Hunting Mice with Microsecond Circuit Switches

Nathan Farrington, George Porter, Yeshaiahu Fainman, George Papen, Amin Vahdat[†]

UC San Diego UC San Diego and Google[†]

ABSTRACT

Recently, there have been proposals for constructing hybrid data center networks combining electronic packet switching with either wireless or optical circuit switching, which are ideally suited for supporting bulk traffic. Previous work has relied on a technique called hotspot scheduling, in which the traffic matrix is measured, hotspots identified, and circuits established to automatically offload traffic from the packet-switched network. While this hybrid approach does reduce CAPEX and OPEX, it still relies on having a well-provisioned packet-switched network to carry the remaining traffic. In this paper, we describe a generalization of hotspot scheduling, called traffic matrix scheduling, where most or even all bulk traffic is routed over circuits. In other words, we don't just hunt elephants, we also hunt mice. Traffic matrix scheduling rapidly time-shares circuits across many destinations at microsecond time scales. The traffic matrix scheduling algorithm can route arbitrary traffic patterns and runs in polynomial time. We briefly describe a working implementation of traffic matrix scheduling using a custom-built data center optical circuit switch with a 2.8 microsecond switching time.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*circuit-switching networks, packet-switching networks, network topology*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Data Center Networks, Circuit Switching

Hotnets '12, October 29–30, 2012, Seattle, WA, USA.

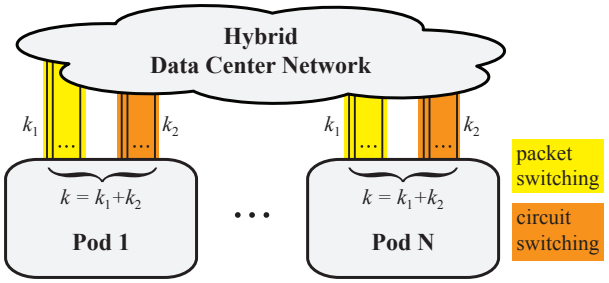

Figure 1: Hybrid data center network architecture: each of the N pods has k uplinks, partitioned between k_1 packet switch ports and k_2 circuit switch ports.

1. INTRODUCTION

Recently, there have been proposals for constructing hybrid data center networks (Fig. 1), combining electronic packet switching with either wireless [8,18] or optical circuit switching [5, 7, 17]. A key advantage of such hybrid networks is that they provide increased amounts of bisection bandwidth with lower CAPEX and OPEX compared to an equivalent packet-switched network [7].

These hybrid networks have all used similar circuit scheduling frameworks that we call hotspot scheduling (HSS). In HSS, (a) the inter-pod[1] traffic matrix is measured, (b) the traffic demand matrix is estimated, (c) hotspots are identified, and (d) a centralized scheduler establishes physical circuits between pods to automatically offload traffic from the congested packet-switched network onto the circuit-switched network. The remaining traffic is routed over the packet-switched network. One of HSS's shortcomings is that expensive algorithms need to be run before each switch reconfiguration. Also, if a hotspot is not large enough to saturate a circuit, then the excess circuit capacity goes to waste because the circuit cannot be shared. These two limitations make HSS static and inflexible.

We argue that first-generation HSS only scratches the surface of what is possible with circuit switching in the data center. In this paper, we describe a generalization of HSS

[1]A *pod* is a unit of server aggregation, e.g., 1024 servers. A pod is roughly equivalent to a row of racks.

called traffic matrix scheduling (TMS), where most or even all bulk traffic is routed over circuits. TMS rapidly time-shares circuits across many destinations at microsecond time scales. In fact, TMS reduces to HSS as a special case. One reason TMS has never been proposed before is that its implementation is impractical using the millisecond time scales of previous hybrid network prototypes. It is only with microsecond time-scale circuit switches that TMS becomes possible. TMS makes circuit switching much more efficient than with first-generation HSS because much more of the network traffic can now be offloaded to circuits.

