Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch

Li Chen and Kai Chen, *The Hong Kong University of Science and Technology;* Zhonghua Zhu, *Omnisensing Photonics;* Minlan Yu, *Yale University;* George Porter, *University of California, San Diego;* Chunming Qiao, *University at Buffalo;* Shan Zhong, *CoAdna*

https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/chen

This paper is included in the Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17).

March 27–29, 2017 • Boston, MA, USA

ISBN 978-1-931971-37-9

Enabling Wide-spread Communications on Optical Fabric with MegaSwitch

Li Chen[1] Kai Chen[1] Zhonghua Zhu[2] Minlan Yu[3]
George Porter[4] Chunming Qiao[5] Shan Zhong[6]
[1]*SING Lab@HKUST* [2]*Omnisensing Photonics* [3]*Yale* [4]*UCSD* [5]*SUNY Buffalo* [6]*CoAdna*

Abstract

Existing wired optical interconnects face a challenge of supporting wide-spread communications in production clusters. Initial proposals are constrained, as they only support connections among a small number of racks (e.g., 2 or 4) at a time, with switching time of milliseconds. Recent efforts on reducing optical circuit reconfiguration time to microseconds partially mitigate this problem by rapidly time-sharing optical circuits across more nodes, but are still limited by the total number of parallel circuits available simultaneously.

In this paper, we seek an optical interconnect that can enable unconstrained communications within a computing cluster of thousands of servers. We present MegaSwitch, a multi-fiber ring optical fabric that exploits space division multiplexing across multiple fibers to deliver rearrangeably non-blocking communications to 30+ racks and 6000+ servers. We have implemented a 5-rack 40-server MegaSwitch prototype with commercial optical devices, and used testbed experiments as well as large-scale simulations to explore MegaSwitch's architectural benefits and tradeoffs.

1 Introduction

Due to its advantages over traditional electrical networks at high link speeds, optical circuit switching technology has been widely investigated for datacenter networks (DCN) to reduce cost, power consumption, and wiring complexity [38]. Many of the initial optical DCN proposals build on the assumption of substantial traffic "concentration". Using optical circuit switches that only support one-to-one connections between ports, they service highly concentrated traffic among a small number of racks at a time (e.g., 2 or 4 [6, 50]). However, evolving traffic characteristics in production DCNs pose new challenges:

- **Wide-spread communications:** Recent analysis of Facebook cluster traffic reveals that servers can concurrently communicate with hundreds of hosts, with the majority of traffic to tens of racks [42]; and traces from Google [46] and Microsoft [15, 17] also exhibit such high fan-in/out wide-spread communications. The driving forces behind such wide-spread patterns are multifold, such as data shuffling, load balancing, data spreading for fault-tolerance, etc. [42, 46].

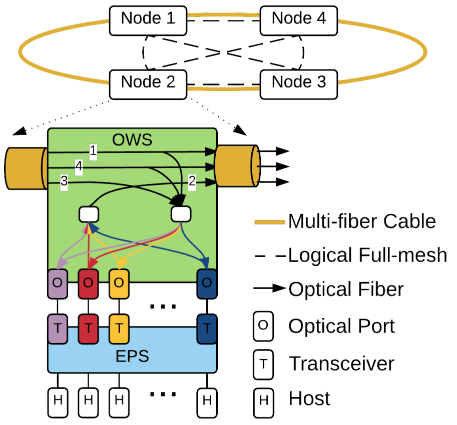


Figure 1: High-level view of a 4-node MegaSwitch.

- **High bandwidth demands:** Due to the ever-larger parallel data processing, remote storage access, web services, etc., traffic in Google's DCN has increased by 50x in the last 6 years, doubling every year [46].

Under such conditions, early optical DCN solutions fall short due to their constrained communication support [6, 7, 12, 50]. Recent efforts [24, 38] have proposed a *temporal* approach—by reducing the circuit switching time from milliseconds to microseconds, they are able to rapidly time-share optical circuits across more nodes in a shorter time. While such temporal approach mitigates the problem, it is still insufficient for the wide-spread communications, as it is limited by the total number of parallel circuits available at the same time [38].[1]

We take a *spatial* approach to address the challenge. To support the wide-spread communications, we seek a solution that can directly deliver parallel circuits to many nodes simultaneously. In this paper, we present an optical DCN interconnect for thousands of servers, called MegaSwitch, that enables unconstrained communications among all the servers.

At a high-level (Figure 1), MegaSwitch is a circuit-switched backplane physically composed of multiple optical wavelength switches (OWS, a switch we implemented §4) connected in a multi-fiber ring. Each OWS is attached to an electrical packet switch (EPS), forming a MegaSwitch node. OWS acts as an access point for EPS to the ring. Each sender uses a dedicated fiber to

[1]Another temporal approach [15] employs wireless free space optics (FSO) technology, However, FSO is not yet production-ready for DCNs, because dust and vibration that are common in DCNs can impair FSO link stability [15].

send traffic to any other nodes; on this multi-fiber ring, each receiver can receive traffic from all the senders (one per fiber) simultaneously. Essentially, MegaSwitch establishes a one-hop circuit between any pair of EPSes, forming a rearrangeably non-blocking circuit mesh over the physical multi-fiber ring.

Specifically, MegaSwitch achieves unconstrained connections by re-purposing wavelength selective switch (WSS, a key component of OWS) as a receiving $w\times 1$ multiplexer to enable non-blocking space division multiplexing among w fibers (§3.1). Prior work [6, 11, 38] used WSS as $1\times w$ demultiplexer that takes 1 input fiber with k wavelengths, and outputs any subset of the k wavelengths to w output fibers. In contrast, MegaSwitch reverses the use of WSS. Each node uses k wavelengths[2] on a fiber for sending; for receiving, each node leverages WSS to intercept all $k\times w$ wavelengths from w other senders (one per fiber) via its w input ports. Then, with a wavelength assignment algorithm (§3.2), the WSS can always select, out of the $k\times w$ wavelengths, a non-interfering set to satisfy any communication demands among nodes.

As a result, MegaSwitch delivers rearrangeably non-blocking communications to $n\times k$ ports, where $n=w+1$ is the number of nodes/racks on the ring, and k is the number of ports per node. With current technology, w can be up to 32 and k up to 192, thus MegaSwitch can support up to 33 racks and 6336 hosts. In the future, MegaSwitch can scale beyond 10^5 ports with AWGR (array waveguide grating router) technology and multi-dimensional expansion (§3.1).

