# CONGA: Distributed Congestion-Aware Load Balancing for Datacenters

Mohammad Alizadeh[†], Tom Edsall[†], Sarang Dharmapurikar[†], Ramanan Vaidyanathan[†], Kevin Chu[†], Andy Fingerhut[†], Vinh The Lam[‡], Francis Matus[†], Rong Pan[†], Navindra Yadav[†], George Varghese[§]

[†]Cisco Systems [‡]Google [§]Microsoft

August 9, 2014

**Abstract**

We present the design, implementation, and evaluation of CONGA, a network-based distributed congestion-aware load balancing mechanism for datacenters. CONGA exploits recent trends including the use of regular Clos topologies and overlays for network virtualization. It splits TCP flows into flowlets, estimates real-time congestion on fabric paths, and allocates flowlets to paths based on feedback from remote switches. This enables CONGA to efficiently balance load and seamlessly handle asymmetry, without requiring any TCP modifications. CONGA has been implemented in custom ASICs as part of a new datacenter fabric. In testbed experiments, CONGA has 5× better flow completion times than ECMP even with a single link failure and achieves 2–8× better throughput than MPTCP in Incast scenarios. Further, the Price of Anarchy for CONGA is provably small in Leaf-Spine topologies; hence CONGA is nearly as effective as a centralized scheduler while being able to react to congestion in microseconds. Our main thesis is that datacenter fabric load balancing is best done in the network, and requires global schemes such as CONGA to handle asymmetry.

**Categories and Subject Descriptors:** C.2.1 [Computer-Communication Networks]: Network Architecture and Design
**Keywords:** Datacenter fabric; Load balancing; Distributed

## 1 Introduction

Datacenter networks being deployed by cloud providers as well as enterprises must provide large bisection bandwidth to support an ever increasing array of applications, ranging from financial services to big-data analytics. They also must provide agility, enabling any application to be deployed at any server, in order to realize operational efficiency and reduce costs. Seminal papers such as VL2 [17] and Portland [1] showed how to achieve this with Clos topologies, Equal Cost MultiPath (ECMP) load balancing, and the decoupling of endpoint addresses from their location. These design principles are followed by next generation overlay technologies that accomplish the same goals using standard encapsulations such as VXLAN [34] and NVGRE [44].

However, it is well known [2, 40, 8, 26, 43, 9] that ECMP can balance load poorly. First, because ECMP randomly hashes flows to paths, hash collisions can cause significant imbalance if there are a few large flows. More importantly, ECMP uses a purely *local* decision to split traffic among equal cost paths without knowledge of potential downstream congestion on each path. Thus ECMP fares poorly with *asymmetry* caused by link failures that occur frequently and are disruptive in datacenters [16, 33]. For instance, the recent study by Gill *et al.* [16] shows that failures can reduce delivered traffic by up to 40% despite built-in redundancy.

Broadly speaking, the prior work on addressing ECMP's shortcomings can be classified as either centralized scheduling (e.g., Hedera [2]), local switch mechanisms (e.g., Flare [26]), or host-based transport

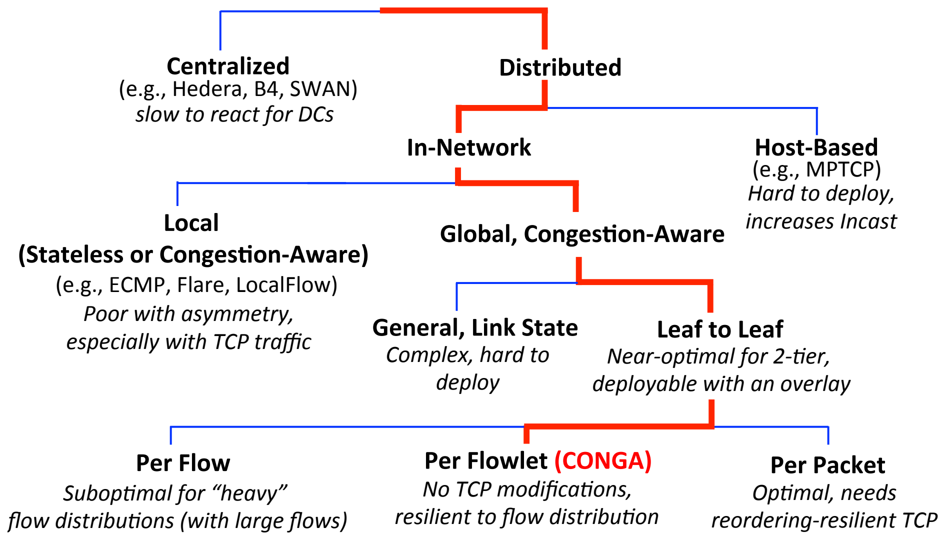

Centralized
(e.g., Hedera, B4, SWAN)
slow to react for DCs
Distributed
In-Network
Host-Based
(e.g., MPTCP)
Hard to deploy, increases Incast
Local
(Stateless or Congestion-Aware)
(e.g., ECMP, Flare, LocalFlow)
Poor with asymmetry, especially with TCP traffic
Global, Congestion-Aware
General, Link State
Complex, hard to deploy
Leaf to Leaf
Near-optimal for 2-tier, deployable with an overlay
Per Flow
Suboptimal for "heavy" flow distributions (with large flows)
Per Flowlet (CONGA)
No TCP modifications, resilient to flow distribution
Per Packet
Optimal, needs reordering-resilient TCP

Figure 1: Design space for load balancing.

protocols (e.g., MPTCP [40]). These approaches all have important drawbacks. Centralized schemes are too slow for the traffic volatility in datacenters [27, 7] and local congestion-aware mechanisms are suboptimal and can perform even worse than ECMP with asymmetry (§2.4). Host-based methods such as MPTCP are challenging to deploy because network operators often do not control the end-host stack (e.g., in a public cloud) and even when they do, some high performance applications (such as low latency storage systems [38, 6]) bypass the kernel and implement their own transport. Further, host-based load balancing adds more complexity to an already complex transport layer burdened by new requirements such as low latency and burst tolerance [3] in datacenters. As our experiments with MPTCP show, this can make for brittle performance (§5).

Thus from a philosophical standpoint it is worth asking: Can load balancing be done in the network without adding to the complexity of the transport layer? Can such a network-based approach compute globally optimal allocations, and yet be implementable in a realizable and distributed fashion to allow rapid reaction in microseconds? Can such a mechanism be deployed today using standard encapsulation formats? We seek to answer these questions in this paper with a new scheme called CONGA (for *Cong*estion *A*ware Balancing). CONGA has been implemented in custom ASICs for a major new datacenter fabric product line. While we report on lab experiments using working hardware together with simulations and mathematical analysis, customer trials are scheduled in a few months as of the time of this writing.

Figure 1 surveys the design space for load balancing and places CONGA in context by following the thick red lines through the design tree. At the highest level, CONGA is a distributed scheme to allow rapid round-trip timescale reaction to congestion to cope with bursty datacenter traffic [27, 7]. CONGA is implemented within the network to avoid the deployment issues of host-based methods and additional complexity in the transport layer. To deal with asymmetry, unlike earlier proposals such as Flare [26] and LocalFlow [43] that only use local information, CONGA uses global congestion information, a design choice justified in detail in §2.4.

Next, the most general design would sense congestion on every link and send generalized link state packets to compute congestion-sensitive routes [15, 47, 50, 35]. However, this is an N-party protocol with complex control loops, likely to be fragile and hard to deploy. Recall that the early ARPANET moved briefly to such congestion-sensitive routing and then returned to static routing, citing instability as a reason [30]. CONGA instead uses a 2-party "leaf-to-leaf" mechanism to convey path-wise congestion metrics between pairs of top-of-the-rack switches (also termed leaf switches) in a datacenter fabric. The leaf-to-leaf scheme is provably near-optimal in typical 2-tier Clos topologies (henceforth called *Leaf-Spine*), simple to analyze, and easy to deploy. In fact, it is deployable in datacenters today with standard overlay encapsulations (VXLAN [34] in our implementation) which are already being used to enable workload agility [32].

With the availability of very high-density switching platforms for the spine (or core) with 100s of 40Gbps ports, a 2-tier fabric can scale upwards of 20,000 10Gbps ports.[1] This design point covers the needs of the overwhelming majority of enterprise datacenters, which are the primary deployment environments for CONGA.

Finally, in the lowest branch of the design tree, CONGA is constructed to work with flowlets [26] to achieve a higher granularity of control and resilience to the flow size distribution while not requiring any modifications to TCP. Of course, CONGA could also be made to operate per packet by using a very small flowlet inactivity gap (see §3.4) to perform optimally with a *future* reordering-resilient TCP.

In summary, our major contributions are:

- We design (§3) and implement (§4) CONGA, a distributed congestion-aware load balancing mechanism for datacenters. CONGA is immediately deployable, robust to asymmetries caused by link failures, reacts to congestion in microseconds, and requires no end-host modifications.
- We extensively evaluate (§5) CONGA with a hardware testbed and packet-level simulations. We show that even with a single link failure, CONGA achieves more than 5× better flow completion time and 2× better job completion time respectively for a realistic datacenter workload and a standard Hadoop Distributed File System benchmark. CONGA is at least as good as MPTCP for load balancing while outperforming MPTCP by 2–8× in Incast [46, 11] scenarios.
- We analyze (§6) CONGA and show that it is nearly optimal in 2-tier Leaf-Spine topologies using "Price of Anarchy" [39] analysis. We also prove that load balancing behavior and the effectiveness of flowlets depends on the coefficient of variation of the flow size distribution.

## 2 Design Decisions

This section describes the insights that inform CONGA's major design decisions. We begin with the desired properties that have guided our design. We then revisit the design decisions shown in Figure 1 in more detail from the top down.

### 2.1 Desired Properties

CONGA is an in-network congestion-aware load balancing mechanism for datacenter fabrics. In designing CONGA, we targeted a solution with a number of key properties:

1. **Responsive:** Datacenter traffic is very volatile and bursty [17, 27, 7] and switch buffers are shallow [3]. Thus, with CONGA, we aim for rapid round-trip timescale (e.g., 10s of microseconds) reaction to congestion.
2. **Transport independent:** As a network mechanism, CONGA must be oblivious to the transport protocol at the end-host (TCP, UDP, etc). Importantly, it should not require any modifications to TCP.
3. **Robust to asymmetry:** CONGA must handle asymmetry due to link failures (which have been shown to be frequent and disruptive in datacenters [16, 33]) or high bandwidth flows that are not well balanced.
4. **Incrementally deployable:** It should be possible to apply CONGA to only a subset of the traffic and only a subset of the switches.
5. **Optimized for Leaf-Spine topology:** CONGA must work optimally for 2-tier *Leaf-Spine* topologies (Figure 4) that cover the needs of most enterprise datacenter deployments, though it should also benefit larger topologies.

[1]For example, spine switches with 576 40Gbps ports can be paired with typical 48-port leaf switches to enable non-blocking fabrics with 27,648 10Gbps ports.

## 2.2 Why Distributed Load Balancing?

The distributed approach we advocate is in stark contrast to recently proposed centralized traffic engineering designs [2, 8, 22, 20]. This is because of two important features of datacenters. First, datacenter traffic is very bursty and unpredictable [17, 27, 7]. CONGA reacts to congestion at RTT timescales ($\sim$100$\mu$s) making it more adept at handling high volatility than a centralized scheduler. For example, the Hedera scheduler in [2] runs every 5 seconds; but it would need to run every 100ms to approach the performance of a distributed solution such as MPTCP [40], which is itself outperformed by CONGA (§5). Second, datacenters use very regular topologies. For instance, in the common Leaf-Spine topology (Figure 4), all paths are exactly two-hops. As our experiments (§5) and analysis (§6) show, distributed decisions are close to optimal in such regular topologies.

Of course, a centralized approach is appropriate for WANs where traffic is stable and predictable and the topology is arbitrary. For example, Google's inter-datacenter traffic engineering algorithm needs to run just 540 times per day [22].

## 2.3 Why In-Network Load Balancing?

Continuing our exploration of the design space (Figure 1), the next question is *where* should datacenter fabric load balancing be implemented — the transport layer at the end-hosts or the network?

