Project Description – "Software Networks" Course

Academic Year 2025

Introduction

As part of the "Software Networks" course, within the module dedicated to Software Defined Networks (SDN), we explored Linux Kernel Networking. In particular, we focused on two complementary approaches to packet processing and analysis: developing kernel modules attached to Netfilter, and programming with eBPF (Extended Berkeley Packet Filter) using the XDP (eXpress Data Path) hook.

The goal of the project is to provide students with hands-on experience in low-level system programming, with a particular emphasis on packet classification, counting, and filtering. The project is structured as a series of individual assignments with increasing difficulty, designed for 16 students.

Technical Background

1. Kernel Modules and Netfilter

Netfilter is the core framework for traffic manipulation in the Linux networking stack. Students were introduced to Netfilter hooks, with practical examples involving the pre-routing stage, where a kernel module inspects incoming packets. One demonstration involved selectively dropping ICMPv6 packets to illustrate how targeted filtering rules can be applied.

2. eBPF/XDP Programming

In parallel, students were also introduced to eBPF programming, focusing on XDP hooks that allow packet interception very early in the network stack, directly at the driver level. Similar filters to those implemented in kernel modules were recreated using eBPF, showing how packet inspection and classification can be handled in a programmable and efficient way.

Project Structure

Each student will be assigned one of 16 available projects, divided into two categories:
- 8 projects using Netfilter-based kernel modules
- 8 projects using eBPF programs with XDP hooks

Each project is organized into three tasks with increasing complexity:

Task 1 – Basic Level (~45 minutes)

- Develop a kernel module or an eBPF program to filter a specific type of packet (e.g., HTTP)
- Compile and load the module or attach the eBPF program
- Verify correct behavior using tools such as ping, tcpdump, ip, and dmesg

Task 2 – Intermediate Level (~2–3 hours)
- Implement packet classification based on IP header (e.g., IPv4 vs. IPv6, application-level protocol)
- Count packets per class and export statistics using procfs, bpffs, or trace pipe
- Write a basic script to gather and print the collected data

Task 3 – Advanced Level (~8–10 hours)
- Perform more advanced traffic analysis, e.g., classify packets based on domain name (DNS), apply thresholds and actions (throttling, drop, logging)
- Potential use of BPF maps, helper functions, and event tracing
- Full test coverage with realistic traffic scenarios

ONLY THE BASIC LEVEL IS REQUIRED, completing also the intermediate and advanced level is optional.

The completion of the project (at either basic, intermediate or advanced level) is needed before accessing the oral exam session.

Educational Objectives

The project aims to strengthen students' understanding of Linux network stack internals while fostering skills in:
- Kernel-space and user-space programming
- Advanced debugging and testing of kernel and eBPF code
- Incremental project development and agile methods
- Traffic analysis and dynamic policy enforcement

Evaluation and Oral Discussion

Project assessment will follow a collaborative and structured process:

- Peer Evaluation: Each student will be required to evaluate a small number of other students' projects. The evaluation will focus on functionality, clarity, and quality of the implementation.
- Meta-Evaluation: How a student evaluates others' projects will also be considered as part of their final assessment, rewarding fairness and technical accuracy.
- Demonstration Requirement: Students must be able to demonstrate that their project works, at either the basic, intermediate, or advanced level.
- Oral Exam: Each student will discuss their project during the oral examination session. The discussion will focus on the design choices, observed behavior, and lessons learned.

This evaluation model encourages clear communication, responsible peer feedback, and practical demonstration over theoretical perfection or deep code analysis.

Student Projects

Each project includes three progressive tasks: basic, intermediate, and advanced. Students must demonstrate their project at one of the three levels and participate in peer evaluation.

Netfilter Kernel Module Projects

M1 – IPv6 Packet Counter (Kernel module)

- - Basic: Count all IPv6 packets.
- - Intermediate: Count IPv6 packets by source subnet.
- - Advanced: Export real-time counters to user space and visualize them.

M2 – TCP vs UDP Classifier (Kernel module)

- - Basic: Identify and count TCP and UDP packets.
- - Intermediate: Maintain counters per port number.
- - Advanced: Implement threshold-based alerts in dmesg.

M3 – HTTP Packet Logger (Kernel module)

- - Basic: Detect HTTP packets on port 80.
- - Intermediate: Log HTTP packets' source IPs to kernel log.
- - Advanced: Drop HTTP packets based on IP blacklist loaded from file.

M4 – DNS Packet Inspector (Kernel module)

- - Basic: Match and count DNS packets (port 53).
- - Intermediate: Extract domain names and log them.
- - Advanced: Drop DNS requests to blacklisted domains (static list).

M5 – Localhost Traffic Monitor (Kernel module)

- - Basic: Detect traffic from/to 127.0.0.1 and ::1.
- - Intermediate: Track packet size distribution.
- - Advanced: Generate summary statistics on shutdown (e.g., via module exit).