On top of its unconstrained connections, MegaSwitch further has built-in fault-tolerance to handle various failures such as OWS, cable, and link failures. We develop necessary redundancy and mechanisms, so that MegaSwitch can provide reliable services (§3.3).

We have implemented a small-scale MegaSwitch prototype (§4) with 5 nodes, and each node has 8 optical ports transmitting on 8 unique wavelengths within 190.5THz to 193.5THz at 200GHz channel spacing. This prototype enables arbitrary communications among 5 racks and 40 hosts. Furthermore, our OWSes are designed and implemented with all commercially available optical devices, and the EPSes we used are Broadcom Pronto-3922 10G commodity switches.

MegaSwitch's reconfiguration delay hinges on WSS, and 11.5μs WSS switching time has been reported using digital light processing (DLP) technology [38]. However, our implementation experience reveals that, as WSS port count increases (for wide-spread communications), such low latency can no longer be maintained. This is mainly because the port count of WSS with DLP used in [38] is not scalable. In our prototype, the WSS reconfiguration is $\sim 3ms$. We believe this is a hard limitation we have to confront in order to scale. To accommodate unstable traffic and latency-critical applications, we develop "basemesh" on MegaSwitch to ensure any two nodes are always connected via certain wavelengths during reconfigurations (§3.2.2). Especially, we construct the basemesh with the well-known Symphony [29] topology in distributed hash table (DHT) literature to provide low average latency with adjustable capacity.

Over the prototype testbed, we conducted basic measurements of throughput and latency, and deployed real applications, e.g., Spark [54] and Redis [43], to evaluate the performance of MegaSwitch (§5.1). Our experiments show that the latency-sensitive Redis experiences uniformly low latency for cross-rack queries due to the basemesh, and the performance of Spark applications is similar to that of an optimal scenario where all servers are connected to one single switch.

To complement testbed experiments, we performed large-scale simulations to study the impact of traffic stability on MegaSwitch (§5.2). For synthetic traces, MegaSwitch provides near full bisection bandwidth for stable traffic of all patterns, but does not sustain high throughput for concentrated, unstable traffic with stability period less than reconfiguration delay. However, it can improve throughput for unstable wide-spread traffic through the basemesh. For real production traces, MegaSwitch achieves 93.21% throughput of an ideal non-blocking fabric. We also find that our basemesh effectively handles highly unstable wide-spread traffic, and contributes up to 56.14% throughput improvement.

2 Background and Motivation

In this section, we first highlight the trend of high-demand wide-spread traffic in DCNs. Then we discuss why prior solutions are insufficient to support this trend.

2.1 The Trend of DCN Traffic

We use two case studies to show the traffic trend.

- **User-facing web applications:** In response to user requests, web servers push/pull contents to/from cache servers (e.g. Redis [43]). Web servers and cache servers are usually deployed in different racks, leading to intensive inter-rack traffic [42]. Cache objects are replicated in different clusters for fault-tolerance and load balancing, and web servers access these objects randomly, creating a wide-spread communication pattern overall.
- **Data-parallel processing:** Traffic of MapReduce-type applications [3, 21, 42] is shown to be heavily cluster-local (e.g. [42] reported 13.3% of the traffic is rack-local, but 80.9% cluster-local). Data is spread to many racks due to rack awareness [19], which requires

[2]Each unique wavelength corresponds to an optical port on OWS, as well as an optical transceiver on EPS.

copies of data to be stored in different racks in case of rack failures, as well as cluster-level balancing [45]. In shuffle and output phases of MapReduce jobs, wide-spread many-to-many traffic emerges among all participating servers/racks within the cluster.

Bandwidth demand is also surging as the data stored and processed in DCNs continue to grow, due to the explosion of user-generated contents such as photo/video, updates, and sensor measurements [46]. The implication is twofold: 1) Data processing applications require larger bandwidth for data exchange both within a application (e.g., in a multi-stage machine learning, workers exchange data between stages) and between applications (e.g., related applications like web search and ad recommendation share data); 2) Front-end servers need more bandwidth to fetch the ever-larger contents from cache servers to generate webpages [5]. In conclusion, optical DCN interconnects must support wide-spread, high-bandwidth demands among many racks.

2.2 Prior Proposals Are Insufficient

Prior proposals fall short in supporting the above trend of high-bandwidth, wide-spread communications among many nodes simultaneously:

- c-Through [50] and Helios [12] are seminal works that introduce wavelength division multiplexing (WDM) and optical circuit switch (OCS) into DCNs. However, they suffer from constrained rack-to-rack connectivity. At any time, high capacity optical circuits are limited to a couple of racks (e.g., 2 [50]). Reconfiguring circuits to serve communications among more racks can incur ∼10 ms circuit visit delay [12, 50].
- OSA [6] and WaveCube [7] enable arbitrary rack-to-rack connectivity via multi-hop routing over multiple circuits and EPSes. While solving the connectivity issue, they introduce severe bandwidth constraints between racks, because traffic needs to traverse multiple EPSes to destination, introducing traffic overhead and routing complexity at each transit EPS.
- Quartz [26] is an optical fiber ring that is designed as an design element to provide low latency in portions of traditional DCNs. It avoids the circuit switching delay by using fixed wavelengths, and its bisection bandwidth are also limited.
- Mordia [38] and REACToR [24] improve optical circuit switching in the temporal direction, by reducing circuit switching time from milliseconds to microseconds. This approach is effective in quickly time-sharing circuits across more nodes, but still insufficient to support wide-spread communications due to the parallel circuits available simultaneously. Furthermore, Mordia is by default a single fiber ring structure with small port density (limited by # of unique wavelengths supported on a fiber). Naïvely stacking

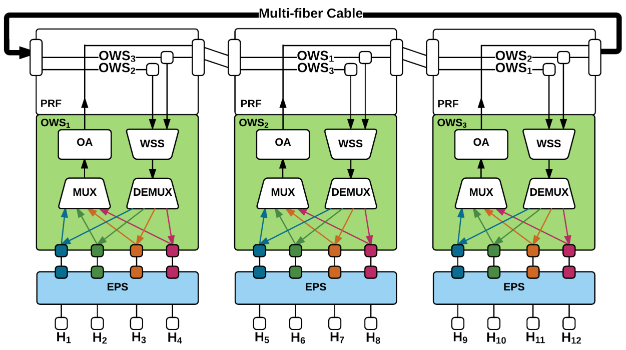


Figure 2: Example of a 3-node MegaSwitch.

multiple fibers via ring-selection circuits as suggested by [38] is blocking among hosts on different fibers, leading to very degraded bisection bandwidth (§5.2); whereas time-sharing multiple fibers via EPS as suggested by [11] requires complex EPS functions unavailable in existing commodity EPSes and introduces additional optical design complexities, making it hard to implement in practice (more details in §3.1).