The state-of-the-art multipath transport protocol, MPTCP [40], splits each connection into multiple (e.g., 8) sub-flows and balances traffic across the sub-flows based on perceived congestion. Our experiments (§5) show that while MPTCP is effective for load balancing, its use of multiple sub-flows actually increases congestion at the edge of the network and degrades performance in Incast scenarios (Figure 13). Essentially, MPTCP increases the burstiness of traffic as more sub-flows contend at the fabric's access links. Note that this occurs despite MPTCP's coupled congestion control algorithm [49] which is designed to handle shared bottlenecks, because while the coupled congestion algorithm works if flows are in steady-state, in realistic datacenter workloads, many flows are short-lived and transient [17].

The larger architectural point however is that datacenter fabric load balancing is too specific to be implemented in the transport stack. Datacenter fabrics are highly-engineered, homogenous systems [1, 33]. They are designed from the onset to behave like a giant switch [17, 1, 24], much like the internal fabric within large modular switches. Binding the fabric's load balancing behavior to the transport stack which already needs to balance multiple important requirements (e.g., high throughput, low latency, and burst tolerance [3]) is architecturally unappealing. Further, some datacenter applications such as high performance storage systems bypass the kernel altogether [38, 6] and hence cannot use MPTCP.

## 2.4 Why Global Congestion Awareness?

Next, we consider local versus global schemes (Figure 1). Handling asymmetry essentially requires *non-local* knowledge about downstream congestion at the switches. With asymmetry, a switch cannot simply balance traffic based on the congestion of its local links. In fact, this may lead to even worse performance than a static scheme such as ECMP (which does not consider congestion at all) because of poor interaction with TCP's control loop.

As an illustration, consider the simple asymmetric scenario in Figure 2. Leaf L0 has 100Gbps of TCP traffic demand to Leaf L1. Static ECMP splits the *flows* equally, achieving a throughput of 90Gbps because the flows on the lower path are bottlenecked at the 40Gbps link (S1, L1). Local congestion-aware load balancing *is actually worse* with a throughput of 80Gbps. This is because as TCP slows down the flows on the lower path, the link (L0, S1) appears *less* congested. Hence, paradoxically, the local scheme shifts more traffic to the lower link until the throughput on the upper link is also 40 Gbps. This example illustrates a fundamental limitation of any local scheme (such as Flare [26], LocalFlow [43], and packet-spraying [9]) that strictly enforces an equal traffic split without regard for downstream congestion. Of course, global congestion-aware load balancing (as in CONGA) does not have this issue.

The reader may wonder if asymmetry can be handled by some form of oblivious routing [31] such as weighted random load balancing with weights chosen according to the topology. For instance, in the above example we could give the lower path half the weight of the upper path and achieve the same traffic split as CONGA. While this works in this case, it fails in general because the "right" traffic splits in asymmetric topologies depend also on the traffic matrix, as shown by the example in Figure 3. Here,

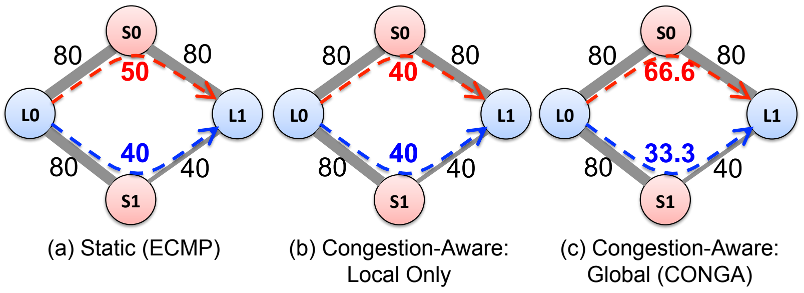


Figure 2: Congestion-aware load balancing needs non-local information with asymmetry. Here, L0 has 100Gbps of TCP traffic to L1, and the (S1, L1) link has half the capacity of the other links. Such cases occur in practice with link-aggregation (which is very common), for instance, if a link fails in a fabric with two 40Gbps links connecting each leaf and spine switch.

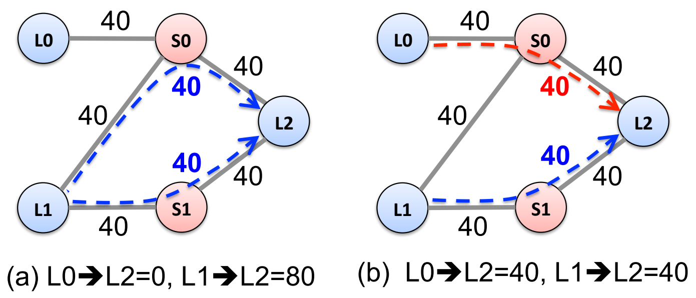


Figure 3: Optimal traffic split in asymmetric topologies depends on the traffic matrix. Here, the L1→L2 flow must adjust its traffic through S0 based on the amount of L0→L2 traffic.

depending on how much L0→L2 traffic there is, the optimal split for L1→L2 traffic changes. Hence, static weights cannot handle both cases in Figure 3. Note that in this example, the two L1→L2 paths are symmetric when considered in isolation. But because of an asymmetry in another part of the network, the L0→L2 traffic creates a *bandwidth asymmetry* for the L1→L2 traffic that can only be detected by considering non-local congestion. Note also that a local congestion-aware mechanism could actually perform worse than ECMP; for example, in the scenario in part (b).

## 2.5 Why Leaf-to-Leaf Feedback?

At the heart of CONGA is a leaf-to-leaf feedback mechanism that conveys real-time path congestion metrics to the leaf switches. The leaves use these metrics to make congestion-aware load balancing decisions based on the global (fabric-wide) state of congestion (Figure 4). We now argue (following Figure 1) why leaf-to-leaf congestion signaling is simple and natural for modern data centers.

**Overlay network:** CONGA operates in an overlay network consisting of "tunnels" between the fabric's leaf switches. When an endpoint (server or VM) sends a packet to the fabric, the source leaf, or source tunnel endpoint (TEP), looks up the destination endpoint's address (either MAC or IP) to determine to which leaf (destination TEP) the packet needs to be sent.[2] It then encapsulates the packet — with outer source and destination addresses set to the source and destination TEP addresses — and sends it to the spine. The spine switches forward the packet to its destination leaf based entirely on the outer header. Once the packet arrives at the destination leaf, it decapsulates the original packet and delivers it to the intended recipient.

Overlays are deployed today to virtualize the physical network and enable multi-tenancy and agility by decoupling endpoint identifiers from their location (see [37, 21, 32] for more details). However, the overlay also provides the ideal conduit for CONGA's leaf-to-leaf congestion feedback mechanism. Specifically, CONGA leverages two key properties of the overlay: (i) The source leaf knows the ultimate

[2]How the mapping between endpoint identifiers and their locations (leaf switches) is obtained and propagated is beyond the scope of this paper.

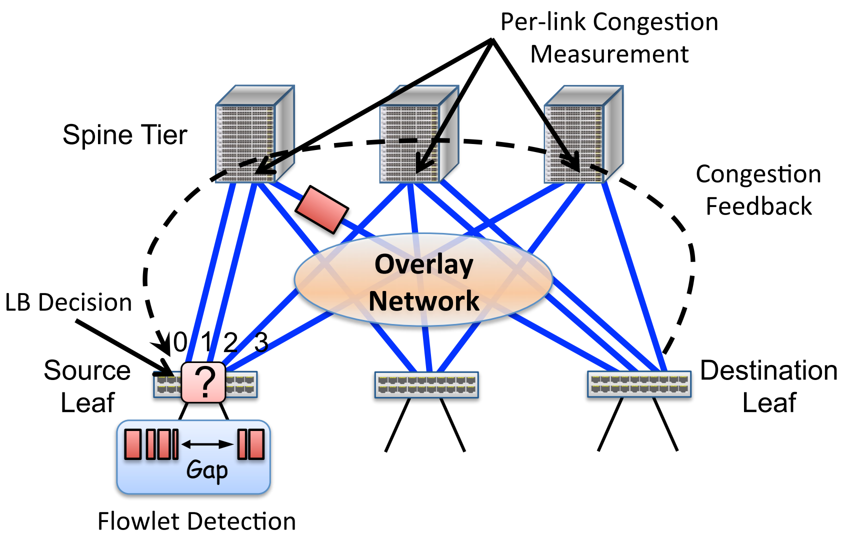


Figure 4: CONGA architecture. The source leaf switch detects flowlets and routes them via the least congested path to the destination using congestion metrics obtained from leaf-to-leaf feedback. Note that the topology could be asymmetric.

destination leaf for each packet, in contrast to standard IP forwarding where the switches only know the next-hops. (ii) The encapsulated packet has an overlay header (VXLAN [34] in our implementation) which can be used to carry congestion metrics between the leaf switches.

**Congestion feedback:** The high-level mechanism is as follows. Each packet carries a congestion metric in the overlay header that represents the extent of congestion the packet experiences as it traverses through the fabric. The metric is updated hop-by-hop and indicates the utilization of the most congested link along the packet's path. This information is stored at the destination leaf on a per source leaf, per path basis and is opportunistically fed back to the source leaf by piggybacking on packets in the reverse direction. There may be, in general, 100s of paths in a multi-tier topology. Hence, to reduce state, the destination leaf *aggregates* congestion metrics for one or more paths based on a generic identifier called the *Load Balancing Tag* that the source leaf inserts in packets (see §3 for details).

## 2.6 Why Flowlet Switching for Datacenters?

CONGA also employs *flowlet switching*, an idea first introduced by Kandula *et al.* [26]. Flowlets are bursts of packets from a flow that are separated by large enough gaps (see Figure 4). Specifically, if the idle interval between two bursts of packets is larger than the maximum difference in latency among the paths, then the second burst can be sent along a different path than the first without reordering packets. Thus flowlets provide a higher granularity alternative to flows for load balancing (without causing reordering).

### 2.6.1 Measurement analysis

Flowlet switching has been shown to be an effective technique for fine-grained load balancing across Internet paths [26], but how does it perform in datacenters? On the one hand, the very high bandwidth of internal datacenter flows would seem to suggest that the gaps needed for flowlets may be rare, limiting the applicability of flowlet switching. On the other hand, datacenter traffic is known to be extremely bursty at short timescales (e.g., 10–100s of microseconds) for a variety of reasons such as NIC hardware offloads designed to support high link rates [28]. Since very small flowlet gaps suffice in datacenters to maintain packet order (because the network latency is very low) such burstiness could provide sufficient flowlet opportunities.

We study the applicability of flowlet switching in datacenters using measurements from actual production datacenters. We instrument a production cluster with over 4500 virtualized and bare metal

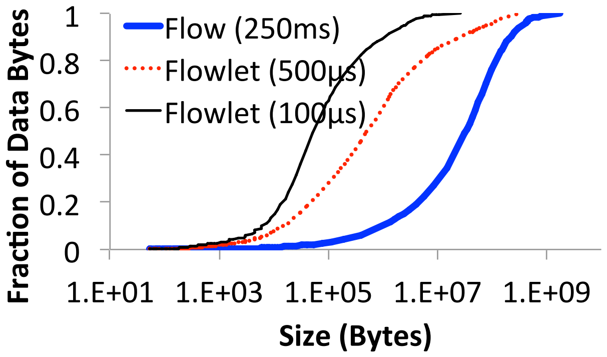


Figure 5: Distribution of data bytes across transfer sizes for different flowlet inactivity gaps.

hosts across ∼30 racks of servers to obtain packet traces of traffic flows through the network core. The cluster supports over 2000 diverse enterprise applications, including web, data-base, security, and business intelligence services. The captures are obtained by having a few leaf switches mirror traffic to an analyzer without disrupting production traffic. Overall, we analyze more than 150GB of compressed packet trace data.

**Flowlet size:** Figure 5 shows the distribution of the data bytes versus flowlet size for three choices of flowlet inactivity gap: 250ms, 500$\mu$s, and 100$\mu$s. Since it is unlikely that we would see a gap larger than 250ms in the same application-level flow, the line "Flow (250ms)" essentially corresponds to how the bytes are spread across flows. The plot shows that balancing flowlets gives significantly more fine-grained control than balancing flows. Even with an inactivity gap of 500$\mu$s, which is quite large and poses little risk of packet reordering in datacenters, we see nearly two orders of magnitude reduction in the size of transfers that cover most of the data: 50% of the bytes are in flows larger than ∼30MB, but this number reduces to ∼500KB for "Flowlet (500$\mu$s)".

