Inside the Social Network's (Datacenter) Network

Arjun Roy, Hongyi Zeng†, Jasmeet Bagga†, George Porter, and Alex C. Snoeren

Department of Computer Science and Engineering
University of California, San Diego

†Facebook, Inc.

ABSTRACT

Large cloud service providers have invested in increasingly larger datacenters to house the computing infrastructure required to support their services. Accordingly, researchers and industry practitioners alike have focused a great deal of effort designing network fabrics to efficiently interconnect and manage the traffic within these datacenters in performant yet efficient fashions. Unfortunately, datacenter operators are generally reticent to share the actual requirements of their applications, making it challenging to evaluate the practicality of any particular design.

Moreover, the limited large-scale workload information available in the literature has, for better or worse, heretofore largely been provided by a single datacenter operator whose use cases may not be widespread. In this work, we report upon the network traffic observed in some of Facebook's datacenters. While Facebook operates a number of traditional datacenter services like Hadoop, its core Web service and supporting cache infrastructure exhibit a number of behaviors that contrast with those reported in the literature. We report on the contrasting locality, stability, and predictability of network traffic in Facebook's datacenters, and comment on their implications for network architecture, traffic engineering, and switch design.

Keywords

Datacenter traffic patterns

CCS Concepts

•**Networks** → **Network measurement; Data center networks;** *Network performance analysis;* Network monitoring; Social media networks;

SIGCOMM '15, August 17–21, 2015, London, United Kingdom

 ISBN 978-1-4503-3542-3/15/08…$15.00

DOI: http://dx.doi.org/10.1145/2785956.2787472

1. INTRODUCTION

Datacenters are revolutionizing the way in which we design networks, due in large part to the vastly different engineering constraints that arise when interconnecting a large number of highly interdependent homogeneous nodes in a relatively small physical space, as opposed to loosely coupled heterogeneous end points scattered across the globe. While many aspects of network and protocol design hinge on these physical attributes, many others require a firm understanding of the demand that will be placed on the network by end hosts. Unfortunately, while we understand a great deal about the former (i.e., that modern cloud datacenters connect 10s of thousands of servers using a mix of 10-Gbps Ethernet and increasing quantities of higher-speed fiber interconnects), the latter tend to be not disclosed publicly.

Hence, many recent proposals are motivated by lightly validated assumptions regarding datacenter workloads, or, in some cases, workload traces from a single, large datacenter operator [12, 26]. These traces are dominated by traffic generated as part of a major Web search service, which, while certainly significant, may differ from the demands of other major cloud services. In this paper, we study sample workloads from within Facebook's datacenters. We find that traffic studies in the literature are not entirely representative of Facebook's demands, calling into question the applicability of some of the proposals based upon these prevalent assumptions on datacenter traffic behavior. This situation is particularly acute when considering novel network fabrics, traffic engineering protocols, and switch designs.

As an example, a great deal of effort has gone into identifying effective topologies for datacenter interconnects [4, 19, 21, 36]. The best choice (in terms of cost/benefit tradeoff) depends on the communication pattern between end hosts [33]. Lacking concrete data, researchers often design for the worst case, namely an all-to-all traffic matrix in which each host communicates with every other host with equal frequency and intensity [4]. Such an assumption leads to the goal of delivering maximum bisection bandwidth [4, 23, 36], which may be overkill when demand exhibits significant locality [17].

In practice, production datacenters tend to enforce a certain degree of oversubscription [12, 21], assuming that either the end-host bandwidth far exceeds actual traffic demands,

Finding	Previously published data	Potential impacts
Traffic is neither rack local nor all-to-all; low utilization (§4)	50–80% of traffic is rack local [12, 17]	Datacenter fabrics [4, 36, 21]
Demand is wide-spread, uniform, and stable, with rapidly changing, internally bursty heavy hitters (§5)	Demand is frequently concentrated and bursty [12, 13, 14]	Traffic engineering [5, 14, 25, 39]
Small packets (outside of Hadoop), continuous arrivals; many concurrent flows (§6)	Bimodal ACK/MTU packet size, on/off behavior [12]; <5 concurrent large flows [8]	SDN controllers [1, 22, 28, 32, 34]; Circuit/hybrid switching [7, 20, 30, 39]

Table 1: Each of our major findings differs from previously published characterizations of datacenter traffic. Many systems incorporate one or more of the previously published features as design assumptions.

or that there is significant locality in demand that decreases the need for full connectivity between physically disparate portions of the datacenter. The precise degree of oversubscription varies, but there is general agreement amongst operators that full connectivity is rarely worthwhile [11]. To mitigate potential "hotspots" caused by oversubscription, researchers have suggested designs that temporarily enhance connectivity between portions of the datacenter [5, 25, 40]. The utility of these approaches depends upon the prevalence, size, and dynamics of such hotspots.

In particular, researchers have proposed inherently non-uniform fabrics which provide qualitatively different connectivity to certain portions of the datacenter through various hybrid designs, typically including either optical [30, 39] or wireless links [25, 40]. If demand can be predicted and/or remains stable over reasonable time periods, it may be feasible to provide circuit-like connectivity between portions of the datacenter [20]. Alternatively, network controllers could select among existing paths in an intelligent fashion [14]. Regardless of the technology involved, all of these techniques require traffic to be predictable over non-trivial time scales [14, 20, 25, 30, 39].

Finally, many have observed that the stylized nature of datacenter traffic opens up many avenues for increasing the efficiency of switching hardware itself. In particular, while some have proposed straightforward modifications like decreased buffering, port count, or sophistication [4] in various layers of the switching fabric, others have proposed replacing conventional packet switches either with circuit or hybrid designs that leverage locality, persistence, and predictability of traffic demands [30]. More extreme, host-based solutions advocate connecting end-hosts directly [21, 23]. Obviously, when, where, or if any of these approaches makes economic sense hinges tightly on offered loads [33].

While there have been a number of studies of university [14] and private datacenters [12], many proposals cannot be fully evaluated without significant scale. Almost all of the previous studies of large-scale (10K hosts or larger) datacenters [5, 12, 14, 17, 21, 25, 26] consider Microsoft datacenters. While Facebook's datacenters have some commonality with Microsoft's, such as eschewing virtual machines [14], they support a very different application mix. As a result, we observe a number of critical distinctions that may lead to qualitatively different conclusions; we describe those differences and explain the reasons behind them.

Our study is the first to report on production traffic in a datacenter network connecting hundreds of thousands of 10-Gbps nodes. Using both Facebook-wide monitoring systems and per-host packet-header traces, we examine services that generate the majority of the traffic in Facebook's network. While we find that the traffic patterns exhibited by Facebook's Hadoop deployments comport well with those reported in the literature, significant portions of Facebook's service architecture [10, 15] vary dramatically from the MapReduce-style infrastructures studied previously, leading to vastly different traffic patterns. Findings of our study with significant architectural implications include:

- Traffic is neither rack-local nor all-to-all; locality depends upon the service but is stable across time periods from seconds to days. Efficient fabrics may benefit from variable degrees of oversubscription and less intra-rack bandwidth than typically deployed.
- Many flows are long-lived but not very heavy. Load balancing effectively distributes traffic across hosts; so much so that traffic demands are quite stable over even sub-second intervals. As a result, heavy hitters are not much larger than the median flow, and the set of heavy hitters changes rapidly. Instantaneously heavy hitters are frequently not heavy over longer time periods, likely confounding many approaches to traffic engineering.
- Packets are small (median length for non-Hadoop traffic is less than 200 bytes) and do not exhibit on/off arrival behavior. Servers communicate with 100s of hosts and racks concurrently (i.e., within the same 5-ms interval), but the majority of traffic is often destined to (few) 10s of racks.

While we do not offer these workloads as any more representative than others—indeed, they may change as Facebook's services evolve—they do suggest that the space of cloud datacenter workloads is more rich than the literature may imply. As one way to characterize the significance of our findings, Table 1 shows how our results compare to the literature, and cites exemplar systems that incorporate these assumptions in their design.

The rest of this paper is organized as follows. We begin in Section 2 with a survey of the major findings of previous studies of datacenter traffic. Section 3 provides a high-level description of the organization of Facebook's datacenters, the services they support, and our collection methodologies. We then analyze aspects of the traffic within a number of Facebook's datacenters that impact provisioning (Section 4), traffic engineering (Section 5), and switch design (Section 6), before concluding in Section 7.

2. RELATED WORK

Initial studies of datacenter workloads were conducted via simulation [6] or on testbeds [18]. Subsequently, however, a number of studies of production datacenter traffic have been performed, primarily within Microsoft datacenters.

It is difficult to determine how many distinct Microsoft datacenters are reported on in literature, or how representative that set might be. Kandula *et al.* observe that their results "extend to other *mining* data centers that employ some flavor of map-reduce style workflow computation on top of a distributed block store," but caution that "*web* or *cloud* data centers that primarily deal with generating responses for web requests (e.g., mail, messenger) are likely to have different characteristics." [26]. By that taxonomy, Facebook's datacenters clearly fall in the latter camp. Jalaparti *et al.* [24] examine latency for Microsoft Bing services that are similar in concept to Facebook's service; we note both similarities to our workload (relatively low utilization coupled with a scatter-gather style traffic pattern) and differences (load appears more evenly distributed within Facebook datacenters).

Three major themes are prevalent in prior studies, and summarized in Table 1. First, traffic is found to be heavily rack local, likely as a consequence of the application patterns observed; Benson *et al.* note that for cloud datacenters "a majority of traffic originated by servers (80%) stays within the rack" [12]. Studies by Kandula *et al.* [26], Delimitrou *et al.* [17] and Alizadeh *et al.* [8] observe similarly rack-heavy traffic patterns.