- Free-space optics proposals [15, 18] eliminate wiring. Firefly [18] leverages free-space optics to build a wireless DCN fabric with improved cost-performance tradeoff. With low switching time and high scalability, ProjecToR [15] connects an entire DCN of tens of thousands of machines. However, they face practical challenges of real DCN environments (e.g., dust and vibration). In contrast, we implement a wired optical interconnect for thousands of machines, which is common in DCNs, and deploy real applications on it. We note that the mechanisms we developed for MegaSwitch can be employed in ProjecToR, notably the basemesh for low latency applications.

3 MegaSwitch Design

In this section, we describe the design of MegaSwitch's data and control planes, as well as its fault-tolerance.

3.1 Data Plane

As in Figure 2, MegaSwitch connects multiple OWSes in a multi-fiber ring. Each fiber has only one sender. The sender broadcasts traffic on this fiber, which reaches all other nodes in one hop. Each receiver can receive traffic from all the senders simultaneously. The key of MegaSwitch is that it exploits WSS to enable non-blocking space division multiplexing among these parallel fibers.

Sending component: In OWS, for transmission, we use an optical multiplexer (MUX) to join the signals from EPS onto a fiber, and then use an optical amplifier (OA) to boost the power. Multiple wavelengths from EPS uplink ports (each with an optical transceiver at a unique wavelength) are multiplexed onto a single fiber. The multiplexed signals then travel to all the other nodes via this fiber. The OA is added before the path to compensate the optical insertion loss caused by broadcasting, which ensures the signal strength is within the receiver sensitivity range (Detailed Power budgeting design is in §4.1). As shown in Figure 2, any signal is amplified once at its sender, and there is no additional amplification stages in the intermediate or receiver nodes. The signal from sender also does not loop back, and terminates at the last receiver on the ring (e.g., signals from OWS_1 terminates in OWS_3, so that no physical loop is formed.).

Receiving component: We use a WSS and an optical demultiplexer (DEMUX) at the receiving end. The design highlight is using WSS as a $w \times 1$ wavelength multiplexer to intercept all the wavelengths from all the other nodes on the ring. In prior work [6, 11, 38], WSS was used as a $1 \times w$ demultiplexer that takes 1 input fiber of k wavelengths, and outputs any subset of these k wavelengths to any of w output fibers. In MegaSwitch, WSS is repurposed as a $w \times 1$ multiplexer, which takes w input fibers with k wavelengths each, and outputs a non-interfering subset of the $k \times w$ wavelengths to an output fiber. With this unconventional use of WSS, MegaSwitch ensures that any node can simultaneously choose any wavelengths from any other nodes, enabling unconstrained connections. Then, based on the demands among nodes, the WSS selects the right wavelengths and multiplexes them on the fiber to the DEMUX. In §3.2, we introduce algorithms to handle wavelength assignments. The multiplexed signals are then de-multiplexed by DEMUX to the uplink ports on EPS.

Supporting $*$-cast [51]: Unicast and multicast can be set up with a single wavelength on MegaSwitch, because any wavelength from a source can be intercepted by every node on the ring. To set up a unicast, MegaSwitch assigns a wavelength on the fiber from the source to destination, and configures the WSS at the destination to select this wavelength. Consider a unicast from H_1 to H_6 in Figure 2. We first assign wavelength λ_1 from node 1 to 2 for this connection, and configure the WSS in node 2 to select λ_1 from node 1. Then, with the routing in both EPSes configured, the unicast circuit from H_1 to H_6 is established. Further, to setup a multicast from H_1 to H_6 and H_9, based on the unicast above, we just need to additionally configure the WSS in node 3 to also select λ_1 from node 1. In addition, many-to-one (incast) or many-to-many (allcast) communications are composites of many unicasts and/or multicasts, and they are supported using multiple wavelengths.

Scalability: MegaSwitch's port count is $n \times k$.

- For n, with existing technology, it is possible to achieve 32×1 WSS (thus $n = w + 1 = 33$). Furthermore, alternative optical components, such as AWGR and filter arrays can be used to achieve the same functionality as WSS+DEMUX for MegaSwitch [56]. Since 48×48 or 64×64 AWGR is available, we expect to see a $1.5\times$ or $2\times$ increase with 49 or 65 nodes on a ring, while additional splitting loss can be compensated.
- For k, C-band Dense-WDM (DWDM) link can support k=96 wavelengths at 50Ghz channel spacing, and k=192 wavelengths at 25Ghz [35]. Recent progress in optical comb source [30, 53] assures the relative wavelength accuracy among DWDM signals and is promising to enable a low power consumption DWDM transceiver array at 25Ghz channel spacing.

As a result, with existing technology, MegaSwitch supports up to $n \times k = 33 \times 192 = 6336$ ports on a single ring.

MegaSwitch can also be expanded multidimensionally. In our OWS implementation (§4), two inter-OWS ports are used in the ring, and we reserve another two for future extension of MegaSwitch in another dimension. When extended bi-dimensionally (into a torus) [55], MegaSwitch scales as $n^2 \times k$, supporting over 811K ports (n=65, k=192), which should accommodate modern DCNs [42, 46]. However, such large scale comes with inherent routing, management, and reliability challenges, and we leave it as future work.

MegaSwitch vs Mordia: Mordia [38] is perhaps the closest related work to MegaSwitch in terms of topology: both are constructed as a ring, and both adopt WSS. However, the key difference is: Mordia chains multiple WSSes *on* one fiber, each WSS acts as a wavelength demultiplexer to set up circuits between ports on the same fiber; whereas MegaSwitch reverses WSS as a wavelength multiplexer *across* multiple fibers to enable non-blocking connections between ports on different fibers.

