Scalable High Resolution Traffic Heatmaps: Coherent Queue Visualization for Datacenters

Andreea Anghel, Robert Birke, and Mitch Gusat

IBM Research, Zurich, Switzerland
{aan,bir,mig}@zurich.ibm.com

Abstract. We propose a new high resolution – temporal and spatial – 10 Gbps Ethernet monitoring technique based on time-coherent congestion 'heatmaps', revealing (all) the queue occupancies at μs granularity. Notably, queues are sampled with a slightly modified version of the new commodity Ethernet hardware congestion management protocol, i.e., IEEE 802 Quantized Congestion Notification. Our technique is evaluated through high-accuracy Layer-2 simulations of a 10 Gbps datacenter Ethernet fabric. Early results reveal that our proposal enables the detection of ephemeral – yet consequential – events and transients essential for datacenter workload characterization: e.g., TCP Incast, Head-of-Line blocking and congestion trees, which may trigger within 10s of μs and were not directly detectable until now.

Keywords: Network monitoring, L2 sampling, Traffic analysis, QCN.

1 Introduction

The proliferation of Big Data applications – such as Hadoop, MapReduce – together with 10-100 Gbps datacenter networks (DCNs) – such as Converged Enhanced Ethernet (CEE), with μs latency – raises new challenges. One of them is the need for faster and scalable network monitoring methods. Deep understanding of such workloads and their communication patterns is key to the design of future workload-optimized datacenters. For example, the ability to accurately detect and optimize (via adapted routing) workload-induced congestion at fine granularity has significant potential in terms of reducing system cost and improving the application run-time [1]. Switches today have outdated monitoring software that cannot cope with the sampling frequency and the network scale needed to orchestrate large datacenters. Hence our target is to coherently (i.e., time-space synchronized) and scalably detect, record and replay ephemeral – yet consequential – transients, essential for workload characterization: e.g., TCP Incast [2], Head-of-Line blocking [3], and congestion trees [4], phenomena that may trigger within few 10s of μs and significantly affect the network and workload performance.

To address these issues, a significant improvement is required in the way we design and implement network monitoring. The challenges of 10-100 Gbps load monitoring and visualization for high resolution datacenter network management

A. Dainotti, A. Mahanti, and S. Uhlig (Eds.): TMA 2014, LNCS 8406, pp. 26–37, 2014.

are three-fold. Space-wise, how to collect, correlate and visualize several 1000s of queues, quasi-simultaneously (correlation may be due to either Hadoop/HPC-like workload phases, or the activation of low-level flow control). Time-wise, we face a resolution challenge, i.e., the "μs wall" – one must detect, record, visualize, and/or react to network events that happen at μs (packet duration) timescales. Finally, the method must conceptually scale to million-node datacenters and Tbps link speeds. Hence the need for commodity hardware queue samplers, possibly reusing existing resources. All while helping the users and operators to better understand their load dynamics and nature.

As an exemplary Big Data workload, the Hadoop Partition/Aggregate pattern consists of multiple congestive episodes within few 100s of μs, known as TCP Incasts. The Aggregate phases actually determine the load, dynamic behavior, stability, and finally the user-perceived performance. However, the Incast phases at and above 10 Gbps are too volatile for the current sFlow [5], NetFlow [6], and SNMP [7] monitoring techniques. Hence the need for a faster and lighter load sensing method able to 'catch' such ephemerally volatile *correlated* transients, scale to large networks, and also drill-in, i.e., zoom into a region of interest of a few 10s-100s queues.

We therefore propose a high-resolution (both) temporal and spatial monitoring technique based on IEEE 802 Quantized Congestion Notification (QCN) [8], Ethernet's new congestion management protocol. Triggering, sampling, aggregating, communicating and visualizing network events are based on the concept of time-coherent *snapshots* of the queue occupancy levels. Coherency is achieved by using a network time difference estimation protocol implemented by the switches – a high accuracy alternative to current clock synchronization protocols such as GPS, NTP [9], and IEEE 1588 [10]. We validate the above via simulations using accurate Layer-2 CEE-compliant 10 Gbps fabric models. As a first, our method detects and captures in real time the TCP Incast-like events in *lossless* fabrics, revealing their inception, evolution and global dynamic behavior at μs scale.