Second, traffic is frequently reported to be bursty and unstable across a variety of timescales—an important observation, since traffic engineering techniques often depend on relatively long-lived, predictable flows. Kapoor *et al.* observe that packets to a given destination often arrive in trains [27]; while Benson *et al.* find a strong on/off pattern where the packet inter-arrival follows a log-normal distribution [13]. Changing the timescale of observation can change the ease of prediction; Delimitrou *et al.* [17] note that while traffic locality varies on a day-to-day basis, it remains consistent at the scale of months. Conversely, Benson *et al.* [14] claim that while traffic is unpredictable at timescales of 150 seconds and longer, it can be relatively stable on the timescale of a few seconds, and discuss traffic engineering mechanisms that might work for such traffic.

Finally, previous studies have consistently reported a bimodal packet size [12], with packets either approaching the MTU or remaining quite small, such as a TCP ACK segment. We find that Facebook's traffic is very different, with a consistently small median packet size despite the 10-Gbps link speed. Researchers have also reported that individual end hosts typically communicate with only a few (e.g., less than 5 [8]) destinations at once. For some Facebook services, an individual host maintains orders of magnitude more concurrent connections.

3. A FACEBOOK DATACENTER

In order to establish context necessary to interpret our findings, this section provides a brief overview of Facebook's datacenter network topology, as well as a description of the application services that it supports; more detail is available elsewhere [2, 9, 10, 15, 19]. We then describe the two distinct collection systems used to assemble the network traces analyzed in the remainder of the paper.

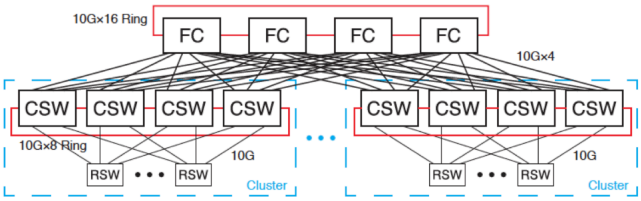


Figure 1: Facebook's 4-post cluster design [19]

3.1 Datacenter topology

Facebook's network consists of multiple datacenter sites (henceforth site) and a backbone connecting these sites. Each datacenter site contains one or more datacenter buildings (henceforth datacenter) where each datacenter contains multiple clusters. A cluster is considered a unit of deployment in Facebook datacenters. Each cluster employs a conventional 3-tier topology depicted in Figure 1, reproduced from a short paper [19].

Machines are organized into racks and connected to a top-of-rack switch (RSW) via 10-Gbps Ethernet links. The number of machines per rack varies from cluster to cluster. Each RSW in turn is connected by 10-Gbps links to four aggregation switches called cluster switches (CSWs). All racks served by a particular set of CSWs are said to be in the same cluster. Clusters may be homogeneous in terms of machines—e.g. Cache clusters—or heterogeneous, e.g. Frontend clusters which contain a mixture of Web servers, load balancers and cache servers. CSWs are connected to each other via another layer of aggregation switches called Fat Cats (FC). As will be seen later in this paper, this design follows directly from the need to support a high amount of intra-cluster traffic. Finally, CSWs also connect to aggregation switches for intra-site (but inter-datacenter) traffic and datacenter routers for inter-site traffic.

The majority of Facebook's current datacenters employ this 4-post Clos design. Work is underway, however, to migrate Facebook's datacenters to a next-generation Fabric architecture [9]. The analyses in this paper are based upon data collected from machines in traditional 4-post clusters, although Facebook-wide statistics (e.g., Table 3) cover hosts in both traditional 4-post clusters and newer Fabric pods.

One distinctive aspect of Facebook's datacenters is that each machine typically has precisely one role: Web servers (Web) serve Web traffic; MySQL servers (DB) store user data; query results are stored temporarily in cache servers (Cache)—including leaders, which handle cache coherency, and followers, which serve most read requests [15]; Hadoop servers (Hadoop) handle offline analysis and data mining; Multifeed servers (MF) assemble news feeds [31]. While there are a number of other roles, these represent the majority, and will be the focus of our study. In addition, a rel-

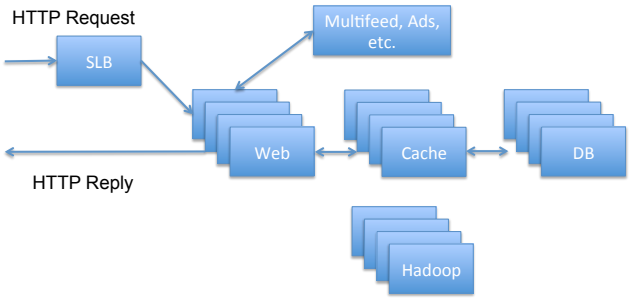


Figure 2: How an HTTP request is served

Type	Web	Cache	MF	SLB	Hadoop	Rest
Web	-	63.1	15.2	5.6	-	16.1
Cache-l	-	86.6	5.9	-	-	7.5
Cache-f	88.7	5.8	-	-	-	5.5
Hadoop	-	-	-	-	99.8	0.2

Table 2: Breakdown of outbound traffic percentages for four different host types

atively small number of machines do not have a fixed role and are dynamically repurposed. Facebook's datacenters do not typically house virtual machines: each service runs on a physical server. Moreover—and in contrast to previously studied datacenters [12]—to ease provisioning and management, racks typically contain only servers of the same role.

3.2 Constituent services

The organization of machines within a cluster—and even a datacenter—is intimately related to the communication patterns between the services they support. We introduce the major services by briefly describing how an HTTP request is served by `http://facebook.com`, shown in Figure 2.

When an HTTP query hits a Facebook datacenter, it arrives at a layer-four software load balancer (SLB) [37]. The query is then redirected to one of the Web servers. Web servers are largely stateless, containing no user data. They fetch data from the cache tier [15]. In case of a cache miss, a cache server will then fetch data from the database tier. At the same time, the Web server may communicate with one or more backend machines to fetch objects such as news stories and ads. Table 2 quantifies the relative traffic intensity between different services by classifying the outbound traffic from four different servers—a Web server, cache leader (cache-l), cache follower (cache-f), and Hadoop—based upon the role of the destination host. (The data is extracted from packet-header traces described in Section 3.3.2.)

In contrast to most service tiers, Hadoop nodes are not involved with serving end-user requests. Instead, Hadoop clusters perform offline analysis such as data mining. HDFS and Hadoop MapReduce are the main applications running on these servers.

3.3 Data collection

Due to the scale of Facebook's datacenters, it is impractical to collect complete network traffic dumps. Instead,

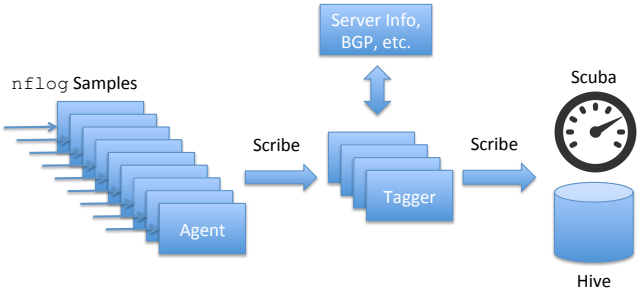


Figure 3: Fbflow architecture

we consider two distinct sources of data. The first, Fbflow, constantly samples packet headers across Facebook's entire global network. The second, port mirroring, focuses on a single machine (or rack) at a time, allowing us to collect complete packet-header traces for a brief period of time at particular locations within a single datacenter.

3.3.1 *Fbflow*

Fbflow is a production monitoring system that samples packet headers from Facebook's entire machine fleet. Its architecture, comprised of two main component types—*agents* and *taggers*—is shown in Figure 3. Fbflow samples packets by inserting a Netfilter `nflog` target into every machine's `iptable` rules. The datasets we consider in this paper are collected with a 1:30,000 sampling rate. A user-level Fbflow agent process on each machine listens to the `nflog` socket and parses the headers, extracting information such as source and destination IP addresses, port numbers, and protocol. These parsed headers—collected across all machines in Facebook's datacenters—along with metadata such as machine name and capture time, are streamed to a small number of taggers using Scribe [2], a log aggregation system.

Taggers, running on a subset of machines, read a portion of the packet-header stream from Scribe, and further annotate it with additional information such as the rack and cluster containing the machine where the trace was collected, its autonomous system number, etc., by querying other data sources. Taggers then convert each annotated packet header into a JSON object and feed it into Scuba [3], a real-time data analytics system. Samples are simultaneously stored into Hive [38] tables for long-term analysis.

3.3.2 *Port mirroring*

While Fbflow is a powerful tool for network monitoring and management, its sampling-based collection prohibits certain types of data analysis. Specifically, in production use, it aggregates statistics at a per-minute granularity. In order to collect high-fidelity data, we deploy a number of special-purpose trace collection machines within the datacenter that collect packet-header traces over short intervals.

We deploy monitoring hosts in five different racks across Facebook's datacenter network, locating them in clusters that host distinct services. In particular, we monitor a rack of Web servers, a Hadoop node, cache followers and leaders, and a Multifeed node. In all but one (Web) instance, we collect traces by turning on port mirroring on the RSW

(ToR) and mirroring the full, bi-directional traffic for a single server to our collection server. For the hosts we monitor, the RSW is able to mirror the selected ports without loss. In the case of Web servers, utilization is low enough that we are able to mirror traffic from a rack of servers to our collection host. We did not measure database servers that include user data in this study.