We note that Farrington et al. [11] further discussed scaling Mordia with multiple fibers non-blocking-ly (referred to as Mordia-PTL below). Unlike MegaSwitch's WSS-based multi-fiber ring, they proposed to connect each EPS to multiple fiber rings in parallel, and rely on the EPS to schedule circuits for servers to support TDM across fibers. This falls beyond the capability of existing commodity switches [24]. Further, on each fiber, they allow every node to add signals to the fiber and require an optical amplifier in each node to boost the signals and compensate losses (e.g., bandpass add/drop filter loss, variable optical attenuator loss, $10/90$ splitter loss, etc.), thus they need multiple amplifiers per fiber. This degrades optical signal to noise ratio (OSNR) and makes transmission error-prone. In Figure 3, we compare OSNR between MegaSwitch and Mordia. We assume 7dBm per-channel launch power, 16dB pre-amplification gain in MegaSwitch, and 23dB boosting gain per-node in Mordia. The results show that, for MegaSwitch, OSNR maintains at $\sim$36dB and remains constant for all nodes; for Mordia, OSNR quickly degrades to 30dB after 7 hops. For validation, we have also measured the aver-

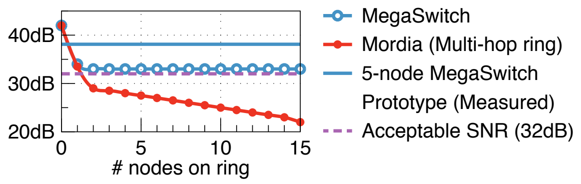


Figure 3: OSNR comparison

age OSNR on our MegaSwitch prototype (§4), which is 38.12dB after 5 hops (plotted in the figure for reference).

Furthermore, there exists a physical loop in each fiber of their design where optical signals can circle back to its origin, causing interferences if not handled properly. MegaSwitch, by design, avoids most of these problems: 1) one each fiber has only one sender and thus one stage of amplification (Power budgeting is explained in §4.1); 2) each fiber is terminated in the last node in the ring, thus there is no physical optical loop or recirculated signals (Figure 9). Therefore, MegaSwitch maintains good OSNR when scaled to more nodes. More importantly, besides above, they do not consider fault-tolerance and actual implementation in their paper [11]. In this work, we have built, with significant efforts, a functional MegaSwitch prototype with commodity off-the-shelf EPSes and our home-built OWSes.

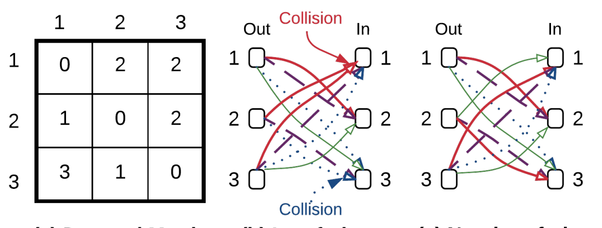
3.2 Control Plane

As prior work [6, 12, 38, 50], MegaSwitch takes a centralized approach to configure routings in EPSes and wavelength assignments in OWSes. The MegaSwitch controller converts traffic bandwidth demands into wavelength assignments and pushes them into the OWSes. The demands can be obtained by existing traffic estimation schemes [2, 12, 50]. In what follows, we first introduce the wavelength assignment algorithms, and then design basemesh to handle unstable traffic and latency sensitive flows during reconfigurations.

3.2.1 Wavelength Assignment

In MegaSwitch, each node uses the same k wavelengths to communicate with other nodes. The control plane assigns the wavelengths to satisfy communication demands among nodes. In a feasible assignment, wavelengths from different senders must not interfere with each other at the receiver, since all the selected wavelengths to a particular receiver share the same output fiber through $w \times 1$ WSS (see Figure 2). We illustrate an assignment example in Figure 4, which has 3 nodes and each has 4 wavelengths. The demand is in (a): each entry is the number of required wavelengths of a sender-receiver pair. If the wavelengths are assigned as in (b), then two conflicts occur. A non-interfering assignment is shown in (c), where no two same wavelengths go to the same receiver.

Given the constraint, we reduce the wavelength assignment problem in MegaSwitch to an edge coloring problem on a bipartite multigraph. We express bandwidth demand matrix on a bipartite multigraph as Figure 4. Multiple edges between two nodes correspond to multiple wavelengths needed by them. Assume each wavelength has a unique color, then a feasible wavelength assignment is equivalent to an assignment of colors to the edges so that no two adjacent edges share the same color—exactly the edge coloring problem [9].



(a) Demand Matrix (b) Interfering (c) Non-interfering

Figure 4: Wavelength assignments (n=3, k=4).

Edge coloring problem is $\mathcal{NP}$-complete on general graphs [6, 9], but has efficient optimal solutions on bipartite multigraphs [7, 44]. By adopting the algorithm in [7], we prove the following theorem[3] (see Appendix), which establishes the rearrangeably non-blocking property of MegaSwitch.

Theorem 1. *Any feasible bandwidth demand can be satisfied by MegaSwitch with at most k unique wavelengths.*

3.2.2 Basemesh

As WSS port count increases, MegaSwitch reconfigures at milliseconds (§4.2). To accommodate unstable traffic and latency-sensitive applications, a typical solution is to maintain a parallel electrical packet-switched fabric, as suggested by prior hybrid network proposals [12, 24, 50].

MegaSwitch emulates such hybrid network with stable circuits, providing constant connectivity among nodes and eliminating the need for an additional electrical fabric. We denote this set of wavelengths as basemesh, which should achieve two goals: 1) the number of wavelengths dedicated to basemesh, b, should be adjustable[4]; 2) With given b, guarantee low average latency for all pairs of nodes.

Essentially, basemesh is an overlay network on MegaSwitch's physical multi-fiber ring, and building such a network with the above design goals has been studied in overlay networking and distributed hash table (DHT) literature [29, 48]. Forwarding a packet on basemesh is similar to performing a look-up on a DHT network. Also, the routing table size (number of known peer addresses) of each node in DHT is analogous to the the number of wavelengths to other nodes, b. Meanwhile, our problem differs from DHT: We assume the centralized controller

[3]We note that this problem can also be reduce to the well-known row-column-row permutation routing problem [4, 39], and the reduction is also a proof of Theorem 1.

[4]The basemesh alone can be considered as a b:k (k is the number of wavelengths per fiber) over-subscribed parallel electrical switching network similar to that of [12, 24, 50].