Whereas HSS couples scheduling and switching together, TMS decouples them. Like HSS, TMS uses expensive algorithms to construct a circuit switch schedule. But unlike HSS, once a schedule has been constructed, it is then implemented in hardware at microsecond time scales. This separation of scheduling and switching is what allows TMS to schedule a larger amount of traffic than HSS despite the fact that the scheduling algorithms are not any faster. In a sense, TMS does a much better job of amortizing the cost of the scheduling algorithms than HSS.

While the circuit switch is busy rapidly setting up and tearing down circuits between pods, hosts in these pods are observing the state of the circuit switch to know when to transmit packets. In other words, the hosts are using time-division multiple access (TDMA) over standard packet-based protocols such as Ethernet [14]. We will say little about TDMA in this paper, other than to point out that variable-length TDMA is our chosen method for hosts to access the circuit-switched network, and instead focus on the design and implementation of the traffic matrix scheduling algorithm.

2. ALL-TO-ALL EXAMPLE

In this section, we walk through an example of TMS. Consider eight pods running Hadoop and generating a perfectly uniform all-to-all communication pattern. Fig. 2 (a) shows the pods physically connected to the same core circuit switch; Fig. 2 (b) shows them logically connected as a full mesh. Fig. 2 (c) shows the inter-pod traffic demand matrix with sources as rows, destinations as columns, and values as fractions of the total link rate. The diagonal is not zero because hosts send to other hosts in the same pod. Although this intra-pod traffic does not transit the core circuit switch, it is still accounted for in the traffic demand matrix. This matrix is the desired transmission rate of the hosts; it is the responsibility of the network to satisfy this demand.

With HSS, each pod would require 7 physical uplinks to implement the logical topology in Fig. 2 (b) even though each physical link would be heavily underutilized (1/8 of the link). But with TMS a single uplink is time-division multiplexed to create 7 virtual (logical) links, each with 1/8 of the capacity of one uplink, meaning that the entire traffic demand can be routed with a single physical uplink, just like a traditional electronic packet switch.

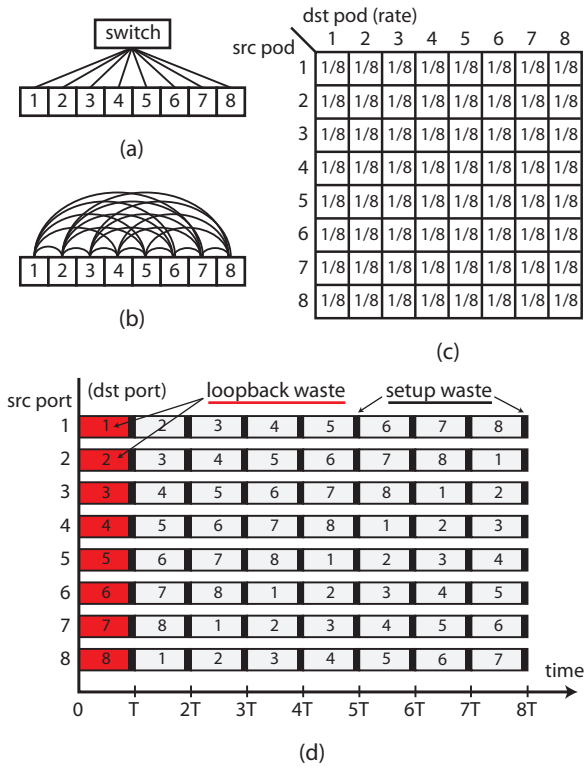

Figure 2: Eight pods running Hadoop with an all-to-all communication pattern. (a) physical topology, (b) logical topology, (c) inter-pod traffic demand matrix, (d) circuit switch schedule

The Gantt chart in Fig. 2 (d) shows a circuit switch schedule that partitions time into 8 equal-duration time slots. Over the course of the schedule, each source port will connect to each destination port for exactly 1/8 of the total time, thus implementing the logical full mesh topology in Fig. 2 (b) and allowing all of the traffic to be routed. The schedule then repeats. A circuit switch schedule has two sources of waste. First, loopback traffic does not leave the pod and transit the circuit switch, so any circuit switch loopback assignments are wasted, such as the assignment from $t = 0$ to $t = T$. Second, the circuit switch takes a non-negligible amount of time to switch and setup new circuits (t_{setup}), which we represent as black bars at the end of each time slot. No traffic can transit the circuit switch during this time. Reducing loopback waste requires careful scheduling whereas reducing setup waste requires using faster switching technologies.

