Baatdaat: Measurement-Based Flow Scheduling for Cloud Data Centers

Fung Po Tso and Dimitrios P. Pezaros
School of Computing Science, University of Glasgow, G12 8QQ, UK
Email: {posco.tso, dimitrios.pezaros}@glasgow.ac.uk

***Abstract*—Software-Defined Networking (SDN) allows for efficient network-wide Traffic Engineering through the logical centralization of the control plane over individual switches that perform packet forwarding independently. Such abstraction is particularly suitable for Data Center (DC) networks that need to react to fluctuating traffic dynamics over short timescales.**

In this paper, we propose a low-cost, SDN-based system that exposes the temporal network-wide utilization through direct measurement, rather than estimation. We then present the *Baatdaat*[1] flow scheduling algorithm which uses spare DC network capacity to mitigate the performance degradation of heavily utilized links. Results show that *Baatdaat* achieves close to optimal Traffic Engineering by reducing network-wide maximum link utilization by up to 18% over ECMP, while at the same time improving flow completion time by as much as 41% - 95% for different types of flows.

I. Introduction

Cloud computing is emerging as an important paradigm where ICT resources are outsourced and hosted over generic Data Center network infrastructures which need to accommodate a wide range of services, from private data processing to public website hosting. With the ability for a tenant to instantaneously initialize resources and use them for a diverse set of processing tasks, this can make DC network traffic highly unpredictable.

Server and network infrastructure typically account for 45% and 15% of the overall cost of a Cloud DC, respectively [1]. However, the network often constitutes a barrier to DC performance since it can fragment resources, leading to low server utilization [2]. Today's DCs mostly run massive data analysis applications, such as, e.g., computing the web search index, that require extensive communication between servers, and hence the speed of computation is hindered by congestion-led propagation delay between servers. Therefore, overall Return-On-Investment (ROI) for Cloud providers can be significantly increased by improving the network performance of the underlying DC topology.

Equal Cost Multipath (ECMP) forwarding is commonly deployed to split traffic flows across redundant shortest paths. ECMP is easy to implement as it statically hashes one or more tuples of packet headers (e.g., protocol, source/destination address, port, etc.) and subsequently schedules flows based on their hashed values, ensuring that packets of the same flow are all scheduled over the same path. However, it is shown that ECMP's static and imperfect hashing produces uneven load distribution for large [3] and small [4] flow sizes, which eventually leads to congestion [5]. An alternative is therefore required that can efficiently route traffic flows through a DC network in real-time based on temporal traffic dynamics, while limiting or avoiding congestion on any link.

It is well-documented that DCs typically exhibit highly bursty traffic characteristics and encounter congestion across a significant number of links [2], especially at higher layers of the topology which are often significantly oversubscribed [5]. Novel DC architectures and traffic engineering (TE) techniques have been proposed to alleviate congestion mainly through distributing load over redundant paths. However, they either don't take current network state into consideration [6], [7], they require application adaptation [8], or they only consider a limited number of shortest paths hence not exploiting the full topological diversity [3].

In this paper, we introduce *Baatdaat*, a novel flow scheduler for reducing congestion in DC networks based on real-time direct measurements of network utilization, and the use of non-shortest albeit lightly-utilized paths (detours) to schedule traffic flows.

Unlike wide-area Internet topologies, DCs are densely packed with servers requiring only a small number of hops and a small propagation delay to reach any other server. With this in mind, we believe congestion can be minimized while at the same time flow completion times can be reduced, since the marginally increased latency due to a detour can be offset by less packet drops and retransmissions over less congested links. In order to realize this traffic engineering scheme, we exploit Software Defined Networking (SDN) to analyze and schedule flows in real-time. SDN is a fairly recent concept of centralizing the network's control plane so that network-wide management can be centrally programmed in software and subsequently enforced through the installation of rules on the switches along the path. OpenFlow [9] is an example of SDN that can run in network switches, with forwarding decisions taken by a centralized controller. OpenFlow is being increasingly supported by switch vendors in production environments to test and deploy novel networking protocols and routing algorithms over legacy infrastructures [10], [11].

