2024 20th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) | 979-8-3503-5632-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICNC-FSKD64080.2024.10702282

# High-Performance Network Firewall Based on XDP

Xiaohang Zhang
School of Computer Science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China

Xinfeng Shu
School of Computer Science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China

Lijun Chen
School of Computer Science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China

Ruilian Xie
School of Computer Science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China

***Abstract*—With the development of computer technology, Internet technology has also witnessed rapid growth. However, the rapid expansion of the internet has led to an increase in the methods and diversity of cyberattacks, making network security a significant challenge that cannot be ignored in the global development of the internet. In the context of escalating cybersecurity threats and increasing network data traffic, enhancing the performance and effectiveness of firewall technology is crucial for strengthening network security protection and ensuring stable network operation. Traditional network firewalls operate in kernel mode, providing packet processing and filtering functions at the end of the protocol stack. When a large number of packets are discarded, it can lead to the CPU being occupied for a long time, resulting in significant resource waste. If packet processing can be done early in the packet reception process, it can greatly improve the performance of the firewall. To address the above issues, this article designs and implements a high-performance firewall technology based on the fast path packet processing capabilities of eXpress Data Path(XDP). With XDP technology, data packets can be processed at the earliest stage when they arrive at the network interface, without having to transfer the packets to the later stages of the kernel network stack. This approach offers better performance compared to traditional network firewall packet filtering methods.**

***Keywords*—Extended Berkeley Packet Filter, Firewall, Fastpath, eXpress Data Path.**

## I. Introduction

With the rapid development of internet technology, the explosive growth of network traffic has also brought about non-negligible network security issues. According to the 45th Statistical Report on the Development of China's Internet Network issued by the China National Computer Network Emergency Technical Treatment Coordination Center (CNNIC)[1], Distributed Denial of Service (DDoS) attacks still occur frequently, and many network devices are still vulnerable to hacking. It can be seen that China's network security is still facing enormous threats. Firewall technology plays a central role in defending against network threats[2]. With the rapid development of internet technology and the increasing demand for security by users and enterprises, firewall technology must continue to evolve to meet more diverse needs and higher-level challenges.

In the evolution of Linux firewalls[3], it has gone through three stages: ipfwadm, ipchains, and iptables[4], but all have faced performance issues. Many researchers have proposed various solutions aimed at improving the processing performance of firewalls. Researchers have actively explored and aimed to enhance the execution efficiency of firewalls by combining traffic statistics with early packet filtering strategies using multi-level expansion trees, especially targeting performance bottlenecks in the packet filtering process[5]. They have introduced an innovative approach, which is multi-level packet filtering with optimized filtering field order, with the goal of improving response speed when dealing with DDoS attacks[6]. However, they pointed out that current policy models based on packet statistics are overly simplified, and relying solely on a single parameter may lead to a certain degree of accuracy loss.

Researchers have attempted to improve firewall performance by reducing the number of rules. They proposed a geometric model approach based on multi-dimensional linear polygons and explored a new firewall policy compression plan to compress firewall rules, verifying its effectiveness. Other research has focused on the framework level to enhance firewall performance. Some studies have concentrated on optimizing existing frameworks[7], such as netfilter, through hardware and software aspects to improve performance. However, these methods are ultimately still limited by the way netfilter handles data packets. In addition, research has also presented a firewall optimization solution that operates entirely within the kernel[8]. While this approach can optimize specific performance metrics through algorithms, its drawback is that it fails to provide sufficient flexibility and has poor observability when working in the kernel.

With the expansion of network scale, researchers have begun

This research is supported by the Key Research and Development Project of Shaanxi Province (No.2024GX-YBXM-110), and Shanghai Automotive Industry Science and Technology Development Foundation (No. 2206).

to delve deeper into the impact of the number of rules on firewall performance[9]. In the discussion of rule management, some scholars have found that excessive predefined rules may lead to rule conflicts, which not only threaten the security of the firewall but also potentially result in performance degradation. To address this issue, they proposed an optimized tree-based rule firewall model that considers the avoidance of rule conflicts, thereby improving overall execution efficiency[10]. However, it is worth noting that once the rule set of this model is constructed, its time complexity remains relatively fixed, which means the flexibility of rule changes is limited[11]. This, to a certain extent, constrains the dynamic optimization of packet processing speed.

