Hey, You Darned Counters! Get Off My ASIC!

Jeffrey C. Mogul
HP Labs
Palo Alto, CA
Jeff.Mogul@hp.com

Paul Congdon
HP Labs/UC Davis
Palo Alto, CA
Paul.Congdon@hp.com

ABSTRACT

Software-Defined Networking (SDN) gains much of its value through the use of central controllers with global views of dynamic network state. To support a global view, SDN protocols, such as OpenFlow, expose several counters for each flow-table rule. These counters must be maintained by the data plane, which is typically implemented in hardware as an ASIC. ASIC-based counters are inflexible, and cannot easily be modified to compute novel metrics.

These counters do not need to be on the ASIC. If the ASIC data plane has a fast connection to a general-purpose CPU with cost-effective memory, we can replace traditional counters with a stream of rule-match records, transmit this stream to the CPU, and then process the stream in the CPU. These *software-defined counters* allow far more flexible processing of counter-related information, and can reduce the ASIC area and complexity needed to support counters.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Packet-switching networks

Keywords

Software-defined counters, OpenFlow

1. INTRODUCTION & MOTIVATION

Software Defined Networking (SDN) not only provides a platform for rapid innovation; it also changes the way that we manage networks. Instead of managing a network piece-by-piece, we can manage an SDN network from the vantage of a central controller, applying global policies and assuring network-wide validity of these policies.

Beyond that, an SDN controller can have a dynamic view of the current state of the network. Before SDN, network management was a slow process, focussed mostly on passively measuring network behavior, occasionally updating configurations, and responding rapidly only to bugs and mis-use. SDN allows us to view network management as a dynamic process, which can respond actively to events on the timescales of individual flows. For example, Hedera [1] identifies and re-routes long-lived "elephant" flows, which are too rare to be load-balanced by ECMP.

For an SDN controller to obtain its dynamic view of network state, switches must maintain per-flow counters (flows can be TCP connections, or coarser-grained aggregates). Switches make these counters visible to the controller. For example, OpenFlow[1] provides three counters per flow-table entry (Received Packets, Received Bytes, and Duration) and provides several mechanisms, both push-based and pull-based, to report counters to the controller.

Our work applies to hardware-based SDN data planes, which typically are implemented as ASICs – either using "merchant silicon," from vendors such as Broadcom, Fulcrum, etc., or using a custom ASIC.

In this paper, we argue that SDN counters should not need be stored on the ASIC itself. If the ASIC data plane has a fast connection to a general-purpose CPU with cost-effective memory (DRAM), we can replace traditional counters with a small buffer to hold a set of records recording information about recent flow-entry matches. The ASIC can stream these buffered records to the CPU, which processes the stream.

These *software-defined counters*, or SDCs, offer several advantages:

- **Increased flexibility**: We can evolve the SDN protocols to support a richer, and possibly more efficient, set of counters, without waiting many years for ASIC designers to implement new counter functions.
- **More efficient access to counters**: By keeping the counters in the switch CPU+DRAM, rather than on the ASIC, SDCs can reduce the overhead required to transfer counter values to the SDN controller.
- **Reduction in ASIC space and complexity**: Traditional counters consume space on the ASIC. While this is a relatively small fraction of the silicon area, removing counter functions from the ASIC can also reduce the amount of logic required to support counter-related functions, which in turn can reduce the time-to-market for the ASIC.

We elaborate briefly on the motivation for these advantages, in Sections 1.1, 1.2, and 1.3. Then we describe our proposed design for SDCs (§2), and we provide some quantified analysis of the approach (§4).

1.1 Increased flexibility

Software-defined counters (SDCs) allow far more flexible processing of counter-related information. If counters are implemented in the ASIC hardware, it may be very difficult to change

[1]For concreteness, in this paper we focus on OpenFlow.

HotSDN'12, August 13, 2012, Helsinki, Finland.

their function as the SDN protocol evolves. Implementing the counters in software, however, would make it much easier to innovate, without re-designing the ASIC or deploying new switch hardware.