**Flowlet concurrency:** Since the required flowlet inactivity gap is very small in datacenters, we expect there to be a small number of concurrent flowlets at any given time. Thus, the implementation cost for tracking flowlets should be low. To quantify this, we measure the distribution of the number of distinct 5-tuples in our packet trace over 1ms intervals. We find that the number of distinct 5-tuples (and thus flowlets) is small, with a median of 130 and a maximum under 300. Normalizing these numbers to the average throughput in our trace (∼15Gbps), we estimate that even for a very heavily loaded leaf switch with say 400Gbps of traffic, the number of concurrent flowlets would be less than 8K. Maintaining a table for tracking 64K flowlets is feasible at low cost (§3.4).

## 3 Design

Figure 6 shows the system diagram of CONGA. The majority of the functionality resides at the leaf switches. The source leaf makes load balancing decisions based on per uplink congestion metrics, derived by taking the maximum of the local congestion at the uplink and the remote congestion for the path (or paths) to the destination leaf that originate at the uplink. The remote metrics are stored in the Congestion-To-Leaf Table on a per destination leaf, per uplink basis and convey the maximum congestion for all the links along the path. The remote metrics are obtained via feedback from the destination leaf switch, which opportunistically piggybacks values in its Congestion-From-Leaf Table back to the source leaf. CONGA measures congestion using the Discounting Rate Estimator (DRE), a simple module present at each fabric link.

Load balancing decisions are made on the *first packet* of each flowlet. Subsequent packets use the same uplink as long as the flowlet remains active (there is not a sufficiently long gap). The source leaf uses the Flowlet Table to keep track of active flowlets and their chosen uplinks.

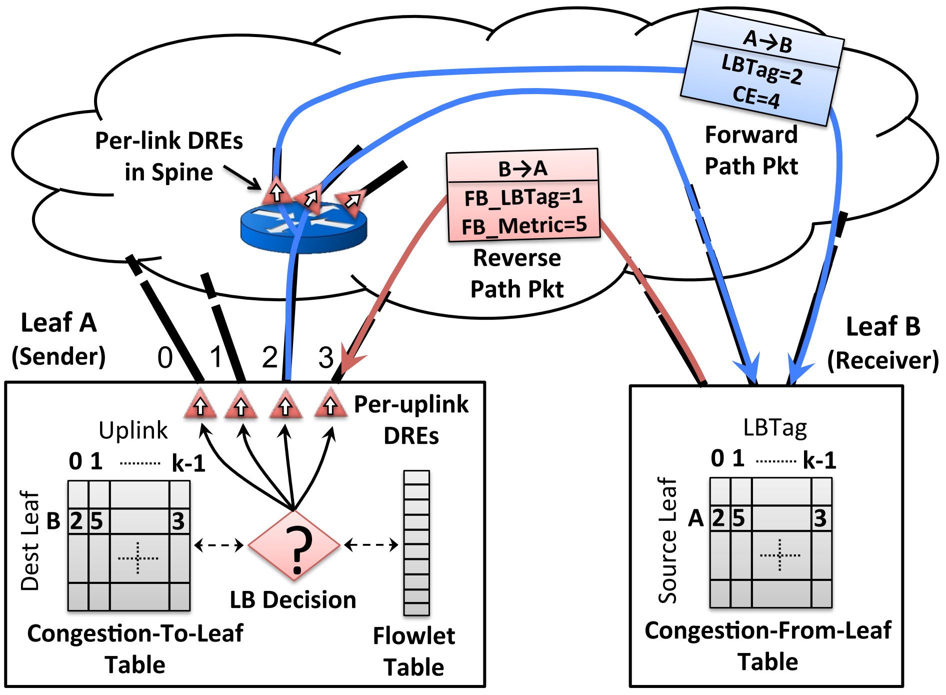


Figure 6: CONGA system diagram.

## 3.1 Packet format

CONGA leverages the VXLAN [34] encapsulation format used for the overlay to carry the following state:

- **LBTag (4 bits):** This field partially identifies the packet's path. It is set by the source leaf to the (switch-local) port number of the uplink the packet is sent on and is used by the destination leaf to aggregate congestion metrics before they are fed back to the source. For example, in Figure 6, the LBTag is 2 for both blue paths. Note that 4 bits is sufficient because the maximum number of leaf uplinks in our implementation is 12 for a non-oversubscribed configuration with $48 \times 10$Gbps server-facing ports and $12 \times 40$Gbps uplinks.
- **CE (3 bits):** This field is used by switches along the packet's path to convey the extent of congestion.
- **FB_LBTag (4 bits) and FB_Metric (3 bits):** These two fields are used by destination leaves to piggyback congestion information back to the source leaves. FB_LBTag indicates the LBTag the feedback is for and FB_Metric provides its associated congestion metric.

## 3.2 Discounting Rate Estimator (DRE)

The DRE is a simple module for measuring the load of a link(see Algorithm 1) The DRE maintains a register, $X$, which is incremented for each packet sent over the link by the packet size in bytes, and is decremented periodically (every $T_{dre}$) with a multiplicative factor $\alpha$ between 0 and 1: $X \leftarrow X \times (1-\alpha)$. It is easy to show that $X$ is proportional to the rate of traffic over the link; more precisely, if the traffic rate is $R$, then $X \approx R \cdot \tau$, where $\tau = T_{dre}/\alpha$. The DRE algorithm is essentially a first-order low pass filter applied to packet arrivals, with a $(1 - e^{-1})$ rise time of $\tau$. The congestion metric for the link is obtained by quantizing $X/C\tau$ to 3 bits ($C$ is the link speed).

The DRE algorithm is similar to the widely used Exponential Weighted Moving Average (EWMA) mechanism. However, DRE has two key advantages over EWMA: (i) it can be implemented with just one register (whereas EWMA requires two); and (ii) the DRE reacts more quickly to traffic bursts (because increments take place immediately upon packet arrivals) while retaining memory of past bursts.

**Algorithm 1** DRE algorithm for measuring congestion

```
**For each packet traversing the link:**
    $X \leftarrow X + packet.size$ (in bytes)
**Periodically with period $\mathbf{T_{dre}}$:**
    $X \leftarrow X \times (1 - \alpha)$
```

## 3.3 Congestion Feedback

CONGA uses a feedback loop between the source and destination leaf switches to populate the remote metrics in the Congestion-To-Leaf Table at each leaf switch. We now describe the sequence of events involved (refer to Figure 6 for an example).

1. The source leaf sends the packet to the fabric with the LBTag field set to the uplink port taken by the packet. It also sets the CE field to zero.
2. The packet is routed through the fabric to the destination leaf.[3] As it traverses each link, its CE field is updated if the link's congestion metric (given by the DRE) is larger than the current value in the packet.
3. The CE field of the packet received at the destination leaf gives the maximum link congestion along the packet's path. This needs to be fed back to the source leaf. But since a packet may not be immediately available in the reverse direction, the destination leaf stores the metric in the Congestion-From-Leaf Table (on a per source leaf, per LBTag basis) while it waits for an opportunity to piggyback the feedback.
4. When a packet is sent in the reverse direction, *one* metric from the Congestion-From-Leaf Table is inserted in its FB_LBTag and FB_Metric fields for the source leaf. The metric is chosen in round-robin fashion while, as an optimization, favoring those metrics whose values have changed since the last time they were fed back.
5. Finally, the source leaf parses the feedback in the reverse packet and updates the Congestion-To-Leaf Table.

It is important to note that though we have described the forward and reverse packets separately for simplicity, every packet *simultaneously* carries both a metric for its forward path and a feedback metric. Also, while we could generate explicit feedback packets, we decided to use piggybacking because we only need a very small number of packets for feedback. In fact, all metrics between a pair of leaf switches can be conveyed in at-most 12 packets (because there are 12 distinct LBTag values), with the average case being much smaller because the metrics only change at network round-trip timescales, not packet-timescales (see the discussion in §3.6 regarding the DRE time constant).

**Metric aging:** A potential issue with not having explicit feedback packets is that the metrics may become stale if sufficient traffic does not exit for piggybacking. To handle this, a simple aging mechanism is added where a metric that has not been updated for a long time (e.g., 10ms) gradually decays to zero. This also guarantees that a path that appears to be congested is eventually probed again.

## 3.4 Flowlet Detection

Flowlets are detected and tracked in the leaf switches using the Flowlet Table according to Algorithm 2 Each entry of the table consists of a port number, a valid bit, and an age bit. When a packet arrives, we lookup an entry based on a hash of its 5-tuple. If the entry is valid ($valid\_bit == 1$), the flowlet is active and the packet is sent on the port indicated in the entry. If the entry is not valid, the incoming packet starts a new flowlet. In this case, we make a load balancing decision (as described below) and cache the result in the table for use by subsequent packets. We also set the valid bit.

[3]If there are multiple valid next hops to the destination (as in Figure 6), the spine switches pick one using standard ECMP hashing.

**Algorithm 2** Flowlet detection algorithm.

```
For each packet:
$ENTRY \leftarrow$ FLOWLET_TABLE[ Hash(5-tuple) ]
if $ENTRY.valid\_bit == 1$ then
  The flowlet is active. Use uplink $ENTRY.port$ for packet.
else
  Packet starts a new flowlet. Make load balancing decision and store the result in $ENTRY.port$.
end if
$ENTRY.valid\_bit \leftarrow 1$
$ENTRY.age\_bit \leftarrow 0$

Periodically with period FLOWLET_TIMEOUT:
if $ENTRY.age\_bit == 1$ then
  $ENTRY.valid\_bit \leftarrow 0$
end if
$ENTRY.age\_bit \leftarrow 1$
```

Flowlet entries time out using the age bit. Each incoming packet resets the age bit. A timer periodically (every $T_{fl}$ seconds) checks the age bit before setting it. If the age bit is set when the timer checks it, then there have not been any packets for that entry in the last $T_{fl}$ seconds and the entry expires (the valid bit is set to zero). Note that a single age bit allows us to detect gaps between $T_{fl}$ and $2T_{fl}$. While not as accurate as using full timestamps, this requires far fewer bits allowing us to maintain a very large number of entries in the table (64K in our implementation) at low cost.

**Remark 1.** Although with hashing, flows may collide in the Flowlet Table, this is not a big concern. Collisions simply imply some load balancing opportunities are lost which does not matter as long as this does not occur too frequently.

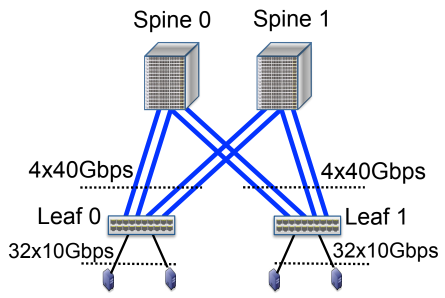
## 3.5 Load Balancing Decision Logic

Load balancing decisions are made on the first packet of each flowlet (other packets use the port cached in the Flowlet Table). For a new flowlet, we pick the uplink port that minimizes the maximum of the local metric (from the local DREs) and the remote metric (from the Congestion-To-Leaf Table). If multiple uplinks are equally good, one is chosen at random with preference given to the port cached in the (invalid) entry in the Flowlet Table; i.e., a flow only moves if there is a strictly better uplink than the one its last flowlet took.

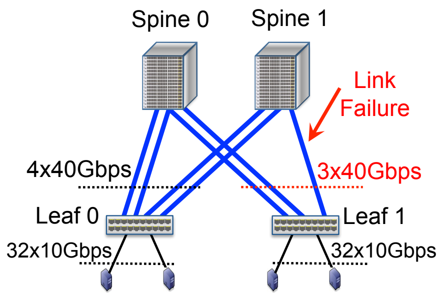
## 3.6 Parameter Choices

CONGA has three main parameters: (i) $Q$, the number of bits for quantizing congestion metrics (§3.1); (ii) $\tau$, the DRE time constant, given by $T_{dre}/\alpha$ (§3.2); and (iii) $T_{fl}$, the flowlet inactivity timeout (§3.4). We set these parameters experimentally. A control-theoretic analysis of CONGA is beyond the scope of this paper, but our parameter choices strike a balance between the stability and responsiveness of CONGA's control loop while taking into consideration practical matters such as interaction with TCP and implementation cost (header requirements, table size, etc).