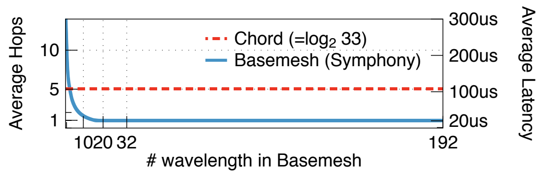


Figure 5: Average latency on basemesh.

calculates routing tables for EPSes, and the routing tables are updated for adjustments of b and peer churn (infrequent, only happens in failures §3.3).

We find the Symphony [29] topology fits our goals. For b wavelengths in the basemesh, each node first uses one wavelength to connect to the next node, forming a directed ring; then each node chooses shortcuts to $b-1$ other nodes on the ring. We adopt the randomized algorithm of Symphony [29] to choose shortcuts (drawn from a family of harmonic probability distributions, $p_n(x)=\frac{1}{xlnn}$. If a pair is selected, the corresponding entry in demand matrix increment by 1.). The routing table on the EPS of each node is configured with greedy routing that attempts to minimize the absolute distance to destination at each hop [29].

It is shown that the expected number of hops for this topology is $O(\frac{log^2(n)}{k})$ for each packet. With n=33, we plot the expected path length in Figure 5 to compare it with Chord [48] (Finger table size is 4, average path length is $O(log(n))$), and also plot the expected latency assuming $20\mu s$ per-hop latency. Notably, if $b\geq n-1$, basemesh is fully connected with 1 hop between every pair; adding more wavelength cannot reduce the average hop count, but serve to reduce congestion.

As we will show in §5.2, while basemesh reduces the worst-case bisection bandwidth, the average bisection bandwidth is unaffected, and it brings two benefits:

- It increases throughput for rapid-changing, unpredictable traffic when the traffic stability period is less than the control plane delay, as shown in §5.2.
- It mitigates the impact of inaccurate demand estimation by providing consistent connection. Without basemesh, when the demand between two nodes is mistakenly estimated as 0, they are effectively cut-off, which is costly to recover (flows need to wait for wavelength setup).

Thanks to the flexibility of MegaSwitch, b is adjustable to accommodate varying traffic instability (§5.2), or be partially/fully disabled for higher traffic imbalance if applications desire more bandwidth in part of network [42]. In this paper, we present b as a knob for DCN operators, and intend to explore the problem of optimal configuration of b as future work.

Basemesh vs Dedicated Topology in ProjecToR [15]: Basemesh is similar to the dedicated topology in ProjecToR, which is a set of static connections for low latency traffic. However, its construction is based on the

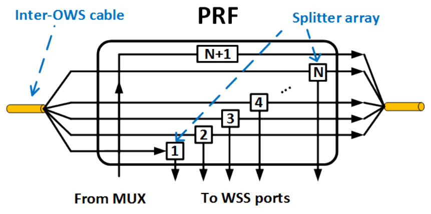


Figure 6: PRF module layout.

optimization of weighted path length using traffic distribution from past measurements, which cannot provide average latency guarantee for an arbitrary pair of racks. In comparison, basemesh can provide consistent connections between all pairs with proven average latency. It is possible to configure basemsesh in ProjecToR as its dedicated topology, which better handles unpredictable, latency-critical traffic.

3.3 Fault-tolerance

We consider the following failures in MegaSwitch.

OWS failure: We implement the OWS box so that the node failure will not affect the operation of other nodes on the ring. As shown in Figure 6, the integrated passive component (Passive Routing Fabric, PRF) handles the signal propagation across adjacent nodes, while the passive splitters copy the traffic to local active switching modules (i.e., WSS and DEMUX). When one OWS loses its function (e.g., power loss) in its active part, the traffic traveling across the failure node from/to other nodes is not affected, because the PRF operates passively and still forwards the signals. Thus the failure is isolated within the local node. We design the PRF as a modular block that can be attached and detached from active switching modules easily. So one can recover the failed OWS without the service interruption of the rest network by simply swapping its active switching module.

Port failure: When the transceiver fails in a node, it causes the loss of capacity only at the local node. We have implemented built-in power detectors in OWS to notify the controller of such events.

Controller failure: In control plane, two or more instances of the controller are running simultaneously, with one leader. The fail-over mechanism follows VRRP [20].

Cable failure: Cable cut is the most difficult failure, and we design[5] redundancy and recovery mechanisms to handle one cable cut. The procedure is analogous to ring protection in SONET [49] by 2 sets of fibers. We propose directional redundancy in MegaSwitch, as shown in Figure 7, the fiber can carry the traffic from/to both east (primary) and west (secondary) sides of the OWS by select-

[5]This design has not been implemented in the current prototype, thus the power budgeting on the prototype (§4.1) does not account for the fault-tolerance components on the ring. The components for this design are all commercially available, and can be readily incorporated into future iterations of MegaSwitch.

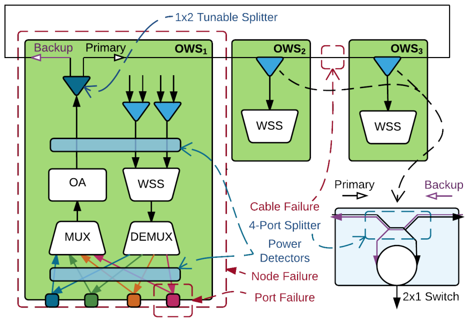


Figure 7: MegaSwitch fault-tolerance.

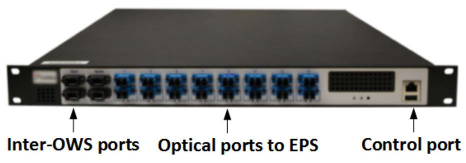


Figure 8: The OWS box we implemented.

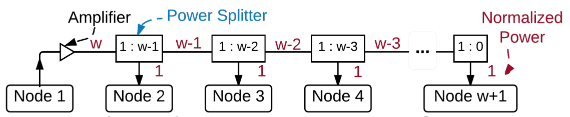


Figure 9: Power budget on a fiber

ing a direction at the sending 1×2 optical tunable splitter[6], and receiving 2×1 optical switches. The splitters and switches are in the PRF module. When there is no cable failure, sending and receiving switches both select the primary direction, thus preventing optical looping. If one cable cut is detected by the power detectors, the controller is notified, and it instructs the OWSes on the ring to do the following: 1) The sending tunable splitter in every OWS transmits on its fiber on both directions[7], so that the signal can reach all the OWSes; 2) The receiving switch for each fiber at the input of WSS selects the direction where there is still incoming signal. Then the connectivity can be restored. A passive 4-port fix splitter is added before the receiving switch as a part of the ring, and it guides the signal from primary and backup directions to different input ports of the 2×1 switch. For more than one cut, the ring is severed into two or more segments, and connectivity between segments cannot be recovered until cable replacement.

