DNS resolver using AF_XDP socket
Midsem Report - Team 13

Guntas Singh Saran*& Hrriday V. Ruparel †& Kishan Ved‡& Pranav Patil§
Department of Computer Science and Engineering
Indian Institute of Technology Gandhinagar
Palaj, GJ 382355, India

Prof. Sameer Kulkarni
Department of Computer Science and Engineering
Indian Institute of Technology Gandhinagar
Palaj, GJ 382355, India

1 Introduction

This midsem report outlines our team's progress on building a high-performance DNS resolver using AF_XDP sockets. As of March 26, 2025, we have focused on understanding the foundational concepts necessary for our implementation, including the Linux kernel networking stack, XDP (eXpress Data Path), and host network stack overheads. We have reviewed our initial project proposal, a detailed report on packet processing in the Linux kernel, and a research paper on network stack overheads at 100 Gbps bandwidths. Additionally, we have met with our Teaching Assistant, Naveen Tiwari, twice, who has provided valuable resources to guide our efforts. This report summarizes our progress, the content we have covered, and our plan moving forward, including the next step suggested by our TA: writing a BPF program for packet filtering.

2 Progress Summary

2.1 Review of Project Proposal

Our proposal outlined the goal of developing a DNS resolver using AF_XDP sockets to achieve high-throughput and low-latency DNS query processing by bypassing the Linux kernel networking stack. Key objectives include filtering DNS packets (UDP port 53) with XDP, processing queries in user space, optimizing with zero-copy techniques, and benchmarking against traditional resolvers. This review reaffirmed our focus on leveraging XDP for packet filtering at the NIC level and AF_XDP for efficient user-space processing.

2.2 The Journey of a Network Packet Through the Linux Kernel

This report provided a detailed breakdown of the Linux kernel's networking stack, covering both ingress and egress paths (Stephan et al. (1)). Key takeaways include:

- **Ingress Path**: Packets move from the NIC (via DMA) through Ethernet, IP, and transport layers (TCP/UDP) to the socket layer, involving functions like `ip_rcv()`, `tcp_v4_rcv()`, and `udp_rcv()`. This highlighted the overheads of kernel processing, such as `sk_buff` allocation and context switching, which our project aims to bypass using XDP.

*Guntas Singh Saran - 22110089 - guntassingh.saran@iitgn.ac.in
†Hrriday V. Ruparel 22110099 - hrriday.ruparel@iitgn.ac.in
‡Kishan Ved - 22110122 - kishan.ved@iitgn.ac.in
§Pranav Patil - 22110199 - pranav.patil@iitgn.ac.in
*Order based on Roll Numbers Only.

- **Egress Path**: The reverse process, from application to NIC, involves functions like `tcp_sendmsg()` and `udp_sendmsg()`, emphasizing the role of kernel buffers and DMA. This reinforced the importance of zero-copy techniques to minimize CPU intervention.
- **Relevance**: Understanding this traditional pipeline clarified the bottlenecks (e.g., data copying, interrupt handling) that XDP and AF_XDP can mitigate by handling packets at the NIC level and in user space.

2.3 Understanding Host Network Stack Overheads

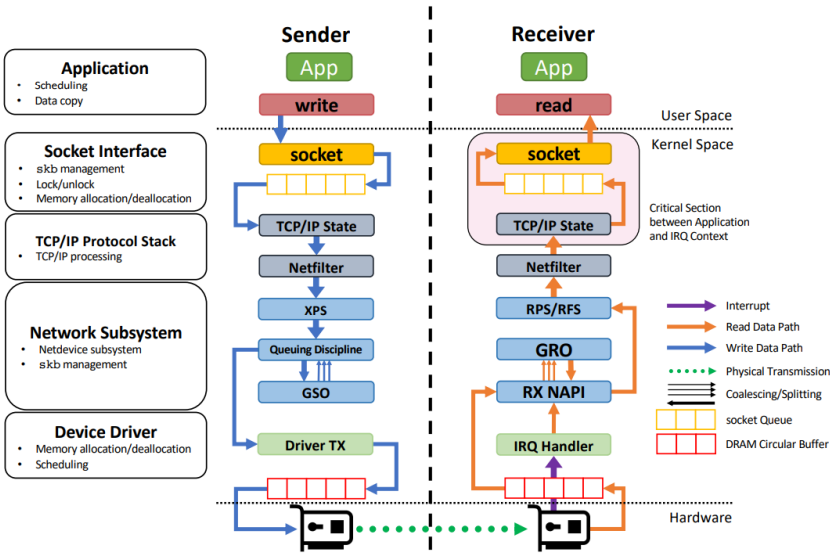

Figure 1: Packet Journey through the Linux Network Stack

This research paper by Cai et al. (2) analyzed Linux network stack performance at 100 Gbps, revealing modern bottlenecks:

