakeshita\Desktop\finding_duplications> Get-Content dup.txt | Group-Object -Property { $_.S  
plit()[0] } | Where-Object { $_.Count -gt 1 } | ForEach-Object { $_.Group | Select-Object -Unique }  
192.168.100.104 b8:6b:23:4c:2e:1f  
192.168.100.104 b8:6b:23:ca:8b:20  
192.168.100.132 00:0c:29:f7:2e:fc  
192.168.100.132 b8:6b:23:ca:8b:20
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# IP address duplications tells us



1. Misconfiguration of address settings client-side in usual, sometimes happens in DHCP
2. MITM attacks such as ARP poisoning.

#sf24us



# Duplication of IPID

- All IP packets have their unique IP Identification fields (Basically)
- We can easily find the same IPID value in the fragmented IPv4 packets. open ipfragment.pcapng and check IP header in frame #5



#sf24us

No.	Time	Source	Destination	Protocol	Length	Info
2	0.000046	Toshiba_65:16:54	GW	ARP	42	192.168.100.110 is at e8:e0:b7:65:16:54
3	6.800478	Toshiba_86:02:54	Broadcast	ARP	60	Who has 192.168.100.110? Tell 192.168.100.105
4	6.800523	Toshiba_65:16:54	Toshiba_86:02:54	ARP	42	192.168.100.110 is at e8:e0:b7:65:16:54
5	6.800761	192.168.100.105	192.168.100.110	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, I...
6	6.800762	192.168.100.105	192.168.100.110	ICMP	62	Echo (ping) request id=0x0001, seq=1/256, ttl...
7	6.800902	192.168.100.110	192.168.100.105	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, I...
8	6.800910	192.168.100.110	192.168.100.105	ICMP	62	Echo (ping) reply id=0x0001, seq=1/256, ttl...
9	7.815772	192.168.100.105	192.168.100.110	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, I...
10	7.912774	103.168.100.105	103.168.100.110	TCP	67	Echo (ping) request id=0x0001, seq=1/256, ttl...

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> 20

# Duplication of IPID



- This is a simple ping packet that is over MTU size.

```
5 6.808761 192.168.100.105 192.168.100.110 IPv4 1514 Fragmented IP pro
* 6 6.808762 192.168.100.105 192.168.100.110 ICMP 62 Echo (ping) reques
> Frame 5: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \De
> Ethernet II, Src: Toshiba_86:02:54 (e8:e0:b7:86:02:54), Dst: Toshiba_65:16:54 (e8:e0:b7:65:
< Internet Protocol Version 4, Src: 192.168.100.105 (192.168.100.105), Dst: 192.168.100.110 (
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x4888 (18568)
  > 001. .... = Flags: 0x1, More fragments
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: ICMP (1)
  Header Checksum: 0x8270 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.100.105 (192.168.100.105)
  Destination Address: 192.168.100.110 (192.168.100.110)
  [Reassembled IPv4 in frame: 6]
  > Data (1488 bytes)
* 5 6.808761 192.168.100.105 192.168.100.110 IPv4 1514 Fragmented IP pro
* 6 6.808762 192.168.100.105 192.168.100.110 ICMP 62 Echo (ping) reques
> Frame 6: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface \Device\NPF
> Ethernet II, Src: Toshiba_86:02:54 (e8:e0:b7:86:02:54), Dst: Toshiba_65:16:54 (e8:e0:b7:65:
< Internet Protocol Version 4, Src: 192.168.100.105 (192.168.100.105), Dst: 192.168.100.110 (
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 48
  Identification: 0x4888 (18568)
  > 000. .... = Flags: 0x0
  ...0 0000 1011 1001 = Fragment Offset: 1488
  Time to Live: 128
  Protocol: ICMP (1)
  Header Checksum: 0xa763 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 192.168.100.105 (192.168.100.105)
  Destination Address: 192.168.100.110 (192.168.100.110)
  > [2 IPv4 Fragments (1508 bytes): #5(1480), #6(28)]
  > Internet Control Message Protocol
```

- Check IPID field value between frame #5 and #6
- Also check for DF/MF flags and offset fields.

# Finding fragments



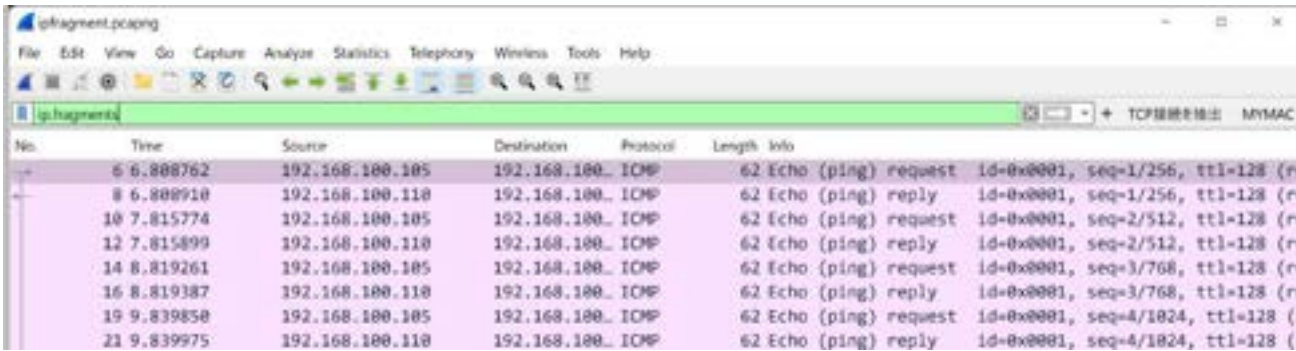
#sf24us

- Open [IPv4 fragments] header in #6
  - ✓ [2 IPv4 Fragments (1508 bytes): #5(1480), #6(28)]
    - [Frame: 5, payload: 0-1479 (1480 bytes)]
    - [Frame: 6, payload: 1480-1507 (28 bytes)]
    - [Fragment count: 2]
    - [Reassembled IPv4 length: 1508]
    - [Reassembled IPv4 data [truncated]: 080064730001000]
- Wireshark adds generated fields about the fragment, also set “ ▪ ” icons at related frames in packet list pane.
- We can use the display filter string “ip.fragments” to find IP fragments, too.

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>22</sup>

# Finding fragments

- Set “ip.fragments” to find fragments



The screenshot shows a Wireshark capture of ICMP traffic. The filter is set to 'ip.fragments'. The table below represents the data shown in the packet list pane.