4 Implementation

4.1 Implementation of OWS

To implement MegaSwitch and facilitate real deployment, we package all optical devices into an 1RU (Rack Unit) switch box, OWS (Figure 2). The OWS is composed of the sending components (MUX+OA) and the receiving components (WSS+DEMUX), as described in §3. We use a pair of array waveguide grating (AWG) as MUX and DEMUX for DWDM signals.

To simplify the wiring, we compact all the 1×2 drop-continue splitters into PRF module, as shown in Figure 6. This module can be easily attached to or detached from active switching components in the OWS. To compensate component losses along the ring, we use a single stage 12dB OA to boost the DWDM signal before transmitting the signals, and the splitters in PRF have different splitting ratios. Using the same numbering in Figure 6, the splitting ratio of i-th splitter is 1:(i−1) for i>1. As an example, the power splitting on one fiber from node 1 is shown in Figure 9. At each receiving transceiver, the signal strength is ∼−9dBm per channel. The PRF configuration is therefore determined by w (WSS radix).

[6]It is implementable with a common Mach-Zehnder interferometer.
[7]Power splitting ratio must also be re-configured to keep the signal within receiver sensitivity range for all receiving OWSes

OWS receives DWDM signals from all other nodes on the ring via its WSS. We adopt 8×1 WSS from CoAdna Photonics in the prototype, which is able to select wavelength signals from at most 8 nodes. Currently we use 4 out of 8 WSS ports for a 5-node ring.

Our OWS box provides 16 optical ports, and 8 are used in the MegaSwitch prototype. Each port maps to a particular wavelength from 190.5THz to 193.5THz at 200GHz channel spacing. InnoLight 10GBASE-ER DWDM SFP+ transceivers are used to connect EPSes to optical ports on OWSes. With the total broadcasting loss at 10.5dB for OWS, 4dB for WSS, 2.5dB for DEMUX, and 1dB for cable connectors, the receiving power is −9dBm ∼−12dBm, well within the sensitivity range of our transceivers. The OWS box we implemented (Figure 8) has 4 inter-OWS ports, 16 optical ports, and 1 Ethernet port for control. For the 4 inter-OWS ports, two are used to connect other OWSes to construct the MegaSwitch ring, and the other two are reserved for redundancy and further scaling (§3.1).

Each OWS is controlled by a Raspberry Pi [37] with 700MHz ARM CPU and 256MB memory. OWS receives the wavelength assignments (in UDP packets) via its Ethernet management interface connected to a separate, electrical control plane network, and configures WSS via GPIO pins.

4.2 MegaSwitch Prototype

We constructed a 5-node MegaSwitch with 5 OWS boxes, as is shown in Figure 10. The OWSes are simply connected to each other through their inter-OWS ports using ring cables containing multiple fibers. The two EPSes we used are Broadcom Pronto-3922 with 48×10 Gbps Ethernet (GbE) ports. For one EPS, we fit 24 ports with 3 sets of transceivers of 8 unique wavelengths to connect to 3 OWSes, and the rest 24 ports are connected to the servers. For the other, we only use 32 ports with 16 ports to OWSes and 16 to servers. We fill the capacity with 40×10GbE interfaces on 20 Dell PowerEdge R320 servers (Debian 7.0 with Kernel 3.18.19), each with a Broadcom NetXtreme II 10GbE network interface card.

The control plane network connected by a Broadcom Pronto-3295 Ethernet switch. The Ethernet management

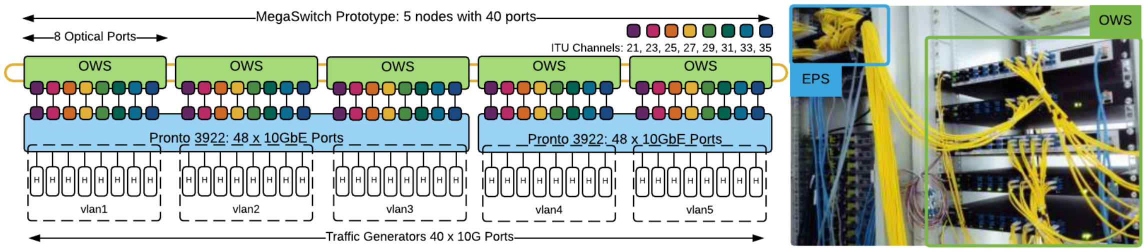

MegaSwitch Prototype: 5 nodes with 40 ports
8 Optical Ports
ITU Channels: 21, 23, 25, 27, 29, 31, 33, 35
OWS
OWS
OWS
OWS
OWS
Pronto 3922: 48 x 10GbE Ports
Pronto 3922: 48 x 10GbE Ports
vlan1
vlan2
vlan3
vlan4
vlan5
Traffic Generators 40 x 10G Ports
EPS
OWS

Figure 10: The MegaSwitch prototype implemented with 5 OWSes.

ports of EPS and OWS are connected to this switch. The MegaSwitch controller is hosted in a server also connected to the control plane switch. The EPSes work in Open vSwitch [36] mode, and are managed by Ryu controller [34] hosted in the same server as the MegaSwitch controller. To emulate 5 OWS-EPS nodes, we divided the 40 server-facing EPS ports into 5 VLANs, virtually representing 5 racks. The OWS-facing ports are given static IP addresses, and we install Openflow [31] rules in the EPS switches to route packets between VLANs in Layer 3. In the experiments, we refer to an OWS and a set of 8 server ports in the same VLAN as a node.

Reconfiguration speed: MegaSwitch's reconfiguration hinges on WSS, and the WSS switching speed is supposedly microseconds with technology in [38]. However, in our implementation, we find that as WSS port count increases (required to support more wide-spread communication), e.g., w=8 in our case, we can no longer maintain 11.5μs switching seen by [38]. This is mainly because the port count of WSS made with digital light processing (DLP) technology used in [38] is currently not scalable due to large insertion loss. As a result, we choose the WSS implemented by an alternative Liquid Crystal (LC) technology, and the observed WSS switching time is $\sim 3ms$. We acknowledge that this is a hard limitation we need to confront in order to scale. We identify that there exist several ways to reduce this time [14, 23].