Recording the packet traces using a commodity server is not entirely trivial, as `tcpdump` is unable to handle more than approximately 1.5 Gbps of traffic in our configuration. In order to support line-rate traces, we employ a custom kernel module that effectively pins all free RAM on the server and uses it to buffer incoming packets. Our kernel module extracts the packets immediately after the Ethernet driver hands the packets to the kernel to avoid any additional delay or overhead. Once data collection is complete, the data is spooled to remote storage for analysis. Memory restrictions on our collection servers limit the traces we collect in this fashion to a few minutes in length.

4. PROVISIONING

The appropriate design, scale, and even technology of a datacenter interconnect depends heavily on the traffic demands of the services it hosts. In this section, we quantify the traffic intensity, locality, and stability across three different types of clusters inside Facebook datacenters; in particular, we examine clusters supporting Hadoop, Frontend machines serving Web requests, and Cache.

Our study reveals that while Facebook's Hadoop deployments exhibit behavior largely consistent with the literature, the same cannot be said for clusters hosting Facebook's other services. In particular, most traffic is *not* rack-local, yet locality patterns remain stable within and across both long (multiple-day) and short (two-minute) time intervals. We define stable traffic as being close to constant (low deviation from a baseline value) over a time interval, and slowly changing across time intervals. Note that this definition is dependent upon the length of the interval being considered; accordingly, we examine several different timescales.

4.1 Utilization

Given that Facebook has recently transitioned to 10-Gbps Ethernet across all of their hosts, it is not surprising that overall access link (i.e., links between hosts and their RSW) utilization is quite low, with the average 1-minute link utilization less than 1%. This comports with the utilization levels reported for other cloud-scale datacenters [12, 17]. Demand follows typical diurnal and day-of-the-week patterns, although the magnitude of change is on the order of $2\times$ as opposed to the order-of-magnitude variation reported elsewhere [12], Even the most loaded links are lightly loaded over 1-minute time scales: 99% of all links are typically less than 10% loaded. Load varies considerably across clusters, where the average link utilization in the heaviest clusters (Hadoop) is roughly $5\times$ clusters with light load (Frontend).

As in other datacenters with similar structure [12, 13], utilization rises at higher levels of aggregation. Focusing on the links between RSWs and CSWs, median utilization varies between 10–20% across clusters, with the busiest 5% of the links seeing 23–46% utilization. These levels are higher than most previously studied datacenters [12, Fig. 9], likely due to the disproportionate increase in edge-link technology (1→10 Gbps) vs. aggregation links (10→40 Gbps). The variance between clusters decreases, with the heaviest clusters running $3\times$ higher than lightly loaded ones. Utilization is higher still on links between CSWs and FC switches, although the differences between clusters are less apparent because different clusters are provisioned with different numbers of uplinks depending on their demand. We examine link utilization at finer timescales in Section 6.

4.2 Locality and stability

Prior studies have observed heavy rack locality in datacenter traffic. This behaviour seems in line with applications that seek to minimize network utilization by leveraging data locality, allowing for topologies with high levels of oversubscription. We examine the locality of Facebook's traffic from a representative sampling of production systems across various times of the day.

Figure 4 shows the breakdown of outbound traffic by destination for four different classes of servers: a Hadoop server within a Hadoop cluster, a Web server in a Frontend cluster, and both a cache follower and a cache leader from within the same Cache cluster. For each server, each second's traffic is represented as a stacked bar chart, with rack-local traffic in cyan, the cluster-local traffic in blue, the intra-datacenter traffic in red, and inter-datacenter traffic in green.

Among the four server types, Hadoop shows by far the most diversity—both across servers and time: some traces show periods of significant network activity while others do not. While all traces show both rack- and cluster-level locality, the distribution between the two varies greatly. In one ten-minute-long trace captured during a busy period, 99.8% of all traffic sent by the server in Figure 4 is destined to other Hadoop servers: 75.7% of that traffic is destined to servers in the the same rack (with a fairly even spread within the rack); almost all of the remainder is destined to other hosts within the cluster. Only a vanishingly small amount of traffic leaves the cluster.

In terms of dispersion, of the inter-rack (intra-cluster) traffic, the Hadoop server communicates with 1.5% of the other servers in the cluster—spread across 95% of the racks—though only 17% of the racks receive over 80% of the server's traffic. This pattern is consistent with that observed by Kandula *et al.* [26], in which traffic is either rack-local or destined to one of roughly 1–10% of the hosts in the cluster.

Hadoop's variability is a consequence of a combination of job size and the distinct phases that a Hadoop job undergoes—any given data capture might observe a Hadoop node during a busy period of shuffled network traffic, or during a relatively quiet period of computation.

By way of contrast, the traffic patterns for the other server classes are both markedly more stable and dramatically different from the findings of Kandula *et al.* [26]. Notably, only a minimal amount of rack-local traffic is present; even

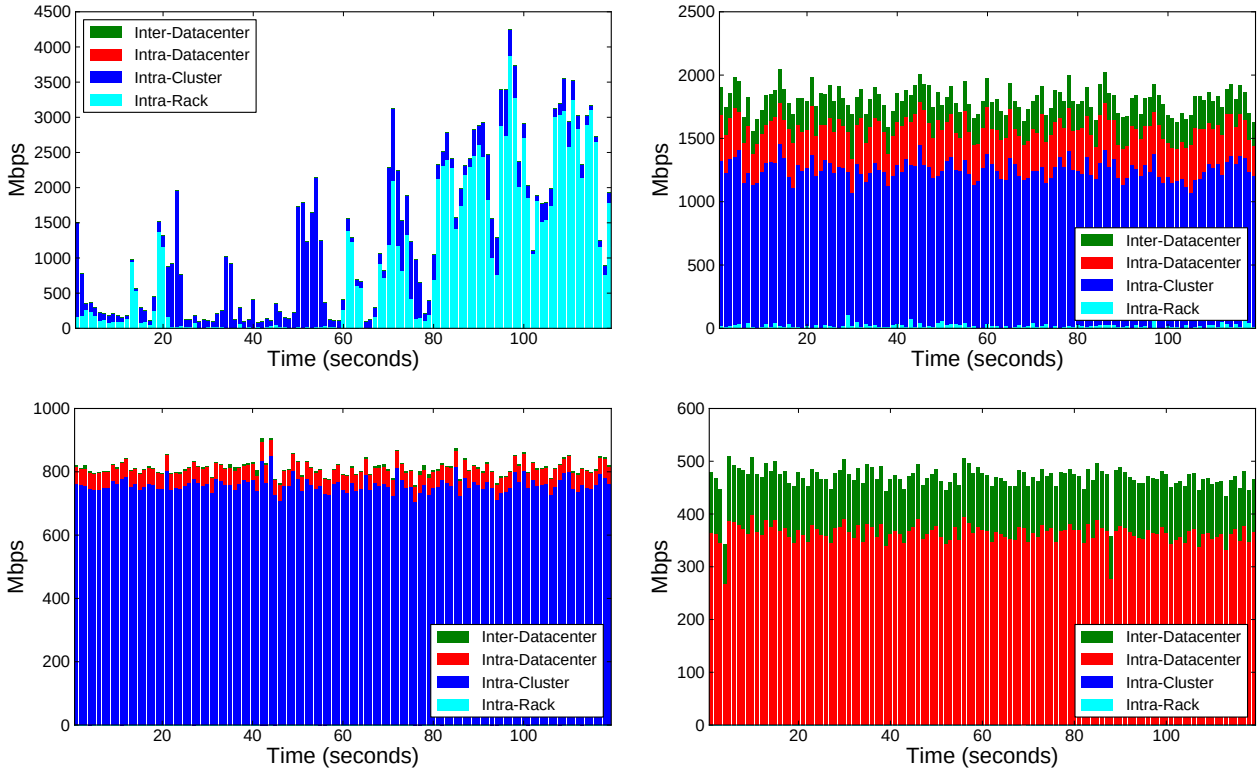


Figure 4: Per-second traffic locality by system type over a two-minute span: Hadoop (top left), Web server (top right), cache follower (bottom left) and leader (bottom right) (Note the differing y axes)

inter-datacenter traffic is present in larger quantities. Frontend cluster traffic, including Web servers and the attendant cache followers, stays largely within the cluster: 68% of Web server traffic during the capture plotted here stays within the cluster, 80% of which is destined to cache systems; the Multifeed systems and the SLB servers get 8% each. While miscellaneous background traffic is present, the volume of such traffic is relatively inconsequential.

Cache systems, depending on type, see markedly different localities, though along with Web servers the intra-rack locality is minimal. Frontend cache followers primarily send traffic in the form of responses to Web servers (88%), and thus see high intra-cluster traffic—mostly servicing cache reads. Due to load balancing (see Section 5.2), this traffic is spread quite widely; during this two-minute interval the cache follower communicates with over 75% of the hosts in the cluster, including over 90% of the Web servers. Cache leaders maintain coherency across clusters and the backing databases, engaging primarily in intra- and inter-datacenter traffic—a necessary consequence of the cache being a "single geographically distributed instance." [15]

The stability of these traffic patterns bears special mention. While Facebook traffic is affected by the diurnal traffic pattern noted by Benson *et al.* [12], the relative proportions of the locality do not change—only the total amount of traffic. Over short enough periods of time, the graph looks essentially flat and unchanging. In order to further investigate the cause and particulars of this stability, we turn our attention to the traffic matrix itself.