The main contributions of this paper on high-performance network firewall technology based on fast data path are as follows:

- **By processing packets at the earliest stage when they arrive at the network interface using XDP[12] technology, there is no need to pass the packets to the later stages of the kernel network stack, thus achieving very low latency, high throughput, and low CPU utilization.**
- **Multiple filtering rules can be loaded using eBPF maps[13], achieving higher accuracy without the need to rely on other tools.**

## II. System Design

### A. System Overview

The XDP-based firewall system is divided into three main modules: the rule matching module, the rule loading module, and the XDP[14] program loading module. The composition of these modules is illustrated in Fig 1.

The XDP program loading module injects ELF files into the XDP hook in the kernel. The XDP hook operates in three modes: Native, Offload, and Generic. In Native mode, the XDP program is loaded and runs in the network card driver, allowing packet parsing and processing using the XDP program before the allocation of the skb, after the packet is DMA'd into the ring buffer. In Offload mode, the XDP program is directly offloaded to the network card, eliminating the need for CPU involvement and providing better performance[15]. However, the prerequisite for using Offload mode is that the network card device supports offloading XDP programs. In cases where the network card and driver of a computer do not support the XDP hook, the XDP Generic mode can be used. This XDP hook operates after the allocation of the skb, positioned later compared to Native and Offload modes.

The rule loading module can load filtering rules into eBPF maps in user space. After the program is loaded onto the XDP hook, the packet parsing module can parse the packets received by the network card, extract packet information, and then run the program on the XDP hook to filter packets by searching for filtering rules in the eBPF map.

The rule matching module is the main logic that implements the firewall functionality, screening data packets based on packet header information. The program loading module injects the XDP program into the XDP hook. After the network card device receives and parses the packets, the rule matching module performs matching based on the rules stored in the eBPF map and the information parsed from the packets.

### B. Rule Loading Module

The rule loading module can load filtering rules into eBPF maps in user space, allowing for the addition, search, update, and deletion of filtering rules. This module enables defining and loading the rules that need to be filtered from a file, achieving the capability of adding multiple rules simultaneously compared to traditional firewalls.

*1) **Parameter Design**:* Compared to traditional firewalls, the fast-path-based firewall achieves more precise filtering by selecting a wider range of parameters. It supports MAC addresses at the data link layer, IPv4 protocol at the network layer, and TCP, UDP, and ICMP protocols at the transport layer. In terms of parameter selection, it supports source MAC address, source IP address, source port, destination IP address, and destination port. The types of parameters are shown in Table I.

TABLE I
PARAMETER TYPES

| Parameter | Type | Description |
|---|---|---|
| src_mac | uint64 | Source MAC Address |
| dst_mac | uint64 | Destination MAC Address |
| saddr | uint32 | Source IP Address |
| daddr | uint32 | Destination IP Address |
| sport | uint16 | Source Port |
| dport | uint16 | Destination Port |
| proto | uint16 | Protocol Type |

*2) **Parameter Storage**:* Programs interact between user space and kernel space through the map mechanism . eBPF stores data in the kernel in the form of key/value data structures. Multiple maps can be created, and all maps can be accessed in user space programs through file descriptors. bpf_map_lookup_elem can be used to search for the key-value pair of an entry in an eBPF map, while bpf_map_update_elem can be used to update the key-value pair of an entry in the map.

### C. Rule Matching Module

The Rule Matching Module is the core component of the eBPF-based firewall. It filters packets based on packet header information and filtering rules. When a packet arrives at the network card and is parsed by the packet parsing module to obtain the packet header information, the Rule Matching Module searches for filtering rules in the eBPF map and performs matching and screening between the obtained packet header information and the filtering rules.