We note that, with current speed on our prototype, MegaSwitch is still possible to satisfy the need of some production DCNs, as a recent study [42] of DCN traffic in Facebook suggests, with effective load balancing, traffic demands are stable over sub-second intervals. On the other hand, even if μs switching is achieved in later versions of MegaSwitch, it is still insufficient for high-speed DCNs (40/100G or beyond). Following §3.2.2, in our prototype, we address this problem by activating basemesh, which mimics hybrid network without resorting to an additional electrical network.

5 Evaluation

We evaluate MegaSwitch with testbed experiments (with synthetic patterns(§5.1.1)) and real applications(§5.1.2)), as well as large-scale simulations (with synthetic patterns and productions traces(§5.2)). Our main goals are to: 1) measure the basic metrics on MegaSwitch's data plane and control plane; 2) understand the performance of real applications on MegaSwitch; 3) study the impact of control plane latency and traffic stability on throughput, and 4) assess the effectiveness of basemesh.

Summary of results is as follows:

- MegaSwitch supports wide-spread communications with full bisection bandwidth among all ports when wavelength are configured. MegaSwitch sees ~20ms reconfiguration delay with ~3ms for WSS switching.
- We deploy real applications, Spark and Redis, on the prototype. We show that MegaSwitch performs similarly to the optimal scenario (all servers under a single EPS) for data-intensive applications on Spark, and maintains uniform latency for cross-rack queries for Redis due to the basemesh.
- Under synthetic traces, MegaSwitch provides near full bisection bandwidth for stable traffic (stability period $\geq$100ms) of all patterns, but cannot achieve high throughput for concentrated, unstable traffic with stability period less than reconfiguration delay. However, increasing the basemesh capacity effectively improves throughput for highly unstable wide-spread traffic.
- With realistic production traces, MegaSwitch achieves $>$90% throughput of an ideal non-blocking fabric, despite its 20ms total wavelength reconfiguration delay.

5.1 Testbed Experiments

5.1.1 Basic Measurements

Bisection bandwidth: We measure the bisection throughput of MegaSwitch and its degradation during wavelength switching. We use the $40\times$10GbE interfaces on the prototype to form dynamic all-to-all communication patterns (each interface is referred as a host). Since traffic from real applications may be CPU or disk I/O bound, to stress the network, we run the following synthetic traffic used in Helios [12]:

- **Node-level Stride (NStride):** Numbering the nodes (EPS) from 0 to $n-1$. For the i-th node, its j-th host initiates a TCP flow to the j-th host in the $(i+l \bmod n)$-th node, where l rotates from 1 to n every t seconds (t is the traffic stability period). This pattern tests the response to abrupt demand changes between nodes, as the traffic from one node completely shifts to another node in a new period.

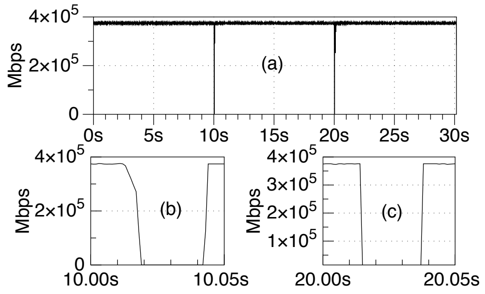


Figure 11: Experiment: Node-level stride

- **Host-level Stride (HStride):** Numbering the hosts from 0 to $n\times k-1$, the i-th host sends a TCP flow to the $(i+k+l \bmod (n\times k))$-th host, where l rotates from 1 to $\lceil k/2 \rceil$ every t seconds. This pattern showcases the gradual demand shift between nodes.
- **Random:** A random perfect matching between all the hosts is generated every period. Every host sends to its matched host for t seconds. The pattern showcases the wide-spread communication, as every node communicates with many nodes in each period.

For this experiment, the basemesh is disabled. We set traffic stability t=10s, and the wavelength assignments are calculated and delivered to the OWS when demand between 2 nodes changes. We study the impact of stability t later in §5.2.

As shown in Figure 11 (a), MegaSwitch maintains full bandwidth of 40×10Gbps when traffic is stable. During reconfigurations, we see the corresponding throughput drops: NStride drops to 0 since all the traffic shifts to a new node; HStride's performance is similar to Figure 11 (a), but drops by only 50Gbps because one wavelength is reconfigured per rack. The throughput resumes quickly after reconfigurations.

We further observe a $\sim 20ms$ gap caused by the wavelength reconfiguration at 10s and 20s in the magnified (b) and (c) of Figure 11. Transceiver initialization delay contributes ~10ms[8], and the remaining can be broken down to EPS configuration ($\sim 5ms$), WSS switching ($\sim 3ms$), and control plane delays ($\sim 4ms$). (Please refer to Appendix for detailed measurement methods and results.)

5.1.2 Real Applications on MegaSwitch

We now evaluate the performance of real applications on MegaSwitch. We use Spark [54], a data-parallel processing application, and Redis [43], a latency-sensitive in-memory key-value store. For this experiment, we form the basemesh with 4 wavelengths, and the others are allocated dynamically.

Spark: We deploy Spark 1.4.1 with Oracle JDK 1.7.0_25 and run three jobs: WikipediaPageRank[9], K-Means[10], and WordCount[11]. We first connect all the 20 servers to a single ToR EPS and run the applications, which establishes the optimal network scenario because all servers are connected with full bandwidth. We record the time series (with millisecond granularity) of bandwidth usage of each server when it is running, and then convert it into two series of averaged bandwidth demand matrices of $0.1s$ and 1s intervals respectively. Then, we connect the servers back to MegaSwitch, and re-run the applications using these two series as inputs to update MegaSwitch every 0.1s and 1s intervals accordingly.

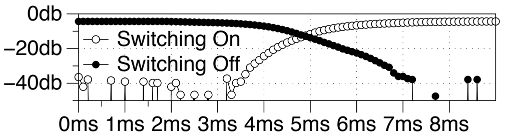


Figure 12: WSS switching speed.