The parameters $Q$ and $\tau$ determine the "gain" of CONGA's control loop and exhibit important tradeoffs. A large $Q$ improves congestion metric accuracy, but if too large can make the leaf switches over-react to minor differences in congestion causing oscillatory behavior. Similarly, a small $\tau$ makes the DRE respond quicker, but also makes it more susceptible to noise from transient traffic bursts. Intuitively, $\tau$ should be set larger than the network RTT to filter the sub-RTT traffic bursts of TCP. The flowlet timeout, $T_{fl}$, can be set to the maximum possible leaf-to-leaf latency to guarantee no packet reordering. This value can be rather large though because of worst-case queueing delays (e.g., 13ms in our testbed), essentially disabling flowlets. Reducing $T_{fl}$ presents a compromise between more packet reordering versus less congestion (fewer packet drops, lower latency) due to better load balancing.



(a) Baseline (no failure)



(b) With link failure

Figure 7: Topologies used in testbed experiments.

Overall, we have found CONGA's performance to be fairly robust with: $Q = 3$ to 6, $\tau = 100\mu$s to 500$\mu$s, and $T_{fl} = 300\mu$s to 1ms. The default parameter values for our implementation are: $Q = 3$, $\tau = 160\mu$s, and $T_{fl} = 500\mu$s.

## 4 Implementation

CONGA has been implemented in custom switching ASICs for a major datacenter fabric product line. The implementation includes ASICs for both the leaf and spine nodes. The Leaf ASIC implements flowlet detection, congestion metric tables, DREs, and the leaf-to-leaf feedback mechanism. The Spine ASIC implements the DRE and its associated congestion marking mechanism. The ASICs provide state-of-the-art switching capacities. For instance, the Leaf ASIC has a non-blocking switching capacity of 960Gbps in 28nm technology. CONGA's implementation requires roughly 2.4M gates and 2.8Mbits of memory in the Leaf ASIC, and consumes negligible die area ($< 2\%$).

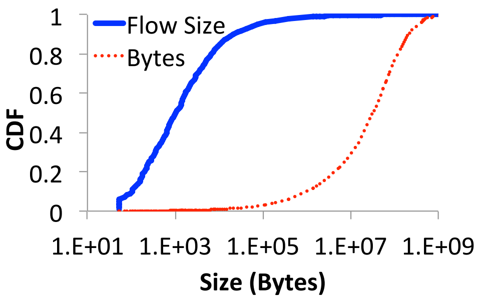
## 5 Evaluation

In this section, we evaluate CONGA's performance with a real hardware testbed as well as large-scale simulations. Our testbed experiments illustrate CONGA's good performance for realistic empirical workloads, Incast micro-benchmarks, and actual applications. Our detailed packet-level simulations confirm that CONGA scales to large topologies.

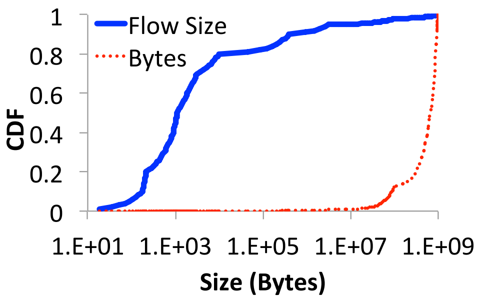
**Schemes compared:** We compare **CONGA**, **CONGA-Flow**, **ECMP**, and **MPTCP** [40], the state-of-the-art multipath transport protocol. CONGA and CONGA-Flow differ only in their choice of the flowlet inactivity timeout. CONGA uses the default value: $T_{fl} = 500\mu$s. CONGA-Flow however uses $T_{fl} = 13$ms which is greater than the maximum path latency in our testbed and ensures no packet reordering. CONGA-Flow's large timeout effectively implies one load balancing decision per flow, similar to ECMP. Of course, in contrast to ECMP, decisions in CONGA-Flow are informed by the path congestion metrics. The rest of the parameters for both variants of CONGA are set as described in §3.6. For MPTCP, we use MPTCP kernel v0.87 available on the web [36]. We configure MPTCP to use 8 sub-flows for each TCP connection, as Raiciu *et al.* [40] recommend.

### 5.1 Testbed

Our testbed consists of 64 servers and four switches (two leaves and two spines). As shown in Figure 7, the servers are organized in two racks (32 servers each) and attach to the leaf switches with 10Gbps links. In the baseline topology (Figure 7(a)), the leaf switches connect to each spine switch with two 40Gbps uplinks. Note that there is a 2:1 oversubscription at the Leaf level, typical of today's datacenter deployments. We also consider the *asymmetric* topology in Figure 7(b) where one of the links between Leaf 1 and Spine 1 is down. The servers have 12-core Intel Xeon X5670 2.93GHz CPUs, 128GB of RAM, and 3 2TB 7200RPM HDDs.



(a) Enterprise workload



(b) Data-mining workload

Figure 8: Empirical traffic distributions. The Bytes CDF shows the distribution of traffic bytes across different flow sizes.

## 5.2 Empirical Workloads

We begin with experiments with realistic workloads based on empirically observed traffic patterns in deployed datacenters. Specifically, we consider the two flow size distributions shown in Figure 8. The first distribution is derived from packet traces from our own datacenters (§2.6) and represents a large enterprise workload. The second distribution is from a large cluster running data mining jobs [17]. Note that both distributions are heavy-tailed: A small fraction of the flows contribute most of the data. Particularly, the data-mining distribution has a very heavy tail with 95% of all data bytes belonging to ∼3.6% of flows that are larger than 35MB.

We develop a simple client-server program to generate the traffic. The clients initiate 3 long-lived TCP connections to each server and request flows according to a Poisson process from randomly chosen servers. The request rate is chosen depending on the desired offered load and the flow sizes are sampled from one of the above distributions. All 64 nodes run both client and server processes. Since we are interested in stressing the fabric's load balancing, we configure the clients under Leaf 0 to only use the servers under Leaf 1 and vice-versa, thereby ensuring that all traffic traverses the Spine. Similar to prior work [19, 4, 12], we use the flow completion time (FCT) as our main performance metric.

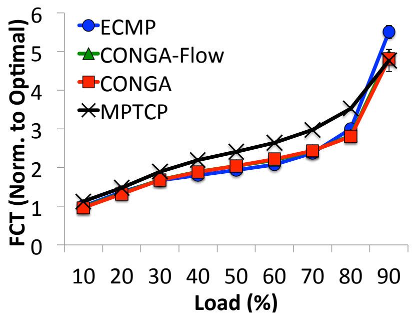
### 5.2.1 Baseline

Figures 9 and 10 show the results for the two workloads with the baseline topology (Figure 7(a)). Part (a) of each figure shows the overall average FCT for each scheme at traffic loads between 10–90%. The values here are normalized to the optimal FCT that is achievable in an idle network. In parts (b) and (c), we break down the FCT for each scheme for small ($< 100$KB) and large ($> 10$MB) flows, normalized to the value achieved by ECMP. Each data point is the average of 5 runs with the error bars showing the range. (Note that the error bars are not visible at all data points.)

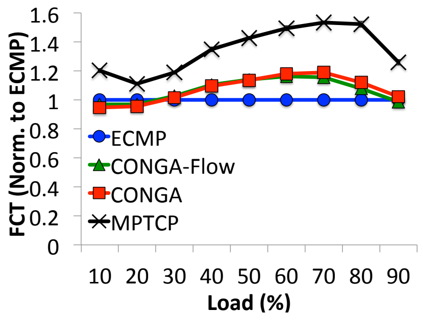
**Enterprise workload:** We find that the overall average FCT is similar for all schemes in the enterprise workload, except for MPTCP which is up to ∼25% worse than the others. MPTCP's higher overall average FCT is because of the small flows, for which it is up to ∼50% worse than ECMP. CONGA and similarly CONGA-Flow are slightly worse for small flows (∼12–19% at 50–80% load), but improve FCT for large flows by as much as ∼20%.

**Data-mining workload:** For the data-mining workload, ECMP is noticeably worse than the other schemes at the higher load levels. Both CONGA and MPTCP achieve up to ∼35% better overall average FCT than ECMP. Similar to the enterprise workload, we find a degradation in FCT for small flows with MPTCP compared to the other schemes.

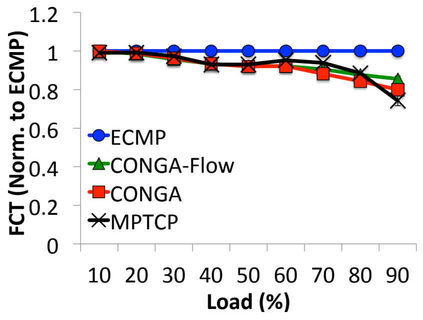
**Analysis:** The above results suggest a tradeoff with MPTCP between achieving good load balancing in the core of the fabric and managing congestion at the edge. Essentially, while using 8 sub-flows per connection helps MPTCP achieve better load balancing, it also increases congestion at the edge links because the multiple sub-flows cause more burstiness. Further, as observed by Raiciu *et al.* [40], the small sub-flow window sizes for short flows increases the chance of timeouts. This hurts the performance of short flows which are more sensitive to additional latency and packet drops. CONGA on the other



(a) Overall Average FCT

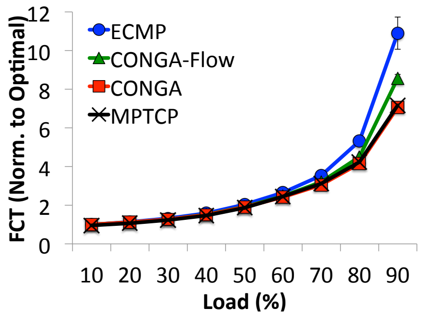


(b) Small Flows (< 100KB)

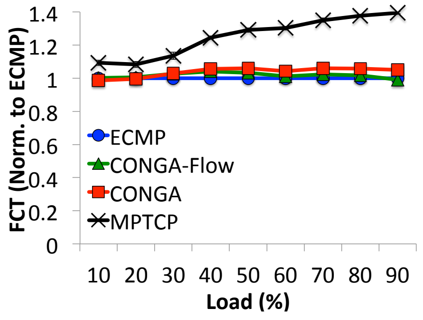


(c) Large Flows (> 10MB)

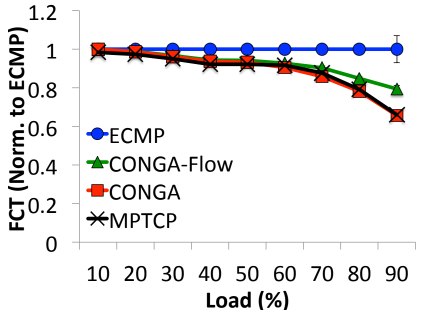
Figure 9: FCT statistics for the enterprise workload with the baseline topology. Note that part (a) is normalized to the optimal FCT, while parts (b) and (c) are normalized to the value achieved by ECMP. The results are the average of 5 runs.



(a) Overall Average FCT

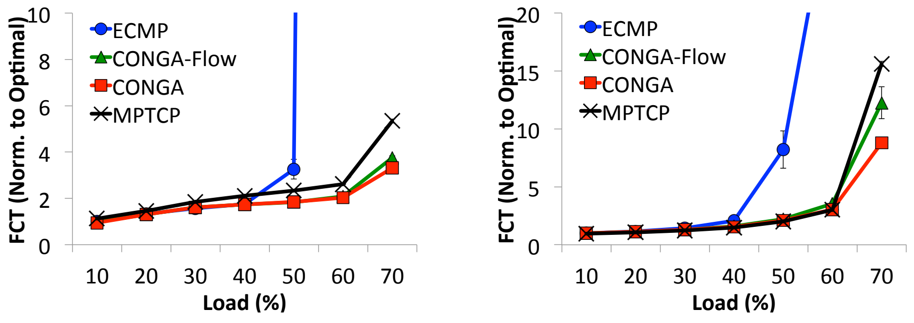


(b) Small Flows (< 100KB)

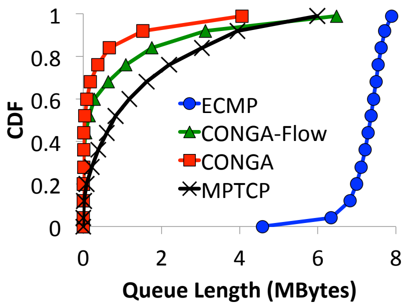


(c) Large Flows (> 10MB)

Figure 10: FCT statistics for the data-mining workload with the baseline topology. Note that part (a) is normalized to the optimal FCT, while parts (b) and (c) are normalized to the value achieved by ECMP. The results are the average of 5 runs.



