creative commons
C O M M O N S D E E D

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

Master's Thesis

An In-depth Latency Measurement Tool for Large-scale System

Bongwon Lee

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

2023

An In-depth Latency Measurement Tool for Large-scale System

Bongwon Lee

Department of Computer Science and Engineering

Ulsan National Institute of Science and Technology

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

An In-depth Latency Measurement Tool for Large-scale System

A thesis/dissertation submitted to
Ulsan National Institute of Science and Technology
in partial fulfillment of the
requirements for the degree of
Master of Science

Bongwon Lee

06.30.2023 of submission

Approved by

[illegible]

Advisor

Youngbin Im

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

An In-depth Latency Measurement Tool for Large-scale System

Bongwon Lee

This certifies that the thesis/dissertation of Bongwon Lee is approved.

06.30.2023 of submission

Signature

[illegible]

Advisor: Youngbin Im

Signature

[illegible]

Myeongjae Jeon

Signature

[illegible]

Yuseok Jeon

Signature

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

Abstract

The existing network performance measurement tools are limited to measuring end-host latency within the scope of the TCP layer or network latency. This limitation hinders the detailed analysis of latency occurring at various points within each component of the network protocol stack. Therefore, this thesis presents eBPF-ELEMENT as a solution to address the aforementioned issues. eBPF-ELEMENT utilizes eBPF (extended Berkeley Packet Filter) and XDP (Express Data Path) to overcome the limitations and challenges associated with implementing a real-time fine-grained latency measurement tool for large-scale systems. eBPF-ELEMENT provides a versatile framework that enables detailed measurement of network performance within servers and across server boundaries and offers network performance metrics, including per-layer latency, packet loss rate, throughput, and system performance metrics, including CPU and memory utilization. Thereby, eBPF-ELEMENT enables holistic system performance measurement and analysis and provides abundant information for troubleshooting different network issues in large-scale systems.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Contents

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

List of Figures

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

I Introduction

The emergence of applications such as live video streaming, teleconferencing, telemedicine, 3D virtual reality (VR), augmented reality (AR), and others has highlighted the growing importance of network performance for utilizing these applications. In response to these developments, extensive research is being conducted on high-speed networks to support these applications. High-speed networks are characterized by low latency, high bandwidth, high reliability, and high availability. They are designed to enable efficient communication between different network edges within a unit of time. However, despite the use of high-speed networks in locations such as data centers where a large amount of data needs to be transferred between servers, the full potential of high-speed networks is not fully utilized. This limitation arises due to the overhead associated with the traditional network stack used in the existing kernels. For example, performance degradation occurs within the network environment due to operations such as data movement between socket buffers and kernel buffers, TCP packet segmentation, checksum calculations, as well as phenomena like the bufferbloat. These factors contribute to the suboptimal performance of the network.

Many tools have been developed for measuring network performance (such as Ping and Traceroute). However, these tools primarily focus on measuring the round trip time or network delay. They ignored the end-host delay that occurs within the host. In order to identify and address the latency caused within the end host, ELEMENT was introduced in [1]. The end-host delay represents the delay between the TCP layer and its upper layer, as depicted in Figure 1. ELEMENT also provided a method to minimize the end-host delay without root privilege. However, all of these tools could not provide in-depth delay information that occurs in different layers of the network stack, between virtual interfaces, over the hypervisors, etc.

To address the aforementioned limitations, this thesis proposes eBPF-ELEMENT as a solution. eBPF-ELEMENT enables the measurement of network performance by tracing all paths of packet movement within the network. This allows for fine-grained analysis of the routes taken by packets, leading to more detailed measurements of network performance. Moreover, eBPF-ELEMENT utilizes eBPF, a powerful tool provided by Linux, to measure network performance. By leveraging the capabilities of eBPF, eBPF-ELEMENT can efficiently capture and analyze network data, enabling comprehensive measurements of network performance. By utilizing eBPF, eBPF-ELEMENT facilitates network performance measurement without the need to modify user-space or kernel-space code. It allows measuring network performance simply by running a single program, enabling its universal usage across different servers. This approach eliminates the need for extensive modifications to the existing infrastructure, making it more convenient and accessible for widespread deployment. The proposed framework provides a variety of insights by analyzing in detail the network performance that existing tools do not measure. In summary, this thesis makes the following contributions.

Per-layer delay information through highly accurate packet matching. It proposed eBPF-ELEMENT, which is a tool that calculates the delay between each layer through accurate packet matching between each layer based on cumulative bytes using sequence numbers. Byte calculation based on sequence

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

numbers accurately matches packets even in various network situations, such as packet retransmissions and out-of-order packet deliveries. It also provides various types of information beyond network delay, including the throughput, packet loss rate, and other system-related metrics.

Accurate delay information through time synchronization across servers. It provides a method of synchronizing each server's time to the central server's time to calculate the precise delays between servers. It especially uses a method of synchronizing the time of each server based on XDP, which minimizes the error of synchronization due to the variance of transmission times of the control packets for synchronization.

Low overhead via sampling of measured data. It proposes a method of adjusting the interval for extracting the data to be used for the calculation of delays. By adjusting the sampling interval, the server's resource overhead for measuring the delay is minimized.

Extensive evaluation in different environments. This thesis evaluates eBPF-ELEMENT in different virtualization environments, network types, and applications and also in realistic environments where multiple clients co-exist.

Insights for controlling the latency. This thesis provides experiment results of the latency variations by adjusting the TCP buffer size and flow priorities and provides insights for controlling the latency of latency-sensitive applications.

In Section II, we provide a background to understand the proposed framework, including ELEMENT, eBPF/XDP. Section III describes research works that utilize eBPF/XDP for various purposes. After that, Section IV explains the structure, operation, and algorithms of eBPF-ELEMENT. In Section V, we provide the results of experiments conducted using eBPF-ELEMENT and an analysis of the obtained results. Section VI provides examples of how to take advantage of the results obtained using eBPF-ELEMENT. After discussing the limitations of the proposed framework and how to overcome in Section VII, we present the functions that we are developing or will develop in the future for improving eBPF-ELEMENT in Section VIII. Then, we conclude the thesis in Section IX.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

II Background

We provide the background needed for understanding eBPF-ELEMENT in this section. We explain ELEMENT first since we extend the functionality of ELEMENT into different network layers in eBPF-ELEMENT, and then eBPF and XDP because they are utilized for the implementation of eBPF-ELEMENT for low-overhead kernel function access and packet modification.

2.1 ELEMENT

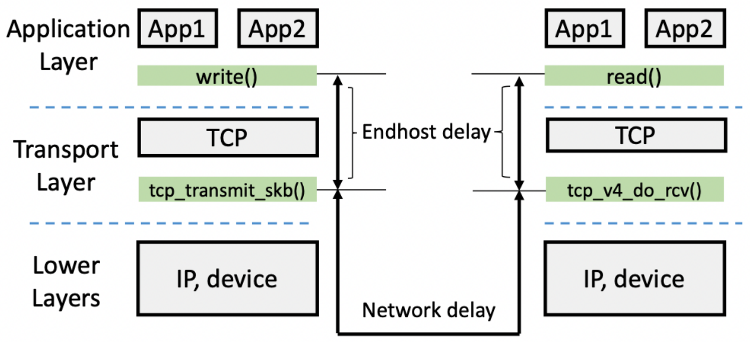


Figure 1: EndHost Delay, End-to-End Delay

ELEMENT [1] is a tool developed for measuring the end host delay. Existing network performance measurement tools have focused on measuring only the round trip time delay or network delay. ELEMENT aims to measure the latency between the TCP layer and the layer above it, i.e., the socket layer, which was previously overlooked but is crucial for network performance. By capturing this latency, ELEMENT enables finding the main delay component within the end-to-end network path. ELEMENT also provides an algorithm for optimizing the end-to-end network delay by minimizing the end-host delay.

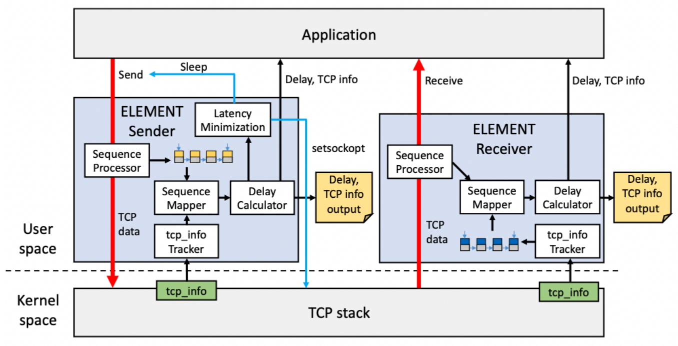


Figure 2: The architecture of ELEMENT

Figure 2 represents the architecture of ELEMENT [1]. ELEMENT utilizes the TCP_INFO option with *getsockopt* to obtain TCP statistical information. The TCP information obtained using *getsockopt*

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

includes the amount of data, acknowledged bytes, congestion window size, RTT (Round Trip Time), and so on. The tcp_info Tracker records TCP statistical information collected through *getsockopt* function. It simultaneously logs the time of data transmission, the amount of data sent, the time of data reception, and the amount of data received by the Sequence Processor. Afterward, the Sequence Mapper attempts to perform sequence number matching between the socket and TCP layers using the sequence numbers recorded by the Sequence Processor. The Delay Calculator utilizes the matched records between the two layers by the Sequence Mapper to calculate the delay. Afterward, the calculated results, along with other TCP information, are periodically saved to a result file. In addition to storing the calculated results, ELEMENT also passes these results to the Latency Minimization component. The Latency Minimization component uses the results to regulate the data transmission speed, aiming to minimize end host delay.

Algorithm 1: Algorithm for calculation the delay at the TCP sender using the TCP-INFO socket option

```
$T_{start} \leftarrow$ // TCP connection start time
$T_{cur} \leftarrow$ // current time
$P \leftarrow 10\, msec$ // sleep period
$T \leftarrow T_{cur} - T_{start}$ // elapsed time
$seq \leftarrow 0$ // total sent bytes
$sendInfo \leftarrow$ // a struct that consists of $bytes, sendTime, next$
tcp_info tracking thread: // Obtain, save the TCP information in $ti$
while $true$ do
    $B_{est} \leftarrow ti.tcpi\_bytes\_acked + ti.tcpi\_unacked * ti.tcpi\_snd\_mss$
    // estimated sent bytes at TCP layer
    while $true$ do
        if $back \neq NULL$ and $back.bytes \leq B_{est}$ then
            $D \leftarrow T - back.sendTime$ // buffer delay
            Print $T, D, ti.tcpi\_snd\_cwnd,$
              $ti.tcpi\_snd\_ssthresh, ti.tcpi\_rtt$
            $old\_back \leftarrow back$
            $back \leftarrow back.next$
            free $old\_back$
            if $back = NULL$ then
                $front \leftarrow NULL$
        else
            break
    Sleep for $P$
data sending thread:
while $true$ do
    if $front = NULL$ then
        Allocate $front$ with $sendInfo$ struct
    else
        $old\_front \leftarrow front$
        Allocate $front$ with $sendInfo$ struct
        $old\_front.next \leftarrow front$
    $front.bytes \leftarrow seq$
    $front.sendTime \leftarrow T$
    $front.next \leftarrow NULL$
    if $back = NULL$ then
        $back \leftarrow front$
    Send a packet and add the packet size to $seq$
```

In order to measure the latency on the sender side, ELEMENT utilizes an algorithm in Algorithm 1. One notable consideration in ELEMENT is that the size of the data unit sent at a given moment may differ between the socket and TCP layers. The factors that affect the variation in data unit are such as the application buffer amount, socket buffer amount, TCP state, etc. This makes the delay calculation hard since the data transmission at different layers is not one-to-one relation. To address this issue, ELEMENT employs a method where it infers the moment of packet transmission at each layer. This approach aims to resolve the disparity in data units by calculating the accumulated bytes at different layers. By recording the accumulated bytes and the corresponding timestamps, ELEMENT tracks the amount of data sent by the application layer and TCP layer over time. In TCP, the cumulative byte count is calculated by adding the acknowledged bytes (tcpi_bytes_acked) value and the number of unacknowledged

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

segments (tcpi_unacked), multiplied by the sender's maximum segment size (tcpi_snd_mss). By utilizing the cumulative bytes from both layers, ELEMENT approximates packet matching and measures the latency between the two layers.

On the receiver side, a similar algorithm to Algorithm 1 is employed in ELEMENT to measure latency. In contrast to the sender-side, on the receiver-side, the TCP layer receives the data first. Accordingly, the *tcp_info* Tracker records the timestamp and sequence number (cumulative received bytes) in the TCP layer. The received cumulative bytes in ELEMENT are obtained by multiplying the total number of segments received (tcpi_segs_in) inside *tcp_info* with the receiver's maximum segment size (tcpi_rcv_mss). The process of calculating the received cumulative byte count at the application layer is similar to the sender side. ELEMENT utilizes the cumulative data quantities on the two layers to perform packet matching, which enables the calculation of latency.

ELEMENT leverages the measurements obtained through the aforementioned method to develop a latency control algorithm. ELEMENT utilizes two approaches to minimize the delay. The first approach is to adjust the rate at which the application sends data based on the measured delay, thereby reducing buffer delay and optimizing the end-to-end delay. The second approach is to enable minimizing latency in TCP connections without requiring modifications to the application's code by dynamic preloading of UNIX to interposition the latency control algorithm between the application and the BSD socket library.

2.2 eBPF, XDP

eBPF-ELEMENT uses eBPF and XDP [2] provided by Linux to measure network delays. eBPF (extended Berkeley Packet Filter) is an extended technology of BPF (Berkeley Packet Filter), which allows users to safely execute user-defined code within specific parts of the kernel without the need to modify the existing kernel source code. eBPF provides security and convenience when executing user-defined code, allowing for the exchange of data between user space and kernel space. This capability enables dynamic system observability, empowering users to effectively monitor and analyze the system's behavior. eBPF's unique characteristics make it applicable for various purposes, such as measuring kernel function delays and analyzing the performance of applications.

XDP, on the other hand, is an eBPF-based high-performance data path used to process network-related processing at high speeds by bypassing most of the operating system's networking stack. XDP is a type of eBPF hook and operates as an instruction set within network drivers. Due to its characteristics, XDP is utilized in scenarios where there is a high volume of traffic, such as packet filtering, and firewall.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

III Related Works

3.1 Performance enhancement by using eBPF/XDP

The increasing amount of data that needs to be transmitted and received within networks has led to a growing demand for high-speed networking systems. To address this issue, active research has been conducted to apply the tools like eBPF, XDP to network systems in order to optimize and enhance their performance. eBPF [2] enables the safe execution of user-defined code within the kernel space without modifying the existing code. XDP (eXpress Data Path) is a high-performance data path used for processing network-related tasks quickly and efficiently.

[3, 4, 5, 6] are research studies that utilize eBPF/XDP for monitoring network and system performance. [3] uses eBPF and XDP to compute network metrics efficiently. By utilizing eBPF, XDP, and devices with computing cores like SmartNICs, the logic for computing metrics is offloaded. [4] investigates the feasibility of using eBPF in high-speed network environments for continuous latency monitoring and demonstrates that eBPF is capable of handling network speeds of up to 10Gbps per core. [5] enables the measurement of performance for isolated containers by utilizing eBPF. It especially performs the profiling and tracing of Interledger connectors. [6] builds *vNetTracer*, a packet profiler for virtualized networks, which relies on the eBPF for inserting user-defined trace programs dynamically into a virtualized network.

[7, 8] explore the use of eBPF/XDP not only for collecting network metrics but also for inspecting the payload of packets for various purposes. [7] introduces the BPFast tool, which utilizes eBPF/XDP in a cloud environment to parse the payloads of packets, inspect them, and protect containers from network attacks. [8] focuses on the use of eBPF/XDP for packet filtering and leveraging them to enhance security measures.

[9] and [10] investigate the use of eBPF/XDP for offloading network functions and creating faster network environments by quickly adapting to network conditions. By leveraging eBPF/XDP, these papers explore how network functionality can be offloaded or accelerated, enabling the creation of more efficient network environments that can dynamically adapt to changing network conditions and requirements. [9] suggests a novel networking platform for next-generation networks (5G and beyond) to achieve advanced data plane control and hardware acceleration. The proposed programmable data plane combines eBPF and XDP, and a new eBPF-XDP packet processing algorithm is proposed. [10] discusses the limitations of the upf-bpf, an approach for 5G User Plane Functions for multi-access edge computing environments using BPF/XDP, and proposes a new design for enhancing flexibility and reducing the run-time adaptation time.

3.2 Traditional performance monitoring approaches

[11, 12] are traditional approaches for efficient performance monitoring without using eBPF/XDP. [11] proposes Pingmesh, a tool for measuring network latency performance in environments with large-scale systems, such as data centers. It helps answer the questions like whether an application's perceived

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

latency issue is due to the network or not, define and track network service level agreement (SLA), and troubleshoot the network automatically. [12] is a high-resolution server performance measurement tool provided by Netflix. By installing an agent on each server, it reads the metrics of each server at regular intervals and provides convenience for users to investigate the correlation between collected metrics by using a dashboard for visualization.

This thesis is differentiated from the research works mentioned above in that it utilizes eBPF/XDP to analyze network delays with high granularity and accuracy, both within the server and among the servers, enabling finding the different latency issues in different layers.