Locality	All	Hadoop	FE	Svc.	Cache	DB
Rack	12.9	13.3	2.7	12.1	0.2	0
Cluster	57.5	80.9	81.3	56.3	13.0	30.7
DC	11.9	3.3	7.3	15.7	40.7	34.5
Inter-DC	17.7	2.5	8.6	15.9	16.1	34.8
Percentage		23.7	21.5	18.0	10.2	5.2

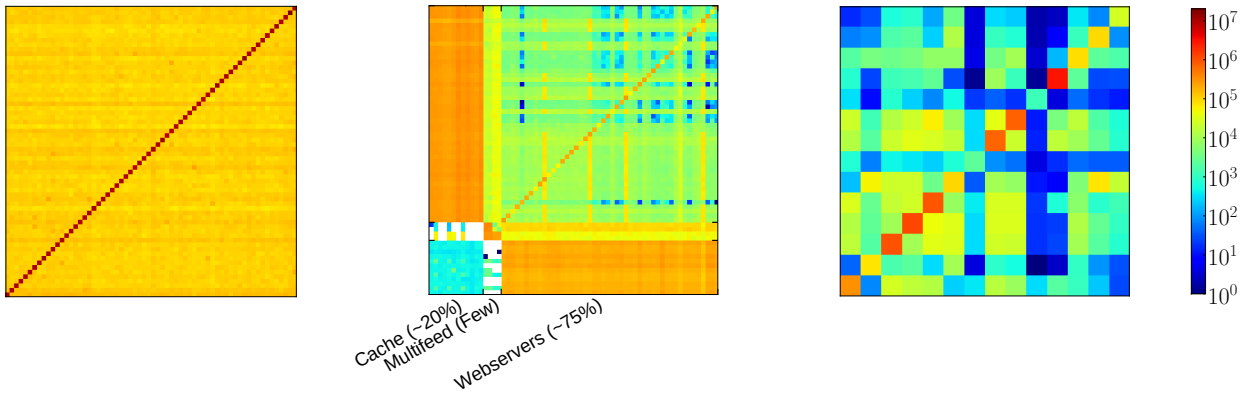
Table 3: Different clusters have different localities; last row shows each cluster's contribution to total network traffic

4.3 Traffic matrix

In light of the surprising lack of rack locality and high degree of traffic stability, we examine traffic from the more long-term and zoomed-out perspective provided by Fbflow.

Table 3 shows the locality of traffic generated by all of Facebook's machines during a 24-hour period in January 2015 as reported by Fbflow. Facebook's traffic patterns remain stable day-over-day—unlike the datacenter studied by Delimitrou *et al.* [17]. The clear majority of traffic is intra-cluster but not intra-rack (i.e., the 12.9% of traffic that stays within a rack is not counted in the 57.5% of traffic labeled as intra-cluster). Moreover, more traffic crosses between datacenters than stays within a rack.

Table 3 further breaks down the locality of traffic generated by the top-five cluster types which, together, account for 78.6% of the traffic in Facebook's network. Hadoop clusters generate the most traffic (23.7% of all traffic), and are significantly more rack-local than others, but even its traffic is far from the 40–80% rack-local reported in the literature [12,



(a) Rack-to-rack, Hadoop cluster (b) Rack-to-rack, Frontend cluster (c) Cluster-to-cluster

Figure 5: Traffic demand by source (x axis) and destination (y axis). The graphs are each normalized to the lowest demand in that graph type (i.e., the Hadoop and Frontend clusters are normalized to the same value, while the cluster-to-cluster graph is normalized independently).

17]. Rather, Hadoop traffic is clearly cluster local. Frontend (FE) traffic is cluster local by design, but not very rack-local, and the locality of a given rack's traffic depends on its constituent servers (e.g., Web server, Multifeed, or cache).

This distinction is clearly visualized in Figure 5, generated in the style of Delimitrou *et al.* [17]. The two left portions of the figure graph the relative traffic demands between 64 racks within clusters of two different types. While we show only a subset of the total set of racks in each cluster, the pattern is representative of the cluster as a whole.

Traffic within the Hadoop cluster (left) is homogenous with a very strong diagonal (i.e., intra-rack locality). The cluster-wide uniformity outside the local rack accounts for intra-cluster traffic representing over 80% of Hadoop traffic—even though traffic to the local rack dominates any given other rack in isolation. Map tasks are placed to maximize read locality, but there are a large number of concurrent jobs which means that it is possible that any given job will not fit entirely within a rack. Thus, some amount of traffic would necessarily need to leave the rack during the shuffle and output phases of a MapReduce job. In addition, the cluster serves data requests from other services which might not strive for as much read locality, which would also contribute to reduced overall rack locality.

The Frontend cluster (center) exhibits three different patterns according to rack type, with none being particularly rack-local. In particular, we see a strong bipartite traffic pattern between the Web servers and the cache followers in Webserver racks that are responsible for most of the traffic, by volume, within the cluster. This pattern is a consequence of placement: Web servers talk primarily to cache servers and vice versa, and servers of different types are deployed in distinct racks, leading to low intra-rack traffic.

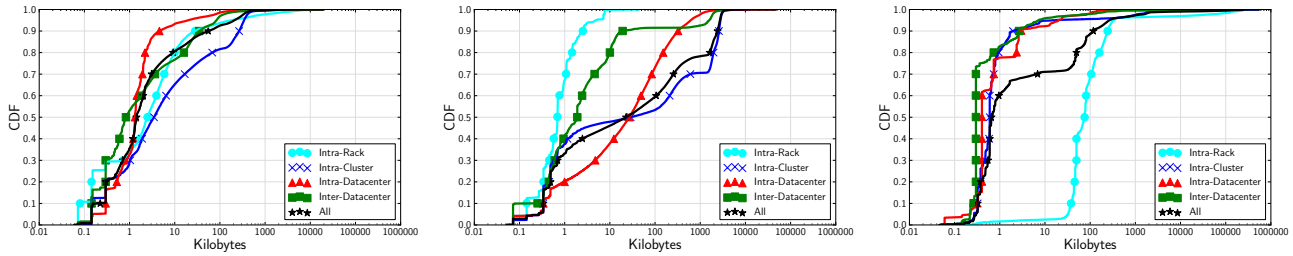
This striking difference in Facebook's locality compared to previously studied Internet-facing user-driven applications is a consequence of the realities of serving a densely connected social graph. Cache objects are replicated across clusters; however, each object typically appears once in a cluster (though hot objects are replicated to avoid hotspots, which we discuss in Section 5). Since each Web server needs to be able to handle any request, they might need to access data in a potentially random fashion due to load balancing.

To make this argument more concrete, loading the Facebook news feed draws from a vast array of different objects in the social graph: different people, relationships, and events comprise a large graph interconnected in a complicated fashion. This connectedness means that the working set is unlikely to reduce even if users are partitioned; the net result is a low cache hit rate within the rack, leading to high intra-cluster traffic locality. In addition, partitioning the graph such that users and their data are co-located on racks has the potential to introduce failure modes which disproportionately target subsets of the user base, leading to a suboptimal experience.

The other three cluster types exhibit additional distinctive behaviors (not shown). Traffic in cache leader clusters, for example, has very little intra-rack demand, instead spreading the plurality of its traffic across the datacenter. Traffic in back-end database clusters is the most uniform, divided almost evenly amongst nodes within the cluster, the same datacenter, and worldwide. Service clusters, which host racks supporting a variety of supporting services, exhibit a mixed traffic pattern that lies between these extreme points.

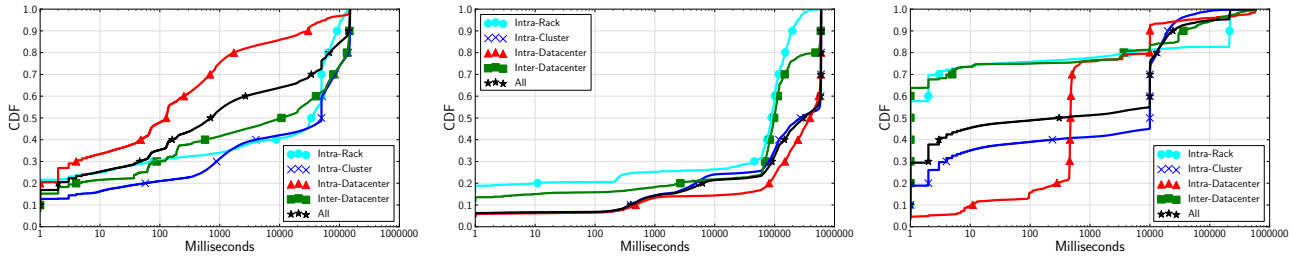
Inter-cluster communication varies considerably by cluster type. Figure 5c plots the traffic demand between 15 clusters within a single datacenter for the a 24-hour period. Hadoop clusters, for example, have a very small proportion of inter-cluster traffic, while cache leader clusters have a large amount of inter-cluster traffic, split between cache followers in other clusters and database clusters. While each cluster may possess the same four-post structure internally, it may make sense to consider heterogenous inter-cluster communication fabrics, as demand varies over more than seven orders of magnitude between cluster pairs.

While the 4-post cluster remains prevalent in Facebook datacenters, Facebook recently announced a new network



(a) Web servers (b) Cache follower (c) Hadoop

Figure 6: Flow size distribution, broken down by location of destination



(a) Web servers (b) Cache follower (c) Hadoop

Figure 7: Flow duration distribution, broken down by location of destination

topology that is being implemented in datacenters going forward [9]. While servers are no longer grouped into clusters physically (instead, they comprise pods where all pods in a datacenter have high connectivity), the high-level logical notion of a cluster for server management purposes still exists to ease the transition. Accordingly, the rack-to-rack traffic matrix of a Frontend "cluster" inside one of the new Fabric datacenters over a day-long period (not shown) looks similar that shown in Figure 5.

