Network Computing Final Project Report

Luca Donato, 10660050

June 10, 2025

1. Introduction

The objective of this project is to analyze, debug, enhance, and validate a simplified TCP connection tracker written in eBPF/XDP. Leveraging the eBPF framework allows for high-performance, stateful packet processing directly in the kernel's networking stack, a critical capability for modern security and monitoring applications like stateful firewalls, DDoS mitigation, and load balancing.

2. Analysis of the Initial Implementation

The initial implementation was affected by several design flaws that rendered it non-functional for its intended purpose. The primary issues identified were:

- **Incomplete State Machine:** The tracker did not implement the full TCP state diagram. Critical states for connection teardown, such as `CLOSE_WAIT` and `CLOSING`, were missing, preventing the correct handling of standard termination sequences.
- **Flawed State Transition and Conditional Logic:** The logic for advancing the state was unreliable due to faulty conditional checks within `if` statements. This included incorrect validation of TCP sequence/acknowledgment numbers and oversimplified logic that failed to distinguish between different packet types (e.g., a pure ACK versus a FIN/ACK). As a result, the tracker would often desynchronize from the actual TCP flow, causing it to incorrectly drop valid packets and thus preventing any end-to-end communication tests from completing successfully.
- **Poor Resource Management and State-Leakage:** The program failed to properly handle events that terminate a connection's lifecycle. It lacked logic to enforce TTL expiration and did not clean up state upon receiving RST packets. This critical omission resulted in a severe state-leakage problem, where the BPF map would inevitably fill with stale entries, exhausting its capacity.

3. Implemented Fixes and Improvements

The tracker was significantly refactored to address all identified issues. The key improvements are summarized below:

- **Complete TCP State Machine:** The state machine was rebuilt to be compliant with the TCP standard. Missing states were added, and the logic was refined to correctly handle connection teardown, including the complete FIN packet sequence.
- **Corrected Conditional Logic:** All conditional checks within `if` statements were fixed. This included implementing precise validation for TCP sequence/acknowledgment numbers

and clarifying flag comparisons to ensure session integrity.

- **Added RST and TTL Handling:** Logic was introduced to prevent state-leakage. The tracker now correctly processes RST packets by deleting the associated connection state from the BPF map. It also enforces TTLs reactively: an expired entry is removed from the map only if a new packet for that specific flow arrives.
- **UDP Flow Tracking:** As an enhancement, support for stateless UDP flows was added. The logic creates a `NEW` entry on the first packet and transitions it to `ESTABLISHED` on subsequent packets, using timeouts to track active flows.

4. Validation Methodology

The tracker's correctness was verified using a suite of tools to test specific scenarios and edge cases. The test environment was configured as described in the project guide.

- `iperf3`: Validated the standard TCP lifecycle, including the 3-way handshake, the `ESTABLISHED` state, and connection teardown with piggybacked FINs.
- `socat`: Employed to test the standard 4-way handshake closing sequence and to verify the optional UDP flow tracking functionality.
- `hping3`: Used to inject RST packets, confirming that the tracker correctly and immediately deletes the corresponding connection state.
- `Python/Scapy`: A custom script was used to test edge cases like the simultaneous close. This required adding a specific `iptables` rule to drop outgoing RST packets, preventing the local kernel's network stack from interfering with the custom packet sequence.

5. Key Future Improvement

While the current implementation is functional, one critical improvement is necessary for its long-term stability in a production environment: proactive garbage collection of expired flows.

- **Implementing a Userspace Garbage Collector:** At present, the eBPF program only removes an expired connection entry when a new packet for that specific flow arrives. This means that flows that go permanently idle are never cleaned up, inevitably leading to state-leakage and the exhaustion of the BPF map's capacity.

  A robust solution is to implement a userspace daemon. This process would periodically scan the entire `connections` map, check the TTL of each entry, and proactively delete any that have expired. This approach guarantees that the map remains clean, offloads the overhead of garbage collection from the kernel's fast path, and ensures the long-term stability of the tracker.

6. Conclusion

This project addressed the required objectives by debugging and extending a provided eBPF connection tracker, resolving its critical flaws. The final implementation now correctly tracks the TCP connection lifecycle, manages resources through RST and TTL handling, and extends its capabilities with UDP flow tracking. The validation process confirmed that the tracker functions as expected under the tested network scenarios, resulting in a functional implementation of an eBPF/XDP connection tracker.