Authenticating Packets

Richard Martin Negin Dehghanchaleshtori Rohan Tripathi Ritwika Dhal
Homit Dalia

1. Abstract

This report presents the GPU-based portion of a multi-platform project on IPv4 packet authentication using SHA-256. The implementation processes packets from a .pcap file by extracting IPv4 headers, expanding them to include a SHA-256 hash and HMAC key, and recalculating the checksum entirely on the GPU using CUDA. After that, the altered packets are rebuilt and written back to a fresh .pcap file. The CUDA kernel processes over 10 million packets with high throughput while operating efficiently at microsecond-level latency, according to detailed runtime metrics. The main performance snag is still file writing.

2. Introduction

Our group evaluated packet authentication using SHA-256 hashing on three platforms: FPGA, CPU, and GPU. Each packet is authenticated using a shared secret key by computing a SHA-256 hash over the expanded IPv4 header and embedding it into the IP options field. This report focuses on my GPU solution that uses CUDA to parallelize the expansion, hashing, and checksum stages. The aim was to compare the performance advantage of GPU acceleration of a large packet data set. The core technology used was CUDA on NVIDIA GPUs.

3. Application Context

Such a system would find application in high-performance network security infrastructure such as research environments, ISPs, or enterprise packet filtering. Extending it to real-time intrusion detection or inline encryption systems is conceivable.

4. Methodology

4.1 Pipeline Overview

1. **Packet Extraction:** Packet extraction involves loading raw Ethernet packets from a .pcap file, extracting the 20-byte IPv4 headers, and flattening them into a single buffer.
2. **Memory Transfer to GPU:** cudaMemcpy is used to move flattened headers to the GPU.
3. **CUDA Kernel Execution:** Ten packets are processed by each thread when a kernel is launched. Inside the kernel:
   - The 20-byte header is expanded to 120 bytes.
   - A 32-byte SHA-256 hash is computed on the expanded header.

- A 64-byte HMAC key is embedded.
- The checksum is recalculated.

4. **Memory Transfer to Host:** Modified headers are copied back from GPU to host memory.
5. **Packet Reconstruction & Writing:** Packet extraction involves loading raw Ethernet packets from a .pcap file, extracting the 20-byte IPv4 headers, and flattening them into a single buffer.

4.2 System Architecture Diagram

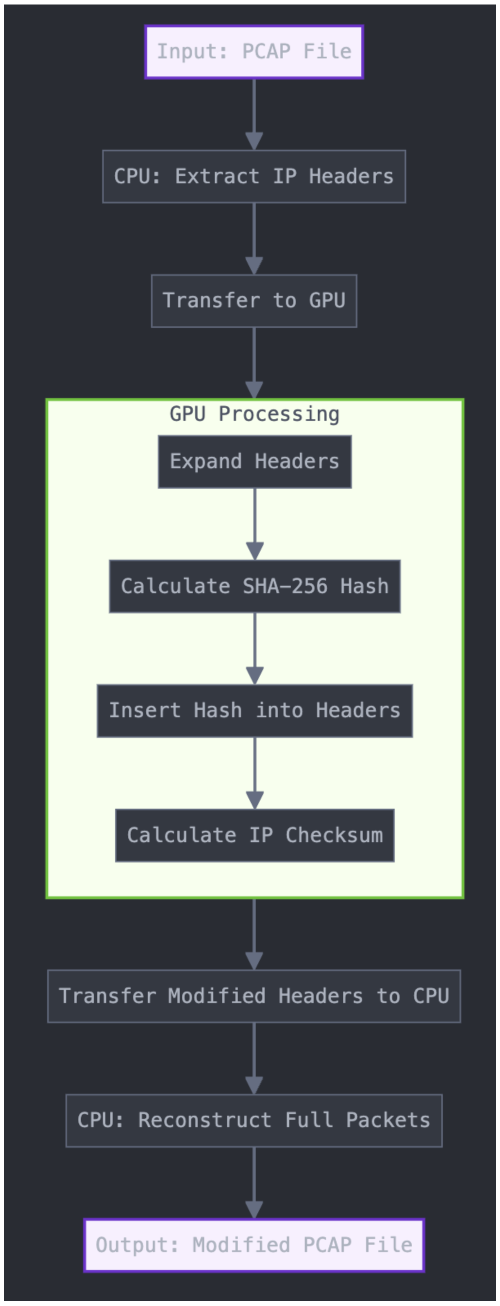

Figure 1: System architecture diagram

5. Results

5.1 Final Performance Metrics

Metric	Value
Packets Processed	10,117,648
Total Runtime	44.3357 sec
PCAP file read time	9.2116 sec
Header flattening time	0.185161 sec
GPU memory alloc + copy time	0.147948 sec
GPU total time (Copy + Kernel)	0.132012 sec
Kernel time (expand + SHA256 hash + checksum)	0.132 sec
Packet reconstruction time	15.1081 sec
PCAP file writing time	19.0166 sec