*1) **Parameter Extraction**:* To further enhance the performance of the network firewall, XDP technology is adopted for the extraction of packet information. This eliminates the

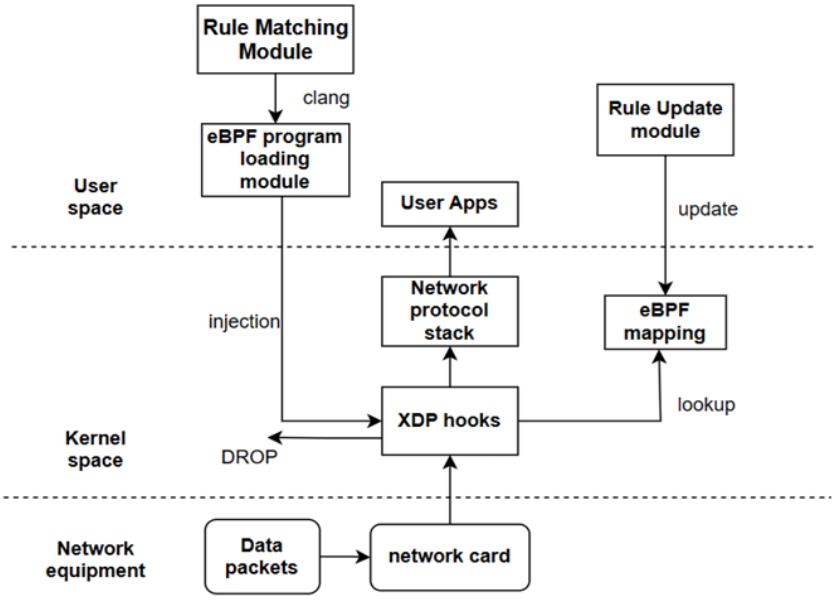


Fig. 1. Relationship between modules

process of packets flowing from the network card within the kernel to ultimately reach user space, without the need for a series of handling processes such as driver initialization, allocation of physical memory, registration of polling methods, handling of hard interrupts, and soft interrupts. Even in cases where the hardware or driver does not support XDP, the generic mode still achieves significant performance gains though it operates at a later stage in the packet processing pipeline.

XDP technology can process packets at the earliest stage when they arrive at the network interface [9], eliminating the need to pass the packets to later stages of the kernel network stack. This results in better performance compared to traditional network firewall packet filtering methods. When a network card receives a packet, it triggers the XDP program. XDP first extracts the packet header information, passing a context object struct xdp_md *ctx to the XDP program. The packet can be parsed through this context object struct xdp_md *ctx. The definition of struct xdp_md *ctx is shown in Table II.

TABLE II
XDP CONTEXT

| Member Variable | Data Type | Data Description |
|---|---|---|
| data | uint32 | Packet Start Pointer |
| data_end | uint32 | Packet End Pointer |
| data_meta | uint32 | Packet Metadata Exchange |

In the XDP hook[16], eBPF programs can quickly process packets, and after the eBPF program execution completes, the XDP program returns with the subsequent handling of the packet. The XDP commands and their functions are shown in Table III.

TABLE III
XDP COMMANDS

| Value | Command | Data Description |
|---|---|---|
| 0 | XDP_ABORTED | Drop Error Packet |
| 1 | XDP_DROP | Directly Discard Packet |
| 2 | XDP_PASS | Packet Forwarded to Kernel |

*2) **Packet Detection**:* Packet inspection involves parsing packets based on their frame structure using the context object struct xdp_md *ctx. This process begins with checking the validity of the packet to ensure that the received packet is compliant. Non-compliant packets can be directly discarded using XDP_DROP, avoiding additional system overhead caused by analyzing erroneous or malformed data packets.

First, the validity of the packet is determined by checking the offset between the data_end and data pointers extracted from the context object struct xdp_md *ctx. Then, key information from the packet is parsed using the offset values of the various header pointers such as *eth, iph, tcph, udph, and so on. The key algorithms for packet validity checking and information extraction are shown in Table IV.

*3) **Rule Matching**:* Rule Matching. Rule matching involves comparing the packet information extracted from the context object struct xdp_md *ctx with the filtering rule information retrieved from a map. This primarily includes matching MAC addresses, IP addresses, ports, and protocol types. Packets that conform to the filtering rules proceed to the next operation, while packets that do not meet the filtering criteria are passed to the kernel for further processing. The rule matching algorithm is shown in Table V.