2.1 Duty Cycle and Effective Link Rate

The time during which the circuit switch is being reconfigured is called t_{setup} and the time during which the circuit switch is stable and can carry traffic is called t_{stable}. The duty cycle, D, for a circuit switch schedule with equal-length time slots, t_{slot}, is given by

$$D = \frac{t_{\text{stable}}}{t_{\text{setup}} + t_{\text{stable}}} = \frac{t_{\text{stable}}}{t_{\text{slot}}} \quad (1)$$

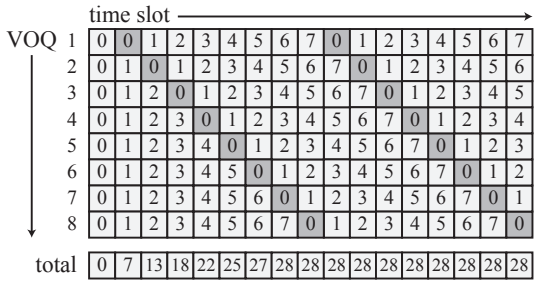

Figure 3: Virtual output queue (VOQ) buffer occupancies for a single host in pod 1 from cold start in units of time (T).

The duty cycle is important because it determines the effective link rate of the circuit switch. For example, if the nominal link rate, L, is 10 Gb/s and $D = \frac{90\mu s}{10\mu s+90\mu s} = 90\%$, then the effective link rate, $L_{\text{effective}}$ = 9 Gb/s. This means that the circuit switch would not actually be able to carry the full traffic demand matrix given in Fig. 2 (c) but only 90% of it.

There are three ways to increase $L_{\text{effective}}$. First, one can reduce t_{setup} by using a faster switch. Second, one can increase t_{stable} at the cost of increased host buffer requirements. Third, one can increase the nominal link rate, L. This third option is especially attractive given technologies such as WDM that can place tens of 10 Gb/s channels on the same physical link. If the traffic demand is less than $L_{\text{effective}}$, then 100% of the traffic can be routed over the circuit-switched network. But once the traffic demand exceeds $L_{\text{effective}}$, the circuit switch has become a bottleneck and we would have to route some traffic over the electronic packet-switched network or suffer a performance penalty.

2.2 Host Buffer Requirements

Traffic destined to hosts in the same pod is called pod-local traffic. Since pod-local traffic does not transit the circuit switch, we have the option of treating it as a special case in our analysis of host buffer requirements. However, we choose to simplify the analysis by not modeling pod-local traffic and leave this optimization to future work.

Each host in each pod maintains a set of N virtual output queues (VOQs) [12], one for every destination pod, including the host's own pod. All traffic generated by a host first goes to one of these queues. There can be additional queueing disciplines within a host to support arbitrary traffic engineering policies, but ultimately we assume traffic ends up in these VOQs before transmission. When a circuit is established, all traffic destined for that particular destination is drained from the respective queue. Fig. 3 shows the buffer occupancies of these VOQs of a single host in pod 1 from a cold start, in units of T. In less than one complete scheduling period a host has filled its VOQs to the steady-state level, in this case $28T$. A particular queue fills at a rate given by the traffic demand matrix until a circuit is established to the appropriate destination, then the queue is drained completely at precisely the time for the next switch reconfiguration, so that there are no standing queues. It is important to note that this example is for completely uniform all-to-all traffic; for other traffic demand matrices, the buffers will fill at different rates, but the circuit switch schedule should be chosen so that the queues are completely drained each scheduling period.