4.4 Implications for connection fabrics

The low utilization levels found at the edge of the network reinforce common practice of oversubscribing the aggregation and core of the network, although it remains to be seen whether utilization will creep up as the datacenters age. The highly contrasting locality properties of the different clusters imply a single homogenous topology will either be overprovisioned in some regions or congested in others—or both. This reality argues that non-uniform fabric technologies that can deliver higher bandwidth to certain locations than others may find use. Researchers are exploring techniques to ameliorate traffic hotspots. The stability of the traffic patterns we observe, however, suggest that rapid reconfigurability may not be as necessary as some have assumed.

Somewhat surprisingly, the lack of significant levels of intra-rack locality (except in the Hadoop cluster) hints that RSWs (i.e., top-of-rack switches) that deliver something less than full non-blocking line-rate connectivity between all of their ports may be viable. In particular, the bipartite traffic pattern between end hosts and RSW uplinks may afford optimizations in switch design. We return to consider further implications for switch design in Section 6.

5. TRAFFIC ENGINEERING

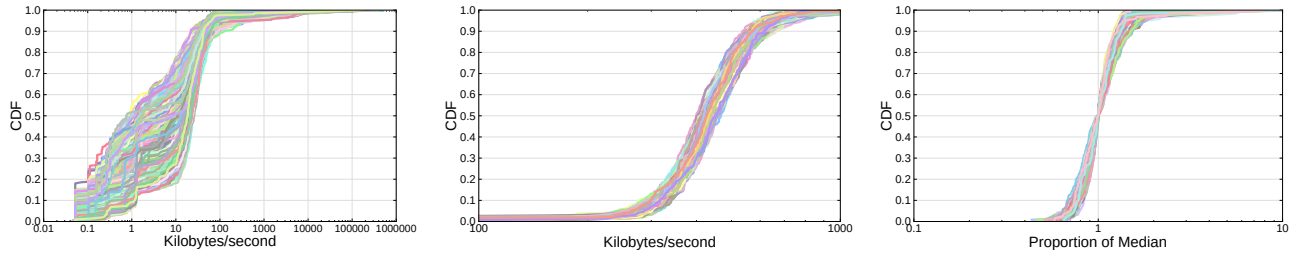
Prior studies suggest that the stability of datacenter traffic depends on the timescale of observation. In this section, we analyze Facebook's traffic at fine timescales, with an eye towards understanding how applicable various traffic engineering and load balancing approaches might be under such conditions.

5.1 Flow characteristics

Figures 6 and 7 plot the size and duration, respectively, of flows (defined by 5-tuple) collected in 10-minute (2.5-minute for the Web-server rack) packet traces of three different node types: a Web-server rack, a single cache follower (cache leader is similar to follower and not shown due to space constraints), and a Hadoop node. We show the overall distribution (in black) as well as per-destination curves.

Consistent with the literature [26, Fig. 9], we find that most flows in Facebook's Hadoop cluster are short. As discussed previously, the traffic demands of Hadoop vary substantially across nodes and time. We plot the results from tracing one node over a relatively busy 10-minute interval; traces from other nodes or even the same node at different times reveal somewhat different distributions, so we caution against examining the specific distribution too carefully. Even in the graphed interval, however, 70% of flows send less than 10 KB and last less than 10 seconds; the median flow sends less than 1 KB and lasts less than a second. Less than 5% of the flows are larger than 1 MB or last longer than 100 seconds; almost none exceed our 10-minute trace.

Conversely, traces from other service types are much more representative due to load balancing. Moreover, many of Facebook's internal services use some form of connec-



(a) Hadoop (rate, each series is 1 second) (b) Cache (rate, each series is 1 second) (c) Cache (stability, each series is a rack)

Figure 8: Per-destination-rack flow rate distribution (for both Hadoop and cache) and stability (cache).

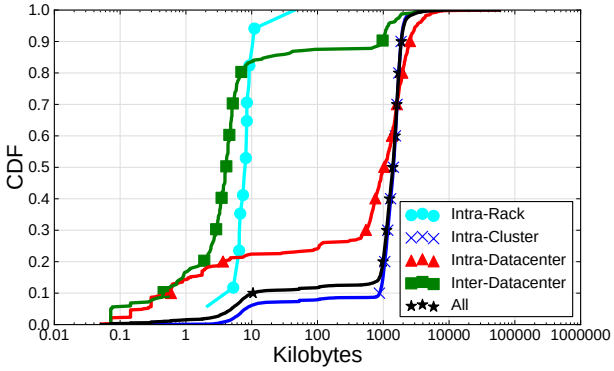


Figure 9: Cache follower per-host flow size

tion pooling [29], leading to long-lived connections with relatively low throughput. Pooling is especially prevalent for cache follower(leader, not shown) nodes, where only 30(40)% of flows are less than 100 seconds in length, with more than 40(25)% of flows exceeding our 10-minute capture period. That said, most flows are active (i.e., actually transmit packets) only during distinct millisecond-scale intervals with large intervening gaps. In other words, regardless of flow size or length, flows tend to be internally bursty. In general cache flows are also significantly larger than Hadoop; Web servers lie somewhere in the middle.

If we consider higher levels of aggregation, i.e., grouping flows by destination host or rack, the distribution of flow sizes simply shifts to the right for Web servers (retaining its basic shape). The behavior is starkly different for cache followers, however: the wide flow-size distribution apparent at a 5-tuple granularity (Figure 6b) disappears at host and rack levels, replaced by a very tight distribution around 1 MB per host (Figure 9). This arises as a consequence of the decision to load balance incoming user requests across all Web servers, combined with the large number of user requests. Since requests and responses are typically small (on the order of a few kilobytes) we do not observe any imbalance created by unequal response sizes.

5.2 Load balancing

Existing traffic engineering efforts seek to leverage variability of traffic; highly regular traffic does not provide much opportunity for improvement. In the previous section, we note that Facebook's approach to load balancing is highly effective on timescales lasting minutes to hours, leaving less room for traffic engineering. We now consider traffic characteristics over the course of a few seconds to determine whether traffic engineering might be effective on short timescales.

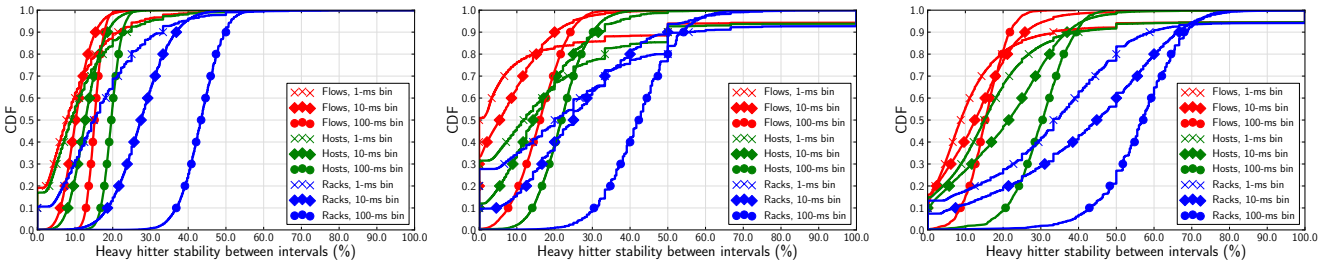
We consider how the traffic from a host varies from one second to next. We examine the distribution of flow rates, aggregated by destination rack, per second over a two-minute period and compare each second to the next. Intuitively, the better the load balancing, the closer one second appears to the next.

We first examine the Hadoop cluster by looking at 120 consecutive 1-second intervals. Figure 8a plots a CDF of per-destination-rack flow sizes for each interval (i.e., there are 120 separate curves). While we do not claim this particular server is representative, it does depict widely varying rates (i.e., more than three orders of magnitude) which are common in our observations.

In and of itself, this is unsurprising—Hadoop has periods of varying network traffic, and a production cluster is likely to see a myriad jobs of varying sizes. It is this variability of traffic that existing network traffic engineering schemes seek to leverage. Orchestra [16] relies on temporal and per-job variation to provide lower task completion times for high-priority tasks, while Hedera [5] provides non-interfering route placement for high bandwidth elephant flows that last for several seconds, which are prevalent within Hadoop workloads.

A different story emerges for Frontend traffic, and the cache in particular. Recall from Table 2 that the largest share of cache follower traffic are responses bound for Web servers. Figure 8b shows the distribution of per-second flow rates on a per-rack basis from a single cache follower node to distinct Web server racks during a two minute period. The distributions for each of the 120 seconds are similar, and all are relatively tight, i.e., the CDFs are fairly vertical about the median of $\approx$2 Mbps. Similar patterns (albeit with different scales) can be observed for other services as well.

From the viewpoint of a single host, each second is similar to the next. However, this analysis does not take per-destination variation into consideration. It is conceivable that there could exist consistently high- or low-rate destina-



(a) Cache follower (b) Cache leader (c) Web servers

Figure 10: Heavy-hitter stability as a function of aggregation for 1/10/100-ms time windows

tions that potentially could be treated differently by a traffic engineering scheme. For each host, we consider outbound traffic rates per destination rack (normalized to the median rate for that rack), and track the rate over time for each rack. Figure 8c plots these distributions for the outbound traffic for the same cache machine as Figure 8b. Each series represents a single destination; a near vertical series represents a destination rack where the rate does not deviate far from the median rate. We find that per-destination-rack flow sizes are remarkably stable across not only seconds but intervals as long as 10 seconds (not shown) as well. All of the flows are within a factor of two of their median size in approximately 90% of the 1-second intervals—the median flow exhibits "significant change" in only 45% of the 1-second intervals according to the 20% deviation cutoff defined by Benson *et al.* [14]. Contrast this to the traffic leaving a Hadoop node—which is not load balanced—where the middle 90% of flows can vary in size by over six orders of magnitude compared to their median size in the trace (not shown).