As an example of such counter-related innovation, consider our previous work on DevoFlow [6], which aimed to reduce bandwidth required for communicating statistics between switches and the controller. We proposed modifying OpenFlow to associate *triggers* with counters, so that only flows which became "significant" would be reported to the controller. DevoFlow triggers could use thresholds on the number of packets for a flow, or the number of bytes, which we asserted (perhaps wishfully) "should be easy to implement within the data-plane." (§2.5 gives other examples.)

We also suggested that triggers could be based on per-flow packet rates or byte rates, but conceded that this feature would require additional state and calculation, and "might be harder to implement." Overall, DevoFlow increases the pressure on ASIC complexity and area, and the potential flexibility offered by exotic trigger conditions would be hard to justify if implemented in hardware.

1.2 Counter access overheads

Our work on DevoFlow showed that, at least in some existing ASICs, the bandwidth available for moving counter values from an ASIC to an OpenFlow controller can significantly limit overall SDN performance. We assume that future ASICs, designed for use in SDN switches, will provide more bandwidth for this path. Nevertheless, we expect this bandwidth will remain limited, and SDN designs that rely on frequent or low-latency counter access could be hobbled by this limit.

Storing the counters in the CPU+DRAM world, rather than on the ASIC, makes it much easier for the switch CPU to return counter values – either individual counters, or large tables – to the SDN controller. This should reduce the overheads for the controller to access the counters, and might enable certain SDN functions that are otherwise infeasible.

1.3 ASIC space and complexity

The cost of switch ASICs depends in large part on their area, and there is probably a practical upper limit for the area of a cost-effective ASIC. (Farrington *et al.*, several years ago, reported example areas of 35x35mm and 40x40mm [7].)

Since ASIC area is precious, this places limits on the sizes of on-chip memory structures, such as TCAMs to support wildcard flow-table entries, hash tables for exact-match lookups, and per-entry counters. Generally, one would like to support as many TCAM (and/or hash-table) entries as possible, since capacity misses in these tables could cause significant performance overheads. But any silicon area devoted to counters is area not available for lookup tables.

Also, designing and testing an ASIC costs a lot of time and money. For economic reasons, current ASICs probably must support traditional (non-SDN) networking as their primary target; ASIC support for SDN functions should therefore be kept as simple as possible.[2] Adding SDN support to create a hybrid (traditional+SDN) ASIC means finding space for the structures not typically found on an ASIC; the per-flow byte counters as used by OpenFlow could be the largest such structure. Adding support for extensions such as DevoFlow would require finding even more space on the ASIC.

[2]Several ASIC vendors have announced OpenFlow support, but we do not know if this involves OpenFlow-specific features on their ASICs.

2. SOFTWARE-DEFINED COUNTERS

In this section, we present our proposed design for software-defined counters (SDCs). We defer discussion of quantitative issues to §4.

2.1 The trouble with on-ASIC counters

OpenFlow (Version 1.1.0) specifies three counters for each flow-table entry: the number of matches, the number of packet bytes in these matches, and the flow duration. Each is specified as 64 bits, so this seems to add 192 bits (24 bytes) of extra storage per table entry.

Possibly, the storage overhead can be reduced, since 64 bits is overkill for any system where the counters are read fairly often. Also, it is not clear if the duration value for standard OpenFlow needs to be stored on-ASIC.

Even so, OpenFlow counters and the logic to support them adds significant ASIC complexity and area. Extending the basic OpenFlow counters, with features such as DevoFlow (and other examples, as in §2.5), adds more bits per table entry, and adds ASIC logic for comparing counters against trigger thresholds. DevoFlow may also require new mechanisms to deliver trigger events to the switch-local CPU.

If counters create problems for ASIC designers, why are they on the ASIC? Although switches generally have a CPU, to manage the ASICs and to engage in routing protocols, etc., in the past, these CPUs have been rather wimpy, and the bandwidth between the ASICs and the CPU has also been limited. (We documented some specific limits in the DevoFlow paper [6].) Prior to SDNs, counters were used primarily for functions with relatively long time scales, which did not stress these CPU and bandwidth limits.

But what if switch CPUs were more powerful, and had high-bandwidth connections to the ASIC? Then we can exploit these two improvements to move counters off of the ASIC, using software-defined counters.

