Aalto University
School of Electrical
Engineering

ELEC-E7130 Internet Traffic Measurements and Analysis

Assignment 4. Traffic with probe packets

Name: Xingji Chen

Student ID: 101659554

E-mail: xingji.chen@aalto.fi

Aalto University School of Electrical Engineering

Task 1: Packet capture with probe packets

1. Use the subprocess module to run Ping commands to test network connectivity and response times for multiple destination addresses, recording the date and time stamp of each command. Write the results of the Ping test to a text file associated with each destination address. Information about the destination addresses and the output files is stored in a list called destinations. Perform a Ping test for each destination address and save the results to the appropriate file for subsequent analysis and monitoring of network performance.

```
import subprocess
import time

def ping_and_save(destination, output_file):
    current_datetime = time.strftime("%Y-%m-%d %H:%M:%S")
    with open(output_file, 'a') as file:
        file.write(current_datetime + '\n')

    try:
        ping_result = subprocess.run(['ping', '-n', '5', destination],
capture_output=True, timeout=5, text=True)
        with open(output_file, 'a') as file:
            file.write(ping_result.stdout + '\n')
    except subprocess.TimeoutExpired:
        with open(output_file, 'a') as file:
            file.write("Ping 超时\n")

destinations = [
    {"name": "ok1.iperf.comnet-student.eu", "ip": "195.148.124.36"},
    {"name": "blr1.iperf.comnet-student.eu", "ip": "142.93.213.224"},
    {"name": "cbg-uk.ark.caida.org", "ip": "128.232.97.9"},
    {"name": "bjl-gm.ark.caida.org", "ip": "196.46.233.22"},
    {"name": "msy-isu.ark.caida.org", "ip": "non"}
]

output_directory = "ping\\"

for destination in destinations:
    output_file = output_directory + destination["name"] + ".txt"
    ping_and_save(destination["name"], output_file)
```

2. Capture network packets from a WLAN interface and save them to a PCAP file. Use PowerShell to retrieve network statistics for the WLAN network adapter.

```
import subprocess

dumpcap_command = 'dumpcap -i "WLAN" -w "D:\\as4\\CHENXINJI2.pcap"'
subprocess.run(dumpcap_command, shell=True)

powershell_command = 'powershell Get-NetAdapterStatistics -Name
"WLAN"'
subprocess.run(powershell_command, shell=True)
```

3. The iperf3 tool for network performance testing is used to connect to separate target servers with the option to specify custom ports.

```
D:\iperf3.exe -c ok1.iperf.comnet-student.eu
D:\iperf3.exe -c blr1.iperf.comnet-student.eu -p 5203
```

4. The tshark tool, which is a command line version of Wireshark, was used to extract session statistics of IP and TCP packets from different PCAP files for network packet analysis and performance evaluation.

```
tshark -r icmp.pcap -q -z conv,ip > icmp.txt
tshark -r iperf.pcap -q -z conv,tcp > iperf.txt
```

0. Sanity checks

Size of trace file	1102294535 bytes
Number of packets in trace file	900712
Total size of packets	1102294535 bytes