2 Selected Related Work

The area of network monitoring, telemetry and topography is exceedingly rich in literature. However, practically only a few schemes have been widely adopted by hardware and software vendors. Thus, despite much larger and faster DCNs, the current state of the art in hardware network monitoring has remained limited to sampling a few, possibly isolated, links with a granularity in the 0.01s to 1s range: e.g., sFlow, NetFlow and SNMP. These methods we consider insufficient for today's 10/100 Gbps DCNs and large distributed workloads.

More recent proposals, such as [11], suggest new router extensions to perform path sampling using hardware synchronization via IEEE 1588. By instrumenting switches with a hash-based primitive, the authors measure latencies down to 10s of μs and 1 loss in a million. By contrast, our proposal (ab)uses the QCN standard to sample queues (instead of paths) with a theoretical sub-μs sampling.

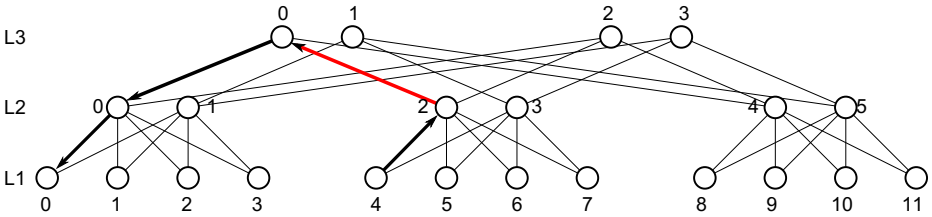

Fig. 1. Folded topology representation of XGFT(3;2,4,3;1,2,2). Links are bidirectional. Node levels start at 0 from bottom to top (L0 to L3). Nodes within a level start at 0 from left to right. For simplicity reasons, only the switch levels (L1, L2, L3) are shown. L0 is populated with 2 (nodes) · 12 (L1-switches) = 24 (nodes). We highlight one path from switch 4 on L1 to switch 0 on L1. The red link represents the upstream queue of port 0 at switch 2 on L2.

Batraneanu et al. [12] propose a tool-set developed to demonstrate aggregate achievable cross-sectional bandwidth for specific traffic profiles, as well as to analyze network hot spot behavior. Queues are being monitored at μs using sFlow and an FPGA implementation. Though the achieved timescale is similar, their solution is fundamentally different in various perspectives: we use the readily available QCN queue samplers, provide a coherent globally synchronized trigger-driven method, developed a record-and-replay tool validated in PFC-enabled 10 Gbps CEE networks, and show that our heatmaps scale with low overhead.

Another related proposal [13] identifies the challenges of datacenter and cloud monitoring and visualization: scale, rapidity, difficulties of detection, localization, and diagnose of performance problems. The proposed tool, i.e., Visual-I, relates directly to our targets. However, while we share goals with Visual-I, our method differs on the use of coherent heatmaps and the novel Layer-2 capabilities of the upcoming CEE-based datacenter networks.

3 Traffic Heatmap Method

We propose a matrix-based traffic visualization scheme as follows. Each packet sent from a source to a destination follows a certain route in the network, as depicted in Fig. 1, and travels through the corresponding set of queues in the forwarding switches, contributing to their respective occupancies. A network congestion/traffic heatmap is an instantaneous time-synchronous 2D snapshot image of the occupancy of all the switch queues in the network. Each heatmap cell represents a single queue providing a fine-grained spatial view of the network.

The occupancy is linearly color-coded on 16 bits using shades of red. The color ranges from white, empty, to dark red, full. Since the heatmaps are time-synchronous, they represent a snapshot of the network queues' status. This allows us to temporally concatenate such sequentially-acquired snapshots into an animated 'movie'. This movie not only provides insights into the network status evolution – congestion levels and locations – but also into the volatile causes and effects of network anomalies such as congestion trees.