(a) Enterprise workload: Avg FCT (b) Data-mining workload: Avg FCT



(c) Queue length at hotspot

Figure 11: Impact of link failure. Parts (a) and (b) show that overall average FCT for both workloads. Part (c) shows the CDF of queue length at the hotspot port [Spine1→Leaf1] for the data-mining workload at 60% load.

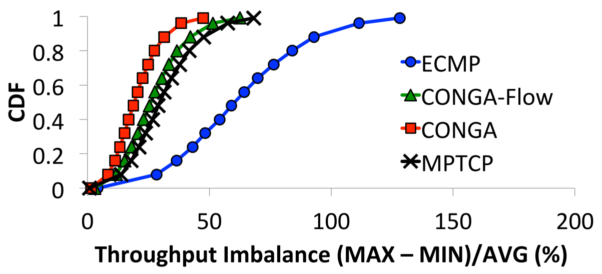
hand does not have this problem since it does not change the congestion control behavior of TCP.

Another interesting observation is the distinction between the enterprise and data-mining workloads. In the enterprise workload, ECMP actually does quite well leaving little room for improvement by the more sophisticated schemes. But for the data-mining workload, ECMP is noticeably worse. This is because the enterprise workload is less "heavy"; i.e., a significant fraction of the traffic in the enterprise workload is due to small flows. In particular, ∼50% of all data bytes in the enterprise workload are from flows that are smaller than 35MB (Figure 8(a)). But in the data mining workload, flows smaller than 35MB contribute only ∼5% of all bytes (Figure 8(b)). Hence, the data-mining workload is more challenging to handle from a load balancing perspective. We quantify the impact of the workload analytically in §6.2.

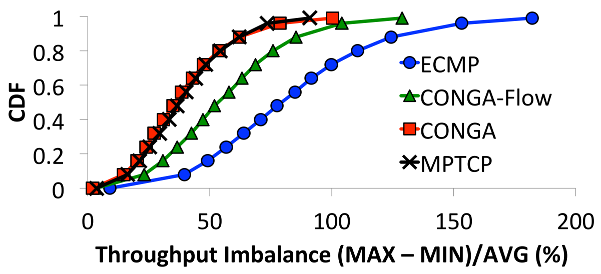
### 5.2.2 Impact of link failure

We now repeat both workloads for the asymmetric topology when a fabric link fails (Figure 7(b)). The overall average FCT for both workloads is shown in Figure 11. Note that since in this case the bisection bandwidth between Leaf 0 and Leaf 1 is 75% of the original capacity (3 × 40Gbps links instead of 4), we only consider offered loads up to 70%. As expected, ECMP's performance drastically deteriorates as the offered load increases beyond 50%. This is because with ECMP, half of the traffic from Leaf 0 to Leaf 1 is sent through Spine 1 (see Figure 7(b)). Therefore the single [Spine1→Leaf1] link must carry twice as much traffic as each of the [Spine0→Leaf1] links and becomes oversubscribed when the offered load exceeds 50%. At this point, the network effectively cannot handle the offered load and becomes unstable.

The other *adaptive* schemes are significantly better than ECMP with the asymmetric topology because they shift traffic to the lesser congested paths (through Spine 0) and can thus handle the offered load. CONGA is particularly robust to the link failure, achieving up to ∼30% lower overall average FCT than MPTCP in the enterprise workload and close to 2× lower in the data-mining workload at 70% load. CONGA-Flow is also better than MPTCP even though it does not split flows. This is



(a) Enterprise workload



(b) Data-mining workload

Figure 12: Extent of imbalance between throughput of leaf uplinks for both workloads at 60% load. The throughput samples are over 10ms intervals.

because CONGA *proactively* detects high utilization paths before congestion occurs and adjusts traffic accordingly, whereas MPTCP is *reactive* and only makes adjustments when sub-flows on congested paths experience packet drops. This is evident from Figure 11(c), which compares the queue occupancy at the hotspot port [Spine1→Leaf1] for the different schemes for the data-mining workload at 60% load. We see that CONGA controls the queue occupancy at the hotspot much more effectively than MPTCP, for instance, achieving a 4× smaller 90th percentile queue occupancy.

### 5.2.3 Load balancing efficiency

The FCT results of the previous section show end-to-end performance which is impacted by a variety of factors, including load balancing. We now focus specifically on CONGA's load balancing efficiency. Figure 12 shows the CDF of the *throughput imbalance* across the 4 uplinks at Leaf 0 in the baseline topology (without link failure) for both workloads at the 60% load level. The throughput imbalance is defined as the maximum throughput (among the 4 uplinks) minus the minimum divided by the average: $(MAX - MIN)/AVG$. This is calculated from synchronous samples of the throughput of the 4 uplinks over 10ms intervals, measured using a special debugging module in the ASIC.

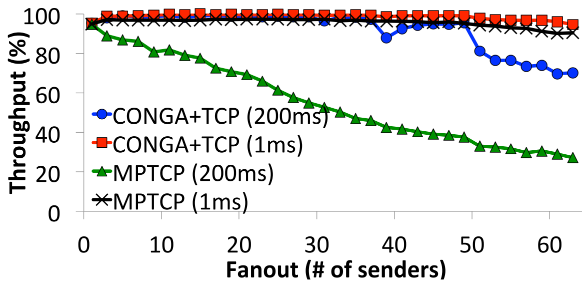
The results confirm that CONGA is at least as efficient as MPTCP for load balancing (without any TCP modifications) and is significantly better than ECMP. CONGA's throughput imbalance is even lower than MPTCP for the enterprise workload. With CONGA-Flow, the throughput imbalance is slightly better than MPTCP in the enterprise workload, but worse in the data-mining workload.

## 5.3 Incast

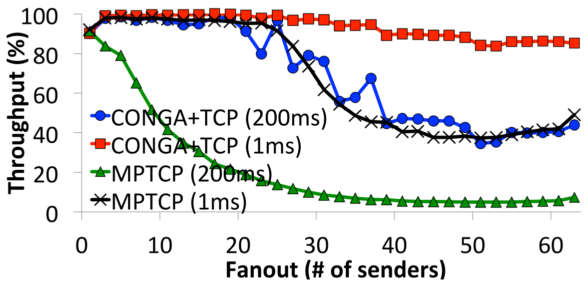
Our results in the previous section suggest that MPTCP increases congestion at the edge links because it opens 8 sub-flows per connection. We now dig deeper into this issue for Incast traffic patterns that are common in datacenters [3].

We use a simple application that generates the standard Incast traffic pattern considered in prior work [46, 11, 3]. A client process residing on one of the servers repeatedly makes requests for a 10MB file striped across $N$ other servers. The servers each respond with 10MB/$N$ of data in a synchronized fashion causing Incast. We measure the effective throughput at the client for different "fan-in" values, $N$, ranging from 1 to 63. Note that this experiment does not stress the fabric load balancing since the total traffic crossing the fabric is limited by the client's 10Gbps access link. The performance here is predominantly determined by the Incast behavior for the TCP or MPTCP transports.

The results are shown in Figure 13. We consider two minimum retransmission timeout ($minRTO$) values: 200ms (the default value in Linux) and 1ms (recommended by Vasudevan *et al.* [46] to cope with Incast); and two packet sizes ($MTU$): 1500B (the standard Ethernet frame size) and 9000B (jumbo frames). The plots confirm that MPTCP significantly degrades performance in Incast scenarios. For instance, with $minRTO$ = 200ms, MPTCP's throughput degrades to less than 30% for large fan-in with 1500B packets and just 5% with 9000B packets. Reducing $minRTO$ to 1ms mitigates the problem to some extent with standard packets, but even reducing $minRTO$ does not prevent significant throughput loss with jumbo frames. CONGA +TCP achieves 2–8× better throughput than MPTCP in similar settings.

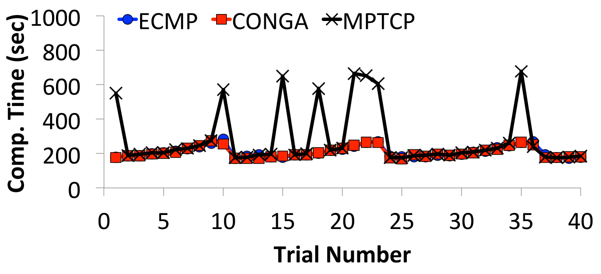


(a) MTU = 1500B

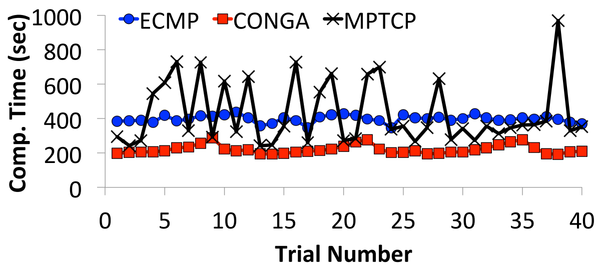


(b) MTU = 9000B

Figure 13: MPTCP significantly degrades performance in Incast scenarios, especially with large packet sizes (MTU).



(a) Baseline Topology (w/o link failure)



(b) Asymmetric Topology (with link failure)

Figure 14: HDFS IO benchmark.

## 5.4 HDFS Benchmark

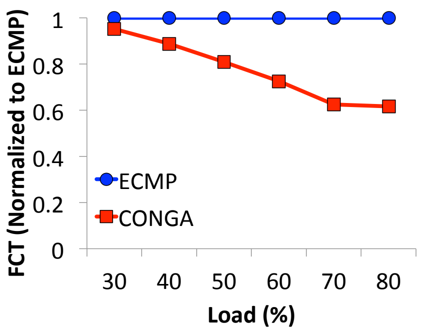
Our final testbed experiment evaluates the end-to-end impact of CONGA on a real application. We set up a 64 node Hadoop cluster (Cloudera distribution hadoop-0.20.2-cdh3u5) with 1 NameNode and 63 DataNodes and run the standard TestDFSIO benchmark included in the Apache Hadoop distribution [18]. The benchmark tests the overall IO performance of the cluster by spawning a MapReduce job which writes a 1TB file into HDFS (with 3-way replication) and measuring the total job completion time. We run the benchmark 40 times for both of the topologies in Figure 7 (with and without link failure). We found in our experiments that the TestDFSIO benchmark is *disk-bound* in our setup and does not produce enough traffic on its own to stress the network. Therefore, in the absence of servers with more or faster disks, we also generate some background traffic using the empirical enterprise workload described in §5.2.

Figure 14 shows the job completion times for all trials. We find that for the baseline topology (without failures), ECMP and CONGA have nearly identical performance. MPTCP has some outlier trials with much higher job completion times. For the asymmetric topology with link failure, the job completion times for ECMP are nearly twice as large as without the link failure. But CONGA is robust; comparing Figures 14 (a) and (b) shows that the link failure has almost no impact on the job completion times with CONGA. The performance with MPTCP is very volatile with the link failure. Though we cannot ascertain the exact root cause of this, we believe it is because of MPTCP's difficulties with Incast since the TestDFSIO benchmark creates a large number of concurrent transfers between the servers.

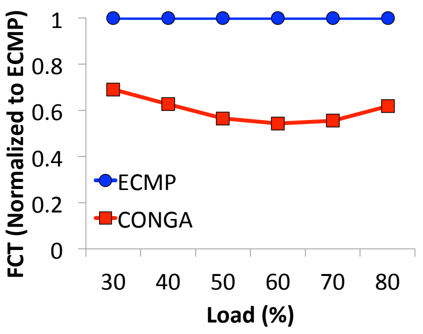
## 5.5 Large-Scale Simulations

Our experimental results were for a comparatively small topology (64 servers, 4 switches), 10Gbps access links, one link failure, and 2:1 oversubscription. But in developing CONGA, we also did extensive packet-level simulations, including exploring CONGA's parameter space (§3.6), different workloads, larger topologies, varying oversubscription ratios and link speeds, and varying degrees of asymmetry. We briefly summarize our main findings.