No.	Time	Source	Destination	Protocol	Length	Info
6	6.808762	192.168.100.105	192.168.100.110	ICMP	62	Echo (ping) request id=0x0001, seq=1/256, ttl=128 (r)
8	6.808910	192.168.100.110	192.168.100.105	ICMP	62	Echo (ping) reply id=0x0001, seq=1/256, ttl=128 (r)
10	7.815774	192.168.100.105	192.168.100.110	ICMP	62	Echo (ping) request id=0x0001, seq=2/512, ttl=128 (r)
12	7.815899	192.168.100.110	192.168.100.105	ICMP	62	Echo (ping) reply id=0x0001, seq=2/512, ttl=128 (r)
14	8.819261	192.168.100.105	192.168.100.110	ICMP	62	Echo (ping) request id=0x0001, seq=3/768, ttl=128 (r)
16	8.819387	192.168.100.110	192.168.100.105	ICMP	62	Echo (ping) reply id=0x0001, seq=3/768, ttl=128 (r)
19	9.839850	192.168.100.105	192.168.100.110	ICMP	62	Echo (ping) request id=0x0001, seq=4/1024, ttl=128 (r)
21	9.839975	192.168.100.110	192.168.100.105	ICMP	62	Echo (ping) reply id=0x0001, seq=4/1024, ttl=128 (r)

- ip.fragment.count
- ip.fragment.error
- ip.fragment.multipletails
- ip.fragment.overlap
- ip.fragment.overlap.conflict
- ip.fragment.toolongfragment
- ip.fragments

- We can also use the count, error, multipletails, overlap, overlap.conflict, toolongfragment fields to find fragments
- Sometimes, long fragments are used for DoS attacks.