The heatmaps are created by a controller which is the central element of the monitoring framework. The forwarding switches sample their queue occupancies – by reusing the commodity QCN hardware mechanism – and send them to the controller (see Subsec. 3.2). The controller then performs two mappings: one in time and one in space. In time, each sample is timestamped using the local clock of the switch it originated from. Since each switch has its own clock, the controller needs to map all the timestamps to a single common timeline (see Subsec. 3.4). The converted timestamps are sorted chronologically. The configuration of the heatmap at any given moment on the timeline is then given by the most recent occupancy value received for every queue previous to that moment. In space, the controller must know the network topology and location of each switch and queue. We assume the existence of a topology discovery protocol that detects and signals topology changes to the controller, for example, through the information gathered by the spanning tree protocol. Each queue has a unique identifier, thus each sample can be mapped to a precise cell using a topology-dependent bijective function. Since we focus on datacenter networks, we only present the mapping for fat trees (Subsec. 3.3). However, our method can be extended to other topologies by changing the mapping bijection.

The final *raw* result is a time-series of complete space-time mappings of all queue occupancy samples of all the switches. The heatmap thus created allows the detection of even ephemeral network anomalies. Once a critical area has been identified – e.g. due to particularly frequent congestion events – it is possible to increase the sampling frequency of those switches/queues to further enhance the detection accuracy. Also, once the culprits are identified, it is possible to take online or offline corrective actions and solve the hotspots or the detected events. An example is given in Sec. 5.1.

3.1 Datacenter Network Topology

Today, fat trees, i.e., multistage k-ary n-fly topologies, are typically the base for large datacenters [14,15]. These topologies can be described by extended generalized fat trees (XGFTs) [16]. An $XGFT(h; m_1, ...m_h; w_1, ...w_h)$ has $h+1$ node levels divided into leaf and inner nodes. The $\prod_{i=1}^{h} m_i$ leaf nodes reside on level 0 and serve as end nodes/servers. Inner nodes occupy levels 1 to h and serve as switches. Each inner node on level i has m_i child nodes and each non-root node on level j has w_{j+1} parent nodes. Fig. 1 shows an exemplary XGFT(3;2,4,3;1,2,2).

3.2 Load Sensing: Queue Sampling

The queue sampling is performed in a distributed manner: each switch samples its own local queues. Sampling can start either periodically or remotely triggered upon a controller request. In the latter case, the controller sends a request to the switch with the desired sampling duration, rate and start time translated to the switch's own clock. The novelty of our approach consists in extending the

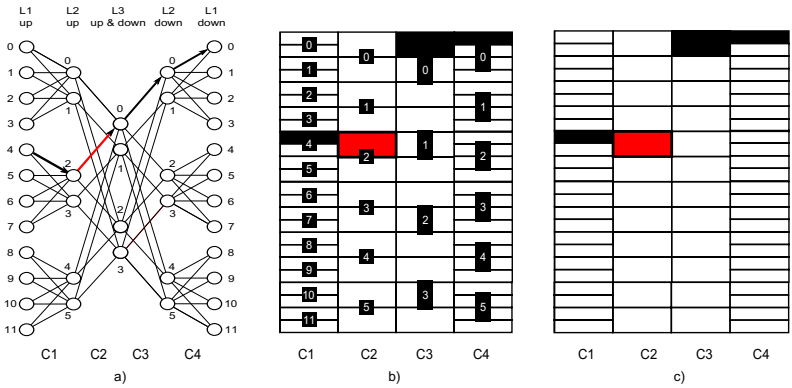

Fig. 2. Spatial mapping example. Intermediate Fig. a) and b) show how the XGFT from Fig. 1 is mapped onto the heatmap in c). a) unfolds and rotates the topology by 90 degrees. Links are unidirectional: traffic flows from left to right. Each level corresponds to the up-/down- stream direction. All figures highlight the same exemplary path from switch 4 on L1 to switch 0 on L1. Similarly, red highlights the send queue(s) of port 0 at switch 2 of level 2 (L2). Each link level in a) corresponds to a column in b) and c). Whereas, each cell in a column represents top-down the output queues ordered by: (1) the switch and (2) the port within that switch. E.g., C3 shows the downstream output queues of the L3 switches: 4 switches · 3 ports · 1 queue = 12 queues. Typical current switches have 1 to 4 hardware queues per port, but for increased clarity of results in paper format we will assume herein a single queue per port – the generalization to several queues is nonetheless trivial.

use of the QCN load sensor to traffic monitoring. QCN is Ethernet's new end-to-end congestion management which, together with its new flow control, i.e., Priority Flow Control (PFC), is being adapted by a growing number of network equipment to ease the network convergence commonly known as CEE.