Is it plausible that switch CPUs will become more powerful? We think so; first, because the need to support SDNs already places more burden on these CPUs, and second, because embedded-processor performance has been improving even while desktop/server CPU speeds have levelled off.

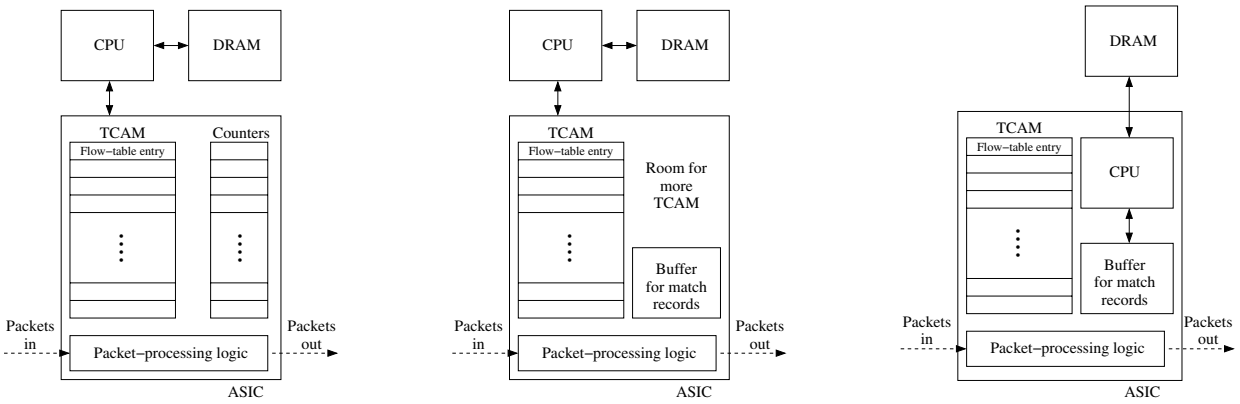
We also believe it is possible to engineer significantly more bandwidth between the ASIC data plane and the switch CPU. One option is to use a modern bus, such as PCI Express (PCIe), especially if we are willing to gain bandwidth at the expense of latency. Or, one could dedicate a 10GbE port for the ASIC-to-CPU connection. Another option is to place a CPU core on the ASIC itself.

2.2 Event records

In our SDC design, the switch ASIC has no counters. Instead, on each packet arrival, the ASIC generates an *event record*, and adds this record to a sequential on-ASIC buffer, of modest size. The buffer is split into several blocks; when a block is full, the ASIC streams the block to the switch-local CPU. The CPU, upon receiving a buffer block, unpacks the event records and updates its representation of the counters, stored in DRAM attached to the CPU.

Each event record represents one packet, and has these two fields:

- **ruleNumber**: the index of the flow-table rule that the packet matched, or a special value if no rules matched.
- **byteCount**: for this packet.
- **timestamp**: for this packet.



(a) Traditional SDN switch (b) SDC switch with off-ASIC CPU (c) SDC switch with on-ASIC CPU

Figure 1: Sketches of switch design alternatives

There is no need for a packetCount field, since each event record implies an increment of one packet. The timestamp field might be omitted, to save buffer bandwidth and to avoid the need for an on-ASIC timebase, if the CPU's own timebase is considered sufficient.

The size of an event record depends on some assumptions about other switch parameters, but one could get by with 32 bits, assuming no timestamp:

- **ruleNumber**: 19 bits supports over 500K flow-table entries.
- **byteCount**: 14 bits supports 9000-byte "jumbo" packets. We could reduce this to 13 bits by rounding up odd packet lengths (e.g., a 63-byte packet would be reported as carrying 64 bytes).

Of course, switches that support larger flow tables or larger packets would need more than 32 bits.

If the timestamp is included, that lengthens the event record. OpenFlow specifies 1-nanosecond precision for the flow-duration counter, but we can represent the timestamps as deltas. With just 10 bits per record, this allows deltas of up to 65 μsec; the ASIC can insert a special record if the gap between packets is larger than that, or a dummy record indicating a match against a special reserved ruleNumber.