# Finding fragments using tshark



#sf24us

When you need frame number, source and destination IP address, IPID and counts on IP fragments

```
tshark -r ipfragment.pcapng -T fields -e frame.number -e ip.src -e ip.dst -e ip.id -e ip.fragment.count -Y ip.fragment.count
```

```
C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r ipfragment.pcapng -T fields -e frame.number -e ip.src -e ip.dst -e ip.id -e ip.fragment.count -Y ip.fragment.count
6      192.168.100.105 192.168.100.110 0x4888 2
8      192.168.100.110 192.168.100.105 0x2527 2
10     192.168.100.105 192.168.100.110 0x4889 2
12     192.168.100.110 192.168.100.105 0x252a 2
14     192.168.100.105 192.168.100.110 0x488a 2
16     192.168.100.110 192.168.100.105 0x252c 2
19     192.168.100.105 192.168.100.110 0x488b 2
21     192.168.100.110 192.168.100.105 0x252d 2
36     192.168.100.110 192.168.100.105 0x252e 2
38     192.168.100.105 192.168.100.110 0x488c 2
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>24</sup>



# Duplication of IPID in another case



- Open two trace file beforenat.pcap before NAT packet captured at LAN #sf24us afternat.pcap post NAT process captured at WAN
- Check the IPID field in the IP header of both

The image displays two side-by-side Wireshark packet capture windows. The left window, titled 'beforenat.pcap', shows a packet list with one entry: a SYN packet from source 192.168.11.3 to destination 202.248.110.225 on port 80, with sequence number 0 and length 52. The packet details pane shows the IP header with Identification: 0x4e0f (19983). The right window, titled 'afternat.pcap', shows a packet list with one entry: a SYN packet from source 114.167.191.37 to destination 202.248.110.225 on port 80, with sequence number 0 and length 52. The packet details pane shows the IP header with Identification: 0x4e0f (19983). Both screenshots show the packet details pane with the IP header expanded, highlighting the Identification field (IPID) with the value 0x4e0f (19983).

# IPID before NAT and after NAT



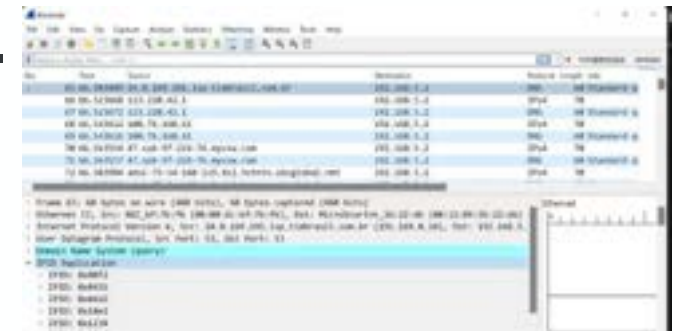
#sf24us

- . IPID does not change during the NAT process
- . We can use IPID to determine the same packet before and after the NAT process

# IPID duplications tells us



1. IP fragments, misconfiguration of MTU, or frame size settings in clients, routers and so on
2. Different data link header but the same IP packets before NAT and after NAT  
IPID does not change during the NAT process.
3. Broken routers and network devices.
4. DoS/DDoS attack (dos.pcap)



# Duplication of seq/ack number

- TCP retransmission / duplicate ACK are very common events during TCP communications.
- Use the display filter to find TCP Retransmission and D\_ACK



#sf24us

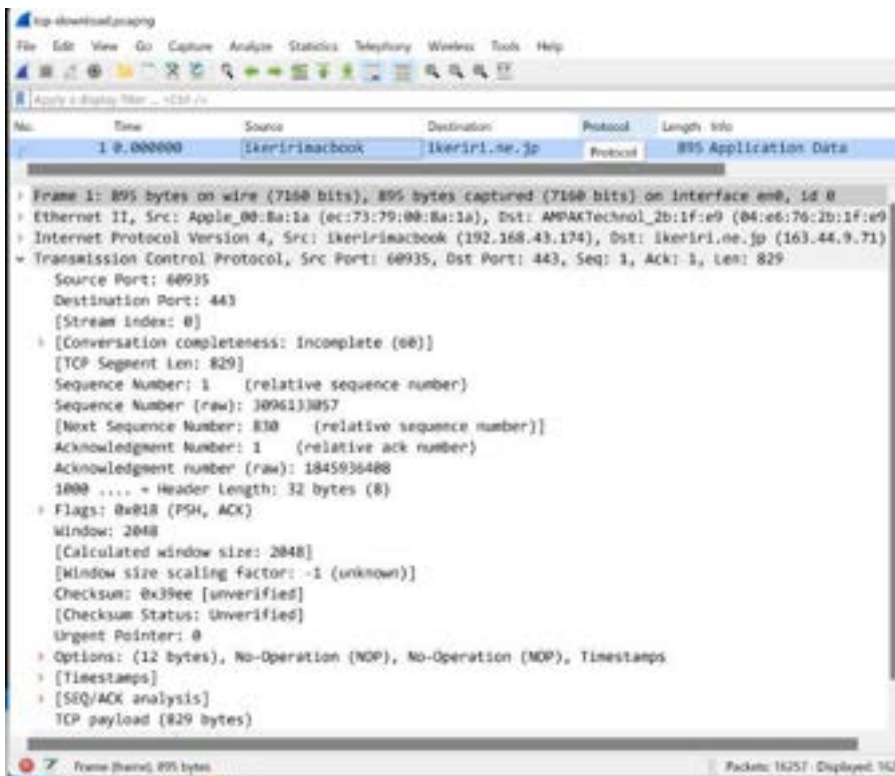
Name	Explanations	Display Filter
TCP retransmission	Same sequence number Max: 5 times (Windows11)	tcp.analysis.retransmission
TCP duplicate ACK	Same acknowledge number Max: 3 times (Fast retransmission)	tcp.analysis.duplicate_ack

- Open tcp-download.pcapng and check the Expert Info window

# Duplication of seq/ack number



- Open the Expert Information window to count the number of TCP retransmission, D\_ACK, fast retransmission



Severity	Summary	Group	Protocol	Count
Error	Record fragment length is too small or too large	Protocol	TLS	124
Warning	Connection reset (RST)	Sequence	TCP	3
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	13
Warning	Ignored Unknown Record	Protocol	TLS	7463
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	10
Note	This frame undergoes the connection closing	Sequence	TCP	1
Note	This frame initiates the connection closing	Sequence	TCP	1
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	9
Note	This frame is a (suspected) retransmission	Sequence	TCP	42
Note	Duplicate ACK	Sequence	TCP	248
Chat	Connection finish (FIN)	Sequence	TCP	2
Chat	TCP window update	Sequence	TCP	106

1 TCP connection using TLS(HTTP)  
18MB file download using WiFi

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# TCP Retransmission Rate

- Expert Info tells us the count of retransmission

”This frame is a (suspected) retransmission”



Severity	Summary	Group	Protocol	Count
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	9
Note	This frame is a (suspected) retransmission	Sequence	TCP	42
14	[TCP Retransmission] 443 → 60935 [ACK] Seq=9857 Ack=830 Win=135 L...	Sequence	TCP	
15	[TCP Retransmission] 443 → 60935 [ACK] Seq=11265 Ack=830 Win=135 ...	Sequence	TCP	
802	[TCP Fast Retransmission] 443 → 60935 [ACK] Seq=677249 Ack=830 Win...	Sequence	TCP	
822	[TCP Retransmission] 443 → 60935 [ACK] Seq=691329 Ack=830 Win=13...	Sequence	TCP	
823	[TCP Retransmission] 443 → 60935 [ACK] Seq=692737 Ack=830 Win=13...	Sequence	TCP	
824	[TCP Retransmission] 443 → 60935 [ACK] Seq=694145 Ack=830 Win=13...	Sequence	TCP	
825	[TCP Retransmission] 443 → 60935 [ACK] Seq=695553 Ack=830 Win=13...	Sequence	TCP	
826	[TCP Retransmission] 443 → 60935 [ACK] Seq=696961 Ack=830 Win=13...	Sequence	TCP	
827	[TCP Retransmission] 443 → 60935 [ACK] Seq=698369 Ack=830 Win=13...	Sequence	TCP	
828	[TCP Retransmission] 443 → 60935 [ACK] Seq=699777 Ack=830 Win=13...	Sequence	TCP	
829	[TCP Retransmission] 443 → 60935 [ACK] Seq=701185 Ack=830 Win=13...	Sequence	TCP	
830	[TCP Retransmission] 443 → 60935 [ACK] Seq=702593 Ack=830 Win=13...	Sequence	TCP	
831	[TCP Retransmission] 443 → 60935 [ACK] Seq=704001 Ack=830 Win=13...	Sequence	TCP	
832	[TCP Retransmission] 443 → 60935 [ACK] Seq=705409 Ack=830 Win=13...	Sequence	TCP	
833	[TCP Retransmission] 443 → 60935 [ACK] Seq=706817 Ack=830 Win=13...	Sequence	TCP	
834	[TCP Retransmission] 443 → 60935 [ACK] Seq=708225 Ack=830 Win=13...	Sequence	TCP	
835	[TCP Retransmission] 443 → 60935 [ACK] Seq=709633 Ack=830 Win=13...	Sequence	TCP	
839	[TCP Retransmission] 443 → 60935 [ACK] Seq=711041 Ack=830 Win=13...	Sequence	TCP	
840	[TCP Retransmission] 443 → 60935 [ACK] Seq=712449 Ack=830 Win=13...	Sequence	TCP	
1622	[TCP Fast Retransmission] Ignored Unknown Record	Sequence	TCP	
1623	[TCP Retransmission] 443 → 60935 [ACK] Seq=1534721 Ack=830 Win=1...	Sequence	TCP	
1624	[TCP Retransmission] 443 → 60935 [ACK] Seq=1536130 Ack=830 Win=1...	Sequence	TCP	

Statistics>Conversations#sf24us>TCP and count the packets from the server ( from port 443)

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Ref Start	Duration	Bits/s A → B	Bits/s B → A
192.168.43.138	80935	192.168.43.71	443	12,253	18 MB	0	4,004	200 KB	12,253	18 MB	175 Mbps	11 Mbps		

12,253 packets are sent from the server and 42 packets may be retransmission.

$$42/12253 \approx 0.00342..$$

Retransmission Rate: 0.34%

# TCP retransmission Rate by tshark



#sf24us

- Counting retransmission rate

(1) All TCP packets from server

```
tshark -r tcp-download -Y tcp.srcport==443 | wc -l
```

(2) retransmitted TCP packets from the server

```
tshark -r tcp-download
```

```
-Y "tcp.srcport==443 and tcp.analysis.retransmission" | wc -l
```

```
C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r tcp-download.pcapng -Y tcp.srcport==443 | wc -l
12253