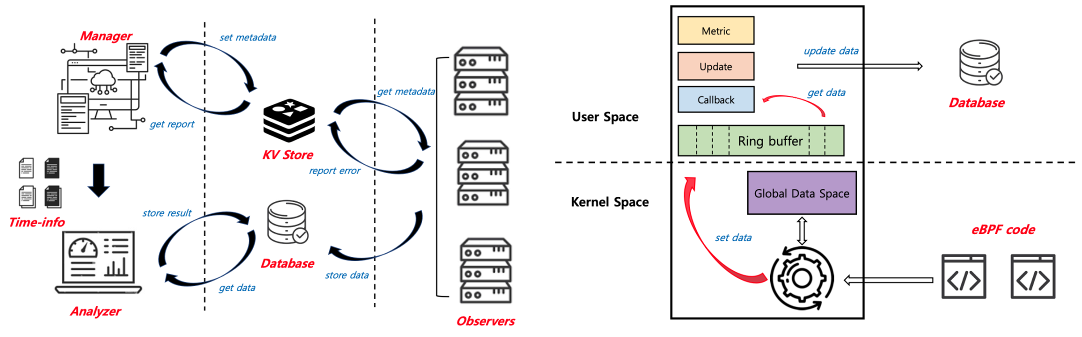
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

IV eBPF-ELEMENT

This section explains the architecture, operation, and algorithms of eBPF-ELEMENT. eBPF-ELEMENT leverages eBPF for the measurement of delays between network segments without the need to modify the existing code of applications, hypervisors, and the kernel. Simply running the eBPF-ELEMENT program on the management server enables the real-time measurement of delays over multiple servers. This makes eBPF-ELEMENT easily applicable for performance monitoring on large-scale systems such as data centers and mobile network cores. It can also provide high accuracy with time synchronization based on XDP and low overhead via sampling methodology.

In Section 4.1, the architecture of eBPF-ELEMENT is provided, and Section 4.2 describes a set of functions within the kernel that eBPF-ELEMENT hooks into. Section 4.3 provides the details of the algorithms that eBPF-ELEMENT utilizes for packet matching, time synchronization, and sampling. Examples that show how eBPF-ELEMENT works in real network environment are provided in Section 4.4.

4.1 eBPF-ELEMENT Architecture



(a) Overall architecture. **(b)** Architecture of Observer.

Figure 3: Architecture of eBPF-ELEMENT.

Figure 3 depicts the overall architecture of eBPF-ELEMENT and the architecture of the Observer, which operates within each server. eBPF-ELEMENT consists of five components: Manager, Observer, Analyzer, Key/Value Store, and Relational Database. In the below, each component is described in detail.

eBPF-ELEMENT Manager

The manager is a component responsible for managing the whole eBPF-ELEMENT program. When the eBPF-ELEMENT program is initiated, the Manager component reads the configuration information in the form of a YAML file. The configuration information includes various details such as the information regarding each server where the eBPF-ELEMENT will be installed, connection details for Key/Value Store and Database, and other relevant information. Based on this information, the Manager installs the eBPF-ELEMENT code on each server and stores the program metadata in the Key/Value Store

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

for executing it. Afterward, the Manager sends the eBPF-ELEMENT program start command and the assigned *key* to each server, which is a unique identifier for each server.

eBPF-ELEMENT Observer

The Observer is a process installed on each server that performs probing and collects the necessary data for network performance analysis. The Observer is installed on each server by the Manager when the program starts. It receives the metadata for configuring itself through the Key-Value Store and runs three processes simultaneously. The first is the Callback process, which is responsible for event handling. The Callback process collects data, which is then updated in the database by the Update process. Additionally, there is the Metric process, which periodically observes CPU and memory usage.

When the Observer starts, it hooks probing functions into the kernel functions it needs to probe. Afterward, if certain conditions are met within those probed kernel functions, the events are triggered. When an event occurs, the Callback process is executed to handle it. During the event processing, the Callback process retrieves event-related data through shared memory (ring buffer). Furthermore, the Callback process transfers this data to the Update process using a shared queue. When the Update process detects the presence of data in the queue it is responsible for, it pops the data from the queue, performs preprocessing if necessary, and then updates the relevant data in the Database. The Metric process operates independently from other two processes. It periodically collects data about current CPU and memory usage through the *proc* file system.

eBPF-ELEMENT Analyzer

The Analyzer analyzes the network delay within each server and between servers based on the information extracted by the Observer running on each server. Analyzer retrieves the raw network data for each server from Database. It distinguished flows using *src_addr, src_port, dst_addr, dst_port*. It then applies the sequence number matching algorithm, which matches packet data between each network layer. After the calculations of delay between layers based on matched data, the results are stored in Database using the flow identifier.

Key/Value Store

The Manager and each server utilize the Key/Value Store for information exchange between them (we use Redis). Each server continuously observes relevant data in the Key/Value Store with the metadata *key* assigned to it at regular intervals. By doing so, it continuously receives the necessary information during run-time from the Manager. In addition, each server reports its error situations to the Manager via the Key/Value Store. This information can be utilized in the future to develop functionality that takes appropriate measures in response to error situations.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Relational Database

table_name	type	field_name
flow_id	bigint	id
	varchar(32)	src_addr
	varchar(32)	dst_addr
	int	src_port
	int	dst_port

(a) *flow_id* table

table_name	type	field_name
log	bigint	flow_id
	int	node_id
	bigint	data_len
	bigint	ts
	int	evt_type
	bigint	tid
	bigint	start_seq
	bigint	cur_seq
	int	cpuid
	int	is_retrans

(b) *log* table

table_name	type	field_name
result	bigint	id
	bigint	flow_id
	int	node_id1
	int	node_id2
	bigint	data_len1
	bigint	data_len2
	bigint	diff_ts
	bigint	manage_ts1
	bigint	manage_ts2
	int	evt_type1
	int	evt_type2

(c) *result* table

table_name	type	field_name
metric	bigint	flow_id
	int	node_id
	bigint	ts
	int	pid
	varchar(32)	name
	int	port
	double	cpu_usage
	double	mem_usage

(d) *metric* table

Figure 4: Tables for the Relational Database.

The Observer is installed on multiple servers to extract network information from each server. eBPF-ELEMENT utilizes the Relational Database (we use MySQL) as the storage for the collected network information by Observers. When analyzing network delay performance, utilizing indexes within the Relational Database can significantly enhance query processing speed, allowing for faster analysis. Indexes optimize the retrieval of data based on specific query criteria, resulting in improved performance when querying network-related data. Figure 4 displays the tables used in Relational Database.

4.2 Probing Points

Figure 5 illustrates the typical functions called within each network layer of the Linux kernel during network packet transmission and reception. The packet transmission process starts with *sock_sendmsg()* and goes through various functions until the packet is sent to the hardware through *dev_hard_xmit()*. On the other hand, during packet reception, the process begins with *netif_receive_skb()* and goes through various functions until the data is moved to the application buffer via *sock_recvmsg()*. While eBPF is a powerful tool, it does have certain limitations in terms of its usage, such as available stack size and probing points. Therefore, we need to carefully choose the probing points in the network stack to collect the appropriate network data efficiently.

Packet transmission process

When a user program sends data to another server, it initiates the data transmission by invoking functions such as *write()*, *send()*, etc. In Linux, devices, sockets, and various other resources are treated uniformly as files. Hence, when invoking functions such as *read()*, *send()*, *sendto()*, the file descriptor passed as an argument is used to differentiate the specific file being referenced. If the argument refers to a socket, the kernel invokes the corresponding function associated with it through the *sock_operation_ops*. Generally, the *send()* function associated with a socket-type file descriptor is implemented as *sock_sendmsg()*. This function is responsible for sending data through the socket. *sock_sendmsg()* moves the data from the application to the socket buffer.

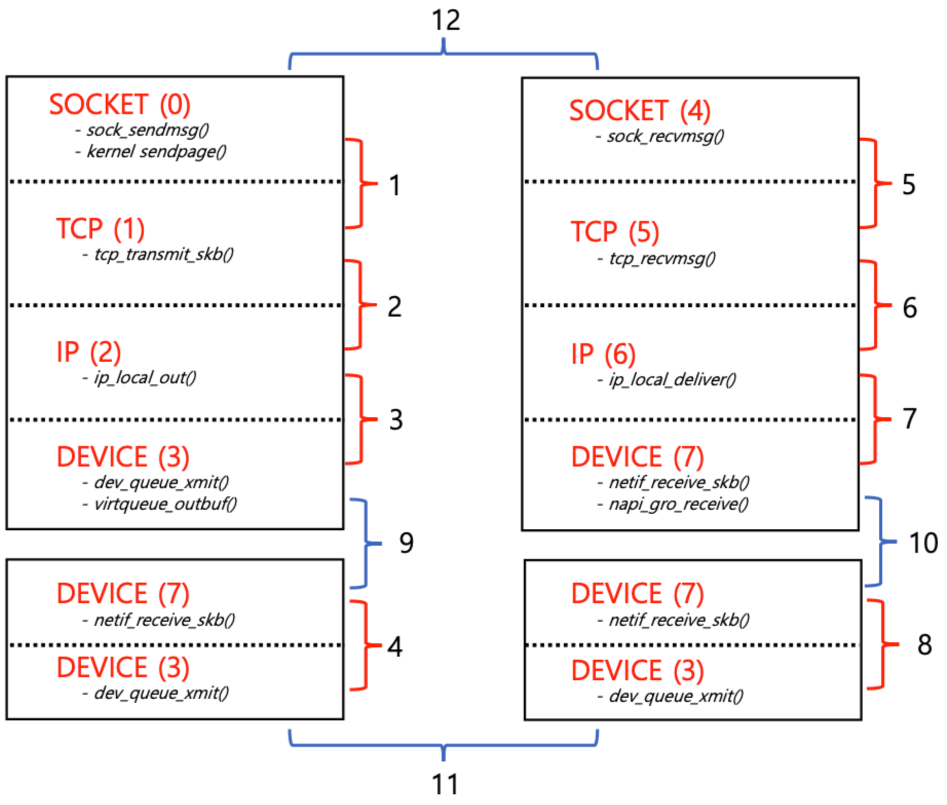


Figure 5: Linux network protocol stack and hooking points of eBPF-ELEMENT.

Afterwards, *sock_sendmsg()* calls either *tcp_sendmsg()* or *udp_sendmsg()* based on the protocol type being used. Indeed, eBPF-ELEMENT focuses on measuring network delays for TCP, so it utilizes *tcp_sendmsg()* for this purpose. *tcp_sendmsg()* moves the data from the socket buffer to the TCP buffer. It then utilizes TCP's flow and congestion control mechanisms to control the transmission speed of the data. Afterward, *tcp_sendmsg()* calls the *tcp_transmit_skb()* function to compute the metadata information required for using the TCP protocol. *tcp_transmit_skb()* receives the *skb* structure as an argument, and sets the TCP header information, configures options such as TCP selective acknowledgment and window scaling if necessary, and generates the TCP packet.

Once a TCP packet is created, it is then passed to the IP layer by invoking *ip_local_out()*. This function is responsible for forwarding the packet to the appropriate IP layer for further processing and eventual transmission over the network. *ip_local_out()* processes the packet and handles the IP header. It verifies the destination IP address and selects the appropriate interface to transmit the packet over the network. Afterward, *dev_queue_xmit()* is called to deliver the packet to the device layer. This function enqueues the packet for transmission and hands it over to the network device for actual physical transmission. *dev_queue_xmit()* performs tasks such as checking the ARP cache to find the destination MAC address. It prepares the packet for transmission by placing it in the interface queue, ensuring that the packet is ready for delivery to the appropriate destination. Once the transmission of the queued packet is completed, a callback function is invoked to notify the completion of the packet transmission process. This callback function finalizes the packet transmission handling.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

eBPF-ELEMENT applies hooks to representative functions of each layer within the network protocol stack to extract information from each layer for in-depth delay analysis. We try to strike a balance between granularity and overhead. Hooking eBPF into multiple functions within each layer allows for extracting various information, leading to a higher granularity of performance metrics. However, this can potentially impact the performance of each server. To mitigate this impact, eBPF is hooked into a single representative function for each layer, reducing the overall overhead. When selecting representative functions for each network layer, we consider whether all the information we need in each layer is accessible in the context of the function. We check what value the function returns (to find the size of actually transmitted bytes in the socket layer) and whether packet headers are completely formed (to get the information we need from the headers).

eBPF-ELEMENT distinguishes the network stack into socket, tcp, ip, and device layers. Figure 5 provides information on which function within each layer has been hooked during packet transmission and reception. During packet transmission, packet information are extracted from the entry points of functions for each layer. The extracted data is then passed to the user space for further analysis or processing. After receiving the data, the user space component stores the data in a queue. Once the number of accumulated data in the queue exceeds a specific threshold, multiple data entries are batched and stored in Database.

Packet reception process

When receiving packets, the kernel initiates the packet reception process by invoking the *netif_rx_action()* function. This function serves as the entry point for packet reception in the network stack. Unlike packet transmission, the *netif_rx_action()* function handles the reception of packets asynchronously with the socket layer, allowing for concurrent processing of incoming packets from multiple sources. It schedules *softirq* handler. *softirq* handler retrieves the driver that requested NAPI poll from the *poll_list* and invokes the driver's *poll()* function. This allows the driver to process incoming packets and perform necessary operations on the received data. The driver encapsulates the received packet into a *sk_buff* structure and then calls *netif_receive_skb()* function. This function is responsible for various operations, such as protocol handling, checksum verification, and delivering the packet to the appropriate protocol layer for further processing.

After receiving the packet, *netif_receive_skb()* passes the packet to the modules registered in the *ptype_all* list for further processing. It reads the header values of the packet and forwards it to the upper layer based on the header information. Subsequently, *ip_rcv()* function is invoked to handle the received packet at the IP layer. *ip_rcv()* function performs various tasks required at the IP layer, such as packet length validation, header checksum verification, and other necessary operations specific to the IP layer. It ensures the integrity and validity of the received IP packet.

Afterwards, the packet is processed by the *netfilter* code to determine whether the packet needs to be routed to another server or if it should be received by the current server. If it is determined that the packet should be received by the current server, the *ip_local_deliver()* function is called. This function

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

is responsible for the local delivery of the IP packet. Depending on the situation, *ip_local_deliver()* may need to reassemble IP fragments before calling the *ip_local_deliver_finish()* function. This function is responsible for the final processing and delivery of the IP packet to the appropriate destination within the server. It extracts information about which protocol the packet is using and delivers the packet to the appropriate upper layer for further processing. In the context of eBPF-ELEMENT analyzing TCP-based packets, the *tcp_v4_rcv()* function is called. This function is responsible for processing and handling incoming TCP packets at the TCP layer. It performs various tasks specific to TCP, such as sequence number verification, checksum validation, congestion control, and handling various TCP flags and options.

Afterwards, the data contained in the packet is moved to the socket buffer. This involves copying the payload of the packet to the socket buffer associated with the receiving socket. To receive the data, the user program can make use of functions such as *read()* or *recv()*. These functions determine whether the file descriptor corresponds to a socket by examining its information. If the file descriptor represents a socket, the appropriate function corresponding to the socket type will be called. In this case, the function *sock_recvmsg()* is called. These functions are typically used to retrieve data from a socket buffer associated with a specific socket. If the data that the user program needs to receive is available in the socket buffer, it will be read. Otherwise, depending on the socket's flags, the process may block until the data arrives in the socket buffer or return immediately.

For similar reasons as the packet transmission process, when receiving data, eBPF-ELEMENT also hooks a representative function for each layer. Figure 5 shows the functions that eBPF-ELEMENT hooks during the reception process, and the selection of these functions is based on whether we can obtain the information we need in the context of the function.

4.3 Algorithm

To measure the network performance within and between servers without modifying the code, eBPF-ELEMENT utilizes techniques for packet matching, time synchronization among servers, optimizing the trade-off between accuracy and server overhead. Here, we describe the techniques employed by eBPF-ELEMENT.

Packet Matching

eBPF-ELEMENT saves the initial sequence number of TCP for each flow and utilizes the difference between subsequent sequence numbers and initial sequence number to calculate the total culmulative bytes sent or received by each layer. In the TCP and the lower layers, there can be issues such as packet loss or out-of-order delivery, which may result in the retransmission of packets with previous sequence numbers. To prevent duplicate calculations for retransmitted packets, eBPF-ELEMENT ignores the packet with smaller sequence number than the largest sequence number discovered up to the current time for a flow. In the case of socket layers, there is no sequence number information. However, since the data is delivered to the socket layer without loss or out-of-order delivery, we can find the exact

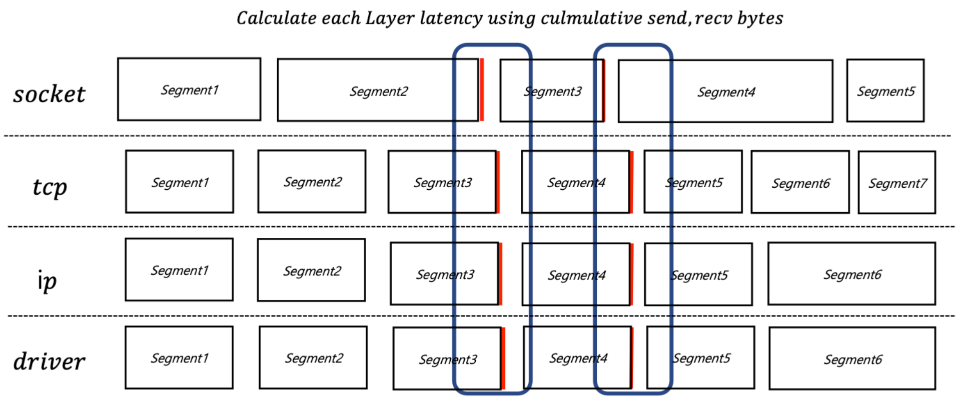


Figure 6: Packet Matching using culmulative send/recv bytes

cumulative bytes for the socket layer.