M6 – Destination IP Classifier (Kernel module)

- - Basic: Identify packets by destination class (A/B/C or IPv6 prefix).
- - Intermediate: Count and log packets per class.
- - Advanced: Implement per-class traffic shaping (simulation only, no tc).

M7 – Fragmented Packet Filter (Kernel module)

- - Basic: Detect fragmented IP packets.
- - Intermediate: Drop first fragments only.
- - Advanced: Implement a timeout-based cache to detect fragment reassembly anomalies.

M8 – ARP Traffic Inspector (Kernel module)

- - Basic: Count ARP requests and replies.
- - Intermediate: Log MAC-IP mappings to the kernel log.
- - Advanced: Drop ARP packets from suspicious MACs (static list).

eBPF/XDP Projects

E1 – Application Protocol Classifier (eBPF)

- - Basic: Identify HTTP, DNS, and SSH traffic by port.
- - Intermediate: Maintain counters for each protocol.
- - Advanced: Add support for reconfiguring ports at runtime.

E2 – Source IP Whitelist (eBPF)

- - Basic: Drop all packets not coming from a fixed IP list.
- - Intermediate: Load whitelist from user space via bpftool.
- - Advanced: Add rate limiting for non-whitelisted IPs.

E3 – Packet Size Analyzer (eBPF)

- - Basic: Track average packet size in real-time.
- - Intermediate: Group statistics by protocol.
- - Advanced: Log anomalies in size distribution.

E4 – High-Volume Traffic Detector (eBPF)

- - Basic: Identify sources generating >100 packets/sec.
- - Intermediate: Maintain per-IP counters and report top talkers.
- - Advanced: Auto-drop top talkers exceeding threshold.

E5 – eBPF-Based Port Scanner Detector (eBPF)

- - Basic: Detect multiple TCP SYNs from same source to different ports.
- - Intermediate: Maintain a rolling window per source IP.
- - Advanced: Blacklist and drop future packets from scanner IPs.

E6 – DNS Response Latency Monitor (eBPF)

- - Basic: Timestamp outgoing DNS queries.
- - Intermediate: Match responses and compute round-trip time.
- - Advanced: Export latency histogram to bpffs.

E7 – Selective Packet Mirror (eBPF)

- - Basic: Mirror packets matching a port to a second interface (simulated).
- - Intermediate: Mirror based on IP + port match.
- - Advanced: Allow dynamic reconfiguration of mirroring criteria.

E8 – VLAN Tag Filter (eBPF)

- - Basic: Drop or accept packets based on VLAN tag (e.g., VLAN 100).
- - Intermediate: Maintain per-VLAN statistics.
- - Advanced: Dynamically change drop policy based on user-space control via maps.

Project Index

Project ID	Project Title
M1	IPv6 Packet Counter (Kernel module)
M2	TCP vs UDP Classifier (Kernel module)
M3	HTTP Packet Logger (Kernel module)
M4	DNS Packet Inspector (Kernel module)
M5	Localhost Traffic Monitor (Kernel module)
M6	Destination IP Classifier (Kernel module)
M7	Fragmented Packet Filter (Kernel module)
M8	ARP Traffic Inspector (Kernel module)
E1	Application Protocol Classifier (eBPF)
E2	Source IP Whitelist (eBPF)
E3	Packet Size Analyzer (eBPF)
E4	High-Volume Traffic Detector (eBPF)
E5	eBPF-Based Port Scanner Detector (eBPF)
E6	DNS Response Latency Monitor (eBPF)
E7	Selective Packet Mirror (eBPF)
E8	VLAN Tag Filter (eBPF)

M1 – Technical Details

Use nf_hook_ops to count packets in pre-routing. Use hash maps indexed by /64 prefix for IPv6. Consider exporting counters via /proc.

M2 – Technical Details

Inspect the transport protocol field in IP headers. Use arrays to count per-port traffic. Use printk for alerts when limits exceeded.

M3 – Technical Details

Match port 80 in TCP header. Use IP header offset to extract source IP. Drop packets with src IP from a statically defined list.

M4 – Technical Details

Inspect UDP port 53. Parse DNS headers to extract domain names. Build a static hash list of forbidden domains.

M5 – Technical Details

Match loopback IPs (127.0.0.1, ::1). Use histograms for packet sizes. Summarize stats in module_exit().

M6 – Technical Details

Extract dest IP and classify into class A/B/C or via IPv6 prefix. Simulate traffic shaping with delays/logging.

M7 – Technical Details

Use IP header flags and fragment offset to detect fragments. First fragment has offset 0. Build fragment cache in kernel memory.

M8 – Technical Details

Hook ARP packets (ETH_P_ARP). Extract sender MAC and IP. Maintain a static MAC blacklist in module.

E1 – Technical Details

Match known ports (80, 53, 22). Use BPF maps for per-protocol stats. Make ports configurable via user-space interaction.

E2 – Technical Details

Create a hash map with allowed source IPs. Drop packets if IP not in map. Add rate limiting via count maps.

E3 – Technical Details

Measure total bytes and divide by packets. Track min/max size. Use BPF_PERF_OUTPUT for anomaly logging.