C:\Users\megumitakeshita\Desktop\finding_duplications>tshark -r tcp-download.pcapng -Y "tcp.srcport==443 and tcp.analysis.retransmission" | wc -l
42
```

- The percentage (frequency) of TCP retransmissions is important  
 $42/12253 \approx 0.0034$  Retransmission Rate: 0.34%

# Duplication of sequence number by tshark

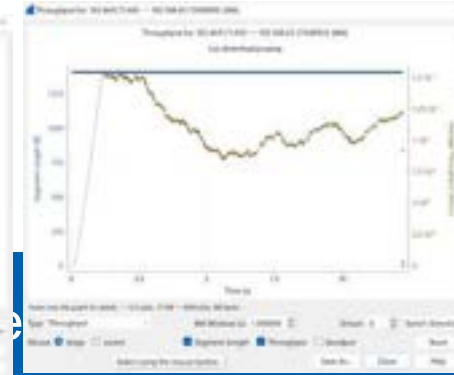
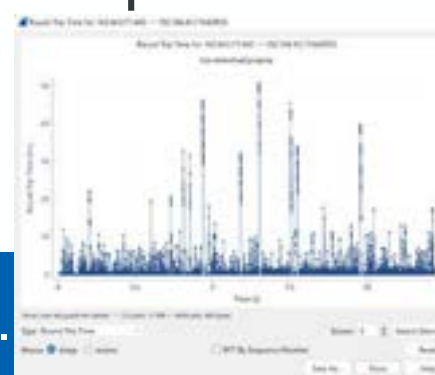
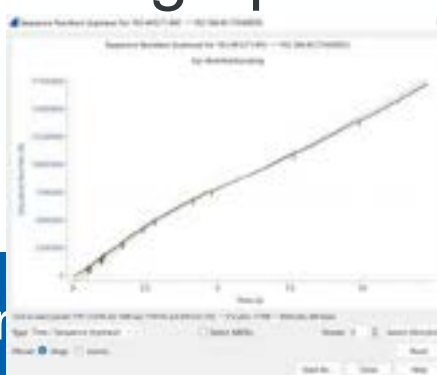


- Threadshold of TCP Retransmission rate

<https://wireshark.marwan.ma/lists/ethereal-users/200601/msg00149.html>

#sf24us

- It depends on the application, but TCP is designed with some retransmissions. Some applications may break 0.5%, but less than 1% is excellent, 3-5% is acceptable, and over 5% may be excessive retransmission.
- Open Statistics>Conversations>TCP and pick up the stream, then create TCP stream graph may help us TCP connection.



Sample traces and configuration



## Calculate TCP Retransmission rate by Post Dissector

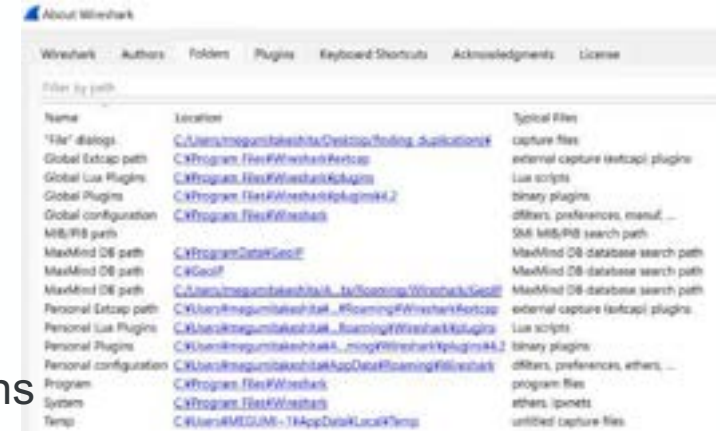


#sf24us

- Wireshark can add Post Dissector to dissect packets after Wireshark finished dissection.
- Dissectors are used for analyzing new protocols, but Post Dissectors are used for additional analyzing process after Wireshark finished packet Dissections.
- We can create a Post Dissector by Lua script to calculate TCP retransmission rate in each connection.  
(It's not so difficult; we have GPTs!!)

# Lua support of Wireshark

- Help>About Wireshark>Folder tab  
Check Personal Lua Plugins location  
C:\Users\megumitakeshita\AppData\Roaming\Wireshark\plugins



Name	Location	Typical Files
"file" dialogs	C:\Users\megumitakeshita\Desktop\Wireshark_Suite\bin	capture files
Global Extraop path	C:\Program Files\Wireshark\Extraop	external capture (extraop) plugins
Global Lua Plugins	C:\Program Files\Wireshark\Plugins	Lua scripts
Global Plugins	C:\Program Files\Wireshark\Plugins4.2	binary plugins
Global configuration	C:\Program Files\Wireshark	filters, preferences, manual, ...
MS/PS path		MS/PS search path
MaxMind DB path	C:\ProgramData\GeoIP	MaxMind DB database search path
MaxMind DB path	C:\GeoIP	MaxMind DB database search path
MaxMind DB path	C:\Users\megumitakeshita\AppData\Roaming\Wireshark\GeoIP	MaxMind DB database search path
Personal Extraop path	C:\Users\megumitakeshita\AppData\Roaming\Wireshark\Extraop	external capture (extraop) plugins
Personal Lua Plugins	C:\Users\megumitakeshita\AppData\Roaming\Wireshark\Plugins	Lua scripts
Personal Plugins	C:\Users\megumitakeshita\AppData\Roaming\Wireshark\Plugins4.2	binary plugins
Personal configuration	C:\Users\megumitakeshita\AppData\Roaming\Wireshark	filters, preferences, ethers, ...
Program	C:\Program Files\Wireshark	program files
System	C:\Program Files\Wireshark	ethers, ipnet
Temp	C:\Users\MEDIUM-1\AppData\Local\Temp	untitled capture files

## Install Visual Studio Code and Extensions

(<https://azure.microsoft.com/ja-jp/products/visual-studio-code>)



- Install VS Code and Lua Extensions
- Recommend to install Lua Debug  
it supports the completion for variable,  
functions and so on.

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# TCP retransmission Rate by Post Dissector



#sf24us

Recommend to read and Wireshark Developer Guide

**Proto** object used for Dissector/PostDissector

**Field** object is used for Wireshark field

**Proto.init()** is necessary function for initializing Dissector

**Proto.dissector** is used for describing packet analysing process.

**Treeltem** is used for Wireshark Dissection tree entry in packet detail pane

1. Define Post Dissector, display filter name is retrans, and Name is "Retransmission Analysis"

-- Define a new Proto for the Post Dissector

```
local retrans_proto = Proto("retrans", "Retransmission Analysis")
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# TCP retransmission Rate by Post Dissector



#sf24us

2. Define Wireshark fields used in post dissecting

```
-- Define the fields to extract
```

```
local ip_src_f = Field.new("ip.src")
```

```
local retrans_f = Field.new("tcp.analysis.retransmission")
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# TCP retransmission Rate by Post Dissector



#sf24us

3. Define the init function to process Post Dissector.