Afterwards, collected information is passed to the user program, which then stores it in Database along with the corresponding timestamp. Analyzer utilizes this information to first differentiate each flow, and uses the each transmission and reception at the socket layers as the matching points with other layers. By matching with transmission and reception in other layers, we can calcualted the delay between socket and other layers by finding the difference between timestamps in different layers. This approach is also applied to calculate the delay between servers. Figure 6 illustrates the method used in packet matching, as described above.

Time synchronization

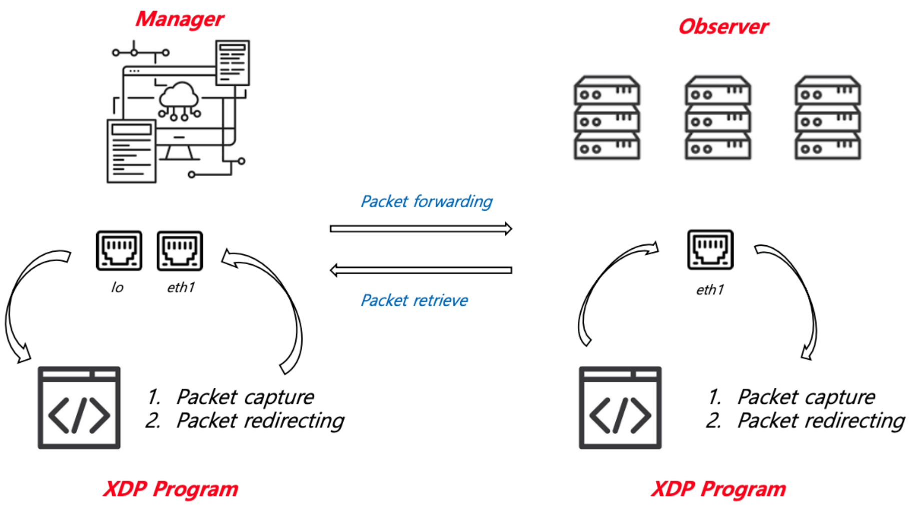


Figure 7: Time synchornization architecture.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Algorithm 2: Algorithm for calibrating the measured data time to the management server time.

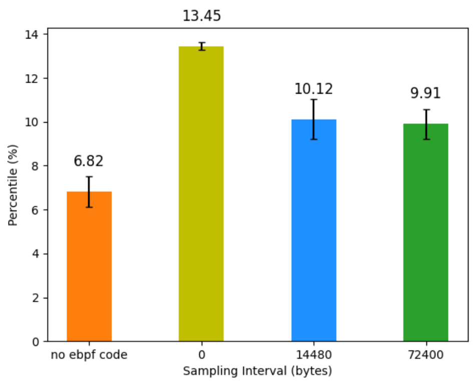
```
$MST \leftarrow Management\ server\ send\ time\ of\ synchronization\ packet$
$MRT \leftarrow Management\ server\ receive\ time\ of\ synchronization\ packet$
$ST \leftarrow Server\ time\ when\ receiving\ the\ synchroniztion\ packet$
$SMST \leftarrow Server\ time\ corresponding\ to\ the\ time\ when\ the\ management\ server\ sends\ the\ syncrhoniztion\ packet$
$TD \leftarrow Time\ difference\ between\ server\ and\ management\ server$
$ST_i \leftarrow Server\ time\ in\ each\ measured\ data$
$STOMT \leftarrow Each\ server's\ time\ converted\ to\ the\ time\ of\ the\ management\ server$

$One\_Way\_Delay \leftarrow \frac{(MRT-MST)}{2}$
$SMST \leftarrow ST\ -\ One\_Way\_Delay$
$TD \leftarrow MST\ -\ SMST$
$STOMT \leftarrow ST_i\ -\ TD$
```

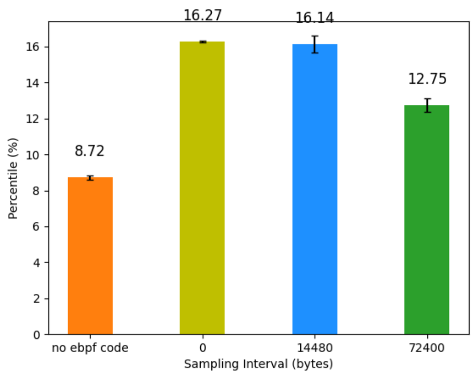
When measuring network latency performance between different servers or between the host and the virtual machines inside the host, it is necessary to synchronize the time because each server uses a different time. Synchronizing the time ensures consistency and accuracy of the measured network latency. The time synchronization is done by aligning the time of each server with the time of the server where the Manager is running. Figure 7 depicts how XDP is used to synchronize the time between servers. Before starting the performance measurement, Manager deploys XDP programs on the manager server and each server. The Manager on the management server creates synchronization packets for all probed servers and sends to its localhost interface. The XDP program intercepts the packets sent to the localhost and uses the metadata contained in those packets to create new packets for each server that needs to be probed. During this process, *dst_addr*, *dst_port*, and checksum are set in the newly created packets. Afterwards, the packets are forwarded to another physical interface for transmission. At this point, the Manager stores the timestamp of when it transmitted each packet. Each server receives the synchronization packet and stores its own timestamp of when it received the packet. After that, the servers exchange the *src_addr, src_port* and *dst_addr, dst_port* and send the packet to the Manager. When Manager server receives the packet, it saves the time at which it received the packet. Then it is passed on to the user program. The user program utilizes the equation shown in Algorithm 2 to synchronize the time of each server with the time of the Manager server. The reason for using XDP is to measure RTT (round-trip time) at the device layer in order to exclude the delay variation introduced by the kernel network stack.

Sampling

When a function hooked by the eBPF-ELEMENT is called, eBPF executes the code to extract raw data which becomes the basis for performance analysis. In high-bandwidth environments, invoking the raw data extraction code in high frequency can introduce high overhead to the server. Executing the data extraction code every time a packet is transmitted or received can increase the accuracy of packet matching during the packet matching process, but the server may experience high overhead. Sampling the execution of the code can help reduce the overhead on the server. By selectively executing the code on a subset of packets or at certain intervals, the server can mitigate the overall impact on performance.



(a) Sender's total cpu usage.



(b) Receiver's total cpu usage.

Figure 8: Total cpu usage when varying the sampling interval.

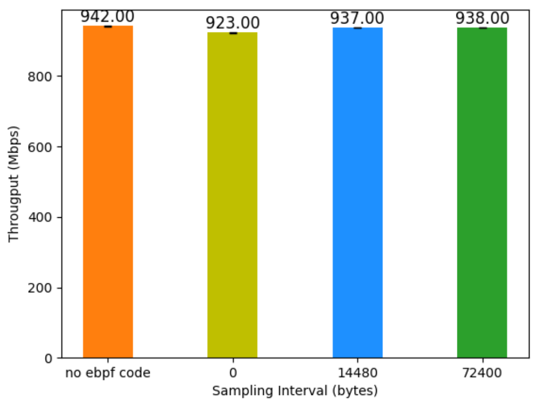
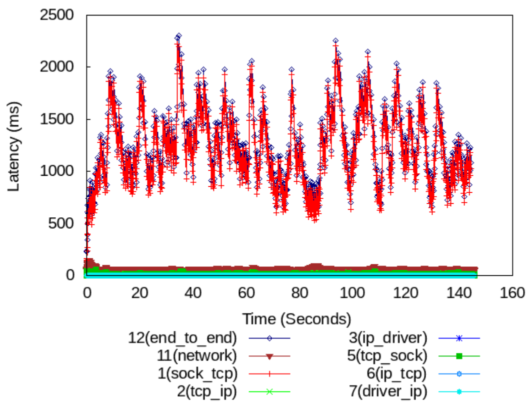
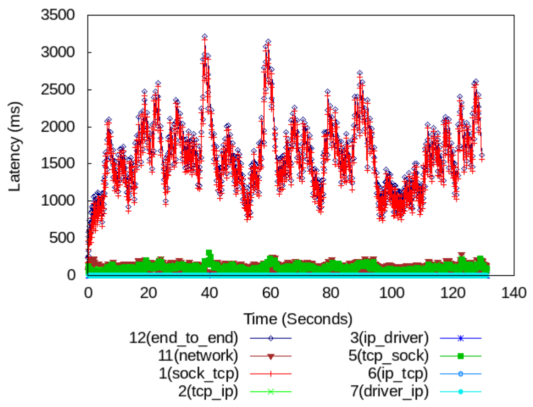


Figure 9: Throughput when varying the sampling interval.



(a) Sampling Interval 0



(b) Sampling Interval 72400

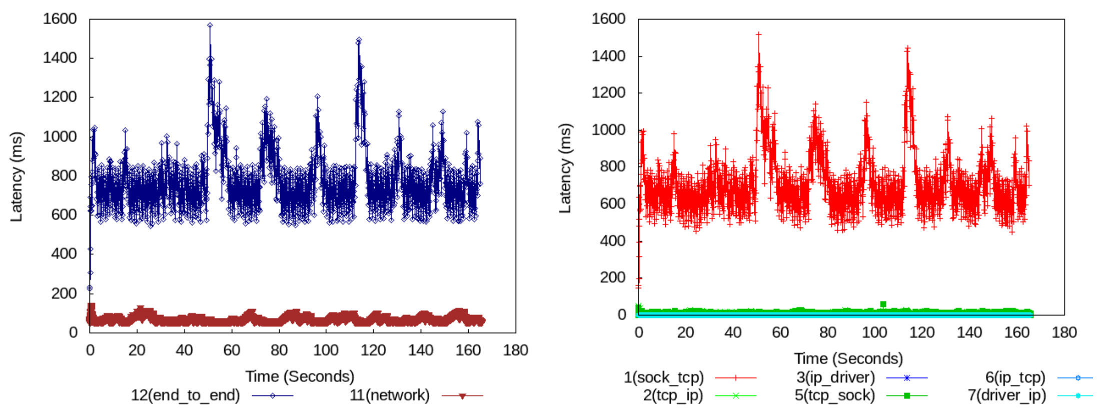
Figure 10: Changes in graph granularity with respect to sampling interval.

This approach balances the accuracy in packet matching with minimizing server overhead. The optimal trade-off will depend on the specific requirements and constraints of the network environment. Figure 8 shows that as the sampling interval increases, the CPU usage decreases. On the other hand, from Figure ,

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

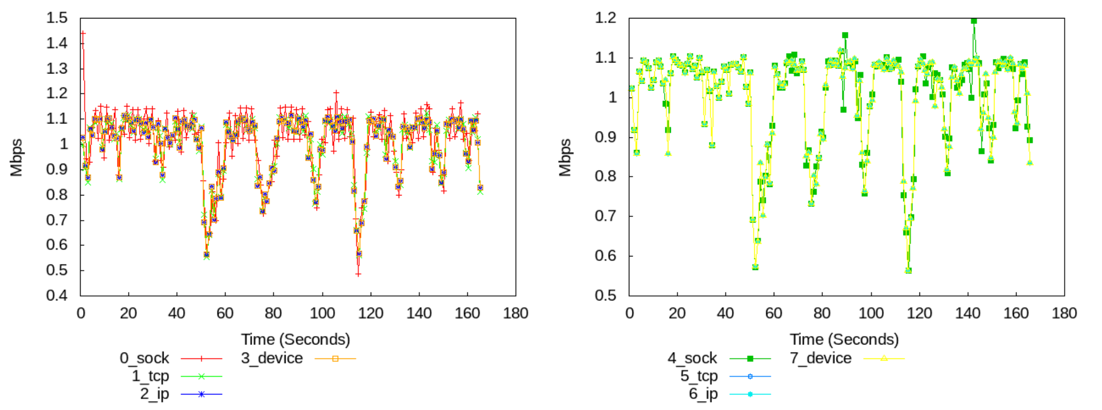
we can observe that a lower sampling interval provides more fine-grained analysis. However, even with the sampling interval of 72400 bytes, delay variations are shown clearly enough to detect the delay problems in the system. Also, we can observe that even without sampling the throughput performance is maintained from Figure . Therefore, system administrators utilizing eBPF-ELEMENT can adjust the sampling interval for each server depending on the system environments and requirements.

4.4 Examples



(a) Intra delay **(b)** Internal delay

Figure 11: Examples of network delay measurement.



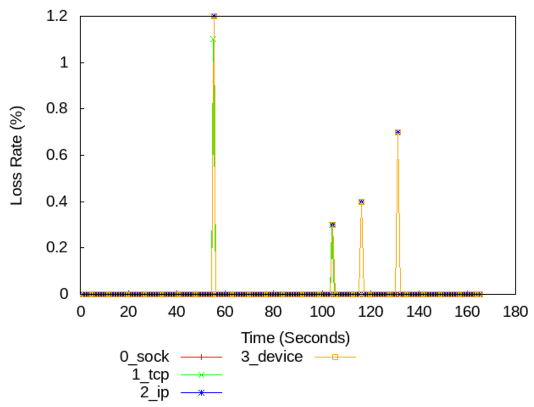
(a) Sender's Throughput **(b)** Receiver's Throughput

Figure 12: Examples of throughput measurement.

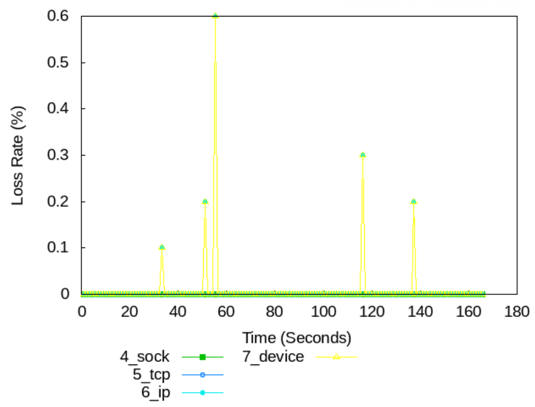
Here, we present the performance metrics measured using eBPF-ELEMENT. eBPF-ELEMENT measures not only the network delays between various network segments but also other performance metrics, including throughput, loss rate, CPU usage, and memory usage of each application.

Examples of eBPF-ELEMENT's measurement results are in Figures 11, 12, 13, 14. We can observe a correlation between packet loss rate, throughput, and network delay. A high packet loss rate leads to a decrease in throughput, and packet retransmission due to loss consumes additional network resources, reduces overall data transfer speed, and increases the delay. By utilizing various metrics such as delay,

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

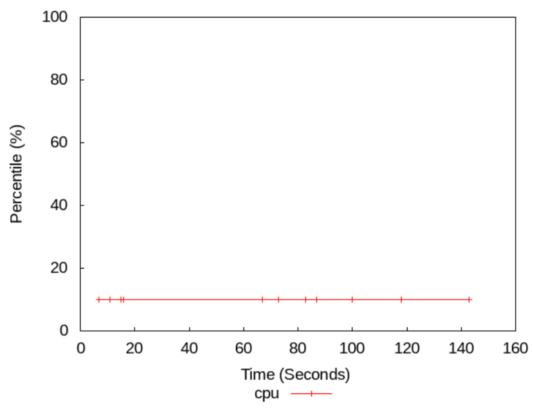


(a) Sender's Loss Rate

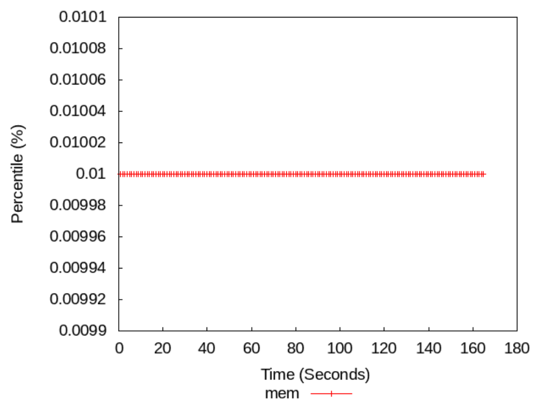


(b) Receiver's Loss Rate

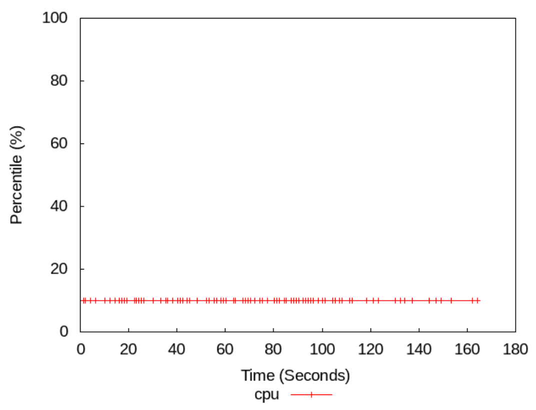
Figure 13: Examples of loss rate measurement.