Note that the format of an event record *is not exposed* outside the implementation of the switch. It does not have to be standardized, and conceivably it could be a "soft" format, reconfigurable when a switch is booted (although we know of no current ASICs that expose this kind of reconfiguration.)

In §3, we discuss techniques to compress event records, to reduce the required ASIC-to-CPU bandwidth.

2.3 System-level design

We assume a switch that includes one or more ASICs and one or more local CPUs.

Figure 1(a) shows a sketch of a traditional SDN switch; this sketch omits a lot of detail, including the possibility of multiple CPU cores. The ASIC includes packet-processing logic, a TCAM for matching against flow-table rules, and storage for counters. (The ASIC might have additional flow-table memory – for example, a hash table for exact-match rules – which we omit for simplicity.) A CPU is attached both to the ASIC and to some DRAM.

Figure 1(b) shows an SDC switch. The counters have been removed from the ASIC. Some of that area has been replaced by a buffer for event records. The remaining area could be used for additional TCAM rows (for example).

We assume that future SDN switches will use a higher-bandwidth connection between ASIC and CPU than used in older switches. For example, this could be a PCI-Express (PCIe) channel, or perhaps a 10GbE connection (§4.2 discusses how much bandwidth is available).

Figure 1(c) shows an SDC switch where the CPU has been moved onto the ASIC. This consumes more ASIC space, but this design simplifies the problem of providing high bandwidth between the data plane and the CPU; it also ensures that each additional ASIC, within a single switch, comes with a proportionate amount of CPU power.

2.4 Design details

OpenFlow version 1.1 introduces the option of replacing a single lookup table with multiple, cascaded tables. This allows factoring of concerns; for example, one could use one table for access-control and another for routing. Multi-table lookups means that an event record would have to carry several ruleNumbers. Fortunately, one rationale for the multi-table model is to avoid the need for a very large single table, so support for multiple ruleNumbers might not seriously increase the event-record size.

The draft OpenFlow 1.3 specification adds two per-rule flags, to disable the use of packet and byte counters for that rule, since the controller might not care about those values. This would be useful for SDC, since it avoid the need to consume ASIC-to-CPU bandwidth and CPU cycles for events that match the flagged rules.

Also, if there are multiple CPU cores available for processing counters, the ASIC can spread the event-stream bandwidth among them. Generally, events for a given ruleNumber should always be sent to the same CPU; a simple hashing scheme obeys this invariant, and might provide reasonable load balance.

2.5 Software opportunities

Our main goal for SDC is to give switch-local software flexibility in how it handles the counters. Therefore, we do not try to cover all the possible opportunities for software-based counter processing, but we discuss a few cases.

In the simplest case, the switch CPU would maintain OpenFlow-style counter tables (for packet counts, byte counts, and flow durations) in DRAM. As it receives each buffer of event records, the event-processing software unpacks each record, and updates the corresponding counters.

To implement DevoFlow-style extensions, the software would keep trigger values in other DRAM tables. On arrival of an event

record, the software would first update the corresponding counters, then compare them against the trigger thresholds, and would generate notifications to the central controller as necessary. The software would also record that a notification has been sent, so as not to repeatedly flood the controller with redundant reports. For rate-based triggers, the software would maintain additional tables (perhaps implemented using sparse data structures) to support rate computations. This allows support for DevoFlow without any impact on the ASIC.

Other possible software features include:

- Immediately discard "uninteresting" records, based on filter conditions specified by a possible extension to OpenFlow. (This might help to support the `Limit` clause in the Frenetic language [8].)
- Support *multi-flow* triggers, such as a trigger that generates a report when any subset of a set of flows reaches a threshold (e.g., "some subset of connections to my Web server together account for 1Gbit/s of download traffic"), or one that looks at the relative rates for a set of flows (e.g., to detect unfair bandwidth usage).
  Multi-flow triggers would be nearly impossible to do on an ASIC. However, this feature would require a more sophisticated load-balancing scheme with a multi-core CPU; for example, adding a few bits per rule to indicate the target CPU.
- Support flow-rate triggers, generating a report when the arrival rate of new flows in a certain category exceeds a threshold. For example, a virus detector or throttle, such as described by Twycross and Williamson [12], can look for abnormal connection-establishment rates. OpenFlow could implement this feature by reporting all new TCP connection attempts to the controller, but that is not a robust way to handle anomalously high connection rates.
- Emulate NetFlow [5] or RMON [13]. While OpenFlow provides a rich set of counter mechanisms (not just per-flow), SDN switches are likely to be used in hybrid environments that are managed by traditional tools. Using SDCs, much of the implementation for these functions could be moved out of the ASIC.