Baatdaat uses OpenFlow running on NetFPGA programmable switches [12], that enables real-time dynamic flow scheduling which can adapt to instantaneous traffic bursts as well as to average link load. Our scheduling system uses hardware-assisted switch-local link utilization measurements that are periodically aggregated to a central controller which in

[1]*Baatdaat* is Cantonese for "reachable in all directions".

turn builds a topology-wide network utilization map. Flows are subsequently scheduled over lightly utilized paths, allowing detours and improving flow completion time. Unlike existing approaches that try to randomize load balancing (e.g., ECMP, VL2), our flow scheduler actively avoids congested links, and does not require any changes at the application level to improve global network performance.

The main contributions of this work are:

- A hardware-assisted traffic monitor that measures link utilization on the switches at line-speed.
- A SDN-based adaptive flow scheduling system that orchestrates and enforces network-wide Traffic Engineering based on link-local metrics.
- Network-wide congestion reduction for DC networks over existing scheduling algorithms by spreading load over shortest and detour paths, while at the same time reducing flow completion times.

Our results show that *Baatdaat* can achieve close to optimal Traffic Engineering, improving over ECMP by up to 18% for different types of load [8]. Although long-term provisioning of DC infrastructures is necessary to accommodate growth in service demand, the improved network utilization offered by *Baatdaat* over short timescales can provide the necessary short-term traffic shaping to avoid resource outages at the onset of sudden traffic dynamics.

The remainder of the paper is structured as follows. Section II discusses the motivation and rationale behind measurement-based flow scheduling for Cloud DC topologies. Section III presents *Baatdaat*'s algorithmic and implementation details, and its different hardware and software components. Section IV presents the experimental results demonstrating *Baatdaat*'s improved DC-wide utilization, flow completion times and low overhead. Section V discusses related work and Section VI concludes the paper.

II. Motivation and Rationale

A. Data Center Traffic Patterns

A number of studies have looked into traffic patterns exhibited by Cloud DCs, revealing some interesting idiosyncrasies. Although a significant fraction of traffic appears to be localized and staying inside a rack, congestion does occur in various layers of the infrastructure despite sufficient capacity being available elsewhere that could be used to alleviate hotspots [13]. Congestion is shown to deteriorate application performance by reducing server-to-server I/O throughput [2]. In terms of flow distribution characteristics, data mining and web service DCs mostly exhibit small flows typically completed within 1s. Flow inter-arrival times vary from 1 flow per 15 milliseconds to 100 flows per millisecond at servers and Top-of-Rack switches, respectively, while on average, there are 10 concurrent flows per server active at any given time [2], [4], [7]. Last but not least, DC traffic patterns change rapidly and unpredictably due to the use of pseudo-random processes to improve DC application performance [7].

ECMP is shown to perform suboptimally with such traffic patterns and not being able to mitigate congestion [2], [4]. Alternative offline traffic engineering techniques that require advance knowledge of traffic demands are also unsuitable for such environments that exhibit long-term unpredictability due to small and bursty flows in the Cloud DCs. Small flow sizes and inter-arrival times also put recently proposed TE techniques based on centralized decision making, e.g., OpenFlow, under question, since the central scheduler would have to deal with a rather high volume of scheduling requests in very short time intervals. For example, MicroTE largely depends on predictable traffic demands, hence not providing any obvious performance gain over ECMP in such environments [8]. Worse still, forwarding every flow to the controller would severely impact the delay-sensitive (i.e., deadline-aware) small flows. Hedera on the other hand, only schedules large flows and would therefore have minimal or no performance gain over ECMP under the observed Cloud DC traffic patterns [3]. The above observations call for an adaptive load-aware scheduler that would react in short timescales, based on the temporal traffic demands and would leverage spare capacity elsewhere to mitigate congestion.

B. Optimality

Consider a network as a directed graph $G = (\mathbb{V}, \mathbb{E})$, where V is the set of nodes (where $N = |\mathbb{V}|$), E is the set of links (where $E = |\mathbb{E}|$), and link (u, v) has capacity $c_{u,v}$. The offered traffic is represented by a traffic matrix $D(s, t)$ for source-destination pairs indexed by (s, t). Let the flow of packets destined to a single destination on link (u,v) be $f_{u,v}$. TE typically considers a link-cost function $\Phi(\{f_{u,v}, c_{u,v}\})$ that is an increasing function of $f_{u,v}$, such as, e.g., the link utilization function $f_{u,v}/c_{u,v}$. Thus, such TE must meet the requirement in Equation 1, assuming $f^t_{u,v}$ is the flow of packets destined to node t over link (u, v).