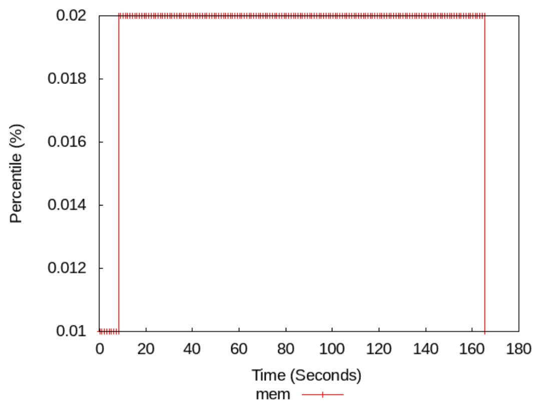
(a) Sender's CPU usage



(b) Sender's memory usage



(c) Receiver's CPU usage



(d) Receiver's memory usage

Figure 14: Examples of resource usage measurement.

throughput, loss rate, CPU usage, and memory usage, it is possible to analyze network performance comprehensively.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

V Evaluation

This thesis aims to validate the effectiveness of eBPF-ELEMENT by presenting network performance measurement results collected using eBPF-ELEMENT in various network environments. This thesis analyzes the collected data to gain insights into network performance and characteristics etc. By examining the network performance, this thesis provides a comprehensive understanding of the network performance under different scenarios. Video streaming services are among the popular applications widely used by many clients today. These services allow users to stream video content over the internet in real-time. Indeed, the field of video streaming is known for its sensitivity to network delay, as it directly affects the quality of the user's experience. Therefore, this thesis utilizes a DASH server, which is one of the video streaming applications, and measures network performance by varying the virtualization environment, network environment, different applications and complex scenarios. The collected data is then analyzed to gain insights into the performance of the network. In the virtualization environment, this thesis measures the network performance when running the program in different environments such as Non-Virtualized, Virtual Machine, and Container. In the network environment, it measures the changes observed when using different network environments such as LAN, WIFI, LTE, etc. In different applications scenarios, it conducts experiments using Iperf, DASH, FTP. In complex scenarios, it conducts experiments in various situations, such as varying the number of connections and introducing background traffic, to observe the impact on network performance.

5.1 Different Virtualization Environments

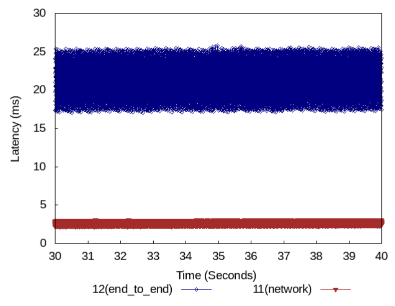
In the virtualization environment, commonly used configurations such as Non-Virtualized, Virtual Machine, and Container are employed. Before presenting the results, let's first provide an overview of each virtualization environment. The Non-Virtualized environment refers to a typical environment where the server operates independently without interference from other servers. Virtual Machine refers to an environment where hardware resources and operating systems are virtualized, allowing it to function as if it were an independent Non-Virtualized environment. To manage Virtual Machines, the host runs a program called a Hypervisor, which is responsible for allocating the physical resources such as cpu and memory to each Virtual Machine. Therefore, Virtual Machines operate as if they are independent server, but in reality, they share physical resources among multiple Virtual Machines. Containers, on the other hand, operate similarly to Virtual Machines as independent server like Non-Virtualized environments, but they do not virtualize hardware or the operating system. Containers utilize Linux namespaces to isolate the environment used by each process, creating an environment where each process behaves as an independent entity rather than as part of a single server. Compared to virtual machines, containers provide a relatively simpler way to configure servers.

Each method of creating environment differs in terms of their distinct characteristics, which in turn impact the utilization of physical resources such as cpu, memory, and NIC. This thesis specifically focuses on experimenting with the Network Stack of each environment used for packet transmission and reception. In the Non-Virtualized environment, network communication typically goes through a set of

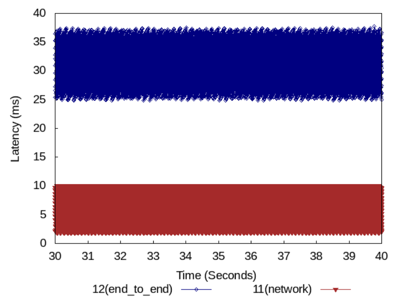
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

representative kernel functions that are called when packets are transmitted and received. Virtual machines virtualize hardware resources and operating systems, allowing them to be used independently. Due to this virtualization, the network path within a virtual machine exhibits similar characteristics to a Non-Virtualized environment. On the contrary, in a virtual machine environment, which is not an actual Non-Virtualized environment, the packets entering or exiting the virtual machine require assistance from the host's hypervisor to manage the process. Therefore, unlike in a typical Non-Virtualized environment, the hypervisor of the host managing the virtual machine needs to invoke additional functions to transmit and receive packets between the virtual machine and the host. These functions also facilitate the transfer of data between the virtual machine and the host's network interface controller. In this case, the experiments are conducted in a environment where a network bridge is established using the tap device, a virtual Ethernet device provided by Linux. The specific configuration and behavior may vary depending on the hypervisor and network environment. Containers are isolated processes that share the physical resources and operating system of the host. Containers provide a lightweight and portable method of application deployment, where each container runs as an independent entity while leveraging the shared resources and underlying operating system provided by the host. This thesis will explain with a focus on Docker, which is a widely used container. Containers, as independent processes, share the operating system with the host and exhibit a network path for packet transmission and reception that is similar to the Non-Virtualized environment.

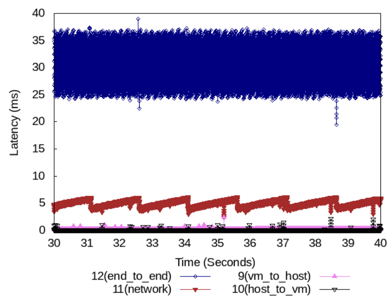
In this experiment, the network traffic in each environment is demonstrated using tools such as Iperf and the DASH application. Iperf is a widely used tool in Linux for measuring network performance. It is commonly used to assess network throughput, latency, and other network-related metrics. The DASH application refers to Dynamic Adaptive Streaming over HTTP (DASH), which is a streaming technology that divides videos into small chunks and dynamically adapts the version of the chunks based on the client's available bandwidth. It optimizes the streaming experience by delivering the most suitable version of video chunks according to the client's network situation.



(a) Non-Virtualized inter delay



(b) Container inter delay

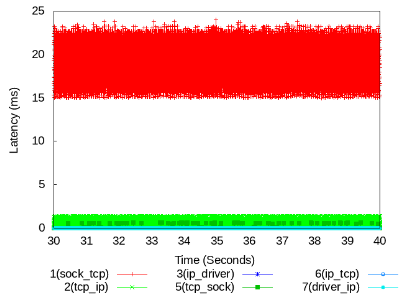


(c) Virtual Machine inter delay

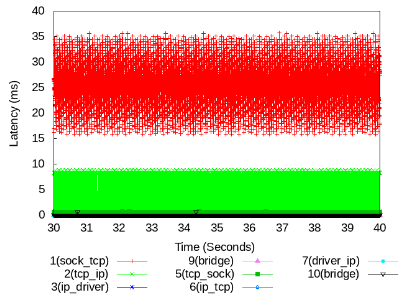
Non-Virtualized-iperf

In the Non-Virtualized environment, among the 12 (end-to-end) delays, the 1 (sock-tcp) interval exhibits the largest delay among the servers which sending data. Furthermore, it shows a periodic pattern of delay increase and decrease, which is attributed to the bufferbloat phenomenon. This observation is consistent

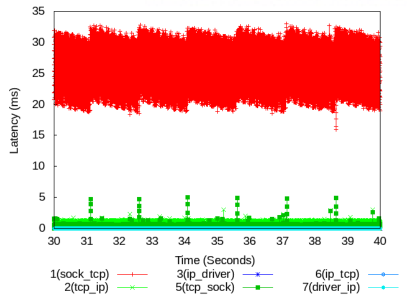
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY



(a) Non-Virtualized intra delay

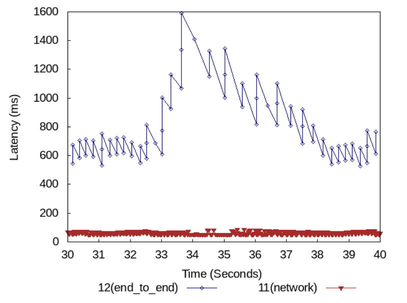


(b) Container intra delay

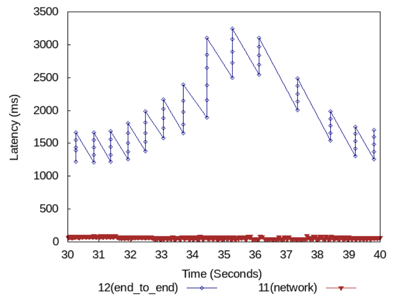


(c) Virtual Machine intra delay

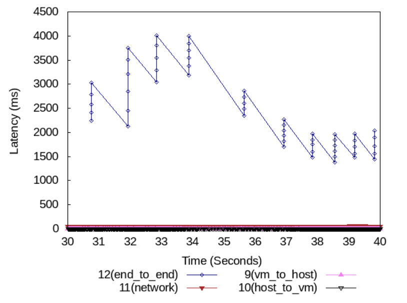
Figure 16: Iperf



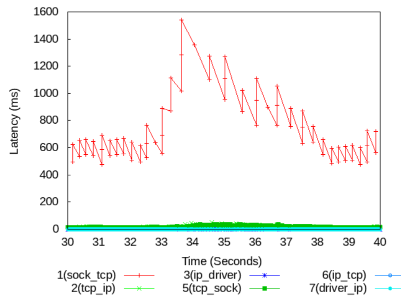
(a) Non-Virtualized inter delay



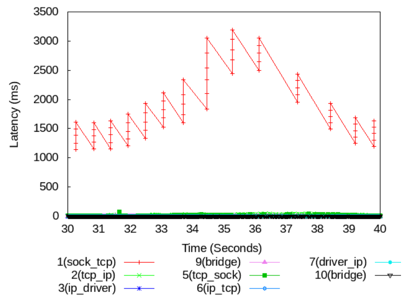
(b) Container inter delay



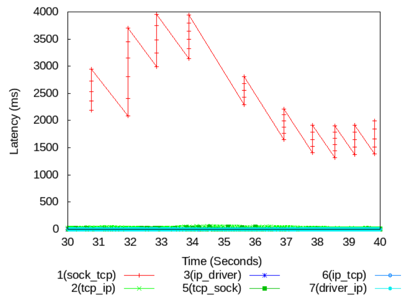
(c) Virtual Machine inter delay



(a) Non-Virtualized intra delay



(b) Container intra delay



(c) Virtual Machine intra delay

Figure 18: Iperf with Network Bandwidth 10mbps limit

with the graph patterns across all virtualization environments.

The above graph represents the network delay with a limited network bandwidth of 10Mbps. It shows an overall increase in delay among the 12 (end-to-end), 1 (sock-tcp), and 5 (tcp-sock). In particular, by limiting the bandwidth, we can observe an increase in delay in the 11 (network). And By limiting the bandwidth, it leads to a noticeable increase in the delay of the 1 (sock-tcp) caused by bufferbloat.

Virtual Machine-iperf

The average delay in the Virtual Machine environment is observed to be higher than in the Non-Virtualized environment. The delay that has the largest impact on the 12 (end-to-end) delay is the delay between the 1 (sock-tcp) interval. This indicates that the major delay occurred within the Virtual Machine rather than outside of it. This can be attributed to the characteristics of Virtual Machine, where hardware resources and operating systems are virtualized, leading to delays caused by factors such as cpu and memory usage. On the other hand, the delay when exchanging packets with the external environment of the Virtual Machine does not have a significant impact on the overall delay. This could be because in this experiment, only one Virtual Machine was running, which might have minimized the influence of the hypervisor. In contrast to the Non-Virtualized environment, the receiving server in the Virtual Machine environment exhibits a periodic increase and decrease in delay. This could be attributed to the fact that the packet loss due to the loss based congestion control. Additionally, as mentioned before, the Virtual Machine's characteristics, including cpu and memory usage limitation, could have a significant impact on the delay in the receiving server. As a result, unlike in other environments, the 11 (network) delay in the Virtual Machine scenario exhibits a staircase-like pattern of increasing and decreasing delays, which is influenced by the delay between the 2(tcp-ip) and 5(tcp-sock) segments of the receiving server.

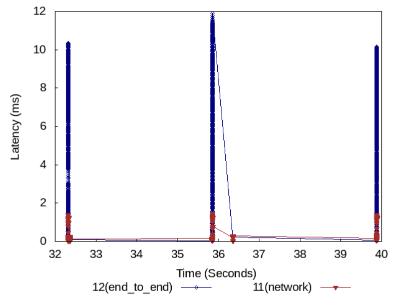
In the Virtual Machine environment as well, similar to the Non-Virtualized environment, there is an overall increase in delay in the 12 (end-to-end), 1 (sock-tcp), and 5 (tcp-sock) segments. Similarly to other environments, we can observe that limiting the network bandwidth results in a significant increase in delay in the 1 (sock-tcp) segment as well.

Container-iperf

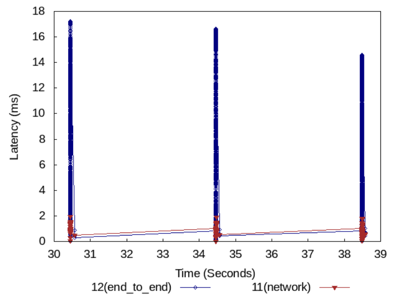
The Figure 16 exhibits a similar pattern to the previous two environments. Currently, Containers are being used in a configuration where they utilize virtual Ethernet and are connected to a Linux Bridge. Therefore, the main difference between Non-Virtualized environment and Virtual Machine environment is that in the Container environment, packets pass through an additional network bridge. Other than that, the characteristics and behavior of the packets remain similar to the Non-Virtualized environment. As shown in the graph 16, the delay introduced by passing through the network bridge 12 (end-to-end delay) does not have a significant impact. One difference from other environments is that the 2 (tcp-ip) delay between the data-receiving server shows a larger variation of delay. Additionally, the average delay of the data-receiving server is larger than the other two servers environments. This is due to the characteristics of the server using the Container environment. Containers utilize techniques like cgroups to limit the physical resources such as cpu and memory used by isolated processes. And Container use virtual ethernet. The performance of virtual ethernet can vary depending on various factors such as hypervisor or other processes. Unlike virtual machines, which virtualize hardware resources, containers share the same kernel as the host. This shared environment can lead to the observed graph 16 patterns, as mentioned before.

In the container environment, similar to Non-Virtualized and Virtual Machine environments, there is an increase in delay in the 12 (end-to-end), 1 (sock-tcp), and 5 (tcp-sock) intervals. And the interval with the most significant increase in delay is the 1 (sock-tcp) interval, which is consistent with the findings in the 2 other environments.