Such stability, both over time and by destination, is the result of a combination of workload characteristics and engineering effort. To a cache system, the offered load per second is roughly held constant—large increases in load would indicate the presence of relatively hot objects, which is actively monitored and mitigated. Bursts of requests for an object lead the cache server to instruct the Web server to temporarily cache the hot object; sustained activity for the object leads to replication of the object or the enclosing shard across multiple cache servers to help spread the load. We note further that the request rate distribution for the top-50 most requested objects on a cache server is close across all cache servers, and that the median lifespan for objects within this list is on the order of a few minutes. Per-destination traffic stability is again a consequence of user request multiplexing across all available Web servers, coupled with relatively small request/response pairs.

5.3 Heavy hitters

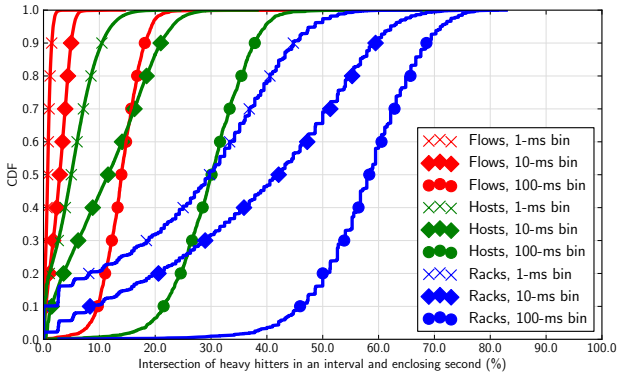
In this section, we examine the behavior of traffic at sub-second timescales to better understand its stability and whether traffic engineering can apply. In particular, we wish to see if certain flows (aggregated or not) stand out in terms of rate, since such flows would provide the largest opportunity for potential impact on network performance. We define a set of flows that we call *heavy hitters*, representing the minimum set of flows (or hosts, or racks in the aggregated case) that is responsible for 50% of the observed traffic volume (in bytes) over a fixed time period. Intuitively, the presence of heavy hitters can signify an imbalance that can be acted upon—if they are persistent for enough time, and large enough compared other flows that treating them differently makes a difference.

Type		Number			Size (Mbps)		
		p10	p50	p90	p10	p50	p90
Web	f	1	4	15	1.6	3.2	47.3
	h	1	4	14	1.6	3.3	48.1
	r	1	3	9	1.7	4.6	48.9
Cache (f)	f	8	19	35	5.1	9.0	22.5
	h	8	19	33	8.4	9.7	23.6
	r	7	15	23	8.4	14.5	31.0
Cache (l)	f	1	16	48	2.6	3.3	408
	h	1	8	25	3.2	8.1	414
	r	1	7	17	5	12.6	427
Hadoop	f	1	2	3	4.6	12.7	1392
	h	1	2	3	4.6	12.7	1392
	r	1	2	3	4.6	12.7	1392

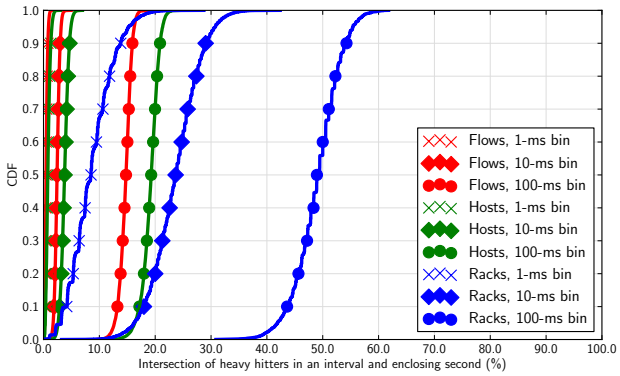
Table 4: Number and size of heavy hitters in 1-ms intervals for each of flow(f), host(h), and rack(r) levels of aggregation.

Table 4 shows statistics regarding the number and size of the heavy hitters that constitute 50% of the traffic in 1-ms intervals for each of the four server classes. Because we are interested in instantaneously large flows, we measure size in terms of rate instead of number of bytes sent over the lifetime of the flow. Next, we consider the the lifespan of heavy hitters, aggregated by 5-tuple, destination host and rack, and measured across intervals of 1, 10 and 100 milliseconds. Figure 10 shows the fraction of the heavy hitters that remain in subsequent time intervals. We do not show the Hadoop nodes, as our heavy-hitter definition almost always results in the identification of 1–3 heavy hitters at each of flow, host, and rack aggregation levels across all three time intervals.

Heavy hitter persistence is low for individual flows (red): in the median case, no more than roughly 15% of flows persist regardless of the length of period, a consequence of the internal burstiness of flows noted earlier. Host-level aggregation (green) fares little better; with the exception of



(a) Web server



(b) Cache follower

Figure 11: Intersection between heavy hitters in a subinterval with enclosing second

destination-host-level aggregation for Web servers, no more than 20% of heavy hitter hosts in a sub-second interval will persist as a heavy hitter in the next interval. Web servers have a higher rate over 100-millisecond periods since they have a relatively small number of cache servers and load balancers with which they communicate, while cache servers converse with many different Web servers.

It is not until considering rack-level flows (blue) that heavy hitters are particularly stable. In the median case, over 40% of cache heavy hitters persist into the next 100-ms interval, and almost 60% for Web servers. Heavy hitters from Web servers are more stable in general, with 32% of rack-level heavy hitters persisting in the median 1-ms interval case. Even so, heavy hitter persistence is not particularly favorable for traffic engineering. With a close to 50% chance of a given heavy hitter continuing in the next time period, predicting a heavy hitter by observation is not much more effective than randomly guessing.

Even if one could perfectly predict the heavy hitters on a second-by-second timescale, it remains to consider how useful that knowledge would be. We compare the heavy hitters from enclosing one-second intervals to the instantaneous heavy hitters from each of the subintervals within the second to see what fraction of the heavy hitters in a subinterval are heavy hitters across the entire enclosing second. A limited degree of overlap implies three things: First, it establishes an upper bound on the effectiveness of traffic engineering—a significant amount of ephemeral heavy hitter traffic would go unseen and untreated by the TE scheme. Second, it serves as an indicator of the difficulty of prediction in the first place; if a one-second prediction interval is not sufficient, smaller timescales (consuming more resources) may be needed. Finally, this metric is an indicator of burstiness, as it indicates the presence of a large number of ephemeral heavy hitters.

Figure 11 plots a CDF of the fraction of a second's overall heavy hitters that are instantaneously heavy in each 1/10/100-ms interval within the second. We show results for a Web server and cache follower—cache leaders are similar. At 5-tuple granularity, predictive power is quite poor, at less than 10–15%. Rack-level predictions are much more effective, with heavy hitters remaining heavy in the majority of 100-ms intervals in the median case for both services. Host-level predictions are more useful for Web servers than cache nodes, but only the 100-ms case is more than 30% effective.

5.4 Implications for traffic engineering

Facebook's extensive use of connection pooling leads to long-lived flows that seem like potential candidates for traffic engineering. These same services use application-level load balancing to great effect, however, leaving limited room for in-network approaches. Many existing techniques work by identifying heavy hitters and then treating them differently (e.g., provisioning a circuit, moving them to a lightly loaded path, employing alternate buffering strategies, etc.). For any such scheme to work, however, it must be possible to first identify the heavy hitters, and then realize some benefit.

Unfortunately, it appears challenging to identify heavy hitters in a number of Facebook's clusters that persist with any frequency. Moreover, even for the timescales and aggregation levels where it is possible (e.g., rack-level flows over intervals of 100-ms or larger), it is not clear there is a great deal of benefit to be gained, as the heavy hitters are frequently not particularly heavy for the vast majority of the period. Previous work has suggested traffic engineering schemes can be effective if 35% of traffic is predictable [14]; only rack-level heavy hitters reach that level of predictability for either Web or cache servers. This somewhat counter-intuitive situation results from a combination of effective load balancing (which means there is little difference in size between a heavy hitter and the median flow) and the relatively low long-term throughput of most flows, meaning even heavy flows can be quite bursty internally.

6. SWITCHING

Finally, we study aspects of the traffic that bear directly on top-of-rack switch design. In particular, we consider the size and arrival processes of packets, and the number of concurrent destinations for any particular end host. In addition, we examine the impact of burstiness over short timesales and its impact on switch buffering.

6.1 Per-packet features

Figure 12 shows the distribution of packet sizes for each of the four host types. Overall, the median packet size is

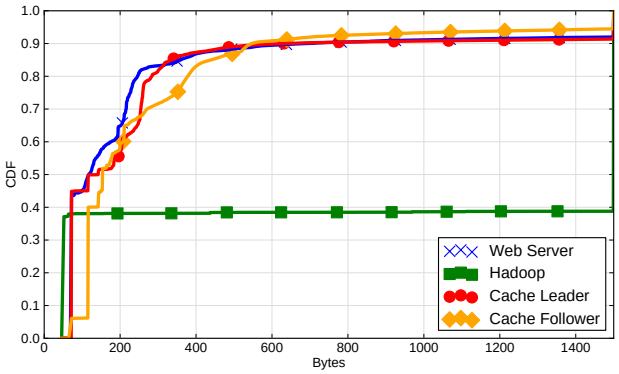
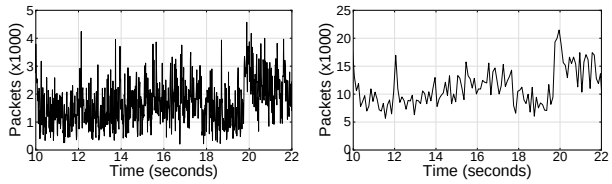


Figure 12: Packet size distribution



(a) 15 ms (b) 100 ms

Figure 13: Hadoop traffic is not on/off at 15 nor 100 ms

approximately 250 bytes, but that is significantly skewed by Hadoop traffic. Hadoop traffic is bimodal: almost all packets are either MTU length (1500 bytes for the servers we study) or TCP ACKs. Packets for the other services have a much wider distribution, but the median packet size for all of them is significantly less than 200 bytes, with only 5–10% of the packets fully utilizing the MTU.