To detect congestion, QCN uses load sensors monitoring all the queues. Its main objective is to keep the queue occupancy at a target equilibrium Q_{eq}. At each frame arrival, QCN samples with probability π the queue occupancy. For each sampled frame, the load sensor computes a feedback value based on the current and past occupancy. Negative feedbacks are sent back to the frame's source, which reduces its injection rate, thus adjusting it to the available bandwidth.

Extending the use of the QCN load sensor to monitoring has the great benefit of readily available hardware which can sample *all* the queues at fine, i.e. subμs timescales, theoretically even per packet arrival – during limited *zoom-in* sampling intervals.

This improves upon other state of the art monitoring tools both in terms of temporal – 10s of ns – and spatial granularity – each output queue. However two additional steps are needed to enhance the native behavior of QCN's load sampler: (1) to timestamp and locally store the QCN samples; and (2) to make the samples available to the heatmap controller.

To avoid flooding the network with monitoring traffic, we use compressive sampling and filtering based on the feedback values computed by QCN. The

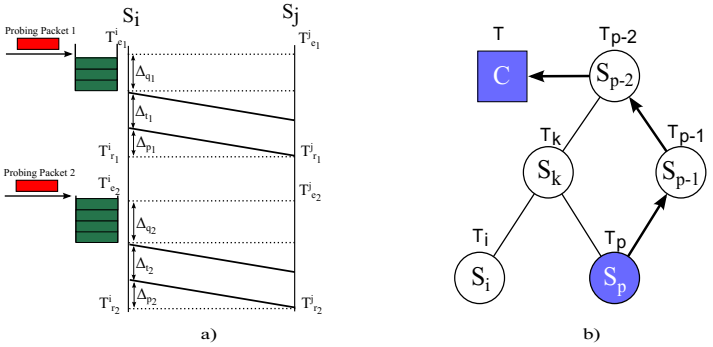

Fig. 3. a) Method to calculate the relative drifts and offsets of two neighboring switches S_i and S_j. b) Method to calculate the absolute drift and offset of a switch S_p to a reference controller system C.

informative samples are mainly related to a congestive event. Hence, all negative feedback samples, signaling a congestion, are sent to the controller. During positive feedback sampling periods, only the first sample is sent back to the controller. During normal network operation, this filter removes most of the samples. To further reduce the monitoring overhead, it is possible to reduce the probability π (see Subsec. 5.3), and compress and aggregate multiple samples into MTU frames. Jumbo frames up to 10-20 KB could be used for high radix switches with thousands of queues (per port and priority). Once the frame is full or upon a timeout – to bound the reporting delay in case of sparse sample reporting – the resulting local snapshot frame is sent to the controller for heatmap assembly.

3.3 XGFT Mapping: Heatmap Bijection

Each heatmap cell represents a specific queue. Fig. 2 shows in steps the heatmap creation of the XGFT in Fig. 1. If level c has S switches, each switch P ports in the upstream/downstream direction, and each port Q queues, then within that level/column, row $r = s \cdot P \cdot Q + p \cdot Q + q$ represents queue q belonging to port p of switch s, where $0 \leqslant s < S$, $0 \leqslant p < P$ and $0 \leqslant q < Q$. Conversely, given a row r within column c, the reverse mapping is: $q = r \ modulo \ Q$, $p = \frac{r}{Q} \ modulo \ P$, $s = \frac{r}{P \cdot Q}$ and the unfolded network level is c. This mapping is valid for homogeneous switches, i.e., same port and queue numbers, which is typical in practice. However, the mapping can be extended to accommodate non-homogeneous configurations. Moreover, although in this paper we only address the mapping of fat tree topologies (which are the most commonly used in datacenters), the mapping can be extended to multi-dimensional topologies by using topological isomorphisms as proposed in [17].

