Scalable, Hardware-Accelerated Network Analytics

Oliver Michel (student)
University of Colorado Boulder
oliver.michel@colorado.edu

John Sonchack (student)
University of Pennsylvania
jsonch@seas.upenn.edu

Adam J. Aviv
United States Naval Academy
aviv@usna.edu

Eric Keller
University of Colorado Boulder
eric.keller@colorado.edu

1 Introduction

Monitoring is a crucial part of network operations, especially as networks grow larger and more complex. Motivated by this, many recent works have introduced switch hardware and software to export fine-grained traffic and performance data [3], *e.g.*, per-packet features such as timestamps or queue depths. While the fine granularity gives operators unprecedented visibility into their networks, processing and extracting useful information from the data, *i.e.*, analytics, remains a challenge.

Currently, the networking community lacks an efficient, scalable, and easily deployable platform for custom analytics of fine-grained monitoring data. Commercial systems, such as [1], are tightly integrated with specific data plane implementations, have limited extensibility, and require heavyweight deployments. Solutions built from general purpose platforms for large scale data processing, *e.g.*, Spark, require complex deployment and careful optimization. In addition to the clear drawbacks for operators, these approaches also make things more challenging for developers because they do not facilitate modularity or code re-use. This is particularly important for network analytics, since many applications are likely to perform similar types of processing, *e.g.*, grouping packet timestamps by a key.

Toward the goal of high-performance per-packet analytics in a practical, easy to operate environment, we propose an architecture and streaming framework in which monitoring applications can be built using reusable components that perform computations. Analytics applications in this framework are C++ applications which link against a library that provides processing primitives, as well as a runtime environment that provides high-performance, parallel computation. Through experiments, we identified that these applications are primarily I/O-bound due to frequent memory operations on small chunks of data, *i.e.*, per-packet data. To further improve processing performance, we propose selective hardware offload of analytics functions, that can be aggregated (while preserving per-packet information) before being processed in software. To implement this pattern, we leverage programmable *Smart NICs*, that can perform initial processing at high data rates. Through initial experiments, we showed that both our software streaming framework, as well as our hardware offload mechanism, scale almost linearly with core count to high traffic rates. This makes per-packet analytics at packet rates commonly observed in ISP backbones or data center networks feasible. Our overall system architecture is depicted in Figure 1 and consists of two components: (1) an initial processing step in hardware, and (2) a scalable and modular streaming analytics framework in software.

2 Practical Fine-Grained Performance Analytics

We implemented a streaming analytics framework for network performance monitoring based on a C++ Parallel Computation Library [2]. This framework serves two main purposes. First, it provides means to easily parallelize complex applications through a stream processing abstraction consisting of computation kernels and self-tuning queues connecting these kernels. This allows high throughput without requiring manual optimizations. Second, we provide a library of standard kernels that implement common functionality, like *map*, *group-by*, *filter*, and *join* to perform computations on a packet or flow data stream. Applications can be written in C++ and compiled to lightweight statically compiled binaries that are easy to ship or deploy. For example, a simple filter kernel can be defined like this:

```
Filter<Flow> filter_tcp([](auto& flow) {
  return flow.key().ip_proto == 6; });
```

Applications can then be composed of a simple topology of such kernels:

```
raft::map m;
m += flow_import >> filter_tcp
  >> tcp_out_of_seq >> printer;
m.exe();
```

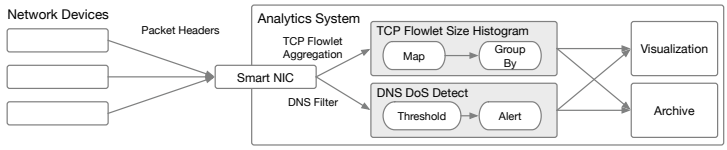

Figure 1: System architecture showing preprocessing steps and two concurrently running analytics applications

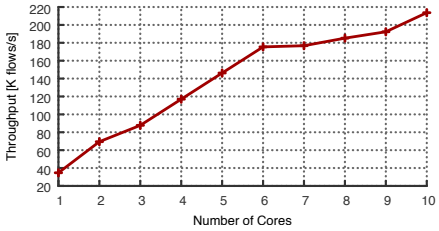

Figure 2: Streaming Library Performance

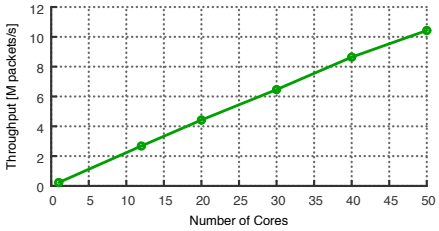

Figure 3: Smart NIC Performance

Through experiments, we have shown that these compute kernels can scale almost linearly with CPU core count. Using proper load-balancing schemes to parallelize computation, we can thus achieve high processing rates. Our example application inspected every single packet of flow records doing simple counter operations. The flow records were synthesized from a CAIDA Internet backbone trace and had an average packet count of 27 per flow. Figure 2 shows the number of flows per second that our system can process in this manner on a 24 Core Intel Xeon E5-2620 3rd Generation CPU. Our system and its dependencies can be easily packaged in a 600MB Docker image, that can easily be deployed and run in almost any environment.

3 Hardware Acceleration

An important trend in networking is the commoditization of programmable hardware accelerators for packet processing, *e.g.*, Barefoot Tofino switches or Netronome NFP Smart NICs, which are both priced comparably to their commodity non-programmable alternatives. We believe that these devices can significantly reduce the workload of the analytics software and, in a NIC form factor, are well-suited to deployment in analytics servers. We envision an optional preprocessing stage in the hardware for dynamic, runtime-configurable filtering and aggregation, that can selectively pass complete flow records, single packet records, or some intermediate flow representation to the software processing system.

To explore the potential, our prototype includes a module that uses a NFP-4000 to aggregate per-packet records into flowlet records, using a cache-based approach similar to Marple [3]. The module reduces the workload of the stream processor by lowering the rate of events it must process, which our initial evaluation showed was a bottleneck for most streaming analytics software. The prototype implementation was written in P4 and showed near-linear scaling behavior per NFP-4000 core and overall high packet throughput (Figure 3). In a 10 Gb/s core Internet router trace, a 1 MB cache in the NIC reduced the software's event rate by a factor of 10 [1].

4 Conclusion

In this paper, we presented early work on a novel approach to high-performance network monitoring using a combination of highly optimized parallel computation in software and hardware offload as a mechanism to preprocess network data at high rates. Furthermore, we presented a programming model for monitoring applications based on reusable and composable operators that enable network operators to write high performing network analytics applications easily and to deploy these applications on general purpose hardware. These two components provide a modular, scalable, and, most importantly, lightweight system that can scale to Internet backbone or data center traffic rates.

References

[1] Cisco Tetration Analytics. https://www.cisco.com/c/en/us/products/data-center-analytics

[2] Beard, Jonathan et.al. 2016. RaftLib: A C++ Template Library for High Performance Stream Parallel Processing. In Intl. Journal of HPC Applications. 2016.

[3] Srinivas Narayana et.al. 2017. Language-Directed Hardware Design for Network Performance Monitoring. In Proceedings of SIGCOMM 17.

[1] CAIDA 02/2015 Chicago Direction A: https://www.caida.org/data/passive/trace_stats/