TurboFlow: Accelerating Flow Collection on Commodity Switches

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

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

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

Jonathan M. Smith
jms@cis.upenn.edu
University of Pennsylvania

1 Introduction

Many systems use transport layer flow records (*i.e.*, of TCP or UDP traffic flows) to profile, debug, secure and optimize computer networks [4]. Flow record generators support these systems by monitoring live traffic to produce flow records. Today, many flow record generators are implemented as software running on commodity servers [7]. It is expensive and time consuming to scale these systems to large or high speed networks because of the limited throughput and high power consumption of commodity servers.

A more scalable approach is to do some of the monitoring with *Programmable Forwarding Engines (PFEs)* [8, 3] that can process packets at high rates according to functions defined as software [2] and are cost and power effective. Commodity switches and line cards with PFEs are beginning to come to market [6, 1]. Flow record generators would ideally deploy stateful routines to these devices that gather information about traffic flows, allowing them to scale and produce feature rich records of high throughput traffic from many vantage points. Such visibility would enable novel systems for improved network monitoring, debugging, and security [5], and allow us to rethink the role of flow records in networking.

However, that potential can only be realized if the PFE component of a flow record generator maps efficiently to the complex architectures of actual hardware. PFEs are specialized for functions that require minimal amounts of state. As a result, there are many architectural restrictions that *have not been considered by previous work* but *come to bear on highly stateful functions that collect information about traffic flows in the PFE.*

- **Limited Memory Accesses**: PFEs operating at line rate only have time for a small number of memory operations per packet. A routine must limit how frequently it modifies persistent state, such as flow information, or it may reduce throughput or fail to compile [8].
- **Parallel Memory Banks:** On many PFEs, the only way to perform multiple memory operations while processing an individual packet is to use parallel execution units that are wired to *separate* memory banks [3, 8]. A routine must be designed so that any persistent state can be partitioned efficiently across these memory banks and execution units.
- **Parallel Processors:** Other designs use shared memory banks that are accessed by many threads processing different packets simultaneously [6]. A routine must be designed so that these threads can efficiently synchronize access to any shared persistent state.
- **Code Space:** PFE routines compile to instructions that are placed into small caches or physical pipelines of execution units. A function must be simple to fit into these resources.

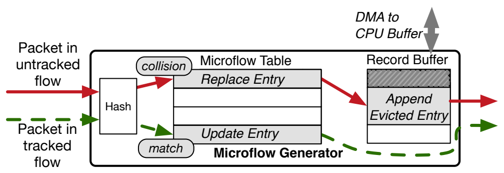

Figure 1: Overview TurboFlow's PFE function.

2 TurboFlow

In this paper, we propose TurboFlow: a flow record generator for commodity network equipment with PFEs that achieves high levels of performance by being designed *specifically for the constraints of real hardware.*

Our insight is that we can leverage the switch CPU to simplify and optimize the PFE component. In TurboFlow, the PFE works closely with the switch CPU to generate flow records, which allows us to move complex logic off of the PFE, leaving it with a simple routine optimized for real PFE architectures. The PFE does *just enough* to enable flow record generation at very high traffic rates without overburdening the switch CPU or sacrificing other design goals.

At a high level, the PFE generates *microflow records*, which are similar to flow records but only account for the most recent subset of packets to minimize the amount of state stored on the PFE. The switch's DMA engine transfers microflows to the switch CPU's main memory, where an aggregator groups them into full flow records that can be exported to a flow collector or network control server.

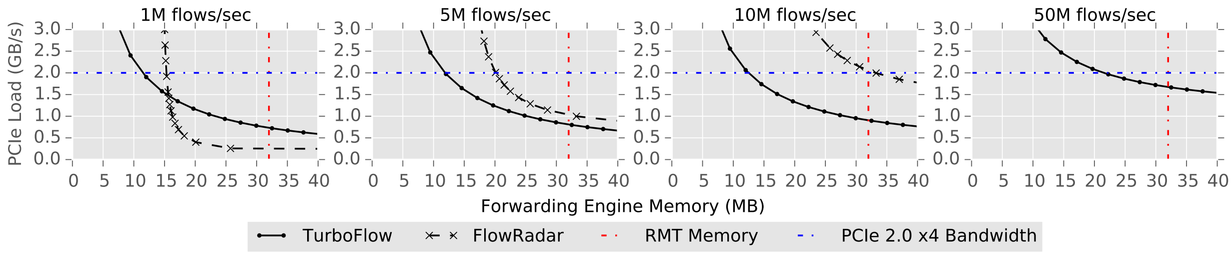

Figure 3: PFE memory versus CPU load.