```
-- Initialize function to reset the table
```

```
function retrans_proto.init()
```

```
  ip_stats = {}
```

```
end
```

ip_stats [1.1.1.1]	ip_stats [2.2.2.2]	...
-----------------------	-----------------------	-----

# TCP retransmission Rate by Post Dissector

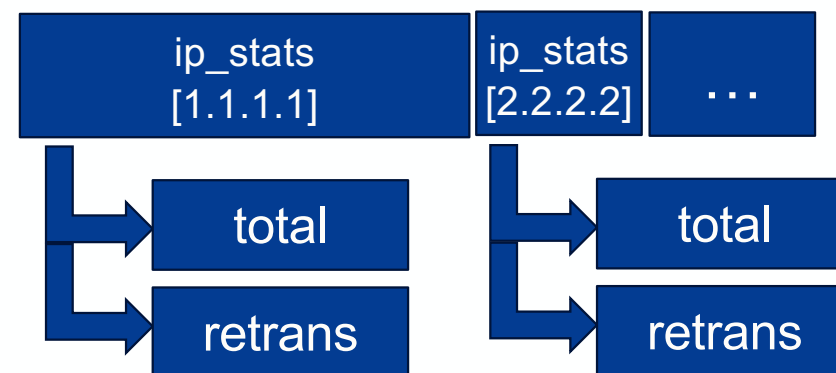


#sf24us

4. Define a helper function

to count retransmissions by each ip address.

```
-- Function to add counts to the table
local function add_ip_count(ip, is_retrans)
  if not ip_stats[ip] then
    ip_stats[ip] = { total = 0, retrans = 0 }
  end
  ip_stats[ip].total = ip_stats[ip].total + 1
  if is_retrans then
    ip_stats[ip].retrans = ip_stats[ip].retrans + 1
  end
end
```



# TCP retransmission Rate by Post Dissector



#sf24us

Now we describe the analyzing process using Wireshark API,

**Proto.dissector** function has three parameters: tvb, pinfo, TreeItem

**tvb** means Testy Virtual buffer, actual packet data objectz

**pinfo** means dissected packet data, we can refer like pinfo.src\_ipz

**TreeItem** means the object of Wireshark dissection trees in the packet detail pane.

5. Define dissector function (buffer as tvb, pinfo and tree as TreeItem)

```
-- Dissector function
```

```
function retrans_proto.dissector(buffer, pinfo, tree)
```

```
-- extract fields from definitions
```

```
local ip_src = ip_src_f()
```

```
local is_retrans = retrans_f()
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# TCP retransmission Rate by Post Dissector



#sf24us

6. Calculate the retransmission rate in an easy way

(It is accurate all packets are TCP, there are no Non-TCP packets)

TCP retransmission packets (tcp.analysis.retransmission)

Retransmission rate =  $\frac{\text{TCP retransmission packets (tcp.analysis.retransmission)}}{\text{all TCP packets by each source IP address}}$

```
if ip_src then
```

```
  add_ip_count(tostring(ip_src), is_retrans ~= nil)
```

```
end
```

```
-- Create a subtree for retransmission analysis
```

```
local subtree = tree:add(retrans_proto, "Retransmission Analysis")
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>40</sup>



# TCP retransmission Rate by Post Dissector



#sf24us