TABLE IV
INSPECTION AND INFORMATION

**Algorithm Packet** Inspection and Information Extraction

**Input** : Data packets received by the network card
**Output** : Extracted data packets
01. packet ← NIC.receive() // Network card receives data packets
02. nh_type=parse_ethhdr(&nh,&data_end,&eth); // Judge if the packet is legitimate
03. if (eth + 1  data_end)
04. struct ethhdr *eth ← NULL; // Initialize Ethernet
05. header struct iphdr *iph ← NULL; //Initialize IP header
06. struct tcphdr *tcph ← NULL; //Initialize TCP header
07. struct udphdr *udph ← NULL; // Initialize UDP header
08. if (nh_type == bpf_htons(ETH_P_IP)) // Determine IPv4
09. u32 saddr ← iph saddr; // Obtain source IP address
10. u32 daddr ← iph daddr; // IP Obtain destination IP address
11. else
12. return xdp_pass; 13. if (nh_type == IPPROTO_TCP)
14. u32 sport ← udph source; // Source port
15. u32 dport ← udph dest; // Destination Port
16. else if(nh_type == IPPROTO_UDP)
17. u32 sport ← udph source; // Source port
18. u32 dport ← udph dest; // Destination port
19. else
10. return xdp_pass; // Continue operation
21. return Network Protocol Processing Result

TABLE V
RULE MATCHING

**Algorithm** Rule Matching

**Input** : Extracted packet data
**Output** : Network Protocol Processing Structure
01. match_rules_ipv4(&conn); //Matching Function
02. bpf_loop(MAX_RULES, match_rules_ipv4_loop, &ctx, 0);
03. struct rules_ipv4 *p_r ← bpf_map_lookup_elem (&rules_ipv4_map, &index); //Retrieve Rules from Map
04. if(!p_r) //Has Been Found
05. return -1.
06. ipv4_cidr_match(p_ctx conn saddr, p_r saddr, p_r saddr_mask).
//IP Matching
07. port_match(p_ctx conn-¿sport, p_r sport). //Port Matching
08. Return Network Protocol Processing Structure

### D. Rule Matching Module

The program loading module can inject the XDP program from an ELF file into the XDP hook point and bind it to the network interface card. When loading an ELF file using the program loading module, it is necessary to specify the network interface card for loading and the loading mode. Linux kernels default to providing a generic mode called XDP Generic, which allows the execution of XDP programs in kernels where the network card driver does not support the native XDP mode.

## III. EXPERIMENT

### A. Experimental Environment

The experiment involves setting up two machines, both running Ubuntu, and placing them within the same local area network. The first machine is configured with an eBPF-based firewall program, while the second machine utilizes the network performance testing tool pktgen to send packets to the first machine, where the eBPF firewall program is deployed. This setup is used to conduct both functional and performance testing. The experimental environment is outlined in Table VI.

TABLE VI
XDP COMMANDS

| Kernel Parameters | Sender Machine | Firewall Machine |
|---|---|---|
| Hard Disk | 60GB | 30GB |
| CPU | 4 | 4 |
| Memory | 4GB | 8GB |
| Operating System | Ubuntu 22.04 LTS | Ubuntu 23.04 LTS |
| Kernel Version | 6.5.0-27-generic | 6.5.0-26-generic |
| IP Address | 192.168.207.179 | 192.168.207.177 |

The host configured with the eBPF firewall program has a 4-core processor, 8GB of RAM, 30GB of hard disk capacity, and runs Ubuntu 23.04 LTS with a Linux kernel version of 6.5.0-26-generic. The packet-sending host is equipped with a 4-core processor, 4GB of RAM, 60GB of hard disk capacity, and uses Ubuntu 22.04 LTS as its operating system. The IP address of the host configured with the eBPF firewall program is 192.168.207.177, while the IP address of the packet-sending host is 192.168.207.179.