*Note:* We used OMNET++ [45] and the Network Simulation Cradle [23] to port the actual Linux



(a) 10Gbps access links



(b) 40Gbps access links

Figure 15: Overall average FCT for a simulated workload based on web search [3] for two topologies with 40Gbps fabric links, 3:1 oversubscription, and: (a) 384 10Gbps servers, (b) 96 40Gbps servers.

TCP source code (from kernel 2.6.26) to our simulator. Despite its challenges, we felt that capturing exact TCP behavior was crucial to evaluate CONGA, especially to accurately model flowlets. Since an OMNET++ model of MPTCP is unavailable and we did experimental comparisons, we did not simulate MPTCP.

**Varying fabric size and oversubscription:** We have simulated realistic datacenter workloads (including those presented in §5.2) for fabrics with as many as 384 servers, 8 leaf switches, and 12 spine switches, and for oversubscription ratios ranging from 1:1 to 5:1. We find qualitatively similar results to our testbed experiments at all scales. CONGA achieves ∼5–40% better FCT than ECMP in symmetric topologies, with the benefit larger for heavy flow size distributions, high load, and high oversubscription ratios.

**Varying link speeds:** CONGA's improvement over ECMP is more pronounced and occurs at lower load levels the closer the access link speed (at the servers) is to the fabric link speed. For example, in topologies with 40Gbps links everywhere, CONGA achieves 30% better FCT than ECMP even at 30% traffic load, while for topologies with 10Gbps edge and 40Gbps fabric links, the improvement at 30% load is typically 5–10% (Figure 15). This is because in the latter case, each fabric link can support multiple (at least 4) flows without congestion. Therefore, the impact of hash collisions in ECMP is less pronounced in this case (see also [40]).

**Varying asymmetry:** Across tens of asymmetrical scenarios we simulated (e.g., with different number of random failures, fabrics with both 10Gbps and 40Gbps links, etc) CONGA achieves near optimal traffic balance. For example, in the representative scenario shown in Figure 16, the queues at links 10 and 12 in the spine (which are adjacent to the failed link 11) are ∼10× larger with ECMP than CONGA.

## 6 Analysis

In this section we complement our experimental results with analysis of CONGA along two axes. First, we consider a game-theoretic model known as the *bottleneck routing game* [5] in the context of CONGA to quantify its Price of Anarchy [39]. Next, we consider a stochastic model that gives insights into the impact of the traffic distribution and flowlets on load balancing performance.

### 6.1 The Price of Anarchy for CONGA

In CONGA, the leaf switches make independent load balancing decisions to minimize the congestion for their own traffic. This uncoordinated or "selfish" behavior can potentially cause inefficiency in comparison to having centralized coordination. This inefficiency is known as the Price of Anarchy (PoA) [39] and has been the subject of numerous studies in the theoretical literature (see [42] for a survey). In this section we analyze the PoA for CONGA using the bottleneck routing game model introduced by Banner *et al.* [5].

**Model:** We consider a two-tier Leaf-Spine network modeled as a bipartite graph $G = (V, E)$. The leaf and spine nodes are connected with links of arbitrary capacities $\{c_e\}$. The network is shared by a set of

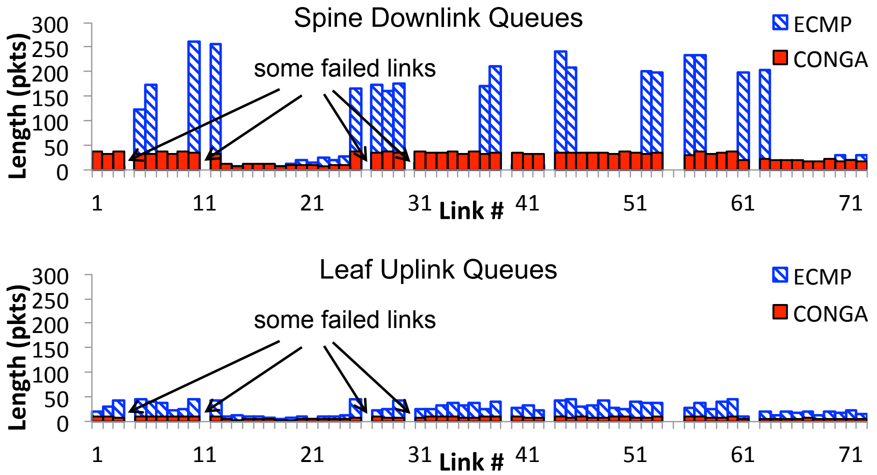


Figure 16: Multiple link failure scenario in a 288-port fabric with 6 leaves and 4 spines. Each leaf and spine connect with **3** $\times$ **40**Gbps links and 9 randomly chosen links fail. The plots show the average queue length at all fabric ports for a web search workload at 60% load. CONGA balances traffic significantly better than ECMP. Note that the improvement over ECMP is much larger at the (remote) spine downlinks because ECMP spreads load equally on the leaf uplinks.

users $U$. Each user $u \in U$ has a traffic demand $\gamma_u$, to be sent from source leaf, $s_u$, to destination leaf, $t_u$. A user must decide how to split its traffic along the paths $P^{(s_u,t_u)}$ through the different spine nodes. Denote by $f_p^u$ the flow of user $u$ on path $p$. The set of all user flows, $f = \{f_p^u\}$, is termed the *network flow*. A network flow is *feasible* if $\sum_{p \in P^{(s_u,t_u)}} f_p^u = \gamma_u$ for all $u \in U$. Let $f_p = \sum_{u \in U} f_p^u$ be the total flow on a path and $f_e = \sum_{p|e \in p} f_p$ be the total flow on a link. Also, let $\rho_e(f_e) = f_e/c_e$ be the utilization of link $e$. We define the *network bottleneck* for a network flow to be the utilization of the most congested link; i.e., $B(f) \triangleq \max_{e \in E} \rho_e(f_e)$. Similarly, we define the bottleneck for user $u$ to be the utilization of the most congested link it uses; i.e., $b_u(f) \triangleq \max_{e \in E | f_e^u > 0} \rho_e(f_e)$.

We model CONGA as a non-cooperative game where each user selfishly routes its traffic to minimize its own bottleneck; i.e., it only sends traffic along the paths with the smallest bottleneck available to it. The network flow, $f$, is said to be at *Nash equilibrium* if no user can improve its bottleneck given the choices of the other users. Specifically, $f$ is a Nash flow if for each user $u^* \in U$ and network flow $g = \{g_p^u\}$ such that $g_p^u = f_p^u$ for each $u \in U \backslash u^*$, we have: $b_{u^*}(f) \leq b_{u^*}(g)$. The game above always admits at least one Nash flow (Theorem 1 in [5]). Note that CONGA converges to a Nash flow because the algorithm rebalances traffic (between a particular source/destination leaf) if there is a path with a smaller bottleneck available.[4] The PoA is defined as the worst-case ratio of the network bottleneck for a Nash flow and the minimum network bottleneck achievable by any flow. We have the following theorem.

**Theorem 1.** *The PoA for the CONGA game is 2.*

The proof is given in Appendix A. Theorem 1 proves that the network bottleneck with CONGA is at most twice the optimal. It is important to note that this is the *worst-case* and can only occur in very specific (and artificial) scenarios such as the example in Figure 17. As our experimental evaluation shows (§5), in practice the performance of CONGA is much closer to optimal.

**Remark 2.** If in addition to using paths with the smallest congestion metric, if the leaf switches also use only paths with *the fewest number* of bottlenecks, then the PoA would be 1 [5], i.e., optimal. For example, the flow in Figure 17(a) does not satisfy this condition since traffic is routed along paths with two bottlenecks even though alternate paths with a single bottleneck exist. Incorporating this criteria into CONGA would require additional bits to track the number of bottlenecks along the packet's path. Since our evaluation (§5) has not shown much potential gains, we have not added this refinement to our implementation.

[4]Of course, this is an idealization and ignores artifacts such as flowlets, quantized congestion metrics, etc.

## 6.2 Impact of Workload on Load Balancing

Our evaluation showed that for some workloads ECMP actually performs quite well and the benefits of more sophisticated schemes (such as MPTCP and CONGA) are limited in symmetric topologies. This raises the question: *How does the workload affect load balancing performance?* Importantly, *when are flowlets helpful?*

We now study these questions using a simple stochastic model. Flows with an arbitrary size distribution, $S$, arrive according to a Poisson process of rate $\lambda$ and are spread across $n$ links. For each flow, a random link is chosen. Let $N^k(t)$ and $A^k(t)$ respectively denote the number of flows and the total amount of traffic sent on link $k$ in time interval $(0, t)$; i.e., $A^k(t) = \sum_{i=1}^{N^k(t)} s_i^k$, where $s_i^k$ is the size of the $i^{th}$ flow assigned to link $k$. Let $Y(t) = \min_{1 \le k \le n} A^k(t)$ and $Z(t) = \max_{1 \le k \le n} A^k(t)$ and define the *traffic imbalance*:

$$\chi(t) \triangleq \frac{Z(t) - W(t)}{\frac{\lambda \mathrm{E}(S)}{n} t}.$$

This is basically the deviation between the traffic on the maximum and minimum loaded links normalized by the expected traffic on each link. The following theorem bounds the expected traffic imbalance at time $t$.

**Theorem 2.** *Assume* $\mathrm{E}(e^{\theta S}) < \infty$ *in a neighborhood of zero:* $\theta \in (-\theta_0, \theta_0)$*. Then for t sufficiently large:*

$$\mathrm{E}(\chi(t)) \le \frac{1}{\sqrt{\lambda_e t}} + O(\frac{1}{t}), \tag{1}$$

*where:*

$$\lambda_e = \frac{\lambda}{8n \log n \left(1 + (\frac{\sigma_S}{\mathrm{E}(S)})^2\right)}. \tag{2}$$

*Here $\sigma_S$ is the standard deviation of $S$.*

*Note:* Kandula *et al.* [26] prove a similar result for the deviation of traffic on *a single link* from its expectation while Theorem 2 bounds the traffic imbalance *across all links.*

The proof follows standard probabilistic arguments and is given in Appendix B. Theorem 2 shows that the traffic imbalance with randomized load balancing vanishes like $1/\sqrt{t}$ for large $t$. Moreover, the leading term is determined by the *effective arrival rate*, $\lambda_e$, that depends on two factors: (i) the *per-link flow arrival rate*, $\lambda/n$; and (ii) the *coefficient of variation* of the flow size distribution, $\sigma_S/\mathrm{E}(S)$. Theorem 2 quantifies the intuition that workloads that consists mostly of small flows are easier to handle than workloads with a few large flows. Indeed, it is for the latter "heavy" workloads that we can anticipate flowlets to make a difference. This is consistent with our experiments which show that ECMP does well for the enterprise workload (Figure 9), but for the heavier data mining workload CONGA is much better than ECMP and is even notably better than CONGA-Flow (Figure 10).

# 7 Discussion

We make a few remarks on aspects not covered thus far.

**Incremental deployment:** An interesting consequence of CONGA's resilience to asymmetry is that it does not need to be applied to *all* traffic. Traffic that is not controlled by CONGA simply creates *bandwidth asymmetry* to which (like topology asymmetry) CONGA can adapt. This facilitates incremental deployment since some leaf switches can use ECMP or any other scheme. Regardless, CONGA reduces fabric congestion to the benefit of all traffic.

**Larger topologies:** While 2-tier Leaf-Spine topologies suffice for most enterprise deployments, the largest *mega* datacenters require networks with 3 or more tiers. CONGA may not achieve the optimal traffic balance in such cases since it only controls the load balancing decision at the leaf switches (recall that the spine switches use ECMP). However, large datacenter networks are typically organized as multiple *pods*, each of which is a 2-tier Clos [1, 17]. Therefore, CONGA is beneficial even in these cases since it balances the traffic within each pod optimally, which also reduces congestion for inter-pod traffic. Moreover, even for inter-pod traffic, CONGA makes better decisions than ECMP at the first-hop.

A possible approach to generalizing CONGA to larger topologies is to pick a sufficient number of "good" paths between each pair of leaf switches and use leaf-to-leaf feedback to balance traffic across them. While we cannot cover all paths in general (as we do in the 2-tier case), the theoretical literature suggests that simple path selection policies such as periodically sampling a random path and retaining the best paths may perform very well [29]. We leave to future work a full exploration of CONGA in larger topologies.