For an all-to-all workload with N pods, an effective link rate of $L_{\text{effective}}$ bits per second, and uniform time slots of duration t_{slot} seconds, the amount of buffering required by each host in bits is given by

$$B = L_{\text{effective}}(N-1)t_{\text{slot}} \quad (2)$$

From (2) we can immediately see why the t_{setup} of 27 ms reported by Helios [6, 7] makes TMS impractical for slow millisecond-scale circuit switches. For 24 pods with 10 Gigabit Ethernet, choosing $T = 270$ ms to achieve a 90% duty cycle yields $B = 7.23$ GB per host. Currently this is an impractical amount of DRAM to dedicate to packet buffering. Since $L_{\text{effective}}$ and N are both likely to increase for future data centers, the only way to make TMS practical is to decrease T by using microsecond-scale switching. Setting T = 100 μs yields B = 2.74 MB of buffering per host, which is much more practical. Therefore we argue that TMS requires microsecond-scale circuit switching.

3. TRAFFIC MATRIX SCHEDULING ALGORITHM

In this section, we describe the TMS algorithm for arbitrary traffic demand matrices. Fig. 4 shows an example.

The TMS algorithm is divided into two phases. In phase 1, the traffic demand matrix (TDM) is scaled into a bandwidth allocation matrix (BAM). A TDM represents the amount of traffic, in terms of percent of circuit switch link rate, that the hosts in a source pod wish to transmit to the hosts in a destination pod. A BAM represents the percentage of bandwidth in a circuit switch and how it is allocated between input-output port pairs over time. In a sense, the BAM is typically "larger" than the TDM since it is not often that the hosts completely drive the network at full utilization. This notion of larger is captured mathematically by the theory of stochastic matrices, even though there is nothing random about our TDM and BAM. If no pod wishes to send more than its link rate (its row sum is less than or equal to 1) and no pod wishes to receive more than its link rate (its column sum is less than or equal to 1), then we say that the TDM is both admissible and doubly substochastic. The BAM is called doubly stochastic because it has the further constraint that its row sums and column sums are exactly equal to 1. By scaling the TDM into a BAM, we are in some sense preserving the demands of the senders and receivers, while also satisfying the constraints of the circuit switch. A matrix scaling algorithm such as Sinkhorn (1964) [13] can be used for this purpose, even when the TDM is not admissible.

In phase 2, the BAM is decomposed into a circuit switch

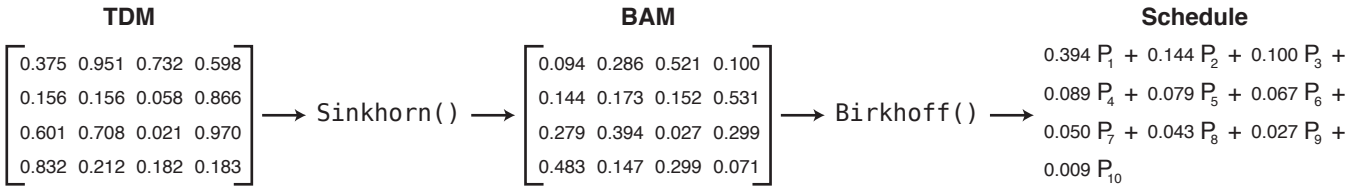

Figure 4: Example run of the traffic matrix scheduling algorithm on a random TDM. In phase 1, the Sinkhorn algorithm scales an inadmissible traffic demand matrix (TDM) into a bandwidth allocation matrix (BAM). In phase 2, the Birkhoff-von Neumann algorithm decomposes the BAM into a circuit switch schedule: a convex combination of permutation matrices (P_i) that sum to the BAM (see equation 3).

schedule, which is a convex combination of permutation matrices that sum to the original BAM

$$\text{BAM} = \sum_{i}^{k} c_i P_i \tag{3}$$

where $0 \leq i \leq k$, and $k = N^2 - 2N + 2$. Each permutation matrix, P_i, represents a circuit switch assignment, and each scalar coefficient, c_i, represents a time slot duration, as a percentage of the total schedule duration. One can use a matrix decomposition algorithm such as Birkhoff-von Neumann (1946) [2, 15], also known as BvN, to compute the schedule. Phase 1 is necessary because BvN requires a doubly stochastic matrix as input. In fact, the Birkhoff-von Neumann theorem states that every doubly stochastic matrix has such a decomposition.