$$min \quad \Phi(\{f_{u,v}, c_{u,v}\}) \tag{1a}$$

$$s.t. \quad \sum_{v:(s,v)\in\mathbb{E}} f^t_{s,v} - \sum_{u:(u,s)\in\mathbb{E}} f^t_{u,s} = D(s,t) \forall s \neq t \tag{1b}$$

$$f_{u,v} \triangleq \sum_{t\in\mathbb{V}} f^t_{u,v} \leq c_{u,v} \forall (u, v) \tag{1c}$$

$$vars. \quad f^t_{u,v}, f_{u,v} \geq 0. \tag{1d}$$

However, finding flow routes in a general network while not exceeding the capacity of any link is the *multicommodity flow problem* which is NP-complete for integer number of flows [14]. The Penalizing Exponential Flow spliTting [15] scheme can achieve optimal traffic engineering for such multicommodity flow by allowing packet forwarding through non-shortest paths and non-integer link weights. [16] modified and implemented PEFT for DC environments, yet it requires substantial modifications of the existing networking hardware. In light of this, *Baatdaat* trades optimality for deploy-ability, and uses practical heuristics to leverage non-shortest paths and significantly improve DC performance over existing schemes.

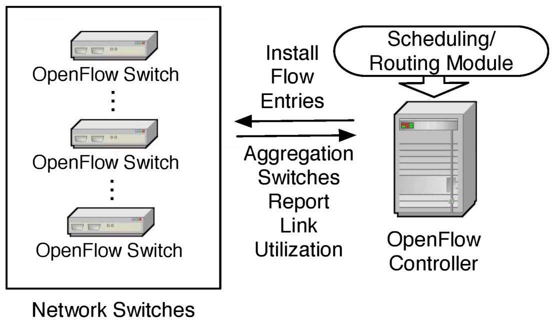

Fig. 1: System architecture.

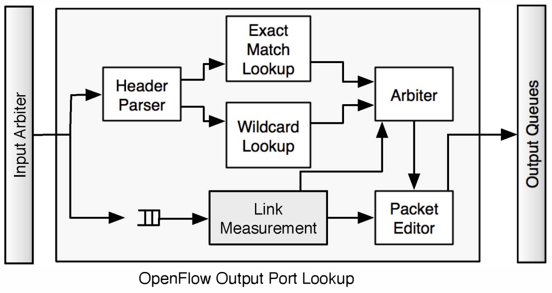

Fig. 2: Design of the link utilization module within the OpenFlow switch pipeline.

III. System Design & Architecture

A. *Design Requirements*

Clearly, a system that performs flow scheduling based on real-time link status feedback has to meet the following design requirements:

- R1: Each switch should be able to monitor link utilization of its associated links at line rate.
- R2: Flows should be allowed to opportunistically take detours (i.e., longer paths) where it is beneficial to do so.
- R3: Global flow scheduling should easily scale to the size of a DC network.

While current switches do not typically support link measurements, R1 requires the ability to conduct line-rate measurement on switches without impacting their forwarding performance. We have implemented such a component alongside NetFPGA's OpenFlow implementation [12]. R2 is highly desirable given the numerous measurement studies unveiling that DC networks generally remain underutilized, albeit with a small fraction of congested links [5]. Using non-shortest paths to route traffic can improve network-wide utilization, so long as it doesn't impact individual flow performance. Scalability, R3, is crucial for centralized approaches since efficiency of the controller has a direct impact on the performance of the network as a whole. In order not to turn the controller into a performance bottleneck, we have adopted a mix of distributed and centralized scheduling approaches for *Baatdaat*. First, all switches individually monitor their link utilization and independently schedule flows onto the links, while the controller only determines detour paths. This way, the controller's workload can be significantly reduced. To this extent, the *Baatdaat* architecture is composed by OpenFlow switches in DC-compatible arrangements such as, e.g., canonical and fat-tree, and a single OpenFlow controller to collect link utilization statistics amongst aggregation switches and to determine flow detours, as shown in Fig. 1.

B. *Measurement Module Design & Implementation*

OpenFlow pushes forwarding decisions onto a logically centralized controller, which can in turn add and remove forwarding entries in OpenFlow switches. This form of network virtualization abstracts complexity from hardware to controller software, allowing control logic to be defined and programmed in software. Each OpenFlow switch matches incoming flows using exact-match or wildcards on specific protocol fields. If a match is not found for an arriving packet, the packet is sent to the controller which registers the new flow and decides on the action(s) that should be applied to all subsequent packets matching the same filter. The action is then sent to the switch and cached as an entry in the switch's flow table. Subsequent packets belonging to the same flow are then directly forwarded at line-rate through the switch without the need for redirection to the controller.