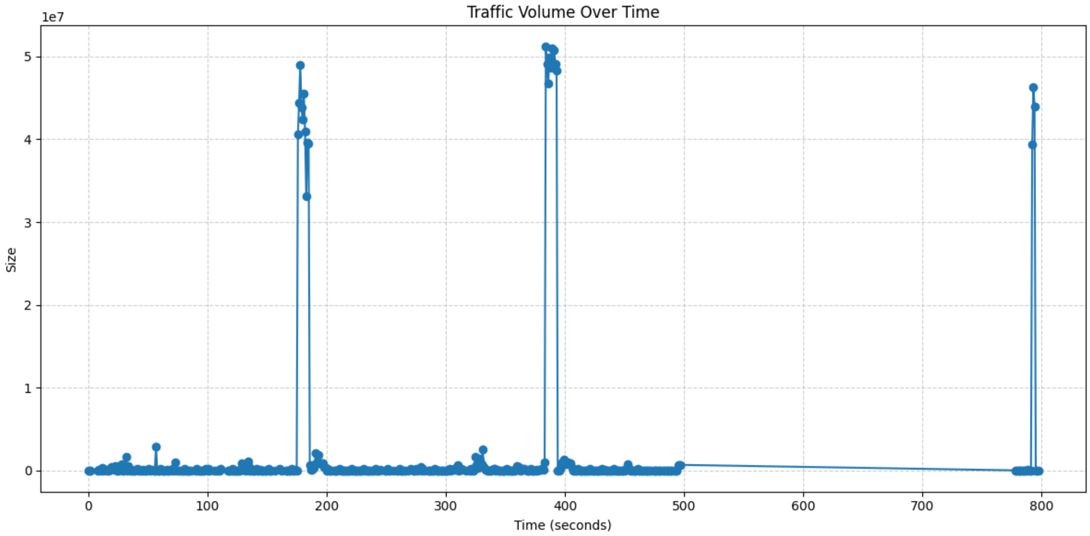
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Tp-LinkT_c3:e6:ed	Broadcast	ARP	60	Who has 192.168.1.102? Tell 192.168.1.1
2	0.495915	192.168.1.110	199.232.166.91	TCP	55	54187 → 443 [ACK] Seq=1 Ack=1 Win=1026 Len=1 [TCP segment of a reassembled PDU]
3	0.799262	199.232.166.91	192.168.1.110	TCP	66	443 → 54187 [ACK] Seq=1 Ack=2 Win=288 Len=0 SLE=1 SRE=2
4	0.953963	192.168.1.110	114.114.114.114	TCP	66	54225 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
5	0.954313	192.168.1.110	114.114.114.114	TCP	66	54226 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
6	1.024201	Tp-LinkT_c3:e6:ed	Broadcast	ARP	60	Who has 192.168.1.102? Tell 192.168.1.1
7	1.096508	114.114.114.114	192.168.1.110	TCP	66	53 → 54225 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
8	1.096508	114.114.114.114	192.168.1.110	TCP	66	53 → 54226 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM
9	1.096896	192.168.1.110	114.114.114.114	TCP	54	54225 → 53 [ACK] Seq=1 Ack=1 Win=64240 Len=0
10	1.097109	192.168.1.110	114.114.114.114	TCP	54	54226 → 53 [ACK] Seq=1 Ack=1 Win=64240 Len=0
11	1.097345	192.168.1.110	114.114.114.114	TCP	56	54226 → 53 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=2 [TCP segment of a reassembled PDU]
12	1.097553	192.168.1.110	114.114.114.114	DNS	97	Standard query 0x83c4 HTTPS gew4-spclient.spotify.com
13	1.097814	192.168.1.110	114.114.114.114	TCP	56	54225 → 53 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=2 [TCP segment of a reassembled PDU]
14	1.097983	192.168.1.110	114.114.114.114	DNS	97	Standard query 0x5109 A gew4-spclient.spotify.com
15	1.123133	192.168.1.110	114.114.114.114	TCP	66	54227 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
16	1.123867	192.168.1.110	114.114.114.114	TCP	66	54228 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
17	1.242964	114.114.114.114	192.168.1.110	TCP	54	53 → 54225 [ACK] Seq=1 Ack=3 Win=64240 Len=0
18	1.242964	114.114.114.114	192.168.1.110	TCP	54	53 → 54226 [ACK] Seq=1 Ack=3 Win=64240 Len=0
19	1.242964	114.114.114.114	192.168.1.110	DNS	164	Standard query response 0x5109 A gew4-spclient.spotify.com CNAME edge-web-gew4.dual-gslb.spotify.com A 35…
20	1.242964	114.114.114.114	192.168.1.110	DNS	222	Standard query response 0x83c4 HTTPS gew4-spclient.spotify.com CNAME edge-web-gew4.dual-gslb.spotify.com …
21	1.243323	192.168.1.110	114.114.114.114	TCP	54	54225 → 53 [ACK] Seq=46 Ack=112 Win=64130 Len=0
22	1.243466	192.168.1.110	114.114.114.114	TCP	54	54226 → 53 [ACK] Seq=46 Ack=170 Win=64072 Len=0
23	1.244079	192.168.1.110	114.114.114.114	TCP	54	54226 → 53 [FIN, ACK] Seq=46 Ack=170 Win=64072 Len=0
24	1.245563	192.168.1.110	114.114.114.114	TCP	54	54225 → 53 [FIN, ACK] Seq=46 Ack=112 Win=64130 Len=0
25	1.246991	192.168.1.110	35.186.224.17	QUIC	1292	Initial, DCID=419603646ccf6df4, PKN: 1, CRYPTO, CRYPTO, PADDING, CRYPTO, CRYPTO, PADDING
26	1.247645	192.168.1.110	35.186.224.17	TLSv1.2	137	Application Data
27	1.247889	192.168.1.110	35.186.224.17	TLSv1.2	93	Application Data
28	1.248091	192.168.1.110	35.186.224.17	TCP	1466	53976 → 443 [ACK] Seq=123 Ack=1 Win=1025 Len=1412 [TCP segment of a reassembled PDU]