**Other path metrics:** We used the *max* of the link congestion metrics as the path metric, but of course other choices such as the *sum* of link metrics are also possible and can easily be incorporated in CONGA. Indeed, in theory, the sum metric gives a Price of Anarchy (PoA) of 4/3 in *arbitrary* topologies [42]. Of course, the PoA is for the worst-case (adversarial) scenario. Hence, while the PoA is better for the sum metric than the max, like prior work (e.g., TeXCP [25]), we used max because it emphasizes the bottleneck link and is also easier to implement; the sum metric requires extra bits in the packet header to prevent overflow when doing additions.

## 8 Related Work

We briefly discuss related work that has informed and inspired our design, especially work not previously discussed.

Traditional traffic engineering mechanisms for wide-area networks [14, 13, 41, 48] use centralized algorithms operating at coarse timescales (hours) based on long-term estimates of traffic matrices. More recently, B4 [22] and SWAN [20] have shown near optimal traffic engineering for inter-datacenter WANs. These systems leverage the relative predictability of WAN traffic (operating over minutes) and are not designed for highly volatile datacenter networks. CONGA is conceptually similar to TeXCP [25] which dynamically load balances traffic between ingress-egress routers (in a wide-area network) based on path-wise congestion metrics. CONGA however is significantly simpler (while also being near optimal) so that it can be implemented directly in switch hardware and operate at microsecond time-scales, unlike TeXCP which is designed to be implemented in router software.

Besides Hedera [2], work such as MicroTE [8] proposes centralized load balancing for datacenters and has the same issues with handling traffic volatility. F10 [33] also uses a centralized scheduler, but uses a novel network topology to optimize failure recovery, not load balancing. The Incast issues we point out for MPTCP [40] can potentially be mitigated by additional mechanisms such as explicit congestion signals (e.g., XMP [10]), but complex transports are challenging to validate and deploy in practice.

A number of papers target the coarse granularity of flow-based balancing in ECMP. Besides Flare [26], LocalFlow [43] proposes spatial flow splitting based on TCP sequence numbers and DRB [9] proposes efficient per-packet round-robin load balancing. As explained in §2.4, such local schemes may interact poorly with TCP in the presence of asymmetry and perform worse than ECMP. DeTail [51] proposes a per-packet adaptive routing scheme that handles asymmetry using layer-2 back-pressure, but requires end-host modifications to deal with packet reordering.

## 9 Conclusion

The main thesis of this paper is that datacenter load balancing is best done in the network instead of the transport layer, and requires global congestion-awareness to handle asymmetry. Through extensive evaluation with a real testbed, we demonstrate that CONGA provides better performance than MPTCP [40], the state-of-the-art multipath transport for datacenter load balancing, without important drawbacks such as complexity and rigidity at the transport layer and poor Incast performance. Further, unlike local schemes such as ECMP and Flare [26], CONGA seamlessly handles asymmetries in the topology or network traffic. CONGA's resilience to asymmetry also paid unexpected dividends when it came to incremental deployment. Even if some switches within the fabric use ECMP (or any other mechanism), CONGA's traffic can work around bandwidth asymmetries and benefit all traffic.

CONGA leverages an existing datacenter overlay to implement a leaf-to-leaf feedback loop and can be deployed without any modifications to the TCP stack. While leaf-to-leaf feedback is not always the best strategy, it is near-optimal for 2-tier Leaf-Spine fabrics that suffice for the vast majority of deployed datacenters. It is also simple enough to implement in hardware as proven by our implementation in custom silicon.

In summary, CONGA senses the distant drumbeats of remote congestion and orchestrates flowlets to disperse evenly through the fabric. We leave to future work the task of designing more intricate rhythms for more general topologies.

**Acknowledgments:** We are grateful to Georges Akis, Luca Cafiero, Krishna Doddapaneni, Mike Herbert, Prem Jain, Soni Jiandani, Mario Mazzola, Sameer Merchant, Satyam Sinha, Michael Smith, and Pauline Shuen of Insieme Networks for their valuable feedback during the development of CONGA. We would also like to thank Vimalkumar Jeyakumar, Peter Newman, Michael Smith, our shepherd Teemu Koponen, and the anonymous SIGCOMM reviewers whose comments helped us improve the paper.

## References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

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

[3] M. Alizadeh et al. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[4] M. Alizadeh et al. pFabric: Minimal Near-optimal Datacenter Transport. In *SIGCOMM*, 2013.

[5] R. Banner and A. Orda. Bottleneck Routing Games in Communication Networks. *Selected Areas in Communications, IEEE Journal on*, 25(6):1173–1179, 2007.

[6] M. Beck and M. Kagan. Performance Evaluation of the RDMA over Ethernet (RoCE) Standard in Enterprise Data Centers Infrastructure. In *DC-CaVES*, 2011.

[7] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *SIGCOMM*, 2010.

[8] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT*, 2011.

[9] J. Cao et al. Per-packet Load-balanced, Low-latency Routing for Clos-based Data Center Networks. In *CoNEXT*, 2013.

[10] Y. Cao, M. Xu, X. Fu, and E. Dong. Explicit Multipath Congestion Control for Data Center Networks. In *CoNEXT*, 2013.

[11] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP Incast Throughput Collapse in Datacenter Networks. In *WREN*, 2009.

[12] N. Dukkipati and N. McKeown. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.*, 2006.

[13] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *INFOCOM*, 2001.

[14] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM*, 2000.

[15] R. Gallager. A Minimum Delay Routing Algorithm Using Distributed Computation. *Communications, IEEE Transactions on*, 1977.

[16] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *SIGCOMM*, 2011.

[17] A. Greenberg et al. VL2: a scalable and flexible data center network. In *SIGCOMM*, 2009.

[18] Apache Hadoop. `http://hadoop.apache.org/`.

[19] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing Flows Quickly with Preemptive Scheduling. In *SIGCOMM*, 2012.

[20] C.-Y. Hong et al. Achieving High Utilization with Software-driven WAN. In *SIGCOMM*, 2013.

[21] R. Jain and S. Paul. Network virtualization and software defined networking for cloud computing: a survey. *Communications Magazine, IEEE*, 51(11):24–31, 2013.

[22] S. Jain et al. B4: Experience with a Globally-deployed Software Defined Wan. In *SIGCOMM*, 2013.

[23] S. Jansen and A. McGregor. Performance, Validation and Testing with the Network Simulation Cradle. In *MASCOTS*, 2006.

[24] V. Jeyakumar et al. EyeQ: Practical Network Performance Isolation at the Edge. In *NSDI*, 2013.

[25] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *SIGCOMM*, 2005.

[26] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic Load Balancing Without Packet Reordering. *SIGCOMM Comput. Commun. Rev.*, 37(2):51–62, Mar. 2007.

[27] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *IMC*, 2009.

[28] R. Kapoor et al. Bullet Trains: A Study of NIC Burst Behavior at Microsecond Timescales. In *CoNEXT*, 2013.

[29] P. Key, L. Massoulié, and D. Towsley. Path Selection and Multipath Congestion Control. *Commun. ACM*, 54(1):109–116, Jan. 2011.

[30] A. Khanna and J. Zinky. The Revised ARPANET Routing Metric. In *SIGCOMM*, 1989.

[31] M. Kodialam, T. V. Lakshman, J. B. Orlin, and S. Sengupta. Oblivious Routing of Highly Variable Traffic in Service Overlays and IP Backbones. *IEEE/ACM Trans. Netw.*, 17(2):459–472, Apr. 2009.

[32] T. Koponen et al. Network Virtualization in Multi-tenant Datacenters. In *NSDI*, 2014.

[33] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A Fault-tolerant Engineered Network. In *NSDI*, 2013.

[34] M. Mahalingam et al. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. `http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-06`, 2013.

[35] N. Michael, A. Tang, and D. Xu. Optimal link-state hop-by-hop routing. In *ICNP*, 2013.

[36] MultiPath TCP - Linux Kernel implementation. `http://www.multipath-tcp.org/`.

[37] T. Narten et al. Problem Statement: Overlays for Network Virtualization. `http://tools.ietf.org/html/draft-ietf-nvo3-overlay-problem-statement-04`, 2013.

[38] J. Ousterhout et al. The case for RAMCloud. *Commun. ACM*, 54, July 2011.

[39] C. Papadimitriou. Algorithms, Games, and the Internet. In *Proc. of STOC*, 2001.

[40] C. Raiciu et al. Improving datacenter performance and robustness with multipath tcp. In *SIGCOMM*, 2011.

[41] M. Roughan, M. Thorup, and Y. Zhang. Traffic engineering with estimated traffic matrices. In *IMC*, 2003.

[42] T. Roughgarden. *Selfish Routing and the Price of Anarchy.* The MIT Press, 2005.

[43] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, Optimal Flow Routing in Datacenters via Local Link Balancing. In *CoNEXT*, 2013.

[44] M. Sridharan et al. NVGRE: Network Virtualization using Generic Routing Encapsulation. `http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-03`, 2013.

[45] A. Varga et al. The OMNeT++ discrete event simulation system. In *ESM*, 2001.

[46] V. Vasudevan et al. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *SIGCOMM*, 2009.

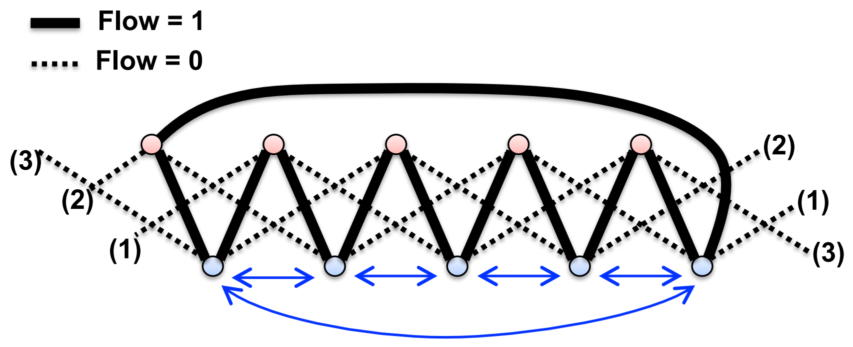
[47] S. Vutukury and J. J. Garcia-Luna-Aceves. A Simple Approximation to Minimum-delay Routing. In *SIGCOMM*, 1999.

[48] H. Wang et al. COPE: Traffic Engineering in Dynamic Networks. In *SIGCOMM*, 2006.

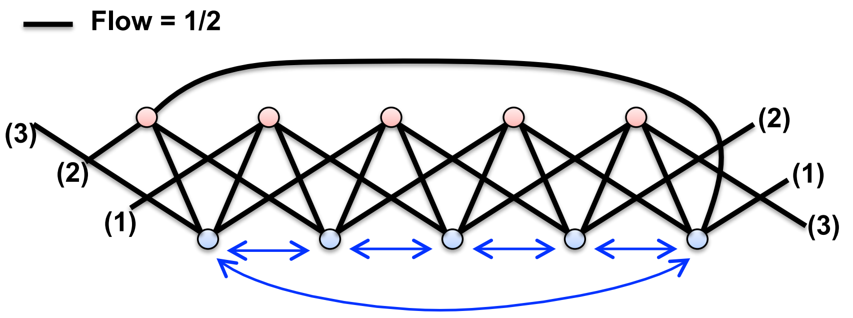
[49] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *NSDI*, 2011.

[50] D. Xu, M. Chiang, and J. Rexford. Link-state Routing with Hop-by-hop Forwarding Can Achieve Optimal Traffic Engineering. *IEEE/ACM Trans. Netw.*, 19(6):1717–1730, Dec. 2011.

[51] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. H. Katz. DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks. In *SIGCOMM*, 2012.



(a) Nash flow



(b) Optimal flow

Figure 17: Example with PoA of 2. All edges have capacity 1. Each pair of adjacent leaves (marked by the blue arrows) send 1 unit of traffic to each other. (a) Nash flow with a network bottleneck of 1. (b) Optimal flow with a network bottleneck of 1/2, where each pair's traffic is spread across two paths with an unused (dashed) edge in the Nash flow. Note that the two ends of the edges marked by (1), (2), and (3) are drawn separately.