```
-- Add statistics to the subtree
for ip, stats in pairs(ip_stats) do
  local retrans_ratio = (stats.retrans / stats.total) * 100
  local ip_node = subtree:add(retrans_proto, "IP: " .. ip)
  ip_node:add(retrans_proto, string.format("Total Packets: %d", stats.total))
  ip_node:add(retrans_proto, string.format("Retransmissions: %d", stats.retrans))
  ip_node:add(retrans_proto, string.format("Retransmission Ratio: %.2f%%",
retrans_ratio))
end
end
```

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>41</sup>

# TCP retransmission Rate by Post Dissector



#sf24us

7. Register this Proto as PostDissector

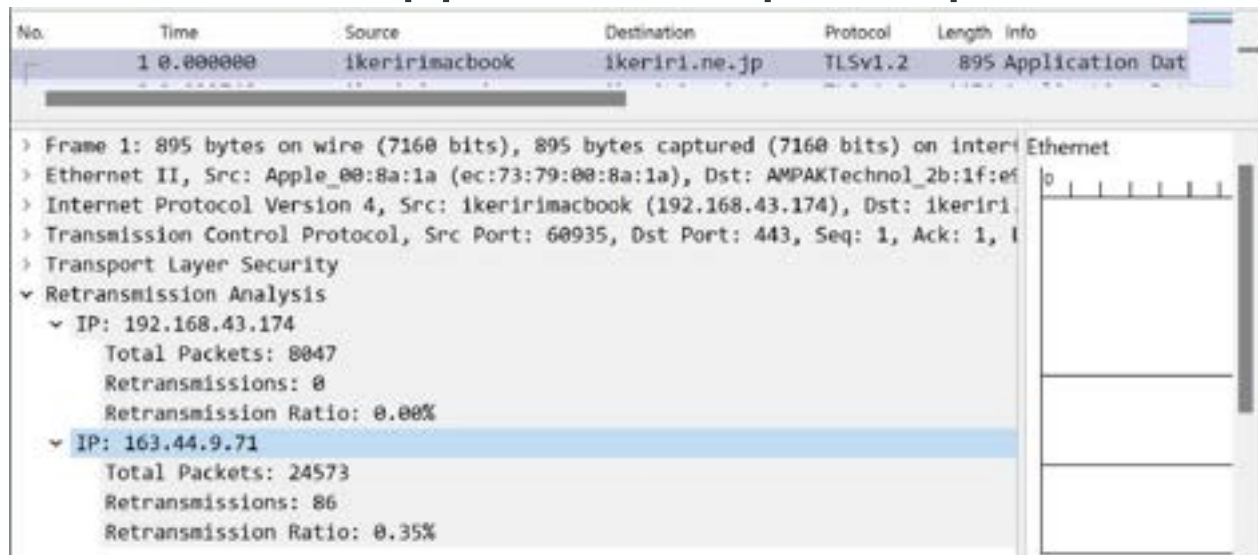
```
-- Register the dissector as a postdissector  
register_postdissector(retrans_proto)
```

# TCP retransmission Rate by Post Dissector



#sf24us

- Copy tcp-download-postdissector.lua into your personal plugin folder at Wireshark personal Configuration (e.x. C:\Users\megumitakeshita\AppData\Roaming\Wireshark\plugins)
- Close all Wireshark apps and reopen tcp-download.pcapng



Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Duplication ACK and Fast retransmission



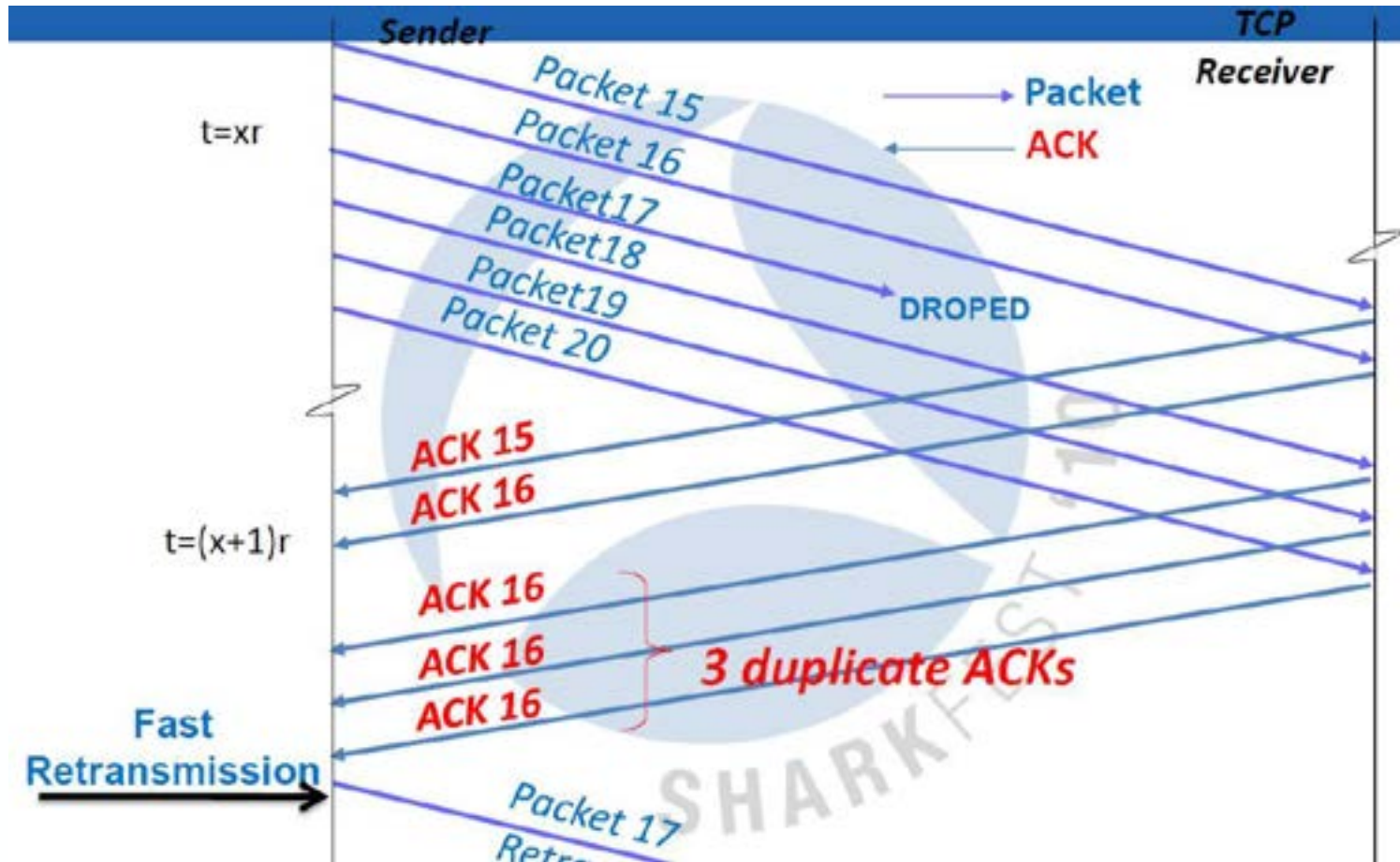
#sf24us

- We capture the trace on the client side, and check the retransmission from the server
- Next, think about Duplication ACK
- The client sends Duplication ACK if the expected segment has not arrived during the RTO timer and sends again if the client does not receive the segment
- If the server receives 3 continuous Duplicate ACK, Server TCP thinks the segment was lost and sends the lost segment immediately (Fast retransmission)

# Fast Retransmission algorithm



#sf24us



Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Frequency of Duplication ACK / Fast retransmission



#sf24us

- Total ACK frame from the client side 4004pkts (tcp.ack and tcp.srcport==60935)
- The same ACK number means Duplicate ACK 248pkts (tcp.analysis.duplicate\_ack) 248/4004=0.0619.. 6.19%
- Three continuous Duplicate ACK causes Fast retransmission 9pkts (tcp.analysis.fast\_retransmission) 9/4004=0.00224.. 0.224%
- The rate of D\_ACK seems bigger than expected, but fast retransmission rate is under 1%(0.224%), it's OK.

Seq	Duplicate ACK	Sequence	TCP
12	[TCP Seq ACK 1141] 60935 -- 440 [ACK] Seq=680 Ack=6837 Win=1916	Sequence	TCP
58	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
388	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
382	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
354	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
446	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
488	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
671	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
671	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
675	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP
678	[TCP Seq ACK 654] 60935 -- 440 [ACK] Seq=680 Ack=677249 Win=L	Sequence	TCP

Note	This frame is a suspected fast retransmission	Sequence	TCP
802	[TCP Fast Retransmission] 443 -- 60935 [ACK] Seq=677249 Ack=830 Win=...	Sequence	TCP
1422	[TCP Fast Retransmission], Ignored Unknown Record	Sequence	TCP
2831	[TCP Fast Retransmission], Ignored Unknown Record	Sequence	TCP
4065	[TCP Fast Retransmission], Ignored Unknown Record	Sequence	TCP
4688	[TCP Fast Retransmission], Encrypted Handshake Message	Sequence	TCP
6373	[TCP Fast Retransmission] 443 -- 60935 [ACK] Seq=683209 Ack=830 Win=...	Sequence	TCP
7132	[TCP Fast Retransmission], Ignored Unknown Record	Sequence	TCP
10299	[TCP Fast Retransmission], Ignored Unknown Record	Sequence	TCP
13139	[TCP Fast Retransmission] 443 -- 60935 [ACK] Seq=11909633 Ack=830 ...	Sequence	TCP

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Sequence/Acknowledgement number duplications tells us



#sf24us

1. It is a common event in TCP connection; don't worry.
2. Count the rate of duplication

TCP retransmission (from Server) `tcp.analysis.retransmission`

Duplicate ACK (from Client side) `tcp.analysis.duplicate_ack`

Fast Retransmission (from Client side) `tcp.analysis.fast_retransmission`

under 1% Excellent 1-5% Acceptable over 5% latency or quality

Use `tracert` to count the latency between the client and server,

Use `pathping` to check the quality (lost %) of the circuit.

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>47</sup>

# Appendix

## Duplication of Port number



[TCP Port numbers reused] 48830 → 1063 [SYN] Seq=0 Win=29200