### B. Experiment One

For this experiment, the parameter selection includes precise filtering based on multiple dimensions, such as IP addresses, ports, and protocol types.

For this experiment, two machines were prepared. One machine with Ubuntu 23.04 LTS operating system was configured with both iptables firewall and the newly developed firewall for this experiment. The other machine, running Ubuntu 22.04 LTS, was equipped with the network performance testing tool pktgen[17], [18]. By continuously sending packets to the Ubuntu 23.04 LTS experimental machine, the number of dropped packets per second under both firewall configurations was recorded. The experimental results are shown in Table VII. As shown in Fig 2.

### C. Experiment Two

Performance testing was conducted on the firewall using the iperf tool. The firewall machine served as the iperf server, while the packet-sending machine functioned as the iperf client[19], [20], sending TCP and UDP packets to the firewall machine for a duration of 30 seconds. The CPU utilization rate was monitored on the firewall machine, and the experimental data is presented in Table VIII. As shown in Fig 3.

## IV. CONCLUSION

This paper proposes a high-performance network firewall technology based on XDP, introducing the overall design and implementation plan of a fast data path firewall, including packet parsing modules, rule matching program modules, rule

TABLE VII
PACKET LOSS PER SECOND

| Packet Length/bytes | Iptables/pps | XDP/pps | Improvement Rate % |
|---|---|---|---|
| 64 | 325784 | 467174 | 143.4 |
| 128 | 294795 | 387360 | 131.8 |
| 512 | 127485 | 148265 | 116.3 |
| 1024 | 106158 | 114120 | 107.5 |

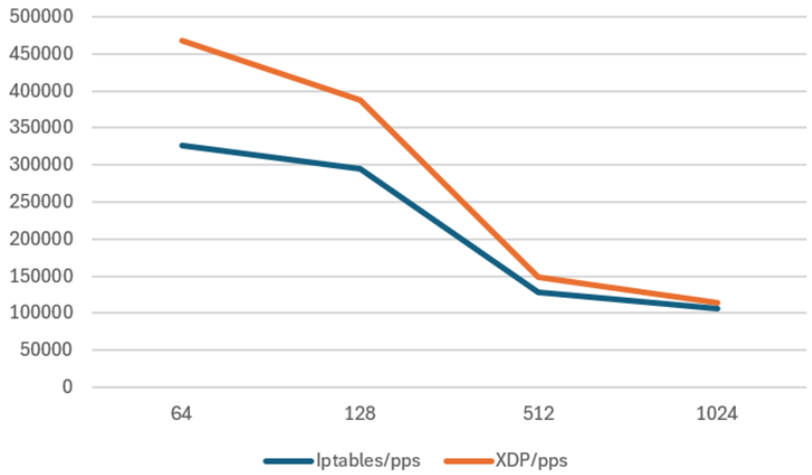


Fig. 2. Comparison Chart of Packet Loss

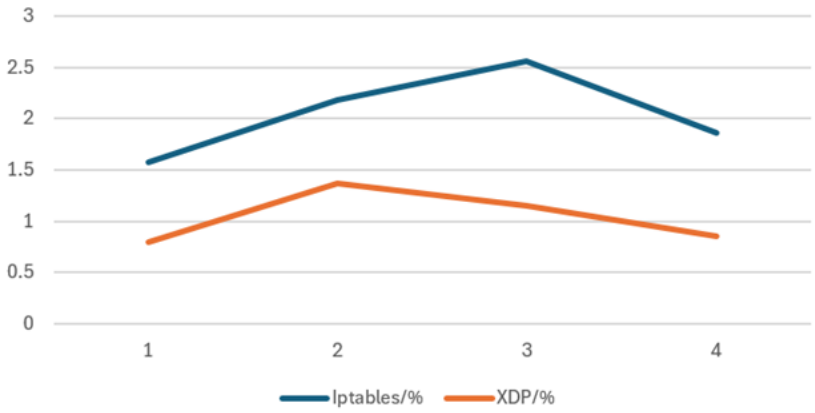


Fig. 3. CPU Utilization Comparison Chart

TABLE VIII
CPU UTILIZATION RATE

| CPU | Iptables % | XDP % | |
|---|---|---|---|
| 1 | 1.57 | 467174 | 0.79 |
| 2 | 2.18 | 387360 | 1.37 |
| 3 | 2.56 | 148265 | 1.15 |
| 4 | 1.86 | 114120 | 0.85 |