We have implemented a hardware-based link measurement module in-line with NetFPGA's OpenFlow switch implementation [17], as illustrated in Fig. 2. As soon as a packet arrives at the input port, the link utilization module adds the size of the packet to the overall bytes arriving on the specific link. Similarly, after the packet is processed by the output port lookup, its size will be added to the total for the outgoing link. After a specified time interval the link utilization will be calculated.

C. Baatdaat *Scheduling*

Intuitively, the scheduling loop should be short enough to capture all flow interarrivals. In this case, centralized schedulers that decide which path to pin a flow on may be hard pressed to keep up and may generate high control traffic overhead. While it is reported that flow inter-arrivals per switch port can be as long as 10-15 ms for Cloud DCs, we set our measurement loop to run every 1 ms by default with an aim to capture instantaneous traffic bursts. The measurement results are stored locally as an array of size equivalent to the number of ports in the switch. *Baatdaat* is a flow-based scheduler that uses 5-tuple header hashing to guarantee packets belonging to the same flow traverse the same path, so that packet re-ordering is avoided. Hence, when a new flow joins, the switch first queries the number of outgoing ports, such as output ports for multiple shortest paths, and then places the flow onto the least utilized link by comparing their measured link utilization. When all outgoing links are reported to have the same utilization, such as all links are 0% utilization right after initialization, the flow is then randomly scheduled using commodity ECMP. Aggregation switches, however, need to report statistics to the controller, which will determine whether or not the new flow should take a detour.

To make the scheduling manageable, we impose three constraints on any flow taking a detour: 1) the flow is downlink traffic from the aggregation switches. This means detours only happen between the aggregation and the Top-of-Rack (ToR) layers of the DC topology. In DCs, cross-aggregation-switch traffic is more common than cross core switch due to the traffic locality nature of DC networks. The multi-root tree architecture provides large amounts of interconnect between aggregation and ToR layers, up to a full mesh in a fat-tree topology. This offers significant path diversity if detour is allowed; 2) the link utilization of shortest paths is $\geq$30%. Without a threshold value, *Baatdaat* would schedule traffic on longer paths even when shortest paths are lightly utilized. However, if the threshold is $\geq$50%, the detour scheme sometimes does not come into effect before shortest paths become substantially congested; 3) the flow can only take a detour of 2 hops longer than the shortest path to prevent oscillation and flooding-like effects in the network.

While each switch keeps link utilization values locally in the form of an array, the controller maintains a *link utilization matrix*, M. Only aggregation switches send and update link utilization of their associated links to the OpenFlow controller. Assuming M is a $m \times n$ matrix, m denotes the number of aggregation switches in a pod, while n is the number of downlink ports on an aggregation switch. For ease of presentation, we assume switches and their ports are sequentially numbered. For example, M_{ij} is the link utilization of port *j* of switch *i*. By allowing the detour, path diversity in the fat tree network increases by $k/2 \times (k/2-1) \times (k/2-2)$ compared to the original shortest path scheme which provides a path diversity of $k/2$. The link utilization of detour paths is defined as $max($*detour links*$) \times c$, where c is a weighting factor to reflect the fact that it is a longer path. In our simulation we found that $c = 1.5$ works well as it gives good detour opportunity. We set $c > 1$ to bias longer paths, but setting $c \geq 2$ will see significant decrease in detour opportunities. The next question is, *how does the controller determine detour paths?* Clearly, since detour paths are limited to being no more than two hops longer than shortest paths and only happen in the aggregation-ToR layer, so starting from any aggregation switch, any detour of 2 more hops will lead to a path of 4 hops. Hence, the algorithm starts by constructing an acyclic tree of depth 3 as shown in Fig. 3, with k switches as vertices and links among them as edges. Depth-first search is then applied to identify and compute link utilization along the path. The OpenFlow controller, after determining a detour path for a new flow, installs the OpenFlow entry to all affected switches.