3.1 Execution Time

We measured the execution time of the TMS algorithm on a 2.66 GHz Intel Core 2 processor. The TMS algorithm was tested with dense uniform random input matrices. In the future we hope to evaluate with actual data center network traces. Sinkhorn 1964 has a time complexity of $O(N^2)$. Fig. 5 shows the measured execution time with input matrices up to size 1024×1024. BvN has a time complexity of $O(N^{4.5})$. Fig. 6 shows the measured execution time with small input matrices.

Our algorithms were implemented in Python for ease of evaluation. Clearly there is an opportunity to improve the runtime performance of these algorithms. Performance could be expected to improve if implemented on a faster platform such as (a) a multicore x86_64 platform with C code, (b) a GPU accelerator platform such as NVIDIA CUDA, OpenCL, or Microsoft's DirectCompute, (c) a multicore integer platform such as Broadcom's XLP, or (d) an FPGA. Second, there may be faster matrix decomposition algorithms, or faster implementations of BvN. Third, we don't yet know the size of the input matrices required in real data center networks; N could be 16, 64, or 1024. Finally, we evaluated using dense uniform random matrices rather than sparse matrices. It is likely that sparse matrices would allow opportunities for performance speedup. For example, the Hopcroft-Karp algorithm [9] used as a building block of BvN performs better on sparse matrices than on dense matrices. However, given that we do not yet have good data center network traffic models, we choose to limit our evaluation to uniform random traffic.

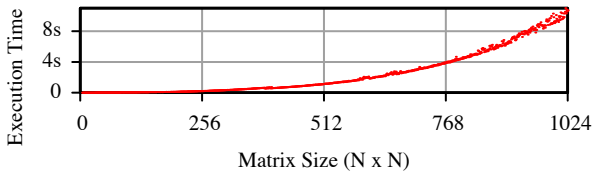

Figure 5: Execution time of the Sinkhorn matrix scaling algorithm.

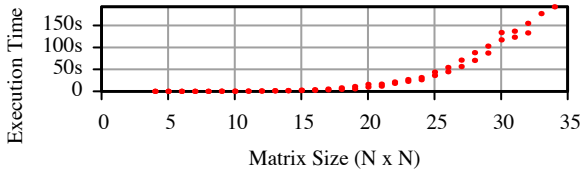

Figure 6: Execution time of the Birkhoff-von Neumann matrix decomposition algorithm.

3.2 Longest Time-Slot First Scheduling

Sometimes it may be better not to schedule all traffic over the circuit switch and to simply schedule only the longest time slots. The reason is that the BvN decomposition algorithm will generate time slots of different lengths, some of which can be quite short. Consider the schedule in Fig. 4. The shortest time slot is only 0.9% of the entire schedule. With such a short time slot, it is likely that t_{setup} will be so large as a percentage that it would have been better to route that traffic over the packet-switched network.

The greatest benefit comes from scheduling the first n timeslots, where n is chosen based on both the minimum required duty cycle, D, as well as the maximum allowed schedule length, T_{schedule}. We extend our definition of D to

variable-length time slots as follows

$$T_{\text{setup}} = nt_{\text{setup}} \tag{4}$$

$$D = \frac{T_{\text{stable}}}{T_{\text{setup}} + T_{\text{stable}}} = \frac{T_{\text{stable}}}{T_{\text{schedule}}} \tag{5}$$

where $n \leq k$ is the number of time slots in the schedule, and T_{schedule} is the duration of the entire schedule period. This definition allows us to choose to schedule only the first n time slots. Traffic that is not scheduled over the circuit-switched network must transit the packet-switched network. Using the example in Fig. 4, Table 1 shows the trade offs in choosing the right number of time slots for the schedule.