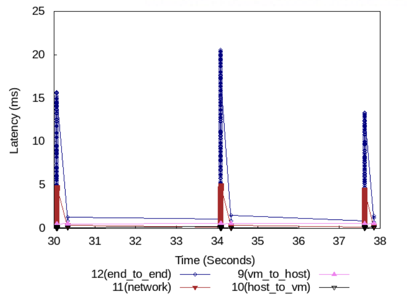
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY



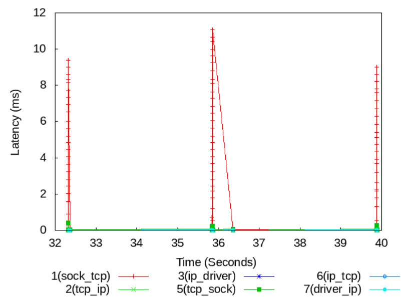
(**a**) Non-Virtualized inter delay



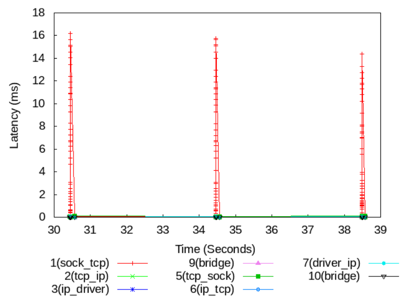
(**b**) Container inter delay



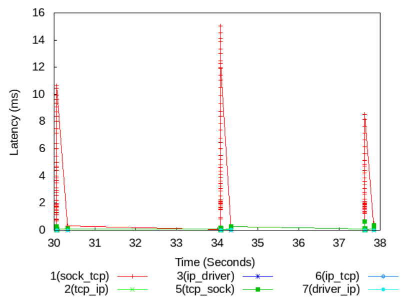
(**c**) Virtual Machine inter delay



(**a**) Non-Virtualized intra delay

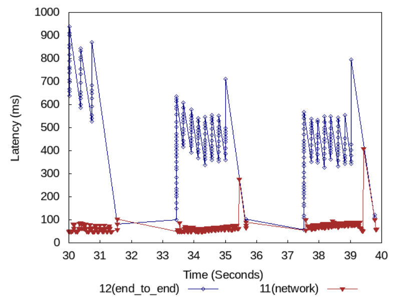


(**b**) Container intra delay

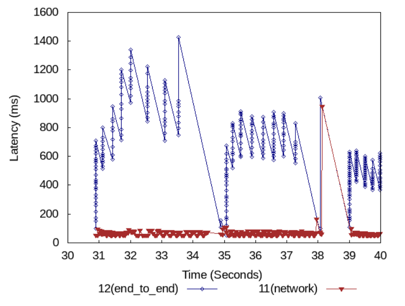


(**c**) Virtual Machine intra delay

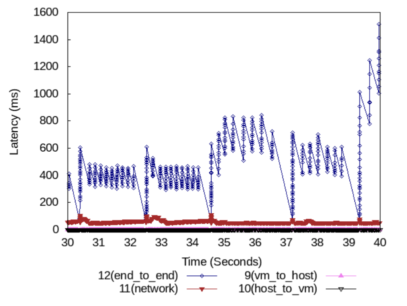
Figure 20: DASH



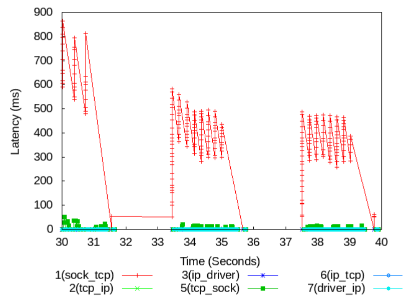
(**a**) Non-Virtualized inter delay



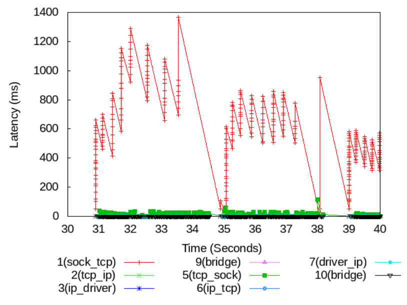
(**b**) Container inter delay



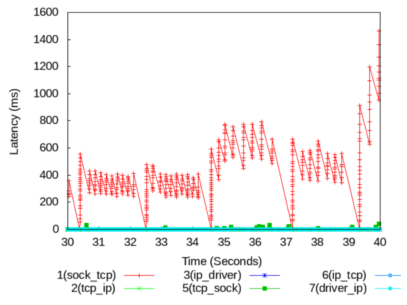
(**c**) Virtual Machine inter delay



(**a**) Non-Virtualized intra delay



(**b**) Container intra delay



(**c**) Virtual Machine intra delay

Figure 22: DASH with Network Bandwidth 10mbps limit

The above graph represents the network graph measured using the DASH application. The DASH allows clients to infer their network situation and request video segments from the server accordingly.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

This process is repeated approximately every 3-4 seconds. From the graph, we can observe that the server sends data for the corresponding video segments in approximately 3-4 second intervals. The graph exhibits a peak-shaped pattern due to the data being sent for the corresponding video segments within that time interval. From the analysis of all the graphs, it is evident that the 1(sock-tcp) which sends data contributes significantly to the overall delay. In this experiment, also bufferbloat plays a significant role in the overall network delay.

When the bandwidth is limited, We observe that there is not a significant difference in network delay across different virtualization environments. This can be attributed to the behavior of the DASH application, which dynamically adapts the data segments based on the network situation. As a result, the application can adjust the transmitted data segments to match the available network bandwidth, leading to relatively consistent network traffic patterns across different virtualization environments. Indeed, we can observe an overall increase of approximately 1-2 seconds in the data transmission duration of each cycle. This can be attributed to the limited bandwidth, which affects the time required to transmit the data.

Non-Virtualized-DASH

In the Non-Virtualzied environment, we can observe that the 1(sock-tcp) delay within the 12(end-to-end) delay has a significant impact. This indicates that the delay introduced in the communication between the Socket and the TCP Layer plays a major role in the overall end-to-end delay. The factors contributing to this delay caused by bufferbloat. When examining the graph at a 1-second granularity, we can observe that the 1(sock-tcp) segment exhibits a stair-like pattern with increasing delay from the start of data transmission. Due to the data transmission occurring in a 3-4 second interval, the overhead associated with sending and receiving the data is relatively smaller compared to iperf. The 1(network) delay remains relatively constant compared to the 12(end-to-end) delay, as the experimental environment is a 1Gbit/s network where network delay is relatively less prominent.

The above graph represents the network performance when the network bandwidth was adjusted to 10mbps in the Non-Virtualized environment. Compared to the previous graph 20, it shows a significant increase in the 12(end-to-end) delay. Overall, there is a substantial increase in the 1(sock-tcp), 5(tcp-sock), and 11(network) delays. In particular, the 11(network) delay shows an increased average delay compared to the previous graph 20. This is due to limiting the network bandwidth to 10mbps, which leads to the occurrence of the 11(network) delay and results in the observed graph pattern. As a result, it can be observed that the server's data transmission interval to the client has increased by 1-2 seconds.

Virtual Machine-DASH

In the Virtual Machine environment, the 12(end-to-end) delay shows the highest end-to-end delay compared to other server environments. Similar to the Non-Virtualized environment, the delay between the DASH server's 1(sock-tcp) interval accounts for a significant portion of the overall end-to-end delay. Looking at the graph, we can see that the delay between the 1(sock-tcp) interval within the Virtual Ma-

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

chine accounts for a significant portion of the 12(end-to-end) delay, while the delay between the 9(vm-to-host) and 10(host-to-vm) intervals in the Virtual Machine's external communication shows smaller delays. This also can be attributed to the characteristics of Virtual Machines, where the virtualization of hardware resources such as cpu and memory can have an impact. The utilization of these resources within the Virtual Machine can influence the performance and contribute to the observed delay. However, when viewed from an overall perspective, the average delay is similar.

The graph 22 represents the network delay when the network bandwidth was limited to 10 Mbps in the Virtual Machine environment. Similar to the Non-Virtualized environment, when compared to the previous graph 22, there is a significant increase in the 12 (end-to-end) delay. The graph pattern shows similarities to the changes observed in the Non-Virtualized environment.

Container-DASH

The 12 (end-to-end) delay in the Container environment shows similar delay patterns with the 2 other environment. This is because Containers utilize Linux Cgroups and Namespaces to isolate and run as individual processes, sharing the same physical resources and operating system. Similar to the Non-Virtualized server environment, the 12 (end-to-end) delay in the Container environment is largely influenced by the delay between the 1 (sock-tcp) segments of the DASH server. The slightly larger delay in the Container environment compared to the regular Non-Virtualized environment can be attributed to the characteristics of containers, where physical resources such as CPU and memory are limited using cgroups. However, when viewed from an overall perspective, the average delay is similar.

The graph 22 represents the network delay in the Container environment when the network bandwidth was limited to 10mbps. Similar to the previous graphs 20, a notable difference is observed in the increased time taken to transmit data within a segment, which is approximately 1-2 seconds longer compared to the original graph. This exhibits a similar graph pattern as seen in the Non-Virtualized and Virtual Machine environments. This similarity can be attributed to the fact that the experiment was conducted on a server with a Non-Virtualized environment. However, from an overall perspective, the average delay is similar across the board.

Common

After continuing the experiments, it was found that there are subtle differences between each server, but no significant differences exist except Virtual Machine environments. As mentioned earlier, the differences arise depending on how you configure Virtual Machines, Containers, and other components.

5.2 Different Network Environments

The network environment is configured using LAN, WIFI, and LTE, and network traffic is measured using tools such as Iperf and DASH. Indeed, LAN (Local Area Network), Wi-Fi, and LTE (Long-Term Evolution) represent different types of network environments with varying characteristics.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

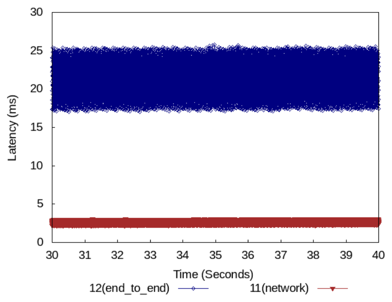
LAN refers to a local network infrastructure typically used in homes, offices, or campus settings. It provides high-speed and reliable wired connections using Ethernet cables.

Wi-Fi, on the other hand, is a wireless network technology that allows devices to connect to a network without the need for physical cables. It is commonly used in homes, public places, and businesses, providing flexibility and mobility.

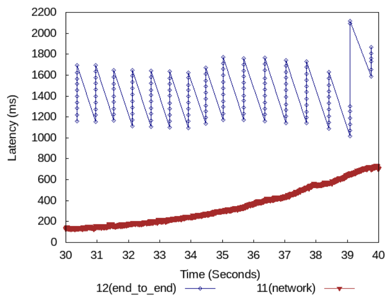
LTE, also known as 4G, is a wireless broadband technology used for cellular networks. It offers high-speed data transmission over long distances and is commonly used for mobile devices, providing internet connectivity outside of Wi-Fi coverage areas.

These different network environments have varying characteristics such as bandwidth, latency, and signal stability, which can impact network performance and the measured network delay.

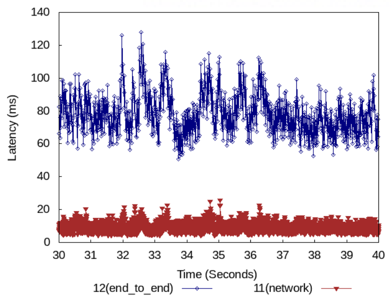
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY



(a) LAN

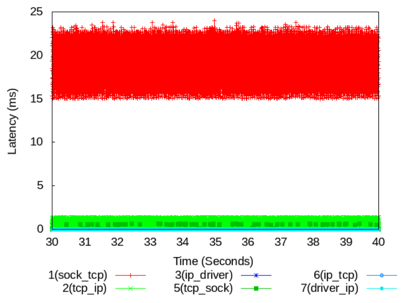


(b) LTE

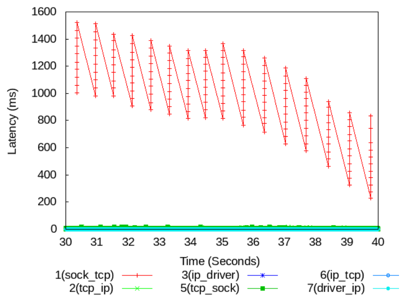


(c) WIFI

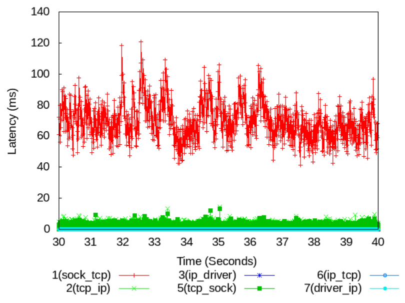
Figure 23: Internal Iperf Network Delay with Different Network Environments.



(a) LAN



(b) LTE



(c) WIFI

Figure 24: Intra Iperf Network Delay with Different Network Environments.

LAN-Iperf

The graph 23 and 24 represent the measurements conducted using Iperf in a LAN environment. It is evident that the 1(sock-tcp) delay contributes the most to the overall 12(end-to-end) delay. This observation consistent with the analysis conducted in the Virtualization environments, where the bufferbloat phenomenon in the sending server leads to this delay. Since we have already explained this phenomenon in the previous experiment, we can move on to the next experiments.

LTE-Iperf

The graph 23 and 24 represent the measurements conducted using Iperf in an LTE environment. Similar to the LAN environment, we observe that the delay between the 1(sock-tcp) contributes the most to the overall delay. This suggests that the bufferbloat phenomenon in the sending server is still present and affects the delay in the LTE network environment as well. One difference compared to the LAN environment is that the average delay is higher in the LTE environment.

In this experiment, the LTE network used a femtocell base station, which resulted in the slowest speeds compared to other network environments. In LTE network, it is observed that network delay increases over time due to bandwidth limitations.

This may also be due to the nature of the wireless network. Wireless communication has limitations

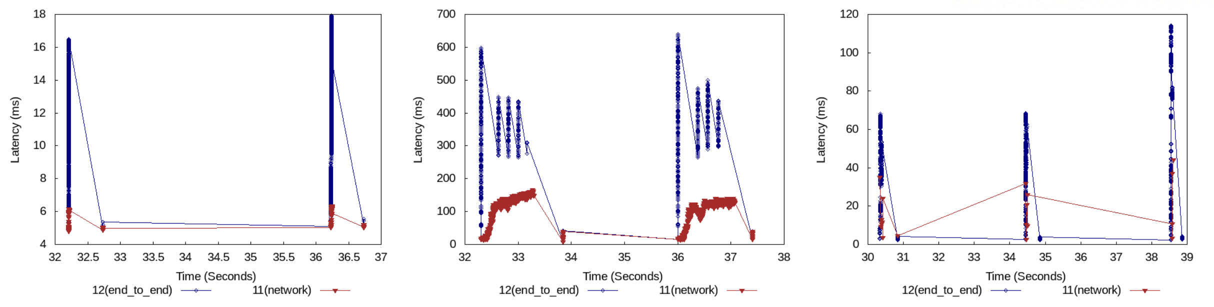
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

in bandwidth due to the transmission of data through the air. Additionally, wireless communication is more susceptible to signal interference compared to wired communication, which can result in slower data transmission speeds. Therefore, in this case as well, wireless communication exhibits lower data transmission speeds compared to wired communication, leading to higher overall delays as depicted in the graph. As a result, it is particularly noticeable that the 11(network) delay is significantly increased.

WIFI-Iperf

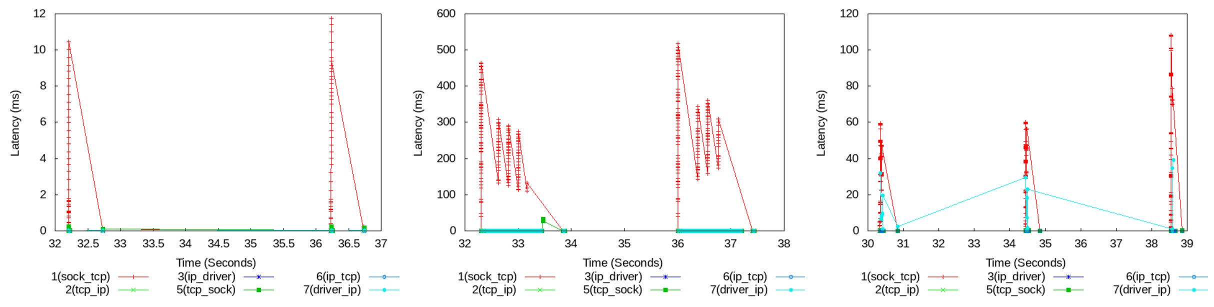
The graph 23 24 represents the network delay measured using Iperf in a WiFi environment. The graph exhibits a similar pattern with 2 other environments. The average delay in the WiFi environment is lower than that of LTE but higher than that of LAN. Similarly to the other network environments, the delay between the 1(sock-tcp) segments of the server contributes the most to the overall end-to-end delay in the WiFi environment. One difference compared to the other network environments is that the delay between the 11(network) segments appears to be more unstable in the WiFi environment. In contrast to the other two network environments, the WiFi environment shows more frequent fluctuations in the delay across all segments that correspond to the paths taken by network packets. The instability in the WiFi network environment can be attributed to factors such as distance from the WiFi router, signal interference, the number of devices connected to the WiFi network, the performance of the WiFi router itself and the medium access mechanism of WIFI. The lower average delay in the WiFi environment compared to the LTE environment indicates that the current WiFi environment using is faster than LTE.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY



(a) LAN **(b)** LTE **(c)** WIFI

Figure 25: Inter-host delay of DASH with different network environments.



(a) LAN **(b)** LTE **(c)** WIFI

Figure 26: Intra-host delay of DASH with different network environments.

LAN-DASH

The graph 25 26 represents the network delay measured using the DASH application in a LAN environment. Similar to previous experiments, we can observe that the delay between the DASH server's 1(sock-tcp) intervals contributes the most to the overall delay. There are no significant differences observed apart from that.

LTE-DASH

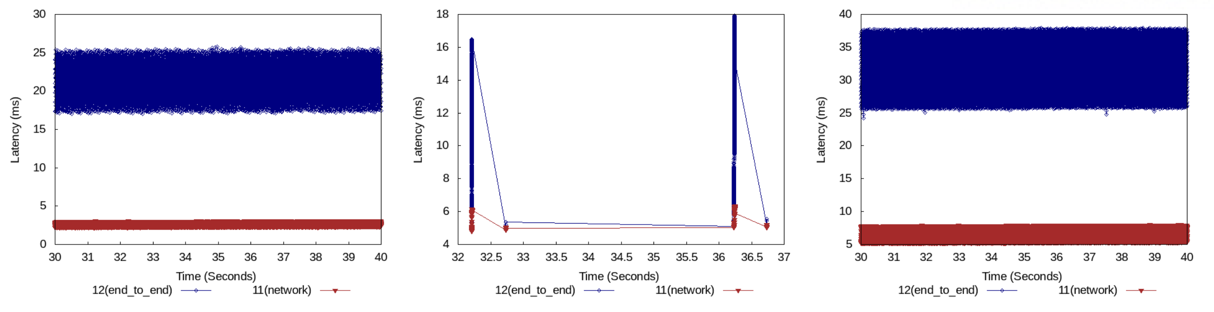
The graph 25 26 represents the network delay measured using the DASH application in an LTE environment. Here as well, similar to LAN-DASH experiment, we can observe that the delay between 1(sock-tcp) in the DASH server is the largest portion of the end-to-end delay. From the graph, we can see that the graph pattern is similar to the case where bandwidth was limited in a regular LAN environment. This indicates that the delay is occurring due to the characteristics of using LTE environment, as mentioned earlier. As a result, we can observe an increase in delay between the 11(network) intervals. Hence, we can observe that the time interval between data transmissions has increased by approximately 1-2 seconds. Additionally, other parts of the graph exhibit similar characteristics to the case where bandwidth was limited in a LAN environment.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

WIFI-DASH

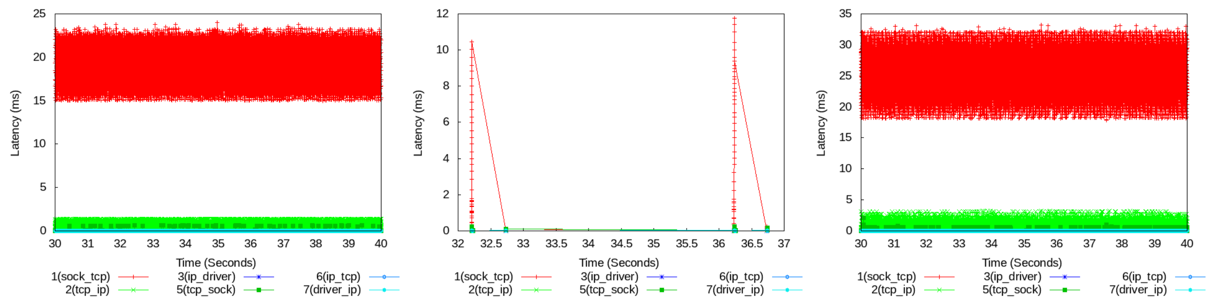
The graph 25 26 represents network delay measured using the DASH application in a WiFi environment. Similar to the previous environment, the graph shows that the average delay is higher than in LTE but lower than in LAN. The larger network delay compared to LAN is also a result of the characteristics of wireless communication. As a result, we can observe a significant delay in the 1(sock-tcp) interval of the DASH server. Apart from the differences in average delay, there are no significant difference.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY



(a) Iperf **(b)** DASH **(c)** FTP

Figure 27: Inter-host delay with different applications.