The goal of SDCs is not necessarily to support arbitrarily complex counter processing in the switch; this could be seen as violating the spirit of software-defined networking, which aims to move intelligence out of the switches and into the controller. Rather, we view SDCs as supporting evolution of the counter support in SDN protocols.

2.6 Adding more event-record fields

The event-record format described in §2.2 is quite simple: it carries a rule number and a packet byte-count. We believe that this data, when coupled with ASIC- or CPU-generated timestamps, is sufficient for the switch CPU to support a wide range of software-defined counter functions.

Of course, packets have features other than their lengths and arrival times. However, we observe that many of these features are implied by the OpenFlow rule that a packet has matched. For example, an exact-match rule fully specifies the values of the packet fields covered by OpenFlow's flow key.

In cases where the OpenFlow application is forced to use wildcard rules (e.g., if using exact-match rules would cause excessive ASIC-table misses), it might be worth adding, to the event-record format, a hash of certain packet-header fields. This would allow, for example, hash-based sampling.[3] Or the switch CPU could then keep separate counters for distinct hash values, providing more detailed, albeit "anonymized," breakdowns of flow statistics.

[3]Suggested by an anonymous reviewer.

2.7 Support for multi-ASIC switches

Switches with large numbers of ports may require multiple ASICs. Implementing OpenFlow in a multi-ASIC switch can introduce complexity not found in a single-ASIC version, since OpenFlow conceptually treats the entire switch as a single entity. In a multi-ASIC switch, some flow table entries will be replicated on several ASICS: for example, a rule that matches all TCP packets to port 80.

With ASIC-resident counters, it is hard to implement counter functions, such as DevoFlow's threshold-based triggers, for a rule that spans multiple ASICs. For example, if we want to know when the rate of connection attempts to port 80 exceeds 100 connections/sec., on a two-ASIC switch, some CPU would have to check the counters on each ASIC every second. However, with SDCs, a single CPU could maintain a unified counter for this trigger, based on event-record streams from both ASICs.

This mechanism has some implications for the system hardware configuration – for example, requiring more complex connections between CPUs and ASICs.

3. COMPRESSING EVENT RECORDS

The ASIC could apply a simple compression algorithm to reduce the bandwidth required between the ASIC and the CPU. We assume that there is some locality in the packet-arrival reference stream that could be exploited to make a compression scheme work. (§4.3 discusses locality.)

For example, if the working set of ruleNumbers is likely to be much smaller than the full range, a simple algorithm such as Huffman coding could be implemented in the ASIC, to avoid the need for sending full-sized ruleNumber values.

Similarly, studies have shown that certain packet sizes are far more common than others; typical distributions are bimodal (e.g., Benson *et al.* [4]). These frequent values could be represented with shorter codes. Note that the worst case for event-arrival rates is the best case for this kind of compression, since all byteCounts would be equal to the minimum packet length.

Another approach to compression might be to keep a small cache representing a few "busy" rules. For each event that hits in this cache, the ASIC would coalesce the counter values locally. When a cache-entry is evicted, the coalesced values (including an explicit packetCount field) would be streamed to the CPU (this is similar to how NetFlow works). Some flow-table entries could be flagged "do-not-coalesce," if needed to support certain fine-grained behaviors.

What if even compression is not enough to prevent the event-record stream from over-running the on-ASIC buffer – that is, the CPU cannot keep up with these events? (§4.3 discusses the CPU requirements.) The obvious solution is to discard some event records, either using drop-tail, or randomly (to reduce bias in the counter values.) Or, the ASIC could have a small "important-rules" filter to prioritize certain event records, and/or a "boring-rules" filter to de-prioritize others; these filters would be applied once the buffer depth reaches a threshold. (We doubt it is necessary to over-engineer this part of the design, if the only reason is to support an unrealistic benchmark at the maximum possible packet rate.)