Thus, while link utilization is low, the packet rate is still high. For example, a cache server at 10% link utilization with a median packet size of roughly 175 bytes generates 85% of the packet rate of a fully utilized link sending MTU-sized packets. As a result, any per-packet operation (e.g., VLAN encapsulation) may still be stressed in a way that the pure link utilization rate might not suggest at first glance.

6.2 Arrival patterns

Benson *et al.* observe that packet arrivals exhibit an on/off pattern at the end-host level [12, 13]. Hosts in Facebook's datacenter do not exhibit this behavior, even within Hadoop clusters. Figure 13 shows a time series of traffic sent by a Hadoop host (arriving at a RSW port) binned by 15- and 100-ms intervals. (c.f. Benson *et al.*'s analogous graphs [13, Figure 5] and [12, Figure 6]). If one considers traffic on a per-destination host basis, on/off behavior remerges (not shown), suggesting its disappearance may be due to a large number of concurrent destinations.

Figure 14 plots the CDF of inter-arrival times between outgoing TCP flows at each of the four types of servers we study. While a significant amount of traffic is routed over long-lived pooled connections, as is the case for request-response traffic between Web servers and cache followers, ephemeral flows do exist. The inter-arrival periods for flows emanating from all four classes of host are shorter than

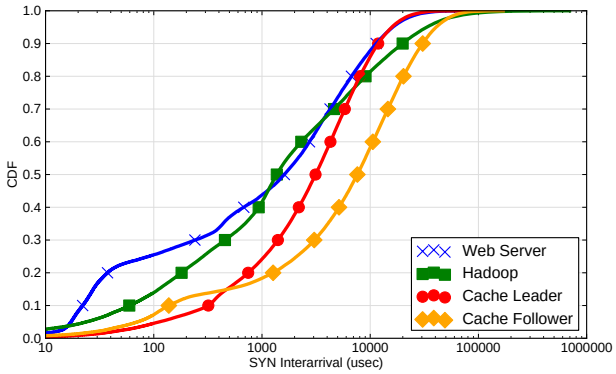


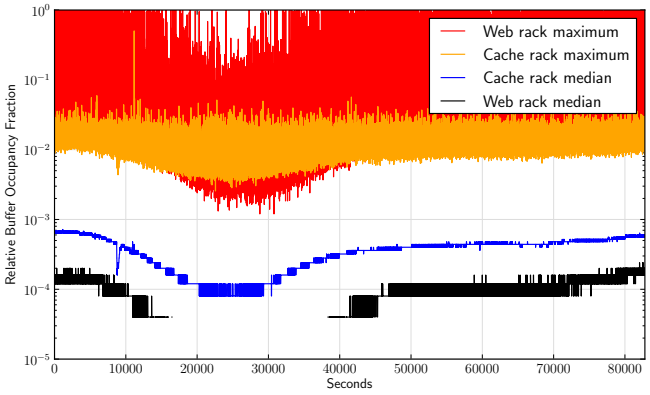
Figure 14: Flow (SYN packet) inter-arrival

those reported in the literature [26, Fig. 11], but to varying degrees. Hadoop nodes and Web servers see an order-of-magnitude increase in flow intensity relative to previous reports—likely due at least in part to the 10× increase in link rate—with median inter-arrival times of approximately 2 ms (i.e., more than 500 flows per second). Perhaps due to connection pooling (which would decouple the arrival of external user requests from the internal flow-arrival rate), the distribution of inter-arrival times for flows at both types of cache node are similar and longer: cache leaders see a slightly higher arrival rate than followers, with median inter-arrival timess of 3 and 8 ms, respectively.

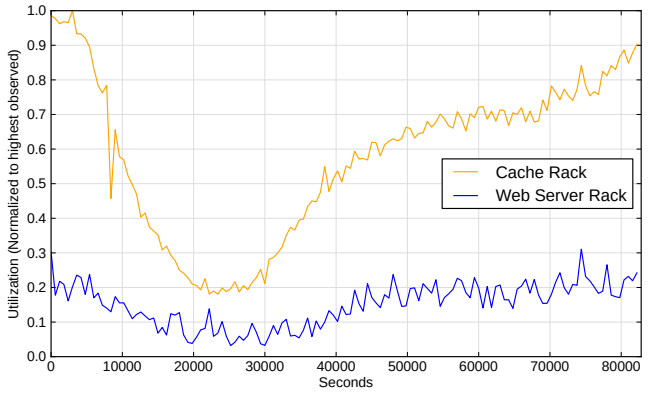
6.3 Buffer utilization

The combination of a lack of on/off traffic, higher flow intensity, and bursty individual flows suggests a potential increase in buffer utilization and overruns. Despite low average link utilization, bursty traffic can still lead to unacceptable loss rates. Recent work at Facebook has led to the development of in-house switching platforms [35], enabling us to gather buffer utilization statistics at fine granularity. In particular, we collect buffer occupancies over a 24-hour period for switches connecting Web servers and cache nodes at a 10-microsecond granularity. Figure 15a plots the median and maximum values per second for the entire period, normalized to the buffer size. In other words, a single point for the median series represents the 50th-percentile buffer occupancy during that second (out of 100,000 samples per second), normalized by the size of the buffer. We also plot the normalized average link utilization (Figure 15b) and egress drop rate (Figure 15c) over the same period, sourced via fbflow and SNMP counters, respectively.

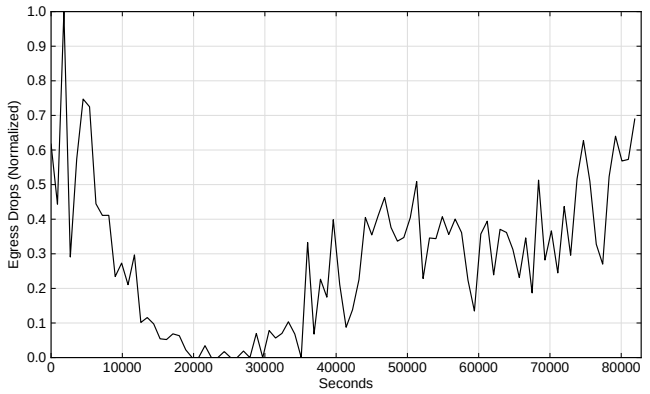
A few trends are apparent from our results. The first is that standing buffer occupancies are non-trivial, and can be quite high in the Web-server case. Even though link utilization is on the order of 1% most of the time, over two-thirds of the available shared buffer is utilized during each 10-μs interval. Diurnal variation exists in buffer occupancies, utilization and drop rate, highlighting the correlation between these metrics over time. Even with the diurnal trafic pattern, however, the maximum buffer occupancy in the Web-server rack approaches the configured limit for roughly three quarters of the 24-hour period. While link utilization is roughly cor-



(a) Normalized buffer occupancy, 10-microsecond resolution



(b) Link utilization, 10-minute average



(c) Web rack egress drops, 15-minute average

Figure 15: Correlating buffer occupancy, link utilization and packet drops in Web server and Cache racks

related with buffer occupancy within the Web-server rack, utilization by itself is not a good prediction of buffer requirements across different applications. In particular, the Cache rack has higher link utilization, but much lower buffer utilization and drop rates (not shown).

These buffer utilization levels occur despite relatively small packet sizes (Section 6.1). As utilization increases in the future, it might be through an increase in the number of flows, in the size of packets, or both. Either will have impacts on buffer utilization: larger packets with the same level of burstiness will use up more of the buffer, while a larger number of flows leads to a greater chance of multiple flows sending bursts of packets simultaneously. Thus, careful buffer tuning is likely to be important moving forward.

6.4 Concurrent flows

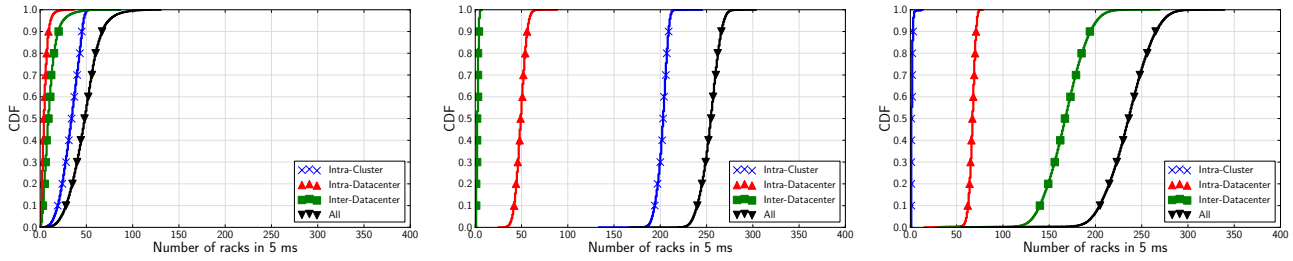
We consider concurrent to mean existing within the same 5-ms window (c.f. the 50-ms window considered by Alizadeh *et al.* while measuring a datacenter of hosts connected with 1-Gbps Ethernet [8]). We find that Web servers and cache hosts have 100s to 1000s of concurrent connections (at the 5-tuple level), while Hadoop nodes have approximately 25 concurrent connections on average—corresponding quite well with the findings of Alizadeh *et al.* [8, Figure 5]. That said, switches are obviously less concerned with individual connections than destination ports. If we group connections destined to the same host, the numbers reduce only slightly (by at most a factor of two)—and not at all in the case of Hadoop.