Capture file statistics are related to the file itself and provide general information about the capture, while counters are statistics related to the content and behavior of network traffic. We can see that values from interface counters to capture file has some differences.

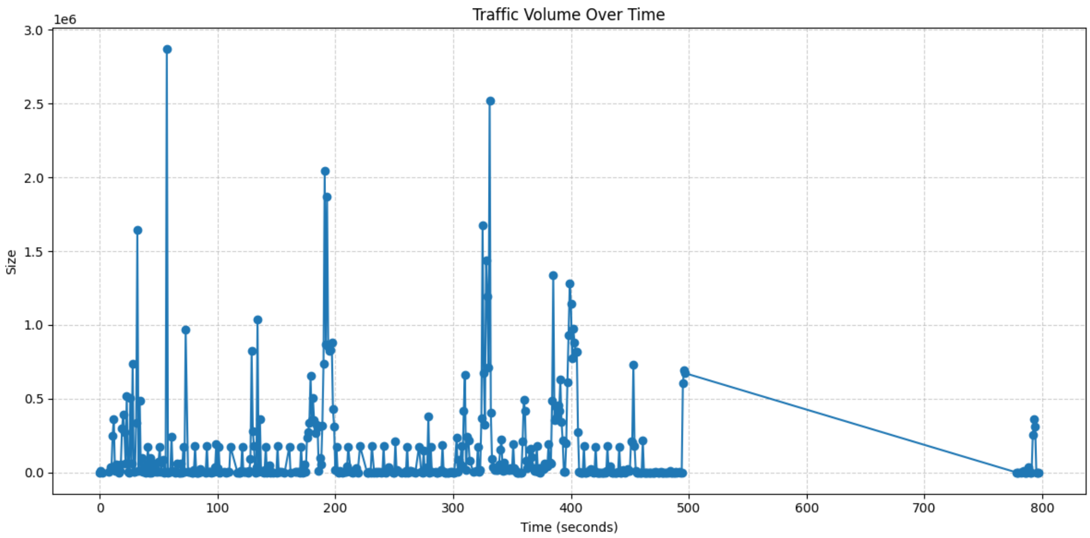
This is because capture file statistics are extracted and computed directly by analytics tools, such as Wireshark, based on file header information without the need to deeply analyze the content of the packets. Counters, on the other hand, require more detailed analysis and counting of each packet, and usually require more processing to obtain information about the packet contents.

Capture File Statistics is used to provide summary information about the capture file itself to help the user understand the overall characteristics of the data capture. Counters are used to analyze the actual content and behavior of network packets in order to gain a deeper understanding of the characteristics of network traffic, such as traffic distribution, error detection and protocol analysis.