3.4 Coherent Sampling: Clock Difference Estimation Protocol

A coherent network heatmap requires: (1) a snapshot of the queues at the same time instant, i.e., 10s of ns timescale; and (2) a common time reference for the

occupancy samples. Since each switch has its own internal clock, this problem is not trivial and goes beyond the scope of this paper. Nevertheless, we shortly introduce a clock estimation method based on the following linear drift model:

$$t_i = \delta_i \cdot t + \omega_i \tag{1}$$

where t is the controller reference time, t_i the time at switch S_i, and δ_i and ω_i the absolute clock drift and offset at switch S_i. We also assume that all δ_i and ω_i are constant and that $\omega_i > 0$. This model entails that given two switches S_i and S_j, we have:

$$t_i = \delta_{ij} \cdot t_j + \omega_{ij} \tag{2}$$

where $\delta_{ij} = \delta_i/\delta_j$ and $\omega_{ij} = \omega_i - \delta_{ij} \cdot \omega_j$ are the relative drift and offset between S_i and S_j.

We rely on an adhoc neighbor-to-neighbor protocol between the switches to minimize estimation errors on the one-way latency due to queuing delays in intermediate hops. Fig. 3a) shows the method to calculate the relative drifts and offsets between two neighboring switches S_i and S_j. S_i sends a probe of size P_s to S_j over a link with speed BW containing the local enqueuing time T_e^i and local queue size Q_s^i. We assume the one-way latency Δ to be the sum of three delays: (i) queuing $\Delta_q = \frac{Q_s}{BW}$, (ii) transmission $\Delta_t = \frac{P_s}{BW}$, and (iii) propagation Δ_p which depends only on the type and cable length, hence assumed to be constant. Using Eq. 1, the arrival time t_r^j at S_j can be expressed as $T_r^j = \delta_{ji} \cdot T + \omega_{ji}$ and $T_r^i = T_e^i + \Delta$. Thus, we have:

$$T_r^j = \delta_{ji} \cdot (T_e^i + \Delta_q + \Delta_t + \Delta_p) + \omega_{ji} \tag{3}$$

containing only three unknowns: δ_{ij}, ω_{ij} and Δ_p. For each probe, the receiving switch adds one equation to an equation system. Solving this system allows to obtain δ_{ji} and ω_{ji} , which are sent to the controller. From all the relative drifts and offsets, the controller then computes the absolute drifts and offsets as shown in Fig. 3b). It computes the absolute δ_i and ω_i for each switch by iteratively applying Eq. 1 on all consecutive switch pairs along the path. The general equation given a switch S_p reachable via the path $S_p \Rightarrow S_{p-1} \Rightarrow ... \Rightarrow S_1 \Rightarrow C$ is shown in Eq. 4.

$$t_p = \prod_{i=2}^{p} \delta_{i,i-1} \cdot t + \sum_{i=1}^{p-1} \left(\prod_{j=i}^{p-1} \delta_{j+1,j} \right) \cdot \omega + \omega_{p,p-1} = \delta_p \cdot t + \omega_p \tag{4}$$

This approach allows for a non-intrusive clock compensation without alteration of the internal switch clocks. The temporal correlation among queue samples, as well as among the start/stop sampling commands, is achieved without explicit synchronization of the switch clocks.

4 Network and Simulation Environment

To show the benefits of such monitoring tool, we identify congestion trees in a lossless datacenter network. More in detail, we consider the same XGFT topology as presented in Sec. 3, comprising 24 end nodes and 22 switches interconnected by 10 Gbps CEE-compliant links. CEE aims at providing a reliable Layer-2 in contrast to the conventional Ethernet which drops frames as soon as a buffer reaches its maximum capacity. CEE achieves this by relying on two distinct protocols: PFC and QCN.

Similar to IEEE 802.3x PAUSE, PFC avoids packet losses during congestion by pausing the sender using explicit control frames. However PFC uses the concept of service classes: a paused priority does not affect the others. QCN counteracts