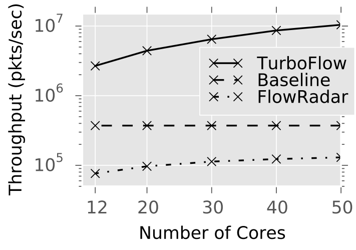

Figure 2: Packet rate throughput on a many cored PFE [6].

Metric	TurboFlow	FlowRadar	Baseline
Total Stateful Units	13	35	2
% RMT [3] Stages	43.7%	-	3.3%
% Banzai [8] Stages	26.6%	13.3%	3.3%
% Banzai [8] Stateful Units	4.3%	11.6%	0.3%

Table 1: Processing requirements on pipeline PFEs.

The PFE component of TurboFlow associates packets with flows using the data structure depicted in Figure 1, which is similar to a hash table but with one important modification. Whenever two flows collide, the PFE sends a microflow record for the older colliding flow up to the switch CPU and replaces it with the newer entry. This design guarantees a small number of memory operations per packet, minimizes the overhead of partitioning and synchronizing PFE state, and compiles to a short PFE function. Further, since the average microflow record summarizes multiple packets, the switch CPU component can generate full flow records that are accurate and contain rich feature sets, even when packet and flow rates are high.

3 Evaluation Highlights

We implemented TurboFlow in P4 [2] and analyzed the performance of its PFE and CPU components. We benchmarked the PFE component on two significantly different architectures that are representative of many existing and proposed PFEs and compared TurboFlow to FlowRadar, the fastest reported P4 flow record generation system [5].

Many-cored PFEs On the NFP-4000 PFE [6], which processes packets in parallel using 200 threads that share a large global memory bank, TurboFlow could monitor up to 10 M packets/s regardless of flow rate, as Figure 2 shows. This is a throughput over 75X higher than FlowRadar. TurboFlow scales better on many cored PFEs because its data structures are simpler to synchronize, which minimizes contention for access to shared memory.

Staged Pipeline PFEs On the Banzai machine, which models the architectures of terabit scale PFEs that use pipelines of processing units with independent memory banks [3, 8], TurboFlow is *compiler guaranteed to run at line rate*. TurboFlow also uses 50% fewer execution units and 40% less PFE memory than FlowRadar, as Table 1 shows. TurboFlow is efficient on this hardware because it minimizes the overhead of partitioning state across stages of the pipeline.

Switch CPU Under a worst case workload, TurboFlow's switch CPU component can monitor up to 30M flows/s while using 2 switch CPU cores, over 4X what FlowRadar supports in the same scenario. TurboFlow scales at the switch CPU level because it leverages PFE memory to reduce switch CPU load when flow rates are high, as Figure 3 shows.

References

[1] Barefoot networks unveils high-speed, programmable network switch. `http://www.zdnet.com/article/barefoot-networks-unveils-high-speed-programmable-network-switch/`.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 99–110. ACM, 2013.

[4] B. Li, J. Springer, G. Bebis, and M. H. Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2):567–581, 2013.

[5] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: a better netflow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 311–324, 2016.

[6] Netronome. Agilio cx intelligent server adapters agilio cx intelligent server adapters. `https://www.netronome.com/products/agilio-cx/`.

[7] QoSient. Argus: Audit records generation and utilization system. `http://qosient.com/argus/`.

[8] A. Sivaraman, M. Budiu, A. Cheung, C. Kim, S. Licking, G. Varghese+, H. Balakrishnan, M. Alizadeh, and N. McKeown. Packet transactions: High-level programming for line-rate switches.