(a) Iperf **(b)** DASH **(c)** FTP

Figure 28: Intra-host delay with different applications.

5.3 Different Applications

Here, we are measuring and analyzing the network performance generated by different applications for the purpose of comparing it with the network performance measured using the Iperf and DASH servers. Additionally, we analyze the network performance generated by the FTP (File Transfer Protocol) application. FTP (File Transfer Protocol) is a standard protocol used for transferring files over a TCP/IP network. It provides a set of rules and commands that devices can use to initiate, authenticate, and transfer files between a client and a server. FTP is commonly used for uploading, downloading, and managing files on remote servers. FTP is indeed a client-server protocol used for file transfer over a TCP/IP network. Clients request files from the server, and the server responds by providing the requested files. Here we will compare and analyze the network performance of the three applications: Iperf, DASH, and FTP.

Iperf

Iperf is a network performance measurement tool provided in Linux, and unlike DASH, it continuously sends data packets. Except for the 1(sock-tcp) interval on the data-sending server, which shows periodic fluctuations due to bufferbloat, the delays in other intervals do not have a significant impact on the end-to-end delay.

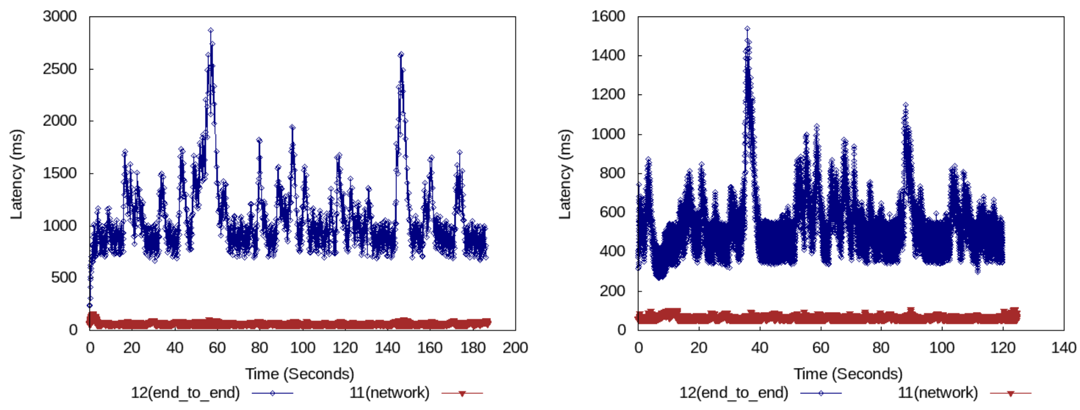
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

DASH

Unlike the other two applications, DASH sends data periodically. Typically, DASH sends segment information about the desired data to the server every 3-4 seconds, and the server delivers the corresponding data to the client. Since DASH sends data in bulk within each cycle, the graph exhibits a peak-shaped.

FTP

Like Iperf, FTP also involves continuous transmission of data. However, compared to the Iperf graph, the FTP graph shows relatively higher delays in the 12 (end-to-end), 11 (network), 1 (sock-tcp), and 2 (tcp-ip) intervals. This is because FTP sends data in a smaller size, which contributes to the higher delays observed in the mentioned intervals. Therefore, we can observe that the delay in the 1(sock-tcp) interval is slightly higher compared to Iperf. Compared to Iperf, there are more segments in FTP where there is byte matching between the sender and receiver. This suggests that FTP transmits data in smaller units or chunks, resulting in more frequent matching of bytes between the sender and receiver. This behavior is inherent to the FTP protocol, which breaks down files into smaller pieces and ensures their accurate delivery.



(a) Iperf **(b)** FTP

Figure 29: Inter-host delay with different applications in 10 Mbps network bandwidth.

The graph 29 30 represents network delay experimented with limited network bandwidth of 10 Mbps using tools such as iperf and FTP. The graph shows similar patterns of network traffic observed during the experiment. From the graph, it can be observed that by limiting the network bandwidth, both applications experienced an increase in network traffic. Additionally, the graph clearly shows the number of byte-matching points, highlighting them more prominently in this graph. Indeed, apart from the specific observation mentioned, the overall shape of the graphs appears to be similar, indicating similar patterns in network traffic.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

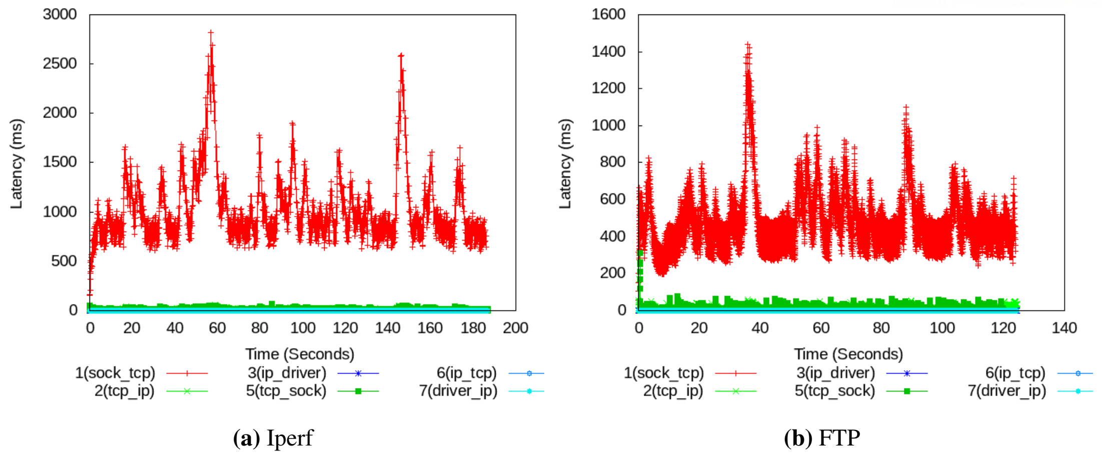


Figure 30: Intra-host delay with different applications in 10 Mbps network bandwidth.

5.4 Complex Scenarios

In this Section, we present network performance measured using Iperf and DASH in various scenarios that can occur in real-life situation. In real world, there are usually multiple clients connecting to the server simultaneously. As the server needs to send data to the clients requesting it, the workload and network complexity increase, leading to potential delays. eBPF-ELEMENT can be used to analyze which specific parts of the network stack contribute to significant delays. Furthermore, a single server typically has multiple network traffic streams. Since the physical resources of a server are limited, having multiple network traffic streams simultaneously can introduce delays between different network traffic streams. Therefore, when running Iperf and DASH, it analyzes the network performance in the presence of other network flows.

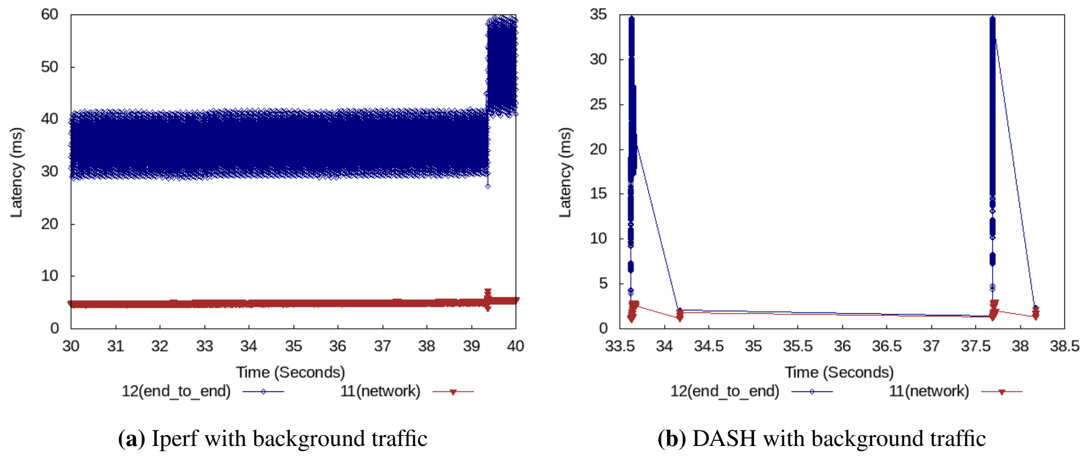
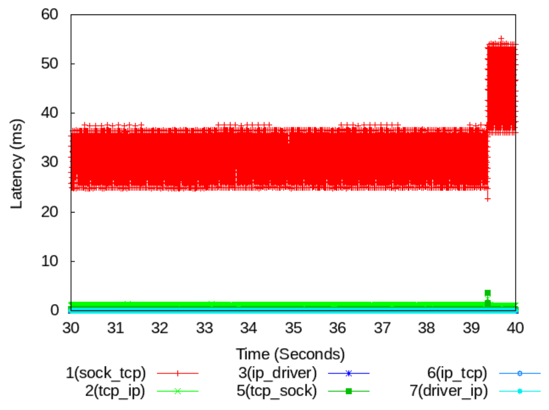
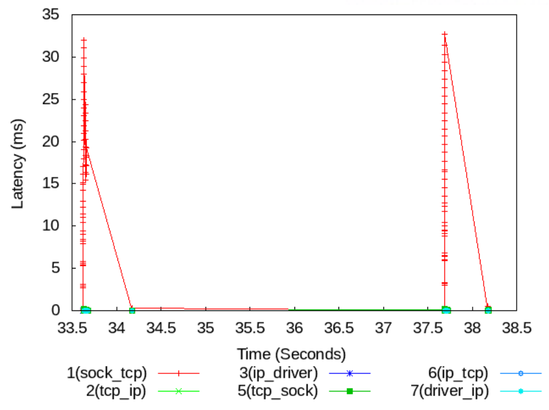


Figure 31: Inter-host delay with background traffic.

The graph 31 32 represents the measurement of network traffic using Iperf and DASH when there is background traffic present within a single server. One of the noticeable observations is that the average delay in the 12 (end-to-end) segments has increased. Other aspects show similar characteristics as observed in other experiments.

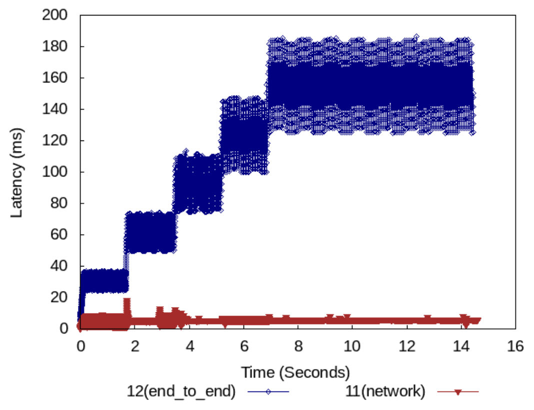


(a) Iperf with background traffic

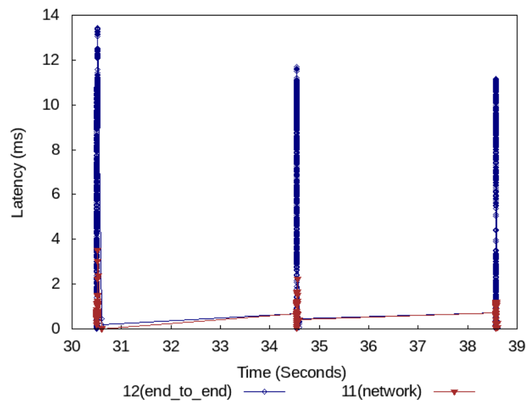


(b) DASH with background traffic

Figure 32: Intra-host delay with background traffic.

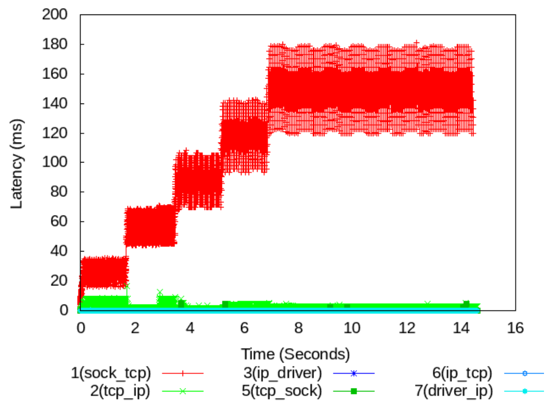


(a) Iperf with Multi Client

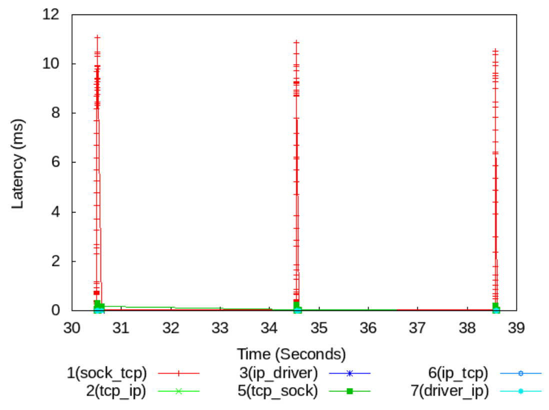


(b) DASH with Multi Client

Figure 33: Inter-host delay in the multiple clients scenario.



(a) Iperf with Multi Client



(b) DASH with Multi Client

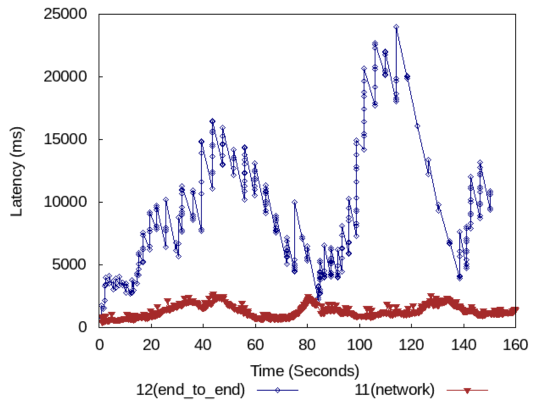
Figure 34: Intra-host delay in the multiple clients scenario.

The graph 33 34 represents the analysis of network performance when multiple clients connect to an application using Iperf and DASH. From the graph, we can observe that the average delay increase in the 12 (end-to-end) segments is larger for Iperf compared to the DASH server. This is because the DASH

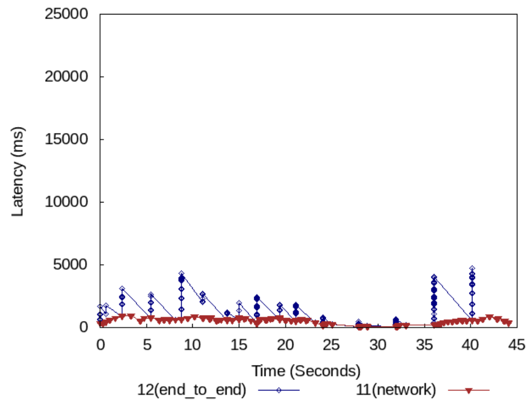
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

server sends Data segments at a periodic interval of 2-3 seconds, while Iperf continuously sends data. In other words, DASH has a lower probability of flow overlapping compared to Iperf. As the number of clients connecting to Iperf increases, the delay in the 1 (sock-tcp) segment exhibits a step-like increase pattern.

DASH server has two distinctive characteristics. The first characteristic is that the DASH server receives information about the required data from the client in intervals of 3-4 seconds and then sends the corresponding data to the client. The second characteristic is that multiple clients dynamically determine the required data chunks based on their own network situation. Due to these two characteristics, even with multiple clients, the probabilty of overlapping in data transmission intervals is reduced, and as observed in the experiments conducted with different server environments, the DASH server can effectively adapt to network conditions, minimizing the occurrence of significant delays.