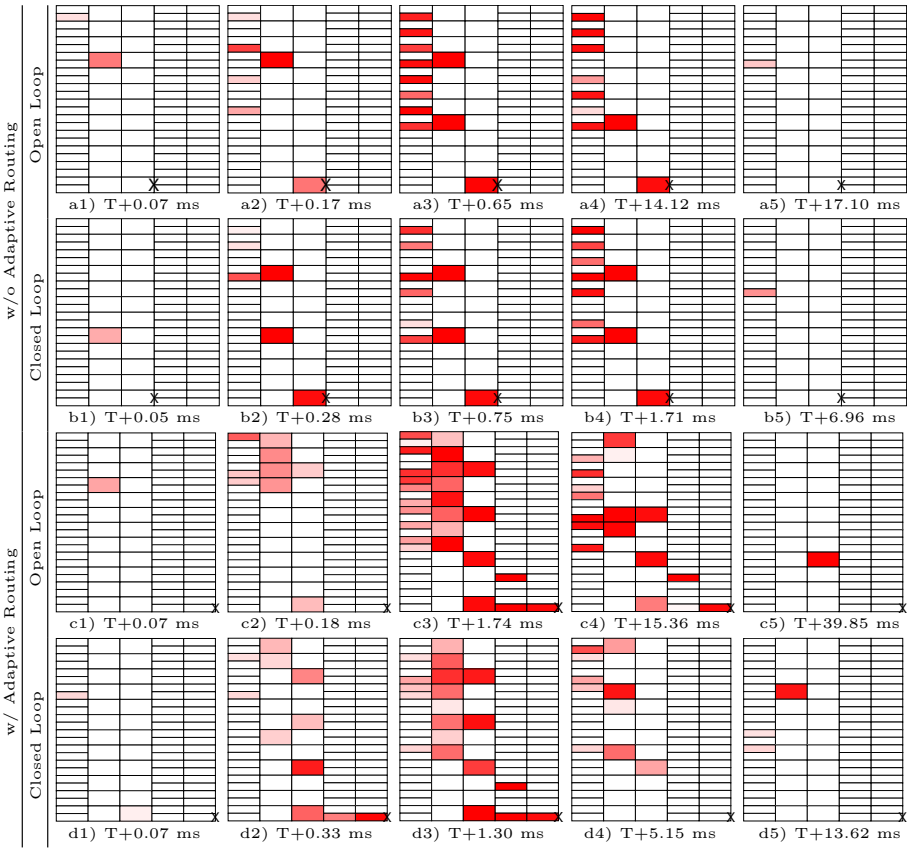

Fig. 4. XGFT(3;2,4,3;1,2,2) under many-to-one traffic (nodes 0-15 send to node 23 at full link speed, i.e. 10 Gbps, for 10 *ms* starting at T). Each row corresponds to a different network configuration and shows 5 time steps of the congestion evolution. The first 3 steps show the congestion tree onset, while the last 2 steps show the network recovery. The congestion tree root is marked by X.

the congestion entity by limiting the injection rates of the culprit sources. Even if mitigated by QCN, PFC can be affected by Head-of-Line blocking building up into congestion trees in the network: a blocked sender may fill up its buffer and recursively other devices will possibly get paused spreading the congestion backwards. Congestion trees can significantly degrade network performance [4], and hence it is useful to detect them to act quickly on their removal.

The occupancy samples are generated using high-accuracy Layer-2 simulations from an event-driven simulator based on OMNeT++ [18]. The simulator is a two-level architecture network modeling platform. The higher level is the topological or system level where we gain a generic overview of the whole simulated infrastructure as an interconnection of modules. The lower level is that of the internal structure of each network module: e.g., hosts, and switches. We used our own network simulator due to its enhanced support for the CEE standards and detailed models of the Layer-2 switch and adapter micro-architectures. We extended our simulator to include: (1) the modified QCN load sensor additionally timestamping, storing and aggregating the queue samples as described in Subsec. 3.2 and (2) the clock difference estimation protocol described in Sec. 3.4. The drifts and offsets values are derived from empirical distributions obtained by instrumenting an Intel e1000 network driver and by analyzing the speed at which clocks of neighboring systems drift. A JAVA-based application collects the queue occupancy samples from the simulator and generates the heatmaps.

5 Results and Discussion

5.1 Congestion Tree Detection Heatmap Movie

As use case we propose a TCP Incast-like scenario, typical of Partition/Aggregate workloads. This bursty and correlated congestive pattern is also typical for barrier-based HPC applications, commonly using lossless fabrics. The surge in correlated packet arrivals overflows the aggregation buffer. In TCP over legacy Ethernet frames will be dropped, and subsequently recovered via TCP timeouts – thus leading to significantly increased delays. In modern CEE fabrics, PFC prevents such drops and improves performance [19]. However, transient congestion trees may still degrade the performance – under certain conditions: flow sizes, buffer settings, hotspot/incast degree. Such congestion trees and their evolution can be observed in Fig. 4, which shows selected *frames* of the heatmap movie under different network configurations: (a) open/closed QCN congestion control loop; (b) without or with adaptive routing. In open loop, the QCN rate limiters are disabled, i.e., congestion notifications are ignored; while in closed loop, the QCN rate limiters react by adjusting the injection rate. The adaptive routing scheme [20] balances the load over the multiple paths available in fat trees. In contrast to Fig. 2c), each heatmap in Fig. 4 includes one additional column (C5) representing the downstream queues of the L1 switches.