Clearly, the time complexity of this search is $O((k/2-1)^2)$ for a fat-tree topology although the search tree has a depth of 3. For a k-ary fat-tree, there are $k/2$ aggregation and ToR switches in a pod, respectively. In the example given in Fig. 3 we can see that the sequence for a detour path is: *(ToR switch* $\xrightarrow{link1}$ *aggregation switch* $\xrightarrow{link2}$ *ToR* $\xrightarrow{link3}$ *aggregation switch* $\xrightarrow{link4}$ *ToR)*. As *link 1* is independently chosen by the ToR switch itself, *link 2 – link 3 – link 4* is the actual detour determined by the controller. Therefore, starting from an aggregation switch, due to the dense interconnected nature of each pod, it has $k/2$ links to ToR switches including the source node. But the source ToR switch has to be excluded to prevent a loop, so the algorithm only needs to search for (and to compute link utilization of) $k/2-1$ links at the first

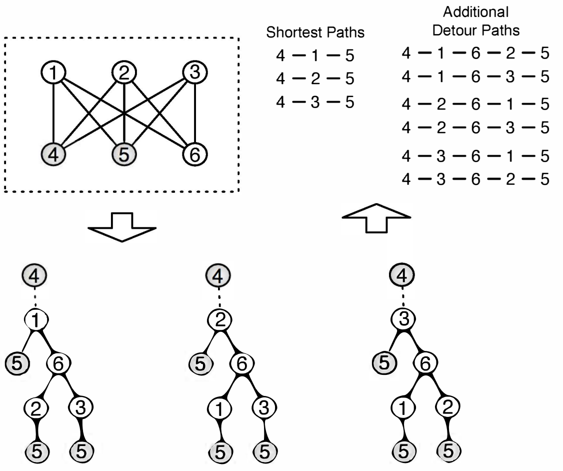

Fig. 3: Example of a calculated detour in a pod, assuming a flow is to be forwarded from node 4 to node 5.

iteration (depth 1) for *link 1*. Similarly, there are $k/2-1$ iterations each at the depth 2 for *link 3*. The complexity so far is $O((k/2-1)^2)$. However, for *link 4*, due to the mesh interconnect of aggregation–ToR switches, the next hop must contain the destination node. Hence, the algorithm does not need to search any further and the overall complexity is $O((k/2-1)^2)$.

IV. Performance Evaluation

In this section, we evaluate the performance of *Baatdaat* with respect to reduced network-wide maximum link utilization and improved load-balancing due to path diversity. In order to test the properties of our system at scale, we have used the ns-3 network simulator for network-wide properties, and our hardware-assisted testbed implementation to evaluate the footprint of the time-critical processing elements.

A. Simulation Setup

We have simulated a $k = 8$ fat-tree topology (128 servers grouped into 8 pods with 8 switches each) with 1Gb/s interconnect links and the OpenFlow module enabled. Simulated flows consist of uniformly chosen 4 KB, 8 KB, and 100 KB flows, to include the range of latency-sensitive flows common in DC networks. The traffic is generated by sending these uniform flows to 100 other servers at different racks to create high and imbalanced network load.

B. Network-wide Experimental Results

Fig. 4 shows the measured Maximum Link Utilization (MLU) for 4 KB, 8 KB and 100 KB flows for ECMP and *Baatdaat* approaches, respectively. The results demonstrate that *Baatdaat* consistently outperforms ECMP for all types of flows by up to 18%, which is almost the amount by which ECMP deviates from optimal (15%-20%) under high load [8].

Looking more closely into the CDFs, the improvement for 4 KB flows is more uniform between the 30% – 70% of the MLU region. Interestingly, for 8 KB flows, the improvement is more significant around the 60% MLU region, and the

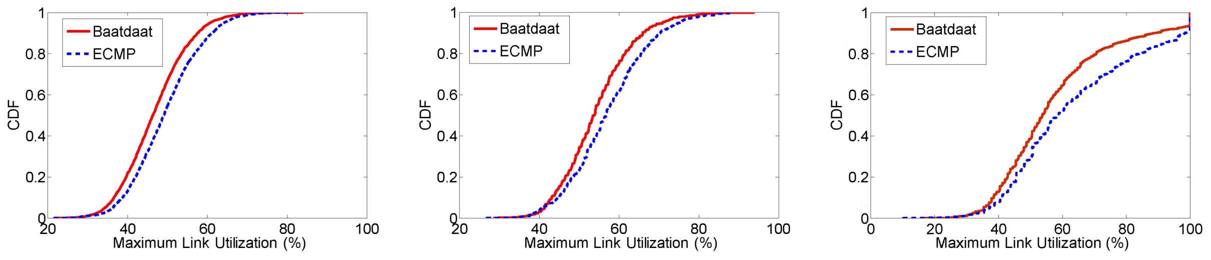

Fig. 4: CDF of maximum link utilization for 4 KB, 8 KB and 100 KB flows respectively.