Table 1: Example of trade offs between the number of schedule time slots (n), the amount of traffic sent over the circuit-switched network (CSN) vs the packet-switched network (PSN), and the duty cycle (D). $\mathbf{t_{setup} = 10}$ μs, $\mathbf{T_{schedule} = 1}$ ms.

n	CSN	PSN	D
0	0%	100.0%	N/A
1	39.4%	60.6%	100.0%
2	53.8%	46.2%	98.0%
3	63.8%	36.2%	97.0%
4	72.7%	27.3%	96.0%
5	80.6%	19.4%	95.0%
6	87.3%	12.7%	94.0%
7	92.3%	7.7%	93.0%
8	96.6%	3.4%	92.0%
9	99.3%	0.7%	91.0%
10	100.0%	0%	90.0%

As the n increases, an increasing fraction of the total traffic is routed over the circuit-switched network. In the limit when $n = k$, all traffic is routed over the circuit-switched network. However, the duty cycle decreases with increasing n. This is because T_{schedule} is held constant, so T_{stable} must decrease as T_{setup} increases. For example, if the minimum required duty cycle was 95%, then by setting $n = 5$, 80.6% of the total traffic would be routed over circuit switches. Alternatively, at the cost of increased host buffering, we could increase T_{schedule} in order to increase n to 6 or 7 while keeping the duty cycle at 95%.

4. IMPLEMENTATION

To evaluate our approach, we have designed and built a microsecond-scale circuit switch called Mordia (Microsecond Optical Research Datacenter Interconnect Architecture). Mordia represents a single (but well-researched) point in the design space, while TMS represents an approach to circuit scheduling for hybrid data center networks that is especially well suited to microsecond-scale circuit switches. Mordia also acts as a testbed to evaluate concepts such as TMS with real hardware and software.

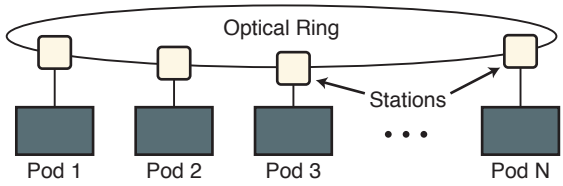

Figure 7: The Mordia prototype is an optical ring of wavelength-selective switches called stations, approximately 2300x faster than the optical space switches used in Helios and c-Through (11.5 μs vs 27 ms).

Fig. 7 shows the high-level topology of Mordia: an optical ring of stations, with each station terminating a pod. Stations use wavelength-selective switching to route traffic to other stations, and hence to other pods. The Mordia prototype is a 24 $\times$ 24-port optical circuit switch (OCS) that is three orders of magnitude faster than current commercial OCS, with a nominal switching time of 2.8 μs, an effective switching time of 11.5 μs when accounting for end-to-end hardware and software initialization delays, supporting a minimum time slot duration of approximately 80 μs and minimum duty cycle of 87.4%. The Mordia architecture can scale to 704 ports with current commodity optical components. Further details on the prototype can be found in related publications and the project website[2].

The Mordia prototype allows us to uncover many different aspects of TMS that were not salient in the equations and algorithms. For example, TMS assumes that all pods are synchronized to the circuit switch, such that when the circuit switch undergoes reconfiguration to implement the next circuit assignment in the schedule, all pods will immediately stop transmitting, wait, and then begin transmitting from the next VOQ. This is nontrivial at microsecond time scales. However, our working prototype shows that it is indeed possible to support microsecond-scale circuit switching over commodity Ethernet technology.

5. RELATED WORK

Recent work by Wang et al. [16, 17] on c-Through has proposed to treat the data center network as a single virtual output queued switch by buffering traffic on hosts and then transmitting via a bufferless crossbar circuit switch. c-Through uses these host buffers to estimate the inter-rack traffic demand matrix, and then relies on a max-weighted assignment on the circuit switch to maximize network throughput. Although operating at a pod-level rather than host/rack-level, Helios [7] also performs demand estimation, and then uses max-weighted assignment to choose circuits. Both of these fully functional hybrid data center network prototypes use HSS. In both systems, the traffic demand matrix is estimated, then communicated to a central point where the max-

[2] `http://mordia.net`

weighted assignment algorithm is executed, at which point the schedule is distributed back to the circuit switches and top-of-rack/pod switches for execution, all of which take a significant amount of time and reside on the critical path for circuit reconfiguration.

TMS leverages several observations made in Flyways [8, 10], showing that for many realistic deployments, such as MapReduce, there is some degree of stability in the rate of change of the traffic demand matrix. In addition, the traffic demand matrix is often sparse, which is good for the algorithms presented in this paper. Finally, the notion of focusing on the traffic demand matrix as an input to a centralized scheduler was inspired by the heat maps in Flyways.