E4 – Technical Details

Use per-IP counters in LRU hash map. Sample packet count every second. Auto-drop via XDP_DROP on threshold exceed.

E5 – Technical Details

Track TCP SYNs from same IP to many ports. Use ring buffer for window. Blacklist IPs after N attempts.

E6 – Technical Details

Record timestamp on outgoing DNS queries. On matching reply, compute delta. Histogram response times in BPF map.

E7 – Technical Details

Use BPF redirect to clone/mirror packets. Match on port or tuple. Mirror target can be set via map.

E8 – Technical Details

Parse VLAN tags using ethhdr and vlanhdr structs. Match VLAN ID. Use map to control drop/accept policy at runtime.

Task Difficulty Ranking

Rank	Task ID	Title
1	M1.basic	IPv6 Packet Counter (Kernel module)
2	M5.basic	Localhost Traffic Monitor (Kernel module)
3	M2.basic	TCP vs UDP Classifier (Kernel module)
4	M4.basic	DNS Packet Inspector (Kernel module)
5	E1.basic	Application Protocol Classifier (eBPF)
6	E3.basic	Packet Size Analyzer (eBPF)
7	E2.basic	Source IP Whitelist (eBPF)
8	M3.basic	HTTP Packet Logger (Kernel module)
9	M6.basic	Destination IP Classifier (Kernel module)
10	E4.basic	High-Volume Traffic Detector (eBPF)
11	E6.basic	DNS Response Latency Monitor (eBPF)
12	M8.basic	ARP Traffic Inspector (Kernel module)
13	E5.basic	Port Scanner Detector (eBPF)
14	E7.basic	Selective Packet Mirror (eBPF)
15	M7.basic	Fragmented Packet Filter (Kernel module)
16	E8.basic	VLAN Tag Filter (eBPF)
17	M1.intermediate	IPv6 Packet Counter (Kernel module)
18	M5.intermediate	Localhost Traffic Monitor (Kernel module)
19	M2.intermediate	TCP vs UDP Classifier (Kernel module)
20	M4.intermediate	DNS Packet Inspector (Kernel module)
21	E1.intermediate	Application Protocol Classifier (eBPF)
22	E3.intermediate	Packet Size Analyzer (eBPF)
23	E2.intermediate	Source IP Whitelist (eBPF)
24	M3.intermediate	HTTP Packet Logger (Kernel module)
25	M6.intermediate	Destination IP Classifier (Kernel module)
26	E4.intermediate	High-Volume Traffic Detector (eBPF)
27	E6.intermediate	DNS Response Latency Monitor (eBPF)
28	M8.intermediate	ARP Traffic Inspector (Kernel module)
29	E5.intermediate	Port Scanner Detector (eBPF)
30	E7.intermediate	Selective Packet Mirror (eBPF)
31	M7.intermediate	Fragmented Packet Filter (Kernel module)
32	E8.intermediate	VLAN Tag Filter (eBPF)
33	M1.advanced	IPv6 Packet Counter (Kernel module)
34	M5.advanced	Localhost Traffic Monitor (Kernel module)
35	M2.advanced	TCP vs UDP Classifier (Kernel module)
36	M4.advanced	DNS Packet Inspector (Kernel module)
37	E1.advanced	Application Protocol Classifier (eBPF)
38	E3.advanced	Packet Size Analyzer (eBPF)
39	E2.advanced	Source IP Whitelist (eBPF)
40	M3.advanced	HTTP Packet Logger (Kernel module)
41	M6.advanced	Destination IP Classifier (Kernel module)
42	E4.advanced	High-Volume Traffic Detector (eBPF)
43	E6.advanced	DNS Response Latency Monitor (eBPF)
44	M8.advanced	ARP Traffic Inspector (Kernel module)
45	E5.advanced	Port Scanner Detector (eBPF)

46	E7.advanced	Selective Packet Mirror (eBPF)
47	M7.advanced	Fragmented Packet Filter (Kernel module)
48	E8.advanced	VLAN Tag Filter (eBPF)

Per-Project Difficulty Summary

Project (ID + Title)	Basic Task Rank	Intermediate Task Rank	Advanced Task Rank
M1 – IPv6 Packet Counter	1	1	1
M5 – Localhost Traffic Monitor	2	2	2
E1 – Application Protocol Classifier	3	4	4
M2 – TCP vs UDP Classifier	4	5	5
E3 – Packet Size Analyzer	5	3	3
M6 – Destination IP Classifier	6	8	6
E2 – Source IP Whitelist	7	6	8
M4 – DNS Packet Inspector	8	7	7
M3 – HTTP Packet Logger	9	9	9
M8 – ARP Traffic Inspector	10	10	10
E6 – DNS Response Latency Monitor	11	12	12
E4 – High-Volume Traffic Detector	12	11	11
E7 – Selective Packet Mirror	13	13	13
E5 – Port Scanner Detector	14	14	14
M7 – Fragmented Packet Filter	15	15	15
E8 – VLAN Tag Filter	16	16	16