```
Apply a display filter ... <Ctrl>F
No.      Time           Source           Destination      Protocol  Length  Info
-----
200351  69.288649268  192.168.100.101  192.168.100.35  TCP       74      [TCP Port numbers reused] 48830 → 1063 [SYN]
200352  69.288670072  192.168.100.101  192.168.100.36  TCP       74      48974 → 1185 [SYN] Seq=0 Win=29200 Len=0 MSS=

Frame 200351: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
Ethernet II, Src: Toshiba_8b:1d:73 (b8:6b:23:8b:1d:73), Dst: VMware_a7:4a:f3 (00:0c:29:a7:4a:f3)
Internet Protocol Version 4, Src: 192.168.100.101 (192.168.100.101), Dst: 192.168.100.35 (192.168.100.35)
Transmission Control Protocol, Src Port: 48830, Dst Port: 1063, Seq: 0, Len: 0
  Source Port: 48830
  Destination Port: 1063
  [Stream Index: 103685]
  [Conversation completeness: Incomplete (37)]
  [TCP Segment Len: 0]
  Sequence Number: 0 (relative sequence number)
  Sequence Number (raw): 1014183697
  [Next Sequence Number: 1 (relative sequence number)]
  Acknowledgment Number: 0
  Acknowledgment number (raw): 0
  1010 ... = Header Length: 40 bytes (10)
  Flags: 0x0002 (SYN)
  Window: 29200
  [Calculated window size: 29200]
  Checksum: 0x4a08 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  [Timestamps]
  [SEQ/ACK analysis]
  [RTT: 0.000305182 seconds]
  [TCP Analysis Flags]
  [Expert Info (Note/Sequence): A new tcp session is started with the same ports as an earlier session in this trace]
  [A new tcp session is started with the same ports as an earlier session in this trace]
  [Severity level: Note]
  [Group: Sequence]
```

Open tcp-portdup.pcapng go to frame #200351 we find [Expert Info (Note/Sequence): A new tcp session is started with the same ports as an earlier session in this trace]

[riri.ne.jp/sharkfest/sf24ikeriri.zip](http://riri.ne.jp/sharkfest/sf24ikeriri.zip)



# Duplication of Port number



- Sake-san's comments in ASK Wireshark  
<https://ask.wireshark.org/question/26247/tcp-port-numbers-reused/>  
“The wireshark note “[TCP Port numbers reused]” means that in the packet capture file, there is a new connection for a 5-tuple (ip-src,ip-dst,protocol,srcport,dstport) that was seen before...”
- Open Expert info and find [TCP Port numbers reused]  
[A new tcp session is started with the same ports as an earlier session in this trace] display filter is tcp.analysis.reused\_port

Note	A new tcp session is started with the same ports as an earlier session in this trace	Sequence	TCP	4
200351	[TCP Port numbers reused] 48830 → 1063 [SYN] Seq=0 Win=29200 Len=0 MSS=14...	Sequence	TCP	
218093	[TCP Port numbers reused] 44244 → 28201 [SYN] Seq=0 Win=29200 Len=0 MSS=1...	Sequence	TCP	
218117	[TCP Port numbers reused] 41282 → 28201 [SYN] Seq=0 Win=29200 Len=0 MSS=1...	Sequence	TCP	
231234	[TCP Port numbers reused] 48524 → 125 [SYN] Seq=0 Win=29200 Len=0 MSS=146...	Sequence	TCP	

# Duplication of Port number



- The server uses well-known or registered port numbers,  
The client uses private port numbers from 49152 to 65535
- If you capture huge traffic for the long term ( 1month of office traffic),  
ip.src, ip.dst, protocol, srcport and dstport are duplicated by chance
- But why this trace records 4 TCP Port number reused? click  
Statistics>Conversations>TCP, there are 131123 TCP connections,  
and most of them are not  
completed half connection
- This trace is a capture of  
TCP port scan by nmap

Address A	Port A	Address B	Port B	Protocols	Bytes	Streams	Protocols A -> B	Bytes A -> B	Protocols B -> A	Bytes B -> A	Rel. Start	Duration	Strs. A -> B	Strs. B -> A
192.168.188.11	55287	192.168.188.101	80	TCP	1462	1	TCP	0 bytes	1	34 bytes	42.120885	0:0000	0	0
192.168.188.11	54666	192.168.188.101	80	TCP	1454	1	TCP	0 bytes	1	34 bytes	42.121210	0:0000	0	0
192.168.188.11	58102	192.168.188.101	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	48.311886	0:0000	0	0
192.168.188.11	52412	192.168.188.101	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	48.287151	0:0000	0	0
192.168.188.101	56230	172.217.161.236	443	TCP	148	1	TCP	0 bytes	1	471 bytes	25.495251	0:0000	0	0
192.168.188.101	32788	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.366605	0:0000	0	0
192.168.188.101	32830	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.367111	0:0000	0	0
192.168.188.101	32838	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.368410	0:0000	0	0