1. Plot the traffic volume over time by considering all captured packets within the most appropriate time interval.



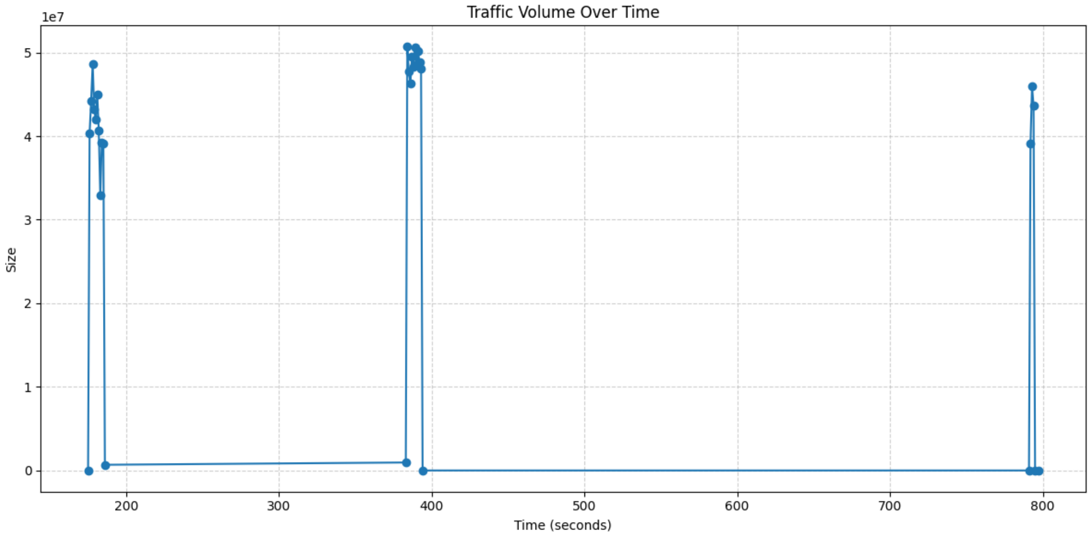
2. Plot the traffic volume without the ping packets and iperf3 packets over time (select the same interval selected in the previous plot).



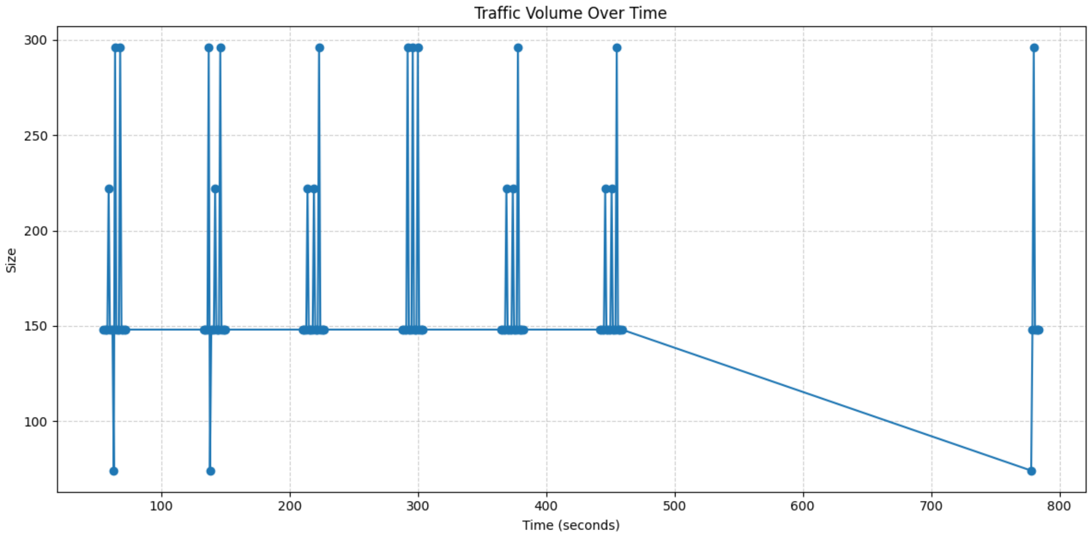
Aalto University
School of Electrical
Engineering