4. EVALUATION

We have not actually implemented SDCs, either in a simulation or in an emulation such as NetFPGA. Therefore, we "evaluate" the feasibility of SDCs using some back-of-the-envelope analyses.

4.1 Event-record rates

The primary limits on the feasibility of SDCs are (1) the bandwidth between the ASIC and CPU(s), and (2) the processing ability of each CPU. What kinds of event-record rates would the system have to handle?

We assume, for concreteness, that a single ASIC can support 48 10GbE ports (although the densest switch ASICs currently on the market seem to be 24x10GbE rather than 48x10GbE, we should assume future ASICs will provide higher aggregate throughput).

The worst-case frame rate for a single 10 GbE port is 14,880,960 frames/sec, or about 714M frames/sec for our 48x10GbE ASIC. Assuming 32 bits per event record (that is, without any compression, and assuming a single flow table), this amounts to 2.86 GByte/sec.

Conversely, the best-case 10GbE frame rate (that is, all packets are 1500 bytes – admittedly, nearly impossible in practice) is 812,740 frames/sec/port, or 39M frames/sec for our 48x10GbE ASIC. This amounts to 156 MByte/sec

Obviously, this is a wide range of possible frame rates. In order to provide some simple real-world calibration, we obtained mean packet sizes from several different settings:

- **OC192 traces provided by CAIDA**: CAIDA collects one-hour traces [10] each month on several Internet backbone links. (OC192 is about 9.9 Gbit/sec.) They provide data on the mean packet size, averaged over each trace. Between Jan. 2011 and Mar. 2012, the lowest one-hour average was 3.2 Kbits/packet, or about 400 bytes/packet.[4] The maximum packet rate, averaged over a one-hour trace, was 902K packets/sec. Our analysis of an older CAIDA OC192 packet trace (Jan. 15 2009) [9] found a peak packet rate, measured over intervals of 1 msec, of 562 pkts/msec, or 562K pkts/sec.
- **Data centers (WREN 2009)**: Benson *et al.* report a mean of 850 bytes [4] for a data center that "hosts several line-of-business applications (e.g., web services)."
- **Data centers (IMC 2010)**: Benson *et al.* [3] reported data obtained from ten data centers, including three university data centers and two private-enterprise data centers. Mean packet sizes were 738 bytes (EDU1), 751 bytes (EDU2), and between 881–888 bytes (different traces from PRV2); peak rates were 215 pkts/msec (EDU1) and 141 pkts/msec (EDU2) [2].

Overall, this data (while sketchy) suggests that real switches encounter a mean packet size between about 400 and 800 bytes. Using 400 bytes as a conservative value, this leads to a rate of about 3,125,000 packets per second per fully-utilized 10GbE port.

With on-ASIC coalescing: We analyzed the potential performance of a small, on-ASIC busy-rule cache (see §3). For the Jan. 15. 2009 CAIDA traces, the reduction in event-record rate ranged from 24% (16 cache entries) to 52% (1024 entries); for the Benson *et al.* traces, reductions ranged from 65% (16 entries) to almost 91% (1024 entries). Thus, even a fairly small on-ASIC cache provides significant event-rate reductions for a data-center workload, and useful rate reductions for an Internet backbone workload.

[4]Data for the "sanjose-B/2011-11-17" trace shows a much lower mean packet size, but CAIDA suspects this trace may be anomalous.

4.2 ASIC-to-CPU bandwidth

The bandwidth available for streaming event records from the ASIC to the CPU, and hence to its DRAM, depends on the specific system configuration:

- **PCIe link to off-ASIC CPU**: In this configuration, the ASIC appears to the CPU as a PCIe device. PCIe provides various bandwidths, ranging from 250 MB/sec (PCIe v1.x) to 8 GB/sec (PCIe v6.0) per "lane." PCIe supports up to 32 lanes, so the maximum available bandwidth is quite large. However, space and power considerations might limit what is feasible.
- **10GbE link to off-ASIC CPU**: One of the switch ASIC ports could be dedicated to communicate with the switch CPU, if the CPU has an integrated NIC. This link could be made loss-free, using 802.1Qbb "Priority-based Flow Control." However, this approach provides relatively low bandwidth compared to the other options, consumes a switch port, and requires an extra NIC on the CPU.
- **On-ASIC CPU reading from on-ASIC buffer**: In this configuration, an on-ASIC CPU would read event records from a small SRAM buffer, and then write data to counters in off-ASIC DRAM. Here, bandwidth between the buffer and the CPU is not likely to be the rate-limiting step. Rather, the CPU itself is likely the bottleneck; we discuss CPU overheads in §4.3.