loading modules, and program loading modules. In addition, the functionality and technical implementation of each module are elaborated in detail. Finally, system performance testing and evaluation were conducted based on the overall design. Experiments have shown that compared with traditional firewall technologies, this technology possesses advantages such as high efficiency, low resource consumption, security, and flexibility, and offers a certain improvement in performance relative to traditional network packet filtering methods.

## REFERENCES


[1] Y. Zhaohui, "Cnnic releases the 45th statistical report on internet development in china," *Network Trusteeship for Civil-Military Integration*, 2020.

[2] M. Rash, *Linux Firewalls: Attack Detection and Response*. No Starch Press, 2007.

[3] M. Bertrone, S. Miano, F. Risso, and M. Tumolo, "Accelerating linux security with ebpf iptables," in *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, 2018, pp. 108–110.

[4] P. Likhar and R. S. Yadav, "Impacts of replace venerable iptables and embrace nftables in a new futuristic linux firewall framework," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2021, pp. 1735–1742.
[5] L. Ceragioli, P. Degano, and L. Galletta, "Are all firewall systems equally powerful?" in *Proceedings of the 14th ACM SIGSAC Workshop on Programming Languages and Analysis for Security*, 2019, pp. 1–17.
[6] Z. Trabelsi and S. Zeidan, "Multilevel early packet filtering technique based on traffic statistics and splay trees for firewall performance improvement," in *2012 IEEE International Conference on Communications (ICC)*. IEEE, 2012, pp. 1074–1078.
[7] M. Nacchia, F. Fruggiero, A. Lambiase, and K. Bruton, "A systematic mapping of the advancing use of machine learning techniques for predictive maintenance in the manufacturing sector," *Applied Sciences*, vol. 11, no. 6, p. 2546, 2021.
[8] T. Chomsiri, X. He, P. Nanda, and Z. Tan, "An improvement of tree-rule firewall for a large network: Supporting large rule size and low delay," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 178–184.
[9] Y. Cheng, W. Wang, J. Wang, and H. Wang, "Fpc: A new approach to firewall policies compression," *Tsinghua Science and Technology*, vol. 24, no. 1, pp. 65–76, 2018.
[10] R. Mohan, A. Yazidi, B. Feng, and J. Oommen, "On optimizing firewall performance in dynamic networks by invoking a novel swapping window–based paradigm," *International Journal of Communication Systems*, vol. 31, no. 15, p. e3773, 2018.
[11] S. Miteff and S. Hazelhurst, "Nfshunt: A linux firewall with openflow-enabled hardware bypass," IEEE, pp. 100–106, 2015.
[12] 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.
[13] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with linux ebpf," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1. IEEE, 2018, pp. 209–217.
[14] W. Findlay, A. Somayaji, and D. Barrera, "Bpfbox: Simple precise process confinement with ebpf," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020, pp. 91–103.
[15] 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.
[16] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with linux ebpf," in *2018 30th International Teletraffic Congress (ITC 30)*, vol. 1. IEEE, 2018, pp. 209–217.
[17] R. Olsson, "Pktgen the linux packet generator," in *Proceedings of the Linux Symposium, Ottawa, Canada*, vol. 2, 2005, pp. 11–24.
[18] D. Turull, P. Sjödin, and R. Olsson, "Pktgen: Measuring performance on high speed networks," *Computer communications*, vol. 82, pp. 39–48, 2016.
[19] C.-H. Hsu and U. Kremer, "Iperf: A framework for automatic construction of performance prediction models," in *Workshop on Profile and Feedback-Directed Compilation (PFDC), Paris, France*. Citeseer, 1998.
[20] A. Tirumala, "Iperf: The tcp/udp bandwidth measurement tool," *http://dast. nlanr. net/Projects/Iperf/*, 1999.