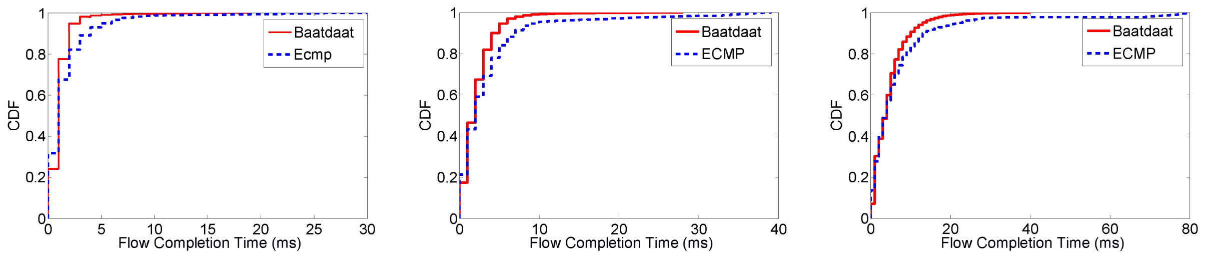

Fig. 5: CDF of flow completion time for 4 KB, 8 KB and 100 KB flows respectively.

improvement for 100 KB flows is more visible around the 70% MLU region, which demonstrates that the detour approach efficiently mitigates the increased congestion by offloading some flows onto less congested albeit longer paths. This is an important feature since it offers an additional 10-18% headroom to DC providers using short-term TE to avoid resource outages due to temporal increases in traffic.

Since DC RTTs can be as low as 250μs [4], can a detour of two more hops degrade individual flow completion time? We show the flow completion times for 4 KB, 8 KB and 100 KB flows for ECMP and *Baatdaat*, respectively, in Fig. 5a, 5b, and 5c. While most traffic flows complete within 1 ms for 4 KB, 3ms for 8 KB and 10 ms for 100 KB flows, we can also see that *Baatdaat* can significantly improve flow completion time, particularly between the 60th - 100th percentile. Obviously, this result demonstrates that allowing detour not only does not deteriorate but rather it improves flow completion time. ECMP is slightly better than *Baatdaat* in the 30th percentile, due to small flows being more latency sensitive, and detour adding a little extra ($\leq$ 1 ms) transmission time. However, we can see in the 70th - 100th percentile, at least 10% flows complete faster in *Baatdaat* than they do in ECMP. Allowing larger number of flows to complete faster is an important feature for data center networks because many Web applications that run over these topologies require strict deadlines [4].

C. Hardware Module Performance

We have evaluated the impact of the real-time hardware measurement module on the NetFPGA's packet switching performance using a back-to-back high-speed throughput test. Two host machines connected via a NetFPGA switch exchanged data at full 1 Gb/s NIC speed. The NetFPGA switch monitors traffic (byte counts) on its input interfaces and then passes packets onto the appropriate output interface. We have compared the NetFPGA's forwarding efficiency to examine whether enabling line-rate monitoring would impact the hardware's forwarding efficiency. Fig. 6 shows the throughput ratio of the instrumented system (with monitoring functionality) over that of a plain switch. It is evident that the in-line traffic monitoring implementation incurs no significant deterioration on the router's forwarding efficiency, since the throughput ratios are mostly in the range between 0.99 and 1. This implies that including real-time network monitoring in the routing protocol algorithm can be seamlessly accommodated using commodity and inexpensive hardware acceleration.

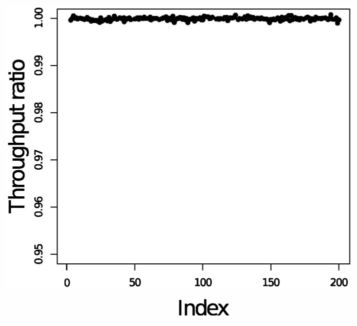

Fig. 6: Throughput ratio of instrumented (monitoring) system over plain switch.

V. Related Work

Traffic Engineering and routing techniques have been widely used for Internet topologies [18], [19], [20]. However, they do not achieve optimal routing or are often based on steady or highly predictable traffic, something that constitutes them unsuitable for DC networks.