How do these bandwidths compare to the demands required for streaming event records? A worst-case analysis (full link utilization with min-size packets, no compression) for a 48x10GbE switch requires 2.86 GByte/sec; a more realistic packet-size distribution (e.g., 400 Bytes/packet) for the same switch, at full utilization, would require 572 MByte/sec. Realistic rates should be sustainable with any of the configurations listed above.

4.3 CPU processing costs

The switch CPU must read event records and update a set of counters for each such record; OpenFlow requires updating two 64-bit counters. Can a reasonable CPU keep up with this processing rate?

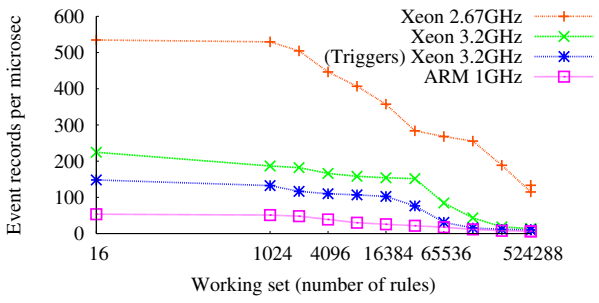


Figure 2: Event-processing microbenchmark

Microbenchmark: We wrote an oversimplified microbenchmark, which loops through a large buffer of event records (too large to fit into the CPU caches, in order to emulate the cache misses that a real input stream would create). We initialized the 32-bit event records in this buffer with random values for the 19-bit ruleNumber and 13-bit byteCount fields. (These event records contain no timestamp field.) The inner loop extracts the ruleNumber, uses this as an index into an in-memory array of counters, and then updates

the 64-bit packetCount and byteCount values for the corresponding counter record.

The microbenchmark allows us to specify the range of randomly-chosen rule numbers in the event records, to emulate working sets of various sizes. Optionally, we can recompile the benchmark so that the inner loop checks, on each event record, whether the total packet or byte count has exceeded a per-rule "trigger" value, as in DevoFlow. This makes the counter records 8 bytes larger, and adds a few more instructions to the inner loop.

Clearly this is an unrealistic benchmark. It ignores any costs for decompression, for communication with the SDN controller, and for processing the OpenFlow duration field (which, probably, does not have to be updated on every packet). It also fails to model most of the advanced counter processing discussed in §2.5. On the other hand, if the ASIC can coalesce multiple events into one record, that should reduce the memory-access workload for the CPU.

We ran the benchmark on several different CPUs: an HP SL390s server, with a 2.66 GHz X5650 ("Nehalem") Xeon and DDR3-1333 memory; an HP xw8200 workstation, with an older 3.2GHz Xeon and DDR2-400 memory; and a Trim-Slice H "miniature desktop," with a 1.0GHz Cortex-A9 ARM CPU and DDR2-667 memory.[5] All of these systems are multi-core, but our benchmark is single-threaded.

Figure 2 shows microbenchmark results for these configurations. The best-case (small working-set) results are 535, 225, and 53 event records processed per microsecond for the SL390s, xw8200, and ARM processors, respectively. These rates are presumably limited by the CPU core. The worst-case (512K-counter working set) results are 133, 13.4, and 6.3 records/μsec, respectively. These rates are probably limited by memory bandwidth, which is affected both by the type of DRAM and by the processor design.

For comparison, using a mean packet size = 400 bytes we expect 3.125 recs/μsec/port, or $\sim$150 recs/μsec for a 10GbEx48 switch. These rates are plausibly within the range of a *dual-core* ARM CPU, if there is enough locality, whereas the worst-case (minimal-packet) rate of 715 recs/μsec clearly is not. If there is no locality, then clearly our design is not currently feasible for such a high-end switch (without adding many cores and memory controllers).