- **CPU Bottlenecks**: A single core struggles to process packets at line rate, with data copying from kernel to user space being a primary bottleneck—directly relevant to our zero-copy goal.
- **Cache Issues**: Increasing bandwidth-delay products outpace L3 cache sizes, causing high cache miss rates (e.g., 49% for a single flow) despite Direct Cache Access (DCA) as offered by DDIO (Data Direct I/O) technology. This suggests that our XDP-based approach must optimize memory access.
- **Multi-Flow Challenges**: Resource contention (e.g., L3 cache pollution) reduces throughput-per-core, emphasizing the need for efficient packet filtering to isolate DNS traffic.
- **Future Directions**: The paper advocates zero-copy mechanisms (e.g., AF_XDP) and dynamic host resource orchestration, aligning with our project's optimization objectives.

2.4 Meetings with TA

We met with Naveen Tiwari twice. In these sessions, he shared the above resources to deepen our understanding and suggested focusing on packet filtering as the next step. His guidance has been instrumental in structuring our learning phase.

3 Current Status

As of now, we have completed the initial learning phase, gaining a solid grasp of:

- The Linux kernel's packet processing pipeline and its inefficiencies.
- XDP's role in bypassing the kernel stack by filtering packets at the NIC.
- AF_XDP's potential for zero-copy user-space processing.
- Performance bottlenecks (e.g., data copying, cache misses) in high-bandwidth scenarios.

We have not yet begun implementation but are well-prepared to transition to coding, starting with the TA's suggested next step.

4 Future Plan

Our future work aligns with the proposed workflow, with each phase tentatively spanning two weeks. Below is the plan ahead:

4.1 Packet Filtering with XDP (This week)

- **Objective**: Write a BPF (Berkeley Packet Filter) program to capture DNS packets (UDP port 53) at the NIC, filter them, and parse headers to extract query details.
- **Tasks**:
  - Study BPF and XDP programming basics (e.g., using `libbpf`).
  - Develop a program to identify UDP packets on port 53 and redirect them to an AF_XDP socket.
  - Parse DNS headers to extract domain names, adhering to RFC 1035 (3) specifications.
- **Deliverable**: A working XDP program integrated with our NIC setup, tested in a Linux VM.

4.2 User-Space Packet Processing (Apr 1 - Apr 7)

- **Objective**: Implement a user-space application to process filtered DNS queries and construct responses.
- **Tasks**:
  - Bind an AF_XDP socket to the NIC's RX queue.
  - Use `nslookup` or a custom resolver library to forward queries to a recursive resolver (e.g., Google Public DNS).
  - Build and send DNS response packets via the TX queue.
- **Deliverable**: A functional user-space resolver handling basic DNS queries.

4.3 Performance Optimization (Apr 8 - Apr 14)

- **Objective**: Enhance efficiency using zero-copy techniques.
- **Tasks**:
  - Enable zero-copy mode in AF_XDP using DMA.

4.4 Benchmarking (Apr 15 - Apr 19)

- **Objective**: Compare our resolver's performance against traditional resolvers.
- **Tasks**:
  - Use `dnsperf` to measure throughput, latency, and CPU usage.
  - Analyze results against a baseline (e.g., BIND or Unbound).
- **Deliverable**: A comprehensive performance report.

5 Acknowledgement

We thank TA Naveen Tiwari for his guidance and for providing critical resources that shaped our understanding. We plan to continue weekly coordination with him to ensure steady progress.

References

[1] A. Stephan and L. Wüstrich, "The path of a packet through the linux kernel,"

[2] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal, "Understanding host network stack overheads," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, (New York, NY, USA), p. 65–77, Association for Computing Machinery, 2021.

[3] "Domain names - implementation and specification." RFC 1035, Nov. 1987.

[4] Linux Kernel Documentation, "AF_XDP Official Documentation." https://www.kernel.org/doc/html/latest/networking/af_xdp.html.

[5] H. P. N. Programming, "Recapitulating af_xdp." https://medium.com/high-performance-network-programming/recapitulating-af-xdp-ef6c1ebead8.

[6] XDP Project, "The eXpress Data Path (XDP) inside the Linux kernel." https://github.com/xdp-project.

[7] DNS-OARC, "dnsperf: Gather accurate latency and throughput metrics for Domain Name Service (DNS)." https://github.com/DNS-OARC/dnsperf.

[8] Libvirt Project, "libvirt: An Open-Source API, Daemon, and Management Tool for Platform Virtualization." https://libvirt.org/.