192.168.188.101	32838	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.368410	0:0000	0	0
192.168.188.101	32846	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.369110	0:0000	0	0
192.168.188.101	32830	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.369110	0:0000	0	0
192.168.188.101	32860	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.369610	0:0000	0	0
192.168.188.101	32860	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.369610	0:0000	0	0
192.168.188.101	32870	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.370110	0:0000	0	0
192.168.188.101	32880	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.370610	0:0000	0	0
192.168.188.101	32884	192.215.16.115	80	TCP	1459	1	TCP	0 bytes	1	34 bytes	77.371110	0:0000	0	0

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>

# Duplication of Port number



- Ex. “ip.addr==192.168.100.121 and tcp.port==80”

No.	Time	Source	Destination	Protocol	Length	Info
8966	41.899066700	192.168.100.101	192.168.100.121	TCP	74	37882 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3315228474 TSecr=0 WS=128
9000	41.899498355	192.168.100.121	192.168.100.101	TCP	74	80 → 37882 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM TSval=16552039 TSecr=3315228474 WS=32
9002	41.899506717	192.168.100.101	192.168.100.121	TCP	66	37882 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315228475 TSecr=16552039
9058	41.899854358	192.168.100.101	192.168.100.121	TCP	66	37882 → 80 [RST, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315228475 TSecr=16552039
130392	68.757860839	192.168.100.101	192.168.100.121	TCP	74	42314 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3315255333 TSecr=0 WS=128
130453	68.758332488	192.168.100.121	192.168.100.101	TCP	74	80 → 42314 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM TSval=16554699 TSecr=3315255333 WS=32
130454	68.758337598	192.168.100.101	192.168.100.121	TCP	66	42314 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315255334 TSecr=16554699
130511	68.758626860	192.168.100.101	192.168.100.121	TCP	66	42314 → 80 [RST, ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315255334 TSecr=16554699

- This trace captured TCP connect scan, accessing a lot of port in a short term, so sometimes there are duplication of port numbers.
- Connection reset (RST) 112149
- Connection establish (SYN) 131142

Severity	Summary	Group	Protocol	Count
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	1
Warning	Previous segment(s) not captured (common at capture start)	Sequence	TCP	1
Warning	D-SACK Sequence	Sequence	TCP	2
Warning	Connection reset (RST)	Sequence	TCP	112149
Note	ACK to a TCP keep-alive segment	Sequence	TCP	1
Note	TCP keep-alive segment	Sequence	TCP	1
Note	This frame undergoes the connection closing	Sequence	TCP	2
Note	A new tcp session is started with the same ports as an earlier session in t...	Sequence	TCP	4
Note	The SYN packet does not contain a SACK_PERM option	Protocol	TCP	4
Note	This frame initiates the connection closing	Sequence	TCP	6
Note	Padding identification may be inaccurate and impact trailer disector	Protocol	Ethertype	25896
Note	This frame is a (suspected) retransmission	Sequence	TCP	26
Chat	TCP window update	Sequence	TCP	7
Chat	Connection finish (FIN)	Sequence	TCP	8
Chat	Connection establish acknowledgement (SYN+ACK)	Sequence	TCP	887
Chat	Connection establish request (SYN)	Sequence	TCP	131142

Sample traces and configurations <https://www.informatica.com/resources/whitepapers/whitepaper>

# Port number duplications tells us



#sf24us

It merely happens Port number duplications,

1. It happens by accident in a long-term huge trace file
2. Port scan
3. Multi-layer NAT/NAPT at WAN side

**duplications are good indicators to find  
the clues of network/security troubleshooting**

# USE WIRESHARK

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip> <sup>52</sup>

# USE WIRESHARK



#sf24us

# Thank you for watching.

Please complete the Google form survey

[trace files and Wireshark profiles are here:](https://www.ikeriri.ne.jp/sharkfest/)

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

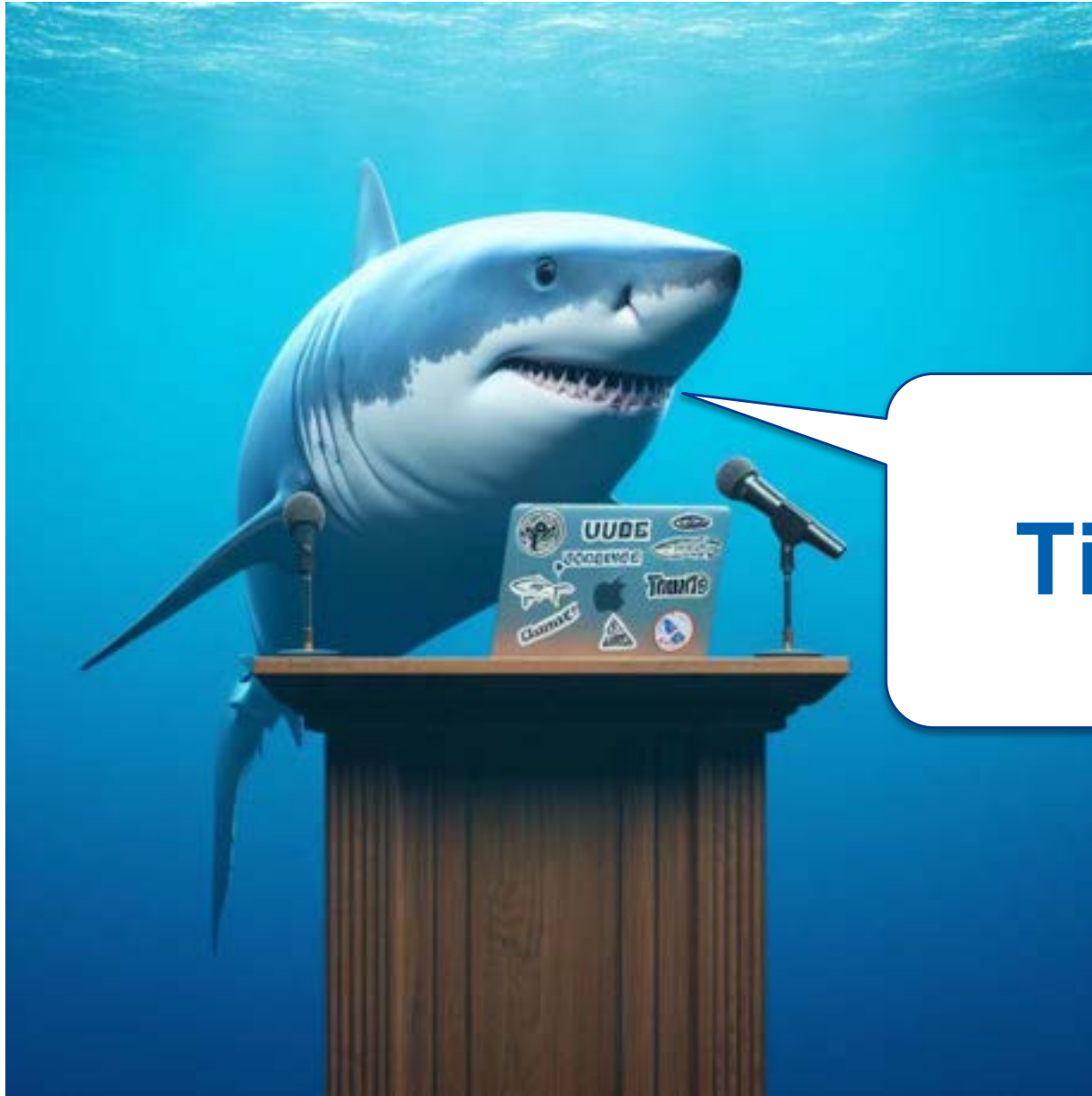
[sf24ikeriri.zip](https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip)



ikeriri network service

<http://www.ikeriri.ne.jp>

Sample traces and configurations <https://www.ikeriri.ne.jp/sharkfest/sf24ikeriri.zip>



**SharkFest'24 US**  
June 15-20 • Fairfax, VA

#sf24us

**Time for Q & A**