In Figure 2, the data set marked "(Triggers)" shows how performance for the 3.2GHz Xeon declines with the optional DevoFlow-style trigger processing. Some of the decline (34%, for small working sets) appears to come from the additional instruction executions; at larger working sets, the decline (23%) may be due in part to higher cache miss rates. We suspect that our implementation of triggers could be further optimized.

Available locality: How much locality is available in real-world network traffic? Benson *et al.* [4] report that, over their university and private data centers, the number of active flows at any single switch never exceeded 10,000, and in most cases was considerably lower. Other researchers have assumed even more flow-level locality for SDN switches, perhaps because most existing switch ASICs support relatively small flow tables. For example, Tavakoli *et al.* analyzed flow-entry requirements for both VL2 and NOX; their scalability analysis suggested that at most 5,000 entries would be required [11].

We analyzed a subset of Benson's traces[6] to model flow-indexed caches of various sizes. A 1K-entry cache would yield hit rates of 78%–97% for these traces. A similar analysis of 1-second slices of the Jan. 15 2009 CAIDA traces shows 43%–52% hit rates for a 1K-entry cache.

[5] `http://trimslice.com/web/trim-slice-h-specifications`

[6] `http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html`

For a working-set size of 1K rules, the data in Figure 2 show that a single ARM core can process 51 event-records/μsec, so three such cores (or one Xeon core) could conceivably handle most of the event-record load for a 48x10GbE switch in a data center similar to those studied by Benson *et al.* An Internet backbone switch, with less locality, might require several Xeon cores to handle the load.

In short, the feasibility of SDCs boils down to a set of assumptions about traffic rates, packet size distributions, flow-level locality, and the amount of hardware one is willing to devote to event-record processing. In the worst case, SDC is probably not feasible with current CPU and memory hardware. In more realistic cases, SDC seems like a promising approach for providing SDN counter support.

5. SUMMARY

Software-defined networking, and OpenFlow in particular, has provided an incentive to reconsider the division of labor between switch ASICs, switch-local CPUs, and server-based controllers. We believe that a "smart controller, dumb switch" model is too naïve, especially with the decreasing cost of putting high-performance embedded CPUs into switches. By exploiting these switch-local CPUs, we can move complexity out of the ASICs *and* provide many more opportunities for interesting evolution in the kinds and features of SDN counters.

Acknowledgments

We would like to thank Theo Benson (U. Wisconsin) and Paul Hick (CAIDA) for access to their data, and Dwight Barron and John Wickeraad of HP for their technical advice. We also thank the anonymous reviewers for their helpful suggestions.

6. REFERENCES

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. NSDI*, 2010.

[2] T. Benson. Pers. communication, 2012.

[3] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. IMC*, pages 267–280, 2010.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding Data Center Traffic Characteristics. In *Proc. WREN*, pages 65–72, 2009.

[5] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proc. SIGCOMM*, pages 254–265, 2011.

[7] N. Farrington, E. Rubow, and A. Vahdat. Data Center Switch Architecture in the Age of Merchant Silicon. In *Proc. Hot Interconnects*, pages 93–102, 2009.

[8] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A Network Programming Language. In *Proc. ICFP*, pages 279–291, 2011.

[9] P. Hick, kc claffy, and D. Andersen. The CAIDA UCSD Anonymized Internet Traces – 15 Jan 2009. `http://www.caida.org/data/passive/passive_2009_dataset.xml`.

[10] P. Hick, kc claffy, and D. Andersen. Trace Statistics for CAIDA Passive OC48 and OC192 Traces. `http://www.caida.org/data/passive/trace_stats/`, 2012.

[11] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying NOX to the Datacenter. In *Proc. HotNets*, 2009.

[12] J. Twycross and M. M. Williamson. Implementing and Testing a Virus Throttle. In *Proc. USENIX Security*, pages 285–294, 2003.

[13] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu. Introduction to the Remote Monitoring (RMON) Family of MIB Modules. RFC 3577 (Informational), 2003.