Row (a), i.e., open loop without adaptive routing, shows that the congestion tree fully builds up within 700 μs. Its root, i.e., designated hotspot, is located on the first downstream output queue, the flows convergence point. After 17

ms the congestion tree is drained, no longer affecting the flow delays. Closing the QCN loop (row b) mitigates the congestion as seen both from the lighter shades of red and the $\sim 2.4\times$ shorter draining time (7 *ms*). As expected, when enabling adaptive routing many more queues share the load - since the traffic is spread across multiple paths (row c and d); now all the 'incast' flows meet only at the destination, relocating the previous tree root. Again closing the QCN loop (row d) reduces the congestion duration, draining faster by a factor $\sim 3\times$. While these results meet the initial expectations, now they are also revealed to external observers without the benefit of apriori knowledge. The events of potential interest are *automatically* detected based on the programmable QCN threshold setpoint, recorded, and visualized for offline analysis.

The fine-grained timescale achievable with our coherent 'snapshooting' method naturally lends itself to movie-like playbacks, with variable temporal resolutions in frames/s. This not only enhances the user experience, but also enables temporal zooming (fast/slow motion, fast forward). Thus we can monitor longer time intervals without loosing track of the – potentially ephemeral – events of interest.

5.2 Time Estimation Accuracy

To test the accuracy of our clock estimation protocol, we use the same network as above, with uniform all-to-all background (i.i.d.) traffic at 60% load. Results show a mean error among all 10 Gbps switches of 380 ns with a standard deviation of 465 ns. The achieved accuracy is comparable to the hardware-aided implementation of IEEE 1588 [21] and 2x-3x better than the software-only IEEE 1588 version [22].

5.3 Monitoring Overhead

The overhead of our QCN-based sampling technique depends on several parameters: average network diameter, injected traffic, average congestion level/severity and its duration. Here we show only the influence of the QCN parameters. In particular we consider the queue sampling probability $\pi \in [1\%, 2\%, 4\%, 10\%]$ and the queue equilibrium setpoint $Q_{eq} \in [2KB, 4KB, 8KB, 16KB]$. We assume a sample size of 8B including a queue identifier (16 bit), queue occupancy (16 bit), queue occupancy variation (16 bit), timestamp offset (16 bit); plus a standard Ethernet-IP-TCP header (54B) extended with a switch identifier 6B and a base timestamp 8B (Timestamp = Base + Offset, for space saving). We compute the sampling overhead as the ratio between the sampling traffic reported to the controller and the total traffic sent by all hosts during the sampling period. The results show that π directly affects the overhead, but even for aggressive sampling probabilities, i.e. 10% (e.g., one in 10 packet arrivals in a 128-port switch fully N-to-1 congested can lead to sub-μs periods at 10 Gbps), the obtained overhead is less than 0.2%. For π of less than 5%, the overhead is below 0.1%. This overhead is approximately constant with respect to Q_{eq}.

5.4 Switch Code Complexity

The additional switch code complexity is low. On the data plane, the switch only needs to locally save and timestamp the occupancy samples already collected by the QCN sampler. On the control plane, the switch needs to perform two sets of operations: (1) aggregate, compress, and send the QCN samples to a centralized controller; (2) collect probes for the clock difference estimation protocol through a heartbeat protocol. The actual derivation of the relative offsets and drifts can be performed either locally or, if too complex for the local switch control processor, offloaded to the controller.

6 Conclusions

We presented a high resolution monitoring method with: (1) μs queue sampling period, (2) based on the newly available IEEE 802 *QCN* hardware load *sensing*; (3) fabric-level *coherence*, i.e., global synchronous trigger-driven method; (4) multi-queue heatmap 'snapshooting' that scales up with low traffic overheads; (5) an intuitive matrix representation of the datacenter system based on a bijective mapping of the common XGFT networks; (6) a record and replay *visualization* tool, validated in lossless link-level flow-controlled 10 Gbps accurate Layer-2 simulations.