## A Proof of Theorem 1

### A.1 PoA $\geq 2$

A lower bound of 2 for the PoA follows from the example shown in Figure 17. The example consists of 5 leaf nodes and 5 spine nodes. Each pair of adjacent leaves and also the left- and right-most leaves send 1 unit of flow to each other. Figure 17(a) shows a Nash flow with a network bottleneck of 1. Note that each source destination pair uses a single path in this flow and half of the edges are unused. However, if the pairs move their traffic to 2 paths, as shown in Figure 17(b), all links are used and a network bottleneck of 1/2 can be achieved.

### A.2 PoA $\leq 2$

We now prove that the PoA is at most 2. The proof follows from a similar argument to that given by Banner *et al.* [5] (Corollary 2 in Section V-B), which consider bottleneck routing games on general graphs. We adapt the argument for the CONGA game, in particular, leveraging the fact that paths in the Leaf-Spine network have length 2. We begin with a few definitions.

For a network flow $f = \{f_p^u\}$ and a path $p \in P$, let $N_f(p)$ be the number of bottleneck links on path $p$ with respect to $f$; i.e., $N_f(p) \triangleq |\{e \in p \mid \rho_e(f_e) = B(f)\}|$, where $|\cdot|$ is the set cardinality. Also, let $\Gamma(f)$ denote the set of users that send traffic through at least one network bottleneck; i.e., $\Gamma(f) \triangleq \{u \in U \mid b_u(f) = B(f)\}$.

**Definition 1.***[Banner* et al. *[5]] The network flow $g = \{g_p^u\}$ is said to be induced by $f$, if and only if*

$$g_p^u = \begin{cases} f_p^u & \text{if } u \in \Gamma(f) \\ 0 & \text{otherwise} \end{cases}$$

*Hence, $g$ consists of exactly the traffic for those users in $f$ that use at least one bottleneck.*

The following are straightforward consequences of this definition.

**Lemma 1.** *If $g$ is induced by $f$, then:*

1. *The network bottleneck of $g$ equals that of $f$: $B(g) = B(f)$.*
2. *The set of bottleneck links with respect to $f$ and $g$ are the same.*
3. *$g$ is feasible for users $u \in \Gamma(f)$;. i.e., it satisfies their demands: $\sum_{p \in P^{(s_u,t_u)}} g_p^u = \gamma_u$ for all $u \in \Gamma(f)$.*
4. *If $f$ is a Nash flow for users $u \in U$, then $g$ is a Nash flow for users $u \in \Gamma(f)$.*

*Proof.* See Lemma A.1 and Lemma 4 in [5]. □

Consider an arbitrary Nash flow $f$ of the CONGA game. We wish to show that the network bottleneck of $f$ is within a factor of 2 of the optimal network bottleneck. The following lemma states that it is sufficient to prove this for the flow $g$ induced by $f$.

**Lemma 2.** *Let $g$ be induced by a Nash flow $f$. If $B(g) \leq 2B(g^*)$, then $B(f) \leq 2B(f^*)$, where $g^*$ and $f^*$ are the optimal flows (that minimize the network bottleneck) among all feasible flows for users in $\Gamma(f)$ and $U$ respectively.*

*Proof.* Using Lemma 1, $B(g) = B(f)$. Also, $B(g^*) \leq B(f^*)$ because $f^*$ must satisfy the demands of all users that $g^*$ satisfies (and possibly more). Hence,

$$B(f) = B(g) \leq 2B(g^*) \leq 2B(f^*),$$

which completes the proof. □

Henceforth, we restrict our attention to $g$. Assume there exists a network flow $h$ which satisfies the demands of users $u \in \Gamma(f)$ (and only those users), and for which $B(g) > 2B(h)$. We will show that this is not possible by contradiction, thereby proving the upper bound of 2 on the PoA for $f$ using Lemma 2.

Let $\bar{E}$ be the set of links that are bottlenecks with respect to $g$; i.e., $\bar{E} = \{e \in E \mid \rho_e(g_e) = B(g)\}$. Also, let $P(e)$ denote the set of all paths that traverse a link $e$. Since $B(g) > 2B(h)$, it follows that $\rho_e(g_e) > 2\rho_e(h_e)$ for each $e \in \bar{E}$, and consequently that $g_e > 2h_e$ for each $e \in \bar{E}$ (recall that $\rho_e(x) = x/c_e$). Thus, for each $e \in \bar{E}$,

$$\sum_{p \in P(e)} g_p > 2 \sum_{p \in P(e)} h_p.$$

Summing these inequalities for all $e \in \bar{E}$, we have

$$\sum_{e \in \bar{E}} \sum_{p \in P(e)} g_p > 2 \sum_{e \in \bar{E}} \sum_{p \in P(e)} h_p. \tag{3}$$

Now let $\bar{P} \triangleq \bigcup_{e \in \bar{E}} P(e)$ be the set of all paths that traverse a bottleneck link in $\bar{E}$. Equation (3) can be rewritten as

$$\sum_{p \in \bar{P}} g_p N_g(p) > 2 \sum_{p \in \bar{P}} h_p N_g(p).$$

Recall that $N_g(p)$ is the number of bottleneck links on path $p$ with respect to $g$. Using the fact that $N_g(p) \leq 2$ for any all $p$ (because the paths in a Leaf-Spine network have length 2), and that $N_g(p) \geq 1$ for $p \in \bar{P}$, we have

$$2 \sum_{p \in \bar{P}} g_p \geq \sum_{p \in \bar{P}} g_p N_g(p) > 2 \sum_{p \in \bar{P}} h_p N_g(p) \geq 2 \sum_{p \in \bar{P}} h_p,$$

and therefore

$$\sum_{p \in \bar{P}} g_p > \sum_{p \in \bar{P}} h_p. \tag{4}$$

In the rest of the proof, we proceed to show that Equation (4) contradicts the feasibility of both $g$ and $h$ for users in $\Gamma(f)$. Define $Q \triangleq \bigcup_{u \in \Gamma(f)} P^{(s_u,t_u)}$ as the set of all paths for users $u \in \Gamma(f)$. Note that $Q \subseteq \bar{P}$. This is because each path $p \in P^{(s_u,t_u)}$ for a user $u \in \Gamma(f)$ must traverse at least one bottleneck link in $\bar{E}$; otherwise, $u$ could reduce its bottleneck by shifting some of its traffic to the path $p$, contradicting the fact that $g$ is a Nash flow (Lemma 1). Further, since the network flows $g$ and $h$ only contain traffic from users $u \in \Gamma(f)$, we have

$$\sum_{p \in \bar{P} \setminus Q} g_p = \sum_{p \in \bar{P} \setminus Q} h_p = 0.$$

This combined with Equation (4) gives

$$\sum_{p \in Q} g_p > \sum_{p \in Q} h_p,$$

which is equivalent to

$$\sum_{u \in \Gamma(f)} \sum_{p \in P^{(s_u,t_u)}} g_p > \sum_{u \in \Gamma(f)} \sum_{p \in P^{(s_u,t_u)}} h_p. \tag{5}$$

But since $g$ and $h$ are feasible for users $u \in \Gamma(f)$, we have

$$\sum_{p \in P^{(s_u,t_u)}} g_p = \sum_{p \in P^{(s_u,t_u)}} h_p = \gamma_u,$$

for each $u \in \Gamma(f)$. This contradicts Equation (5). □

## B Proof of Theorem 2

Let $Z(t) = \max_{1 \le k \le n} A^k(t)$. For any $\theta \in (0, \theta_0)$, we have

$$e^{\theta \mathrm{E}(Z(t))} \overset{(a)}{\le} \mathrm{E}\left(e^{\theta Z(t)}\right) = \mathrm{E}\left(\max_{1 \le k \le n} e^{\theta A^k(t)}\right) \le \sum_{k=1}^{n} \mathrm{E}\left(e^{\theta A^k(t)}\right) \overset{(b)}{=} n\mathrm{E}\left(e^{\theta A^1(t)}\right). \tag{6}$$

Here, (a) results from Jensen's inequality and (b) is true because the random variables $A^k(t)$ are identically distributed. Let $M_S(\theta) = \mathrm{E}\left(e^{\theta S}\right)$ be the moment generating function of $S$. Note that $M_S(\theta)$ exists for $\theta \in (-\theta_0, \theta_0)$. We have:

$$\begin{aligned}
\mathrm{E}\left(e^{\theta A^1(t)}\right) &= \mathrm{E}\left(\exp\left\{\theta \sum_{i=1}^{N^1(t)} S_i^1\right\}\right) \\
&\overset{(a)}{=} \sum_{k=0}^{\infty} e^{-\frac{\lambda t}{n}} \frac{\left(\frac{\lambda t}{n}\right)^k}{k!} \mathrm{E}\left(\exp\left\{\theta \sum_{i=1}^{k} S_i^1\right\} \middle| N^1(t) = k\right) \\
&\overset{(b)}{=} \sum_{k=0}^{\infty} e^{-\frac{\lambda t}{n}} \frac{\left(\frac{\lambda t}{n}\right)^k}{k!} \mathrm{E}\left(\exp\left\{\theta \sum_{i=1}^{k} S_i^1\right\}\right) \\
&\overset{(c)}{=} \sum_{k=0}^{\infty} e^{-\frac{\lambda t}{n}} \frac{\left(\frac{\lambda t}{n}\right)^k}{k!} \mathrm{E}\left(e^{\theta S}\right)^k \\
&= e^{\frac{\lambda t}{n}(M_S(\theta)-1)}.
\end{aligned} \tag{7}$$

In the above, (a) follows because $N^1(t)$ is Poisson with mean $\frac{\lambda t}{n}$; (b) follows because $N^1(t)$ and $\{S_i^1\}$ are independent; and (c) follows because $\{S_i^1\}$ are i.i.d. random variables. Equations (6) and (7) give

$$e^{\theta \mathrm{E}(Z(t))} \le n e^{\frac{\lambda t}{n}(M_S(\theta)-1)}$$

for $\theta \in (0, \theta_0)$. Therefore,

$$\mathrm{E}\left(Z(t)\right) \le \frac{\log n}{\theta} + \frac{\lambda t}{n}\left(\frac{M_S(\theta) - 1}{\theta}\right),$$

and plugging in the Taylor series expansion,

$$M_S(\theta) = 1 + \mathrm{E}(S)\theta + \frac{1}{2}\mathrm{E}\left(S^2\right)\theta^2 + O(\theta^3),$$

we obtain

$$\mathrm{E}\left(Z(t)\right) \leq \frac{\lambda t \mathrm{E}(S)}{n} + \frac{\log n}{\theta} + \frac{\lambda t \mathrm{E}(S^2)}{2n}\theta + \frac{\lambda t}{n}O(\theta^2), \tag{8}$$

for $\theta \in (0, \theta_0)$. Let $\theta^* = \sqrt{\frac{2n \log n}{\lambda \mathrm{E}(S^2) t}}$. Note that $\theta^* \in (0, \theta_0)$ for $t > \frac{2n \log n}{\lambda \mathrm{E}(S^2)\theta_0^2}$. Setting $\theta = \theta^*$ in Equation (8) yields

$$\mathrm{E}\left(Z(t)\right) \leq \frac{\lambda \mathrm{E}(S)}{n}t + \sqrt{\frac{2\lambda \mathrm{E}\left(S^2\right)\log n}{n}t} + O(1). \tag{9}$$

A nearly identical argument for $W(t) = \min_{1 \leq k \leq n} A^k(t)$ gives

$$\mathrm{E}\left(W(t)\right) \geq \frac{\lambda \mathrm{E}(S)}{n}t - \sqrt{\frac{2\lambda \mathrm{E}\left(S^2\right)\log n}{n}t} + O(1), \tag{10}$$

for $t > \frac{2n \log n}{\lambda \mathrm{E}(S^2)\theta_0^2}$. Equations (9) and (10), along with the fact that $\mathrm{E}\left(W(t)\right) \leq \mathrm{E}\left(Z(t)\right)$, give

$$\mathrm{E}\left(\chi(t)\right) = \frac{\mathrm{E}\left(Z(t)\right) - \mathrm{E}\left(W(t)\right)}{\frac{\lambda \mathrm{E}(S)}{n}t} \leq \sqrt{\frac{8n \log n \mathrm{E}\left(S^2\right)}{\lambda \mathrm{E}\left(S\right)^2 t}} + O(\frac{1}{t}), \tag{11}$$

which is the same as Equation (1). □