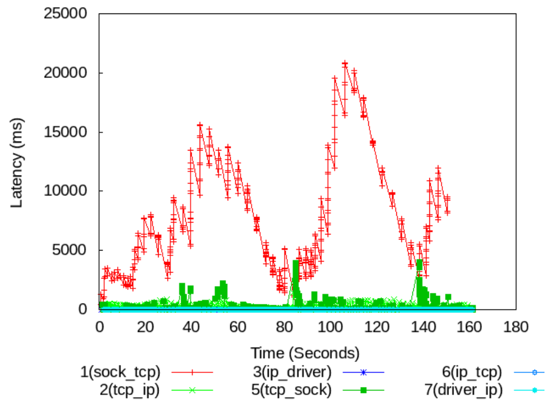


(a) Iperf

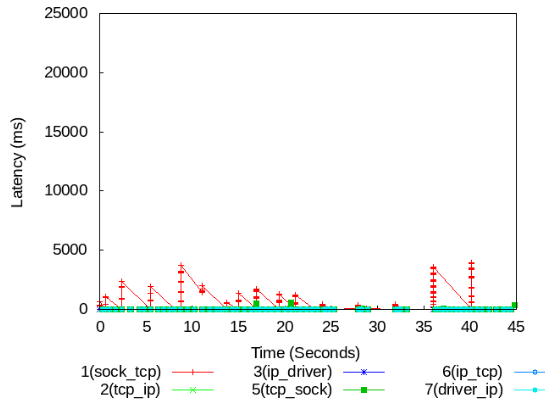


(b) DASH

Figure 35: Delay in the multiple clients and 10 Mbps bandwidth scenario.



(a) Iperf with Multi Client



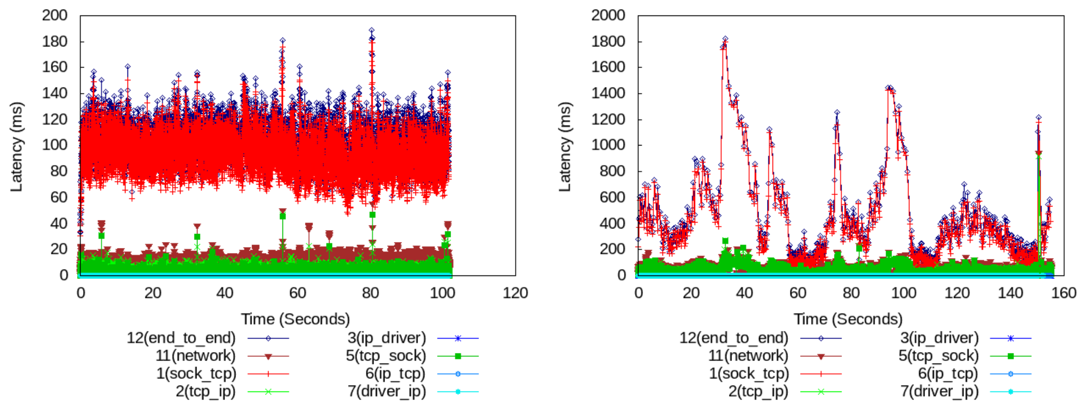
(b) DASH with Multi Client

Figure 36: Delay in the multiple clients and 10 Mbps bandwidth scenario.

The graph 35 36 represents the network delay when there are multiple clients which are connected Iperf and DASH servers with the network bandwidth set to 10 Mbps. In the current experiment with more than 10 clients, it can be observed that Iperf shows an increase in delay that is proportional to the

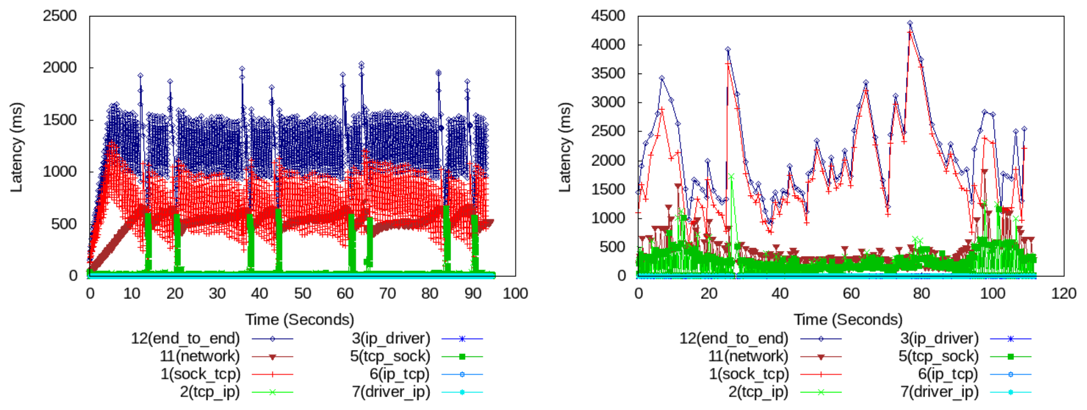
UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

number of connected clients, reaching a higher peak compared to the case with only one client. On the other hand, in the case of DASH, the delay does not increase proportionally to the number of clients. This can be attributed to the characteristics of the DASH mentioned earlier, where it dynamically adjusts its behavior based on the network situation. As a result, DASH exhibits lower average delay even with multiple clients.



(a) WIFI **(b)** Multi Client in WIFI, small buffer size

Figure 37: Delay in the WIFI



(a) LTE **(b)** Multi Client in LTE, small buffer size

Figure 38: Delay in the LTE

The above figure 37 38 represents the network performance measured when multiple clients are created in an environment with limited bandwidth, such as LTE and WIFI, and the network resources become saturated. The left figure 37 38 represents the network performance when one client is connected in an LTE, WIFI environment. The right figure 37 38 represents the network performance when more than 20 clients are connected. Additionally, to simulate a situation with limited memory, the TCP buffer size was decreased.

The results indicate that significant delays occur not only in the 1 (sock-tcp) but also in other segment, which cannot be ignored. Looking at graphs 37 38, we can observe an increase in delay between the 2 (tcp-ip) and 5 (tcp-sock). Decreasing the TCP buffer size in an environment where multiple clients

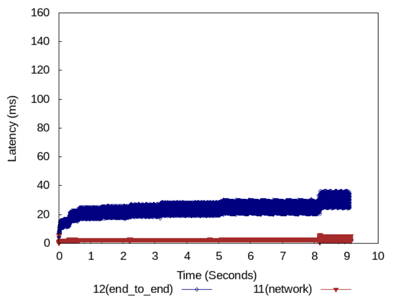
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

are connecting simultaneously has led to an increase in delay in 2 (tcp-ip). Additionally, in an environment using loss-based congestion control, there has been an increase in delay during the 5 (sock-tcp) due to packet loss.

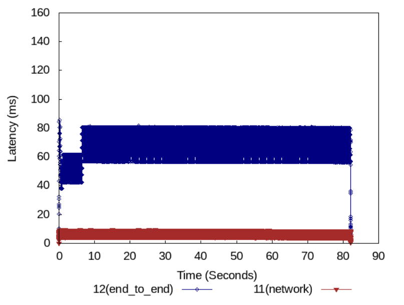
This shows that using eBPF-ELEMENT in a constantly changing network situations enables detailed analysis of delays between various network intervals.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

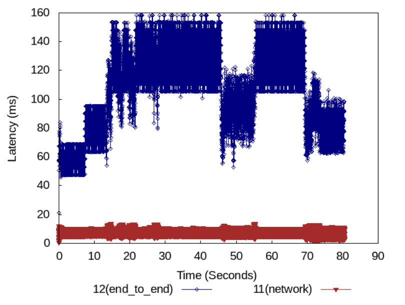
VI Latency Control



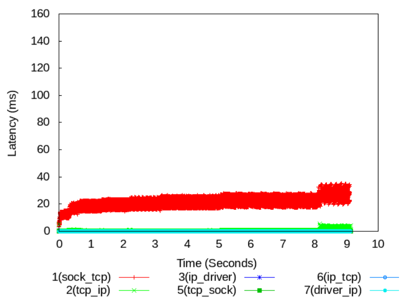
(a) Iperf with High Priority



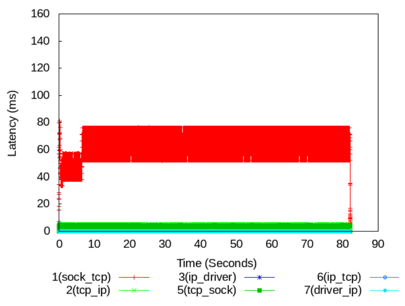
(b) Iperf with Middle Priority



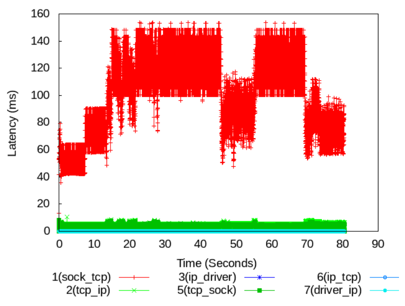
(c) Iperf with Low Priority



(a) Iperf with High Priority



(b) Iperf with Middle Priority



(c) Iperf with Low Priority

Figure 40: Delay caused by changes in packet priority

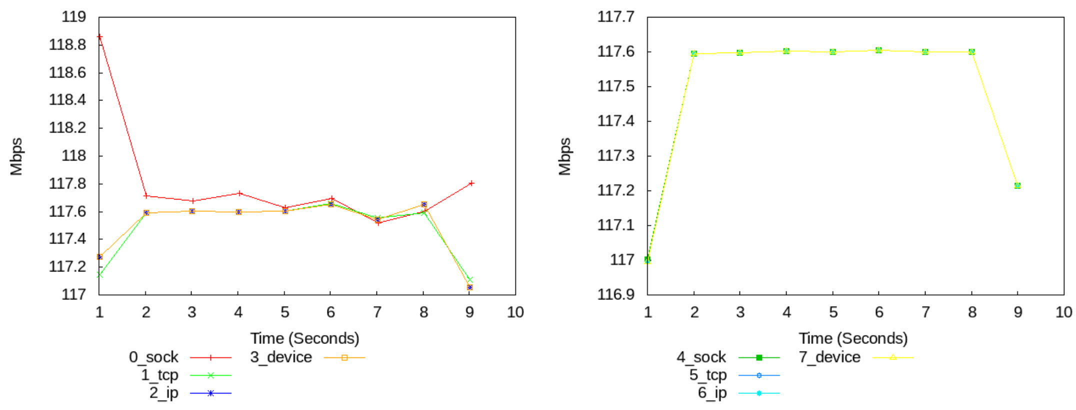


Figure 41: Throughput with High Priority

Here, we provide an example of how you can optimize network environments using eBPF-ELEMENT in a real networking environment. The graph 40 represents the measurements taken while varying the priority of one flow among multiple flows when there are more than 10 Iperf clients. Here, we use the TC (Traffic Control) command to classify packets and enqueue them in the queue according to their respective priority value.

From the graph, we can observe that as the priority level increases, the average delay decreases, while it increases as the priority level decreases. Through this, we can understand that by adjusting the

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY

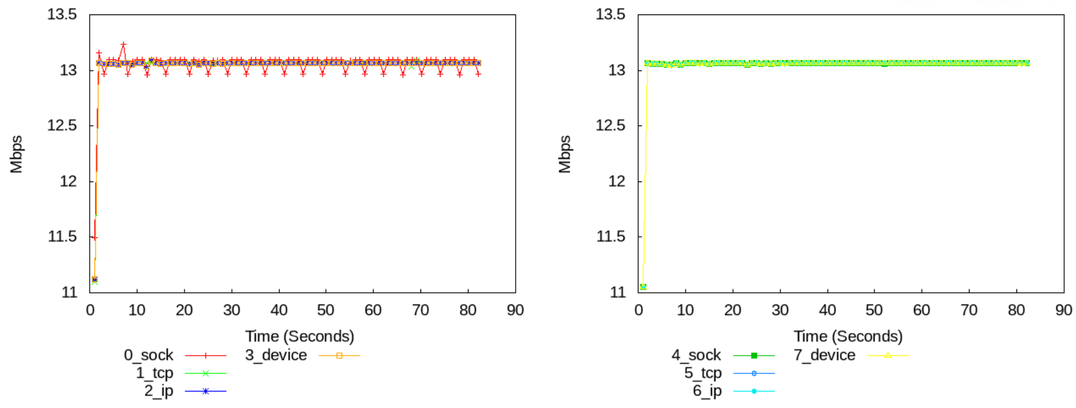


Figure 42: Throughput with Middle Priority

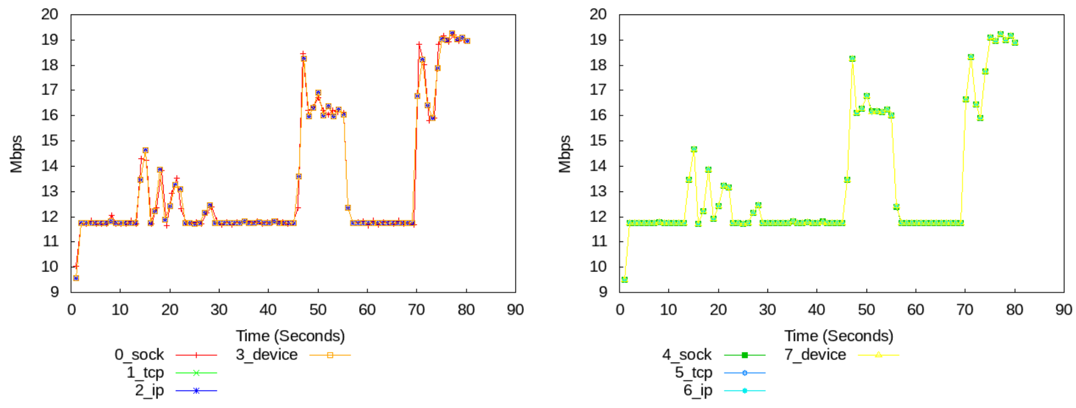
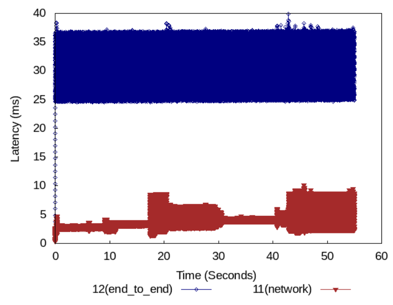


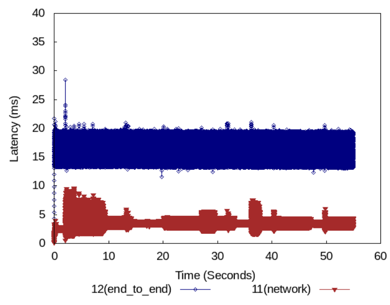
Figure 43: Throughput with Low Priority

packet priority, we can ensure a certain level of Quality of Service (QoS) even in situations where there are many existing flows.

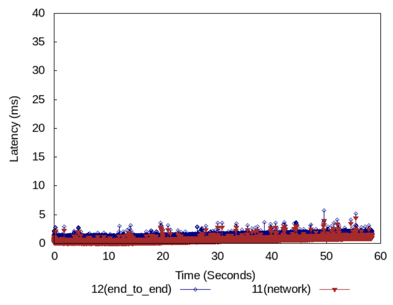
Through the use of eBPF-ELEMENT, we can analyze network performance in real-time and utilize the results to ensure Quality of Service (QoS).