As proof of concept, we have designed and implemented a method that displays the time-coherent heatmap snapshots and movies showing the simultaneous occupancy of – some or all – the queues in a simulated network. For the first time now, the detection and visualization – in quasi-real time – of congestion trees and Incast events, correlated across the entire fabric, have been shown to operate in a lossless 10 Gbps commodity Ethernet. Thus, this method enables finer resolution insights into workload-specific traffic patterns and detects persistent and transient anomalies. This fast and efficient monitoring approach can benefit researchers, architects, network administrators, Big Data application developers, etc., to better understand the interaction between complex workloads and the underlying networks in large multitenant datacenters.

References

1. Prisacari, B., et al.: Fast pattern-specific routing for fat tree networks. ACM TACO 10(4), 36:1–36:25 (2013)
2. Zhang, J., et al.: Modeling and understanding TCP incast in data center networks. In: IEEE INFOCOM, pp. 1377–1385 (2011)
3. Jurczyk, M., et al.: Phenomenon of higher order head-of-line blocking in multistage interconnection networks under nonuniform traffic patterns. IEICE Transactions on Information and Systems (Special Issue on Architectures, Algorithms and Networks for Massively Parallel Computing) 79(8), 1124–1129 (1996)
4. Pfister, G.F., et al.: Hot spot contention and combining in multistage interconnection networks. IEEE Transactions on Computers 100(10), 943–948 (1985)

5. Phaal, P., et al.: InMon corporation's sFlow: A method for monitoring traffic in switched and routed networks. Technical report, RFC 3176 (2001)
6. Claise, B.: Cisco systems NetFlow services export version 9 (2004)
7. Case, J., et al.: A simple network management protocol (SNMP). In: Network Information Center, SRI International (1989)
8. IEEE: P802.1Qbb/D1.3 Virtual bridged local area networks - Amendment: Priority-based flow control. Technical report. IEEE (2010)
9. Mills, D., et al.: Network time protocol version 4: Protocol and algorithms specification (2010)
10. IEEE/ANSI: 1588 Standard for a precision clock synchronization protocol for networked measurement and control systems. Technical report, IEEE/ANSI (2008)
11. Kompella, R.R., et al.: Every microsecond counts: tracking fine-grain latencies with a lossy difference aggregator. In: ACM SIGCOMM CCR, vol. 39, pp. 255–266. ACM (2009)
12. Batraneanu, S., et al.: Operational model of the ATLAS TDAQ network. IEEE Transactions on Nuclear Science 55(2), 687–694 (2008)
13. Fisher, D., et al.: Using visualization to support network and application management in a data center. In: IEEE INM 2008, pp. 1–6 (2008)
14. Leiserson, C.E.: Fat-trees: universal networks for hardware-efficient supercomputing. IEEE Transactions on Computers 100(10), 892–901 (1985)
15. Petrini, F., et al.: Performance evaluation of the Quadrics interconnection network. Cluster Computing 6(2), 125–142 (2003)
16. Ohring, S.R., et al.: On generalized fat trees. In: Parallel Processing Symposium, pp. 37–44. IEEE (1995)
17. Dally, W., et al.: Principles and practices of interconnection networks. Morgan Kaufmann Publishers Inc., San Francisco (2003)
18. Minkenberg, C., et al.: Trace-driven co-simulation of high-performance computing systems using OMNeT++. In: SIMUTools 2009, p. 65 (2009)
19. Crisan, D., et al.: Short and Fat: TCP performance in CEE datacenter networks. In: HOTI 2011, pp. 43–50 (2011)
20. Gusat, M., et al.: R3C2: Reactive route and rate control for CEE. In: HOTI 2010, pp. 50–57 (2010)
21. Ferrari, P., et al.: Synchronization of the probes of a distributed instrument for real-time Ethernet networks. In: ISPCS 2007, pp. 33–40 (2007)
22. Correll, K.: et al.: Design considerations for software only implementations of the IEEE 1588 precision time protocol. In: Conference on IEEE 1588 Standard, vol. 1588, pp. 11–15 (2005)