3. Plot the traffic volume comparing the ping packets with the iperf3 packets over time (keeping the same interval).

iperf3:



ping:



4. Provide the average throughput.

Average throughput	1382196.1011820212 bytes per second

5. Do you have any observations from the above plot of network traffic?

For the traffic volume over time by considering all captured packets within the most appropriate time interval, high traffic volume occurs only at certain moments, such as the 180th, 390th and 790th seconds, when the size of the traffic volume suddenly increases to 5*10^7 bytes, while at other moments the flow fluctuates and is basically very small.

For the traffic volume without the ping packets and iperf3 packets over time, the trend of the traffic volume is not so clear, we can observe that it has been fluctuating and the peaks occur irregularly.

For the traffic volume with the iperf3 packets over time, similar to the traffic volume over time by considering all captured packets, peaks occur at some certain moments which are different. Peaks occur at 100th second, 380th second and 790th second, reaching 5*10^7 bytes. And before or after these moments, the traffic volume stays at a very low level. For the ping packets, peaks occur more frequently, at 50th, 130th, 210th, 290th, 370th, 450th and 780th seconds. But the peak is much smaller than the former, and only reaches a maximum of 300 bytes, which indicates that the number of ping packets is very small.

Aalto University
School of Electrical
Engineering

Task 2: Compare active and passive measurements

1. This code performs the calculation and accumulation of packet lengths, separately calculating the total length of all packets, ICMP packets, and packets containing specific information.

```
with open('cxj.csv', 'r') as csvfile:
    # Create a CSV reader
    csvreader = csv.reader(csvfile)

    # Skip the header row
    next(csvreader)

    # Initialize variables
    total_packet_length = 0
    icmp_packet_length = 0
    custom_info_packet_length = 0

    # Iterate through each row
    for row in csvreader:
        # Get the length field
        packet_length = int(row[5])

        # Calculate the total packet length across all packets
        total_packet_length += packet_length

        # Calculate the packet length for ICMP protocol packets
        if row[4] == 'ICMP':
            icmp_packet_length += packet_length

        # Calculate the packet length for packets containing '5201' in
the Info field
        if '  5201 ' in row[6]:
            custom_info_packet_length += packet_length
```

2. The main function of this code is to perform byte to kilobyte unit conversion, and will be converted to the value and other relevant information returned to the Pandas Series.

```
ef convert_to_bytes(row):
    units = {'bytes': 1, 'kb': 1024, 'mb': 1024**2}
```

```
    try:
        ld_bytes_unit = str(row['ld_bytes_unit']).lower()
        factor = units[ld_bytes_unit]
        ld_kb = row['ld_bytes'] * factor

        rd_bytes_unit = str(row['rd_bytes_unit']).lower()
        factor = units[rd_bytes_unit]
        rd_kb = row['rd_bytes'] * factor

        total_bytes_unit = str(row['total_bytes_unit']).lower()
        factor = units[total_bytes_unit]
        total_kb = row['total_bytes'] * factor

        return pd.Series({'ld_bytes': ld_kb, 'rd_bytes': rd_kb,
'total_bytes': total_kb, 'server_ip': row['second_ip_interface']})
    except KeyError as e:
        print(f"Error processing row {row}: {e}")
        raise ValueError("Invalid unit. Supported units are 'bytes',
'kb', 'mb.")
```

3. The purpose of this code is to draw horizontal lines of different colors based on the conditions of active and passive measurements, depending on whether server_ip matches a particular IP address and port combination.