With the increasing use of Cloud computing over DC infrastructures, new routing schemes have been developed to exploit their highly-connected nature. Fat-tree, the most dominant topology in DCs today, typically employs ECMP [6] to balance traffic load amongst redundant shortest paths. However, ECMP suffers from hash collisions, which can result in an imbalance of flow allocations across paths. VL2 [7] is a virtual layer 2 infrastructure which uses Valiant Load Balancing (VLB) to randomize packet forwarding, and essentially exhibits similar limitations to those of ECMP [3].

Hedera [3] complements ECMP by identifying large flows exceeding 100 Mb/s via monitoring of throughput at network edge switches and then scheduling these flows along a redundant path with suitable capacity. However, this approach becomes limited as network utilization increases and flows fair-share the bottleneck bandwidth. On the contrary, our work attempts to place flows based on minimum link utilization and independent of flow size. PEFT uses a mix of shortest and non-shortest paths with exponential penalisation on the latter [15]. It has been shown to achieve optimal routing for Internet networks with decisions made on a hop-by-hop basis, but it is designed for offline traffic engineering over ISP topologies with predictable traffic matrices, not the highly unpredictable DC traffic characteristics that occur even over long timescales. DeTail [4] attempts to reduce flow completion time in DC networks for web site load times by using flow priorities and switch port buffer occupancies to determine next hop behavior. However, DeTail requires major changes at several layers of the networking stack, and applications must be modified to communicate flow priorities for latency-sensitive traffic scheduling. *Baatdaat* is much simpler to deploy, requiring no in-depth knowledge of individual application flows, yet still significantly reducing flow completion time in congested networks. MicroTE [8] uses short-term traffic patterns and partial predictability to achieve its goals. However, with highly unpredictable DC traffic, it becomes equivalent to or worse than ECMP, and requires significant changes to end hosts as traffic monitoring duties are pushed to servers.

VI. Conclusion

In this paper we have presented *Baatdaat*, a novel flow scheduling system for DC networks. *Baatdaat* uses a modified NetFPGA implementation of OpenFlow to directly measure the temporal, network-wide utilization of the infrastructure and to subsequently schedule flows over redundant lightly-utilized and non-shortest paths. Unlike existing flow scheduling algorithms, *Baatdaat* takes into consideration the current state of the network and dynamically adapts scheduling to make use of local link utilization information available at each switch, and improves network performance on a global scale. Moreover, it can be readily deployed in DCs already running OpenFlow-enabled switches.

Simulation results have shown that *Baatdaat* can substantially improve flow completion time in the presence of congestion, and achieves close to optimal Traffic Engineering through reducing network-wide utilization by up to 18% over ECMP. Results on the NetFPGA platform demonstrate that traffic monitoring and real-time flow-level scheduling can be seamlessly incorporated into appropriately instrumented switches without impacting their forwarding performance.

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, January 2009.

[2] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM SIGCOMM Internet Measurement Conference (IMC'09)*, 2009, pp. 202–208.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," *The 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10)*, 2010.

[4] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM*, 2012.

[5] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," *Internet Measurement Conference (IMC)*, 2010.

[6] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992, IETF*, 2000.

[7] A. Greenberg, J. 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," *ACM SIGCOMM*, 2009.

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

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, 2008.

[10] *IBM RackSwitch G8264 & G8264T http://www-03.ibm.com/systems/networking/switches/rack/g8264/*.

[11] "NEC ProgrammableFlow networking," *http://www.necam.com/PFlow/*.

[12] NetFPGA, "http://www.netfpga.org/."

[13] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM Internet Measurement Conf. (IMC'10)*, 2010, pp. 267–280.

[14] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, no. 184-193, 1975.

[15] D. Xu, M. Chiang, and J. Rexford, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on Networking*, April 2011.

[16] F. P. Tso and D. P. Pezaros, "Improving data centre network utilisation using near-optimal traffic engineering," *IEEE Transactions on Parallel and Distributed Systems*.

[17] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an openflow switch on the netfpga platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '08. New York, NY, USA: ACM, 2008, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/1477942.1477944

[18] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," *IEEE INFOCOM*, 2001.

[19] H. Wang, H. Xie, L. Qiu, R. Yang, Y. Zhang, and A. Greenberg, "Cope: Traffic engineering in dynamic networks," *ACM SIGCOMM*, 2006.

[20] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," *ACM SIGCOMM*, 2005.