Given the general lack of intra-rack traffic, almost all flows will traverse an up-link port. Hence, it is perhaps more interesting to consider flows at the rack level—i.e., considering the number of different racks with which an individual host is communicating. Figure 16 shows the number of concurrent flows sent by a single host over a 5-ms interval to different classes of destination hosts for three different host types: cache follower, cache leader, Web server. Cache followers communicate with 225–300 different racks, while leaders talk to 175–350. In the median interval, both types of cache nodes communicate with approximately 250 other racks—the location of the racks varies dramatically as discussed previously, however. Web servers communicate with 10–125 racks concurrently, 50 in the median interval.

Some proposed switch designs [25, 30] employ different technologies for large flows. Hence, we restrict our focus to the heavy hitter racks, namely those destination racks that constitute the majority of the traffic. The median number of heavy-hitter racks is 6–8 for Web servers and cache leaders with an effective max of 20–30, while the cache follower has 29 heavy hitter racks in the median case and up to 50 in the tail. Due to the differences in locality, Web servers and cache followers have very few rack-level heavy hitters of their cluster, while the cache leader displays the opposite pattern. Even considering only heavy hitter racks, the number of concurrent destinations is still significantly larger than that reported by Alizadeh *et al.* [8]. In addition, the relative impermanence of our heavy hitters suggests that, for Frontend clusters at least, hybrid circuit-based approaches may be challenging to employ.

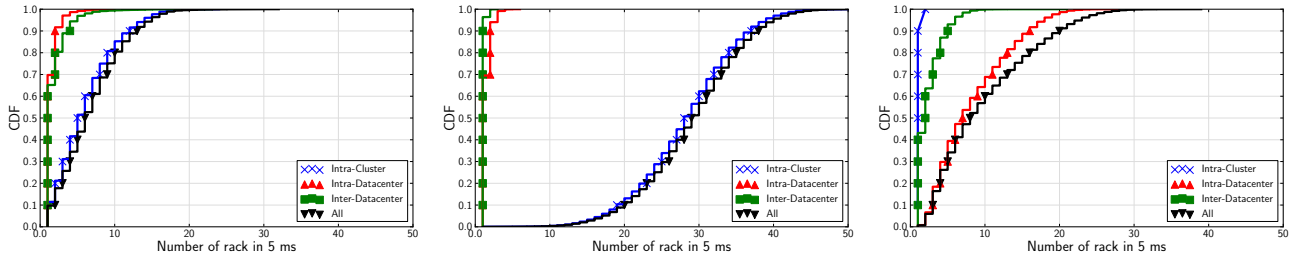
7. CONCLUSION

Facebook's datacenter network supports a variety of distinct services that exhibit different traffic patterns. We find that several deviate substantially from the services considered in the literature. The different applications, combined



(a) Web server (b) Cache follower (c) Cache leader

Figure 16: Concurrent (5-ms) rack-level flows



(a) Web server (b) Cache follower (c) Cache leader

Figure 17: Concurrent (5-ms) heavy-hitter racks

with the scale (hundreds of thousands of nodes) and speed (10-Gbps edge links) of Facebook's datacenter network result in workloads that contrast in a number of ways from most previously published datasets. Space constraints prevent us from providing an exhaustive account; we describe features that may have implications for topology, traffic engineering, and top-of-rack switch design.

Our methodology imposes a few limitations on the scope of this study. Using end hosts to capture and timestamp packets introduces scheduler-based variations on timestamp accuracy. In addition, we can only capture traffic from a few hosts at a time without risking drops in packet collection. Together, these constraints prevent us from evaluating effects like incast or microbursts, which are noted as being contributors to poor application performance [24]. Further, per-host packet dumps are necessarily anecdotal and ad hoc, relying on the presence of an unused capture host on the same rack as the target. While Fbflow is deployed datacenter-wide, the sheer amount of measurement data it provides presents another challenge—specifically, one of data processing and retention—which limits the resolution at which it can operate. We thus view effective network monitoring and analysis to be an ongoing and constantly evolving problem.

Acknowledgements

This work is supported in part by the National Science Foundation through grants CNS-1314921 and CSR-1018808. We are indebted to Theo Benson, Nick McKeown, Remzi Arpaci-Dusseau, our shepherd, Srikanth Kandula, and the anonymous reviewers for their comments and suggestions on earlier drafts of this manuscript. Petr Lapukhov, Michal Burger, Sathya Narayanan, Avery Ching and Vincent Liu provided invaluable insight into the inner workings of various Facebook services. Finally, and most significantly, Omar Baldonado catalyzed and faciliated the collaboration that enabled this study.

8. REFERENCES

[1] An open network operating system. http://onosproject.org.

[2] Scribe (archived). https://github.com/facebookarchive/scribe.

[3] L. Abraham, J. Allen, O. Barykin, V. Borkar, B. Chopra, C. Gerea, D. Merl, J. Metzler, D. Reiss, S. Subramanian, J. L. Wiener, and O. Zed. Scuba: Diving into data at Facebook. *Proc. VLDB Endow.*, 6(11):1057–1067, Aug. 2013.

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

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX NSDI*, Apr. 2010.

[6] A. Alameldeen, M. Martin, C. Mauer, K. Moore, X. Min, M. Hill, D. Wood, and D. Sorin. Simulating a $2M commercial server on a $2K PC. *IEEE Computer*, 36(2):50–57, Feb. 2003.

[7] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proc. ACM SIGCOMM*, Aug. 2014.

[8] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, Aug. 2010.

[9] A. Andreyev. Introducing data center fabric, the next-generation Facebook data center network. https://code.facebook.com/posts/360346274145943, 2014.

[10] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *Proc. ACM SIGMETRICS/Performance*, June 2012.

[11] L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer:An Introduction to the Design of*

Warehouse-Scale Machines. Morgan & Claypool, 2nd edition, 2013.

[12] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. ACM IMC*, 2010.

[13] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic charachteristics. In *Proc. ACM SIGCOMM WREN*, Aug. 2009.

[14] T. Benson, A. Anand, A. Akella, and M. Zhang. MicroTE: Fine grained traffic engineering for data centers. In *Proc. ACM CoNEXT*, Dec. 2011.

[15] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, M. Marchukov, D. Petrov, L. Puzar, Y. J. Song, and V. Venkataramani. TAO: Facebook's distributed data store for the social graph. In *Proc. USENIX ATC*, June 2013.

[16] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 98–109, New York, NY, USA, 2011. ACM.

[17] C. Delimitrou, S. Sankar, A. Kansal, and C. Kozyrakis. ECHO: Recreating network traffic maps for datacenters with tens of thousands of servers. In *Proc. IEEE International Symposium on Workload Characterization*, Nov. 2012.

[18] D. Ersoz, M. S. Yousif, and C. R. Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *Proc. IEEE International Conference on Distributed Computing Systems*, June 2007.

[19] N. Farrington and A. Andreyev. Facebook's data center network architecture. In *Proc. IEEE Optical Interconnects*, May 2013.

[20] N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. In *Proc. ACM SIGCOMM*, Aug. 2010.

[21] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A scalable and flexible data center network. In *Proc. ACM SIGCOMM*, Aug. 2009.

[22] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. *SIGCOMM CCR*, 38(3), July 2008.

[23] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *Proc. ACM SIGCOMM*, Aug. 2009.

[24] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan. Speeding up distributed request-response workflows. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 219–230, New York, NY, USA, 2013. ACM.

[25] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *Proc. ACM HotNets*, Oct. 2009.

[26] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: Measurements & analysiss. In *Proc. ACM IMC*, Nov. 2009.

[27] R. Kapoor, A. C. Snoeren, G. M. Voelker, and G. Porter. Bullet trains: A study of NIC burst behavior at microsecond timescales. In *Proc. ACM CoNEXT*, Dec. 2013.

[28] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A distributed control platform for large-scale production networks. In *Proc. USENIX OSDI*, 2010.

[29] A. Likhtarov, R. Nishtala, R. McElroy, H. Fugal, A. Grynenko, and V. Venkataramani. Introducing mcrouter: A memcached protocol router for scaling memcached deployments. https://code.facebook.com/posts/296442737213493, Sept. 2014.

[30] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter. Circuit switching under the radar with REACToR. In *Proc. USENIX NSDI*, Apr. 2014.

[31] R. Mack. Building timeline: Scaling up to hold your life story. https://www.facebook.com/note.php?note_id=10150468255628920, Jan. 2012.

[32] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proc. ACM HotNets*, 2009.

[33] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A cost comparison of datacenter network architectures. In *Proc. ACM CoNEXT*, Dec. 2010.

[34] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Proc. USENIX OSDI*, 2010.

[35] A. Simpkins. Facebook open switching system (fboss) and wedge in the open. https://code.facebook.com/posts/843620439027582/facebook-open-switching-system-fboss-and-wedge-in-the-open/, 2015.

[36] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *Proc. USENIX NSDI*, Apr. 2012.

[37] D. Sommermann and A. Frindell. Introducing Proxygen, Facebook's C++ HTTP framework. https://code.facebook.com/posts/1503205539947302, 2014.

[38] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive – a petabyte scale data warehouse using Hadoop. In *Proc. IEEE ICDE*, Mar. 2010.

[39] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, and M. Ryan. c-Through: Part-time optics in data centers. In *Proc. ACM SIGCOMM*, Aug. 2010.

[40] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng. Mirror mirror on the ceiling: Flexible wireless links for data centers. In *Proc. ACM SIGCOMM*, Aug. 2012.