Bazzaz et al. [1] enumerate limitations of circuit switching, such as the need to support correlated flows within an application in the context of realistic data center workloads. They suggest augmenting the central controller with additional application semantics, and using OpenFlow [11] to provide a richer control plane.

HSS uses "one loop" to both compute and execute the schedule, whereas TMS uses "two loops", one for computation and one for execution. Vattikonda et al. [14] pursue a similar two-level approach, with a slower-running central scheduler relying on very fast running TDMA in switches, however in the context of wireline data center networks.

We are not the first to suggest using the Birkhoff-von Neumann decomposition algorithm to compute schedules for input queued switches [3, 4]. Previous work focused on crossbar scheduling for packet switches, whereas this paper focuses on circuit switch scheduling for data center networks with packet buffers distributed among the hosts.

6. CONCLUSION

In this paper, we described traffic matrix scheduling, a technique for scheduling circuit switches to route potentially all bulk data center traffic, not just traffic from hotspots. This makes circuit switches much more useful than previously thought. In fact, the advantages of circuit switching compared to packet switching, namely (i) CAPEX and OPEX reduction, (ii) low latency, (iii) no jitter, and (iv) the ability to scale the nominal link rate to hundreds of Gb/s per link, make circuit switching a viable contender for future data center network architectures.

Acknowledgements

We would like to thank Ronald Graham, Bill Lin, our shepherd Atul Adya, and our reviewers, for their valuable feedback. We received support from the National Science Foundation through CIAN NSF ERC under grant #EEC-0812072.

7. REFERENCES

[1] H. H. Bazzaz, M. Tewari, G. Wang, G. Porter, T. S. E. Ng, D. G. Andersen, M. Kaminsky, M. A. Kozuch, and A. Vahdat. Switching the Optical Divide: Fundamental Challenges for Hybrid Electrical/Optical Datacenter Networks. In *SoCC '12*.

[2] G. Birkhoff. Tres Observaciones Sobre el Algebra Lineal. *Univ. Nac. Tucumán Rev. Ser. A*, 5:147–151, 1946.

[3] C. Chang, W. Chen, and H. Huang. On Service Guarantees for Input-buffered Crossbar Switches: A Capacity Decomposition Approach by Birkhoff and von Neumann. In *Intl. Workshop on Quality of Service (IWQoS '99)*.

[4] C. Chang, D. Lee, and Y. Jou. Load Balanced Birkhoff–von Neumann Switches, Part I: One-stage Buffering. *Computer Communications*, 25(6):611–622, 2002.

[5] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, and X. Wen. OSA: An Optical Switching Architecture for Data Center Networks with Unprecedented Flexibility. In *NSDI '12*.

[6] N. Farrington, Y. Fainman, H. Liu, G. Papen, and A. Vahdat. Hardware Requirements for Optical Circuit Switched Data Center Networks. In *OFC/NFOEC*, Los Angeles, Mar. 2011.

[7] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM '10*.

[8] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting Data Center Networks with Multi-Gigabit Wireless Links. In *SIGCOMM '11*.

[9] J. Hopcroft and R. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[10] S. Kandula, J. Padhye, and P. Bahl. Flyways to Decongest Data Center Networks. *HotNets '09*.

[11] 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.*, 38(2):69–74, Mar. 2008.

[12] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% Throughput in an Input-queued Switch. *Communications, IEEE Transactions on*, 47(8):1260–1267, 1999.

[13] R. Sinkhorn. A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[14] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical TDMA for Datacenter Ethernet. *ACM EuroSys '12*.

[15] J. von Neumann. A Certain Zero-sum Two-person Game Equivalent to the Optimal Assignment Problem. *Contributions to the Theory of Games*, 2:5–12, 1953.

[16] G. Wang, D. Andersen, M. Kaminsky, M. Kozuch, T. Ng, K. Papagiannaki, M. Glick, and L. Mummert. Your Data Center is a Router: The Case for Reconfigurable Optical Circuit Switched Paths. In *HotNets '09*.

[17] 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 *SIGCOMM '10*.

[18] 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 *SIGCOMM '12*.