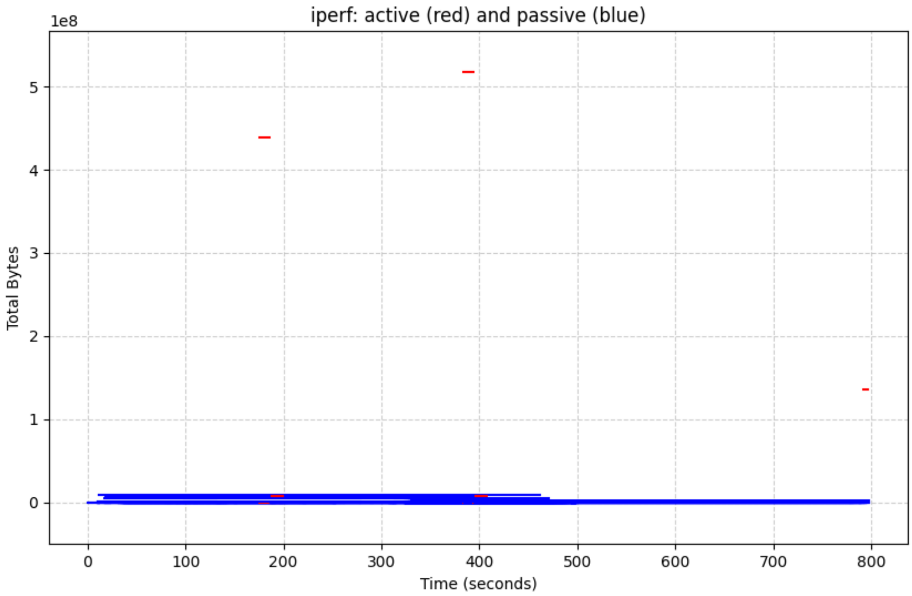
```
for (name1, name2, server_ip), group in groups:
    if server_ip.strip() in {'195.148.124.36:5201',
'142.93.213.224:5203'}:
        ax.plot([group['start'], group['start'] + group['duration']],
[group['total_bytes'], group['total_bytes'],], color='r')
    else:
        ax.plot([group['start'], group['start'] + group['duration']],
[group['total_bytes'], group['total_bytes'],], color='b')
```

Aalto University
School of Electrical
Engineering

1. How much traffic was there that was not iperf or ping traffic?

all packets	1102294535 bytes
packets use icmp	18870 bytes
packets use 5201 port	1034465646 bytes
traffic was there that was not iperf or ping traffic	67810019 bytes

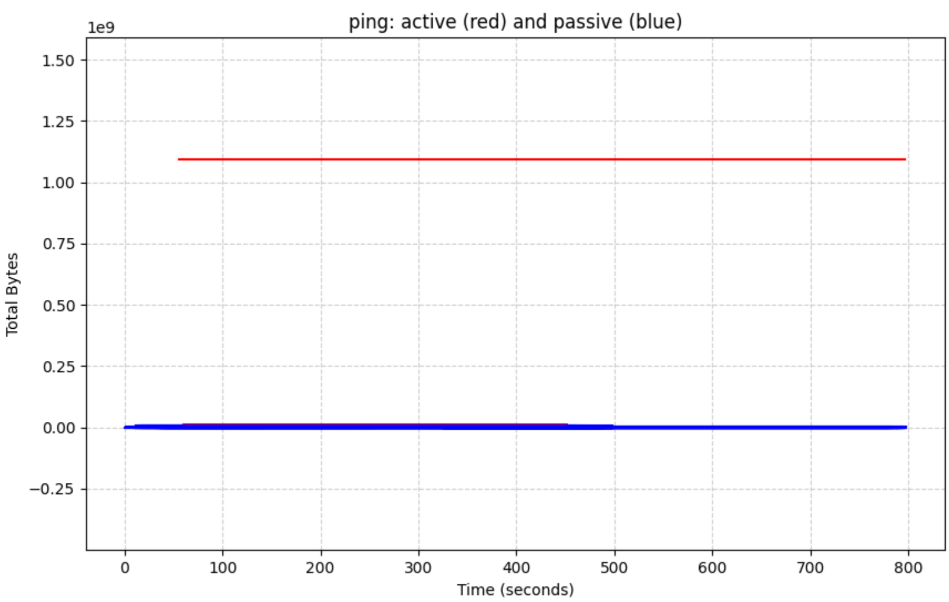
2. Compare iperf results from active and passive measurements. Provide a table and plot a time series.