(a) Iperf with High Buffer



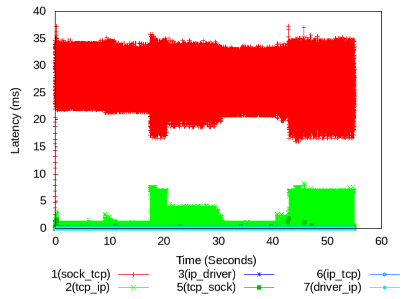
(b) Iperf with Middle Buffer



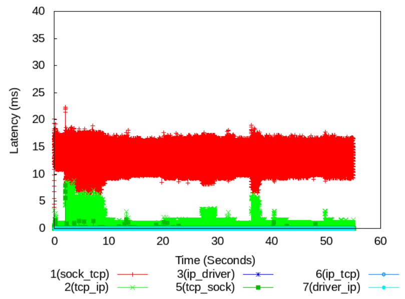
(c) Iperf with Low Buffer

The graph 45 is an example intended to demonstrate another use case. This graph 45 represents an experiment conducted to reduce the delay in the 1 (sock-tcp) segment, which accounts for the majority of the 12 (end-to-end) delay. The experiment involved adjusting the TCP buffer size. As demonstrated

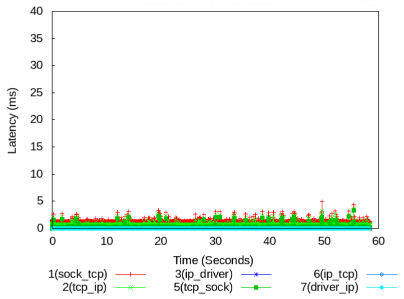
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY



(a) Iperf with High Buffer



(b) Iperf with Middle Buffer



(c) Iperf with Low Buffer

Figure 45: Delay caused by changes in tcp's buffer size

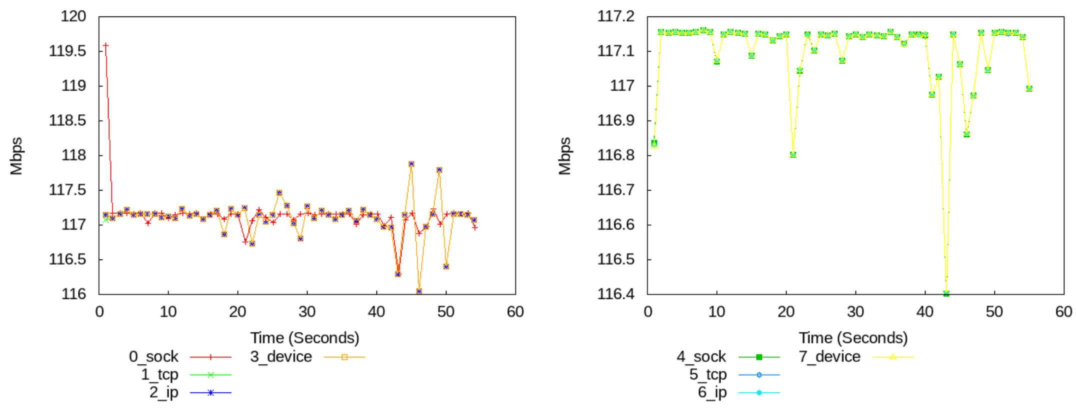


Figure 46: Throughput with High Buffer

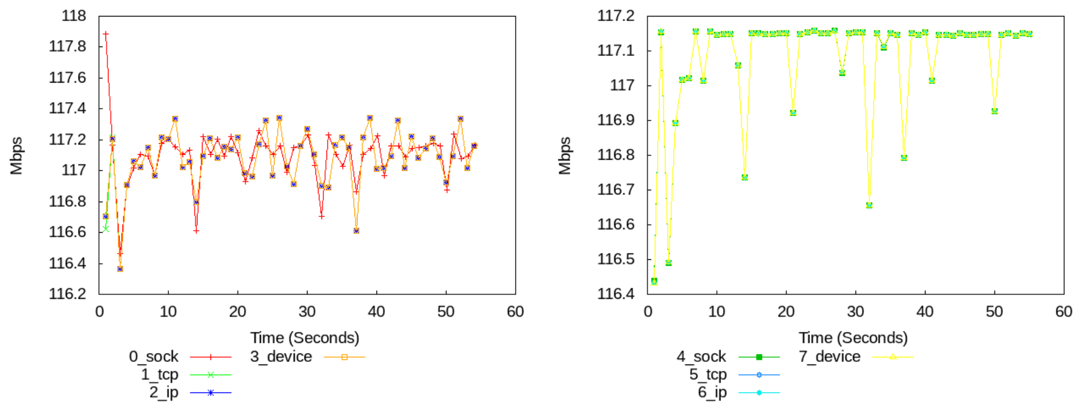


Figure 47: Throughput with Middle Buffer

in the existing research [1], the delay between the 1 (sock-tcp) segments is caused by the phenomenon known as bufferbloat. By appropriately tuning the buffer size, it is possible to mitigate the impact of bufferbloat and regulate the delay.

From the graph, we can observe that as the TCP buffer size decreases, the average delay also decreases. However, it is worth noting that reducing the buffer size too much can result in an unstable graph, indicating potential issues with reducing througput and increased variability in delay.

Indeed, using the results obtained from eBPF-ELEMENT, it is possible to dynamically adjust the TCP buffer size using 'sysctl' to strike a trade-off between network stability, throughput and delay. By

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

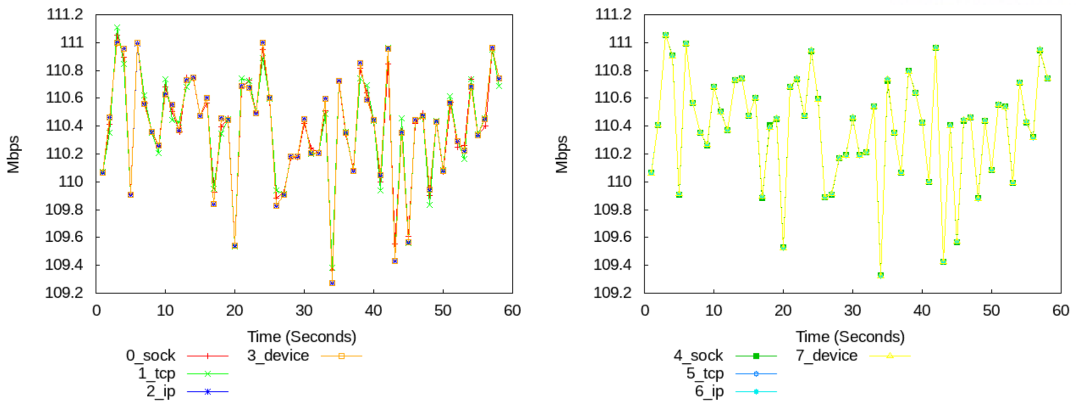


Figure 48: Throughput with Low Buffer

considering the network situation, we can effectively control the 12 (end-to-end) delay by adjusting the TCP buffer size dynamically.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

VII Discussion

eBPF-ELEMENT is a tool designed for measuring network performance for a large-scale system. As the number of servers requiring probing increases, the amount of data to be managed will grow exponentially. The reason is that even when adding just one server for probing, that server may contain multiple flows. Therefore, an efficient Architecture is needed to manage the proposed framework while enabling fast calculation of latency. Currently, eBPF-ELEMENT maintains only one instance of the Manager and Analyzer components, performing computations for all flows. This limits scalability and the ability to handle a large number of flows efficiently. In the future, to enhance scalability, it would be necessary to transform eBPF-ELEMENT into a Master-Slave architecture, enabling fast network performance measurement.

As the data collected from each server grows exponentially, it becomes necessary to partition and store the data in Database tables to facilitate fast querying. By partitioning and optimizing the storage of data in Database, it becomes possible to reduce the query time for the most time-consuming operations. This optimization leads to faster overall processing time, enabling more efficient and timely analysis of network performance data in eBPF-ELEMENT.

Currently, eBPF-ELEMENT is limited to the TCP protocol. Indeed, UDP is commonly used in scenarios where fast network communication is required, such as real-time applications or multimedia streaming. Currently, there is significant research underway to add additional functionality to UDP by incorporating fields in UDP packets like selective packet send, retransmission etc. Many organizations are already leveraging the speed of the UDP protocol while incorporating additional network functionalities by layering programs on top of the UDP protocol like Google QUIC. So, it becomes necessary for eBPF-ELEMENT to extend its capabilities to include network performance measurement for UDP-based communication.

eBPF-ELEMENT can analyze various segments within the network path in detail. However, currently, it is not being utilized to improve network conditions. Since eBPF-ELEMENT allows for more detailed analysis of network performance compared to the existing network performance tools, utilizing it effectively involves analyzing network traffic and developing network performance optimization algorithms.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

VIII Future Works

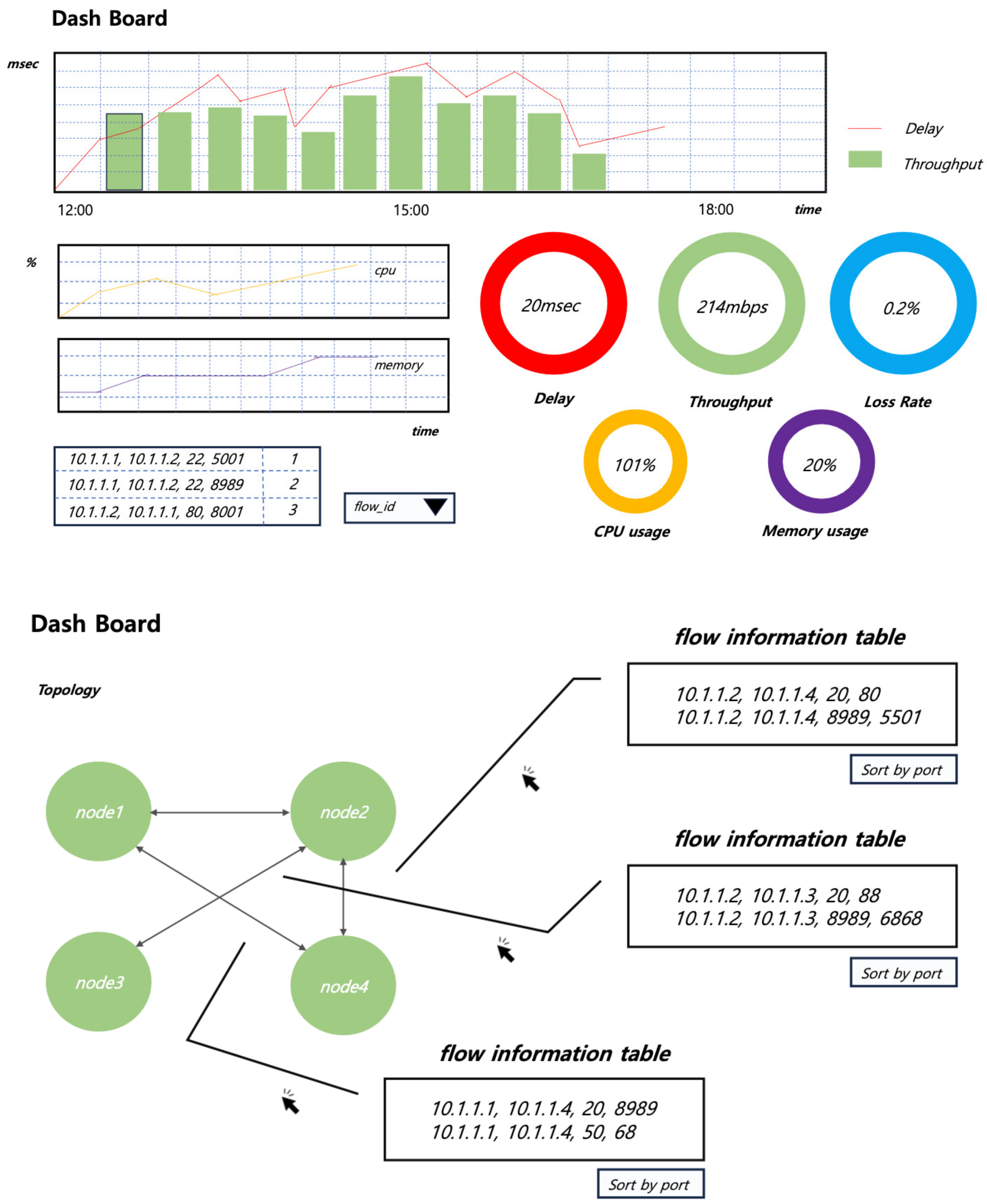


Figure 49: Dashboard Sample

The sampling interval has a trade-off relationship between the accuracy of network performance measurement and the server overhead. Currently, before the program starts, user can specify the sampling interval, and the network performance is continuously measured using the specified sampling

interval. Since the network environment is constantly changing, using a fixed sampling interval may be unsuitable for specific network situations. Hence, To adapt to different network environments, the sampling interval can be adjusted based on the frequency of events generated by each server within a given time frame. This allows for adjusting the sampling interval according to the specific network situations. By adjusting the sampling interval based on the frequency of events in each situation, it becomes possible to achieve more accurate network performance measurements in different scenarios.
Currently, the results of network performance measurements are being stored in a database for future use. In the future, a unified graphical user interface (GUI) will be developed, utilizing the computed results to display real-time network performance. This GUI aims to provide users with an easy-to-use interface to visualize network performance effectively.

Previously, the ELEMENT [1] utilized the results to create an algorithm for minimizing latency. However, with eBPF-ELEMENT, it currently does not utilize the results. The more granular network performance measurement capabilities of eBPF-ELEMENT allow for more precise measurements between each layer. As a result, there are plans to leverage this capability to develop an more advanced algorithm for fine-tuning latency minimization in the future.

The results of eBPF-ELEMENT can be utilized in a variety of places as well as Latency Minization. For example, the phenomenon of server failure in a distributed environments has become a problem and various studies are being conducted to solve it. These studies are trying to create a way to find out possibility of failure before server fail and take quick action. [13, 14, 15] suggest how to detect gray failure, a phenomenon that occurs before the server failure. Detection of gray failure is important in terms of accuracy and speed. As proved in eariler section, eBPF-ELEMENT enables detailed network performance measurements. In addition, eBPF-ELEMENT provides various metrics for servers by measuring system metrics. Using this, eBPF-ELEMENT when grap failure occurs later compares the measurement graph with the graph of previously studied papers to present an algorithm that can reinforce the detection of gray failure along with existing research.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

IX Conclusion

This thesis suggested a tool that extends the existing ELEMENT paper by utilizing eBPF, a tool provided by Linux, to measure network performance within servers and between servers without modifying the application code. Through this tool, various experiments were conducted, including different server environments, network environments, applications, and complex scenarios. The purpose was to validate the proper functioning of eBPF-ELEMENT and analyze the network traffic obtained from these experiments. The key difference from the original ELEMENT is that it allows for granular analysis of all paths traversed by the packets. This enables a more detailed and comprehensive analysis compared to the original ELEMENT approach.

In all experiments conducted, it is consistently observed that the delay between the 12(end-to-end) measurement points is predominantly influenced by the delay between the 1(sock-tcp) points of the data-sending server. This finding aligns with the previous ELEMENT paper, which demonstrated that such delays are primarily caused by bufferbloat. The shape of the network traffic graph can vary depending on the application utilizing the network traffic. Furthermore, depending on how the experimental environment is set up, we may observe variations in the average delay even if the graph shapes remain similar.

Through the use of eBPF-ELEMENT, it was discovered that detailed analysis of network traffic in various environments is possible. This provides an opportunity to explore strategies for reducing network delay by applying optimal techniques, such as packet scheduling, based on the results obtained.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

References

[1] Y. Im, P. Rahimzadeh, B. Shouse, S. Park, C. Joe-Wong, K. Lee, and S. Ha, “I sent it: Where does slow data go to wait?” in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–15.

[2] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, “Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[3] M. Abranches, O. Michel, E. Keller, and S. Schmid, “Efficient network monitoring applications in the kernel with ebpf and xdp,” *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*.

[4] S. Sundberg, A. Brunstrom, S. Ferlin-Reiter, T. Hoiland-Jorgensen, and J. D. Brouer, “Efficient continuous latency monitoring with ebpf,” *Lecture Notes in Computer Science(LNCS)*.

[5] C. Cassagnes, L. Trestioreanu, C. Joly, and R. State, “The rise of ebpf for non-intrusive performance monitoring,” *IEEE*.

[6] K. Suo, Y. Zhao, W. Chen, and J. Rao, “vnettracer: Efficient and programmable packet tracing in virtualized networks,” *IEEE*.

[7] M. sung You, J. woo Kim, S. Shin, and T. june Park, “Bpfast: An ebpf/xdp-based high-performance packet payload inspection system for cloud environments,” *Korea Inststute of Information Security and Cryptology*, vol. 32, no. 2, pp. 213–225.

[8] Y. Choe, J.-S. Shin, S. Lee, and J. Kim, “ebpf/xdp based network traffic visualization and dos mitigation for intelligent service protection,” *Lecture Notes on Data Engineering and Communications Technologies*.

[9] P. Salva-Garcia, R. Ricart-Sanchez, E. Chirivella-Perez, Q. Wang, and J. M. A. Calero, “Xdp-based smartnic hardware performance acceleration for next-generation networks,” *Network and Systems Management*.

[10] T. A. N. do Amaral, R. V.Rosa, D. F. Moura, and C. E. Rothenberg, “Run-time adaptive in-kernel bpf/xdp solution for 5g upf,” *FEEC*.

[11] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, “Pingmesh: A large-scale system for data center network latency measurement and analysis,” *SIGCOMM*, pp. 139–152.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

[12] M. Spier, A. Ather, and B. Gregg, “Introducing vector: Netflix’s on-host performance monitoring tool,” 2015, https://netflixtechblog.com/introducing-vector-netflixs-on-host-performance-monitoring-tool-c0d3058c3f6f.

[13] P. Huang, C. Guo, L. Zhou, J. R.Lorch, Y. Dang, M. Chintalapati, and R. Yao, “Gray failure: The achilles’heel of cloud-scale systems,” *HOTOS*, pp. 150–155.

[14] J. B.Leners, H. Wu, W.-L. Hung, M. K.Aguilera, and M. Walfish, “Detecting failures in distributed systems with the falcon spy network,” *SOSP*, pp. 279–294.

[15] J. B.Leners, T. Gupta, M. K.Aguilera, and M. Walfish, “Improving availability in distributed systems with failure informers,” *NSDI*.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Acknowledgements

I would like to express my gratitude to Prof. Youngbin Im for guiding my research. Through the guidance of my advisor, Prof. Youngbin Im, I have learned how to study and research. I am grateful for the dedicated guidance of my Prof, who has diligently led my mediocre research and helped me achieve the best possible results with limited outcomes. Also, I would like to thank Prof. Myeongjae Jeon, Yuseok Jeon for participating as judges and advising me on the experiment.

During my time in the lab, I have not only focused on network research but also extensively studied computer systems. I have discovered areas in which I struggle, and this has served as a catalyst for personal growth, motivating me to take steps forward. Based on the attitude I learned from my advisor and the knowledge I gained in the lab, I will make further efforts to grow in areas where I can excel. Thank you.

ULSAN NATIONAL INSTITUTE OF
SCIENCE AND TECHNOLOGY