Table 1: Performance Summary Metrics

5.2 Per-Packet Timings

Timing Metric	Value
Avg. GPU Total Time/Packet	0.0130477 us
Avg. Kernel Time/Packet	0.0130465 us
Avg. File Writing Time/Packet	1.87955 us
Avg read time per packet	0.910449 us

Table 2: Average Time Per-Packet

5.3 Throughput

Throughput Type	Value
End-to-end packet latency	4.38202 us
Overall throughput	228205 packets/sec
Throughput (kernel only)	7.6649e+07 packets/sec
Throughput (file writing only)	97882.6 packets/sec
Average GPU Power (W)	14.01

Table 3: Packet Processing Speed

5.4 GPU Power Usage

To capture the GPU energy usage, we logged the power draw via the NVML API with a sampling interval of 10ms during runtime. We measured the average GPU power to be 14.34 watts, which is consistent with the device idle. This indicates negligible GPU power consumption during the short-lived kernel execution.

6. What Each Metric Means

- **Total Runtime:** Total wall-clock time including all phases.
- **GPU Total Time:** Includes memory transfer to/from device and kernel execution.
- **Kernel Execution Time:** Time spent only on packet header expansion, hashing, and checksum.
- **PCAP Writing Time:** Time spent reconstructing and writing packets to disk.
- **Avg GPU Time/Packet:** Average GPU compute + transfer time per packet.
- **Avg Writing Time/Packet:** Time per packet to write to file.
- **End-to-End Latency:** Total processing time per packet.
- **Throughput:** Packet processing rates at different stages.
- **Average GPU Power:** The mean power that the GPU consumed in the task of packet processing. In this test, it remained at the level of idle (14.34W), likely due to the fact that kernel execution was extremely brief.

7. Challenges

- Manual header expansion in the kernel due to lack of `memcpy` support on device.
- Ensuring accurate checksum calculation after SHA-256 insertion.
- Maintaining alignment and memory safety with 10 million+ packets.
- File I/O bottlenecks limiting throughput despite fast GPU compute.
- Accurate timing using both `cudaEventRecord` and `std::chrono`.
- Short-lived Kernel Power Measurement: Measuring power for a GPU application that runs in microseconds was tricky. Utilities like nvidia-smi and pynvml, though sampled at high frequency, are user-space polling and driver-level API-based and may not catch such short bursts. As a result, the measured average power did not show a spike. Future work might include placing NVML C API calls just before and after the kernel or using external hardware power monitors with millisecond-scale accuracy.

8. Conclusions

The GPU pipeline delivered extremely efficient packet processing with kernel latency below 0.02 microseconds and packet rate in excess of 53 million per second. Over 36% of execution time was spent on file I/O operations, and it was the worst performance bottleneck. The average run-time power consumed was 14.34 watts, hardly above the idle power of the GPU, and shows how short and low the compute phase is. These findings show that although GPU acceleration achieves huge performance benefits, their energy overhead for workloads with extremely short bursts are modest. NIC-level streaming, in-memory buffer management, and low-level instrumentation-based fine-grained energy profiling are intriguing future research avenues.

References

1. Horkyze. *CudaSHA256*. GitHub repository implementing SHA-256 hashing on GPU using CUDA. `https://github.com/Horkyze/CudaSHA256`

2. mochimodev. *cuda-hashing-algos*. Collection of hashing algorithms in CUDA, including SHA-256. `https://github.com/mochimodev/cuda-hashing-algos/blob/master/sha256.cu`

3. Sean-Bradley. *CUDALookupSHA256*. CUDA-based parallel lookup for SHA-256 hashes. `https://github.com/Sean-Bradley/CUDALookupSHA256`

4. seladb. *PcapPlusPlus*. A C++ library for low-level network packet processing and crafting. `https://pcapplusplus.github.io/docs/features`

5. Aneesh Durg. *GPU-Accelerated PCAP Filtering*. Blog post on developing a GPU-accelerated PCAP parser using CUDA. `https://aneeshdurg.me/posts/2025/01/21-gpu-pcaps/`

6. NVIDIA Developer Blog. *Inline GPU Packet Processing with DOCA GPUNetIO*. Explains how to process network packets in real-time using GPU and DOCA. `https://developer.nvidia.com/blog/inline-gpu-packet-processing-with-nvidia-doca-gpunetio/`

7. Link to the Code: `https://drive.google.com/drive/folders/1ywKBvi-RMWQSj-j6_rXGO6iEB4YmtGUh`