Ip	Left direction frames	Left direction bytes	Right direction frames	Right direction bytes	Total frames	Total bytes	Start	Duration	Server ip
192.168.1.110:54405 <-> 195.148.124.36:5201	65565	3624960	324183	512753664	389748	516947968	383.924019	10.0548	195.148.124.36:5201

192.168.1.110:54405 <-> 195.148.124.36:5201	50298	2781184	275495	436207616	325793	438304768	175.960358	10.1746	195.148.124.36:5201
192.168.1.110:57283 <-> 195.148.124.36:5201	16527	913408	85246	134217728	101773	135266304	792.020745	5.0714	195.148.124.36:5201
192.168.1.110:54171 <-> 146.75.82.250:443	5511	8289280	2783	162816	8294	8452096	11.344449	450.2186	146.75.82.250:443
192.168.1.110:54269 <-> 208.80.154.240:443	5420	7857152	2756	221184	8176	8078336	18.438310	381.1625	208.80.154.240:443

3. Compare ping results from active and passive measurements. Provide a table and plot a time series.



Ip	Left direction frames	Left direction bytes	Right direction frames	Right direction bytes	Total frames	Total bytes	Start	Duration
192.168.1.110 <-> 195.148.124.36	132457	7153k	684989	1034M	817446	1041M	55.755887	741.3363
192.168.1.110 <-> 114.114.114.114	4743	394k	6728	425k	11471	819k	0.953963	792.6291
192.168.1.110 <-> 142.93.213.224	1738	95k	9248	13M	10986	13M	59.959400	391.3487
192.168.1.110 <-> 146.75.82.250	5511	8095k	2783	159k	8294	8254k	11.344449	450.2186
192.168.1.110 <-> 208.80.154.240	5420	7673k	2756	216k	8176	7889k	18.438310	381.1625

In active measurements, the measurer actively generates traffic that may interfere with the network to some extent. This interference may affect the true reflection of network performance. In passive measurements, there may be bias in which flows are selected for observation and analysis. The sample traffic selected may not be representative of the entire network traffic

Comparing iperf results from active and passive measurements, the red line indicates an active measurement, while the blue line indicates a passive measurement. Active measurements introduce a lot of flow, but the duration is short, usually only a few seconds. Passive measurements introduces very little flow, but lasts a long time, almost the entire measurement.

Compare ping results from active and passive measurements, the red line indicates an active measurement, while the blue line indicates a passive measurement. Similarly, passive measurements introduce very little flow and last for a long time, almost the entire measurement. However active measurements also introduce a lot of traffic, but last longer, unlike iperf results.

4. Comparisons of active and passive measurements

	Active measurements	Passive measurements
Measurement Tools	Ping, Iperf, Traceroute	CoralReef, Wireshark, tcpdump
Measured Features	Latency (RTT), Packet Loss, Bandwidth Utilization, Real-time Performance	Availability, Bandwidth Utilization, Errors, Packet Drops, Traffic Analysis
Use of Tools	Ping is used to measure host reachability and latency	CoralReef is used for traffic analysis and extraction of traffic features
	Iperf is used for measuring bandwidth, throughput, latency and network performance testing	Wireshark is used for packet analysis, troubleshooting, and traffic analysis
	measuring bandwidth, throughput, and latency	tcpdump is used for packet capture and analysis
Issues and Limitations	Active measurement can introduce network interference and affect real performance	Passive measurement requires support from network devices and may not be applicable to all networks
	Active measurement may not accurately simulate complex real-world traffic	The accuracy of passive measurement is influenced by sample selection and sampling rates
	Active measurement is affected by network conditions and paths	Passive measurement may not capture encrypted and privacy-protected traffic