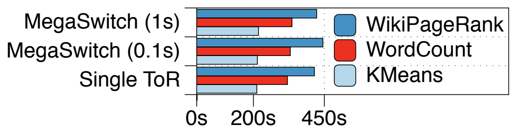


Figure 13: Completion time for Spark jobs

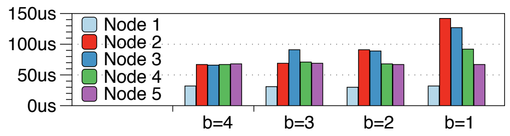


Figure 14: Redis intra/cross-rack query completion

We plot the job completion times in Figure 13. With ~20ms reconfiguration delay and 0.1s interval, the bandwidth efficiency is expected to be $(0.1-0.02)/0.1=80\%$ if every wavelength changes for each interval. However, MegaSwitch performs almost the same as if the servers are connected to a single EPS (on average 2.21% worse). This is because, as we observed, the traffic demands of these Spark jobs are stable: e.g., for K-Means, the number of wavelength reassignments is only 3 and 2 times for the update periods of 0.1 and 1s, respectively. Most wavelengths do not need reconfiguration, and thus provide uninterrupted bandwidth during the experiments. The performance of both update periods is similar, but the smaller update interval has slightly worse performance due to one more wavelength reconfiguration. For example, in WikiPageRank, MegaSwitch with 1s update interval shows 4.94% less completion time than that with 0.1s. We further examine the relationship between traffic stability and MegaSwitch's control latencies in §5.2.

Redis: For this Redis in-memory key-value store experiment, we initiate queries to a Redis server in the first node from the servers in all 5 nodes, with a total number of 10^6 SET and GET requests. Key space is set to 10^5. Since the basemesh provides connectivity between all the nodes, Redis is not affected by any reconfigura-

[8]Our transceiver's receiver Loss of Signal Deassert is -22dBm.

[9]A PageRank instance using a 26GB dataset [32]

[10]A clustering algorithm that partitions a dataset into K clusters. The input is Wikipedia Page Traffic Statistics [47].

[11]It counts words in a 40G dataset with 7.153×10^9 words.

tion latency. The average query completion times from servers in different racks are shown in Figure 14. Query latency depends on hop count. With 4 wavelengths in basemesh (b=4), Redis experiences uniform low latencies for cross-rack queries, because they only traverse 2 EPSes (hops). For b=3, queries also traverse 2 hops, except the ones from node 3. When b=1, basemesh is a ring, and the worst case hop count for a query is 5. Therefore, for latency-critical bandwidth-insensitive applications like Redis, setting b=$n-1$ guarantees uniform latency between all nodes on MegaSwitch. In comparison, for other related architectures, the number of EPS hops for cross-rack queries can reach 3 (Electrical/Optical hybrid designs [12, 24, 50]) or 5 (Pure electrical designs [1, 16, 25]) for cross-rack queries.

5.2 Large Scale Simulations

Existing packet-level simulators, such as ns-2, are time consuming to run at 1000+-host scale [2], and we are more interested in traffic throughput rather than per-packet behavior. Therefore, we implemented a flow-level simulator to perform simulations at larger scales. Flows on the same wavelength share the bandwidth in a max-min fair manner. The simulation runs in discrete time ticks with the granularity of millisecond. We assume a proactive controller: it is informed of the demand change in the next traffic stability period and runs the wavelength assignment algorithm before the change, therefore it configures the wavelengths every t seconds with a total reconfiguration delay of 20ms (§5.1.1). Unless specified otherwise, basemesh is configured with b=32.

We use synthetic patterns in §5.1.1 as well as realistic traces from production DCNs in our simulations. For the synthetic patterns, we simulate a 6336-port MegaSwitch (n=33, k=192), and each pattern runs for 1000 traffic stability periods (t). For the realistic traces, we simulate a 3072-ports MegaSwitch (n=32, k=96) to match the scale of the real cluster. We replay the realistic traces as dynamic job arrivals and departures.

- **Facebook:** Hive/MapReduce trace is collected by Chowdhury et al. [8] from a 3000-server, 150-rack Facebook cluster. For each shuffle, the trace records: start time, senders, receivers, and the received bytes. The trace contains more than 500 shuffles (7×10^5 flows). Shuffle sizes vary from 1MB to 10TB, and the number of flows per shuffle varies from 1 to 2×10^4.
- **IDC:** We collected 2-hour running trace from the Hadoop cluster (3000-server) of a large Internet service company. The trace records shuffle tasks (the same information as above is collected), as well as HDFS read tasks (sender, receiver, and size are collected). We refer to them as **IDC-Shuffle** and **IDC-HDFS** when used separately. The trace contains more than 1000 shuffle and read tasks, respectively (1.6×10^6 flows). The size of shuffle and read varies from 1MB to 1TB, and the number of flows per shuffle is between 1 and 1.7×10^4.

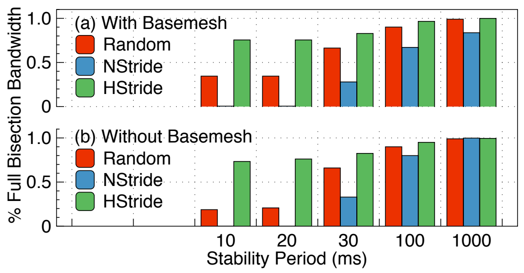


Figure 15: Throughput vs traffic stability

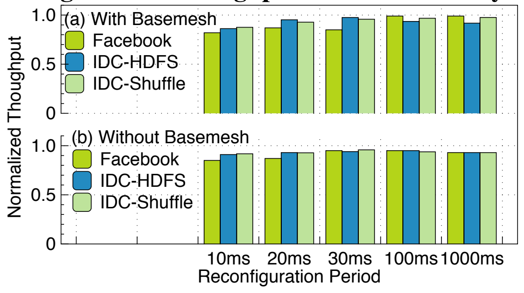


Figure 16: Throughput vs reconfiguration frequency

Impact of traffic stability: We vary the traffic stability period t from 10ms to 1s for the synthetic patterns, and plot the average throughput with and without basemesh in Figure 15. We make three observations: 1) If stability period is $\leq$20ms (reconfiguration delay), traffic patterns that need more reconfigurations have lower throughput, and NStride suffers the worse throughput; 2) All the three patterns see the throughput steadily increasing to full bisection bandwidth with longer stability period; 3) Although basemesh reserves wavelengths to maintain connectivity, throughput of different patterns is not negatively affected, except for NStride, which cannot achieve full bisection bandwidth even for stable traffic (t=1s), since the basemesh takes 32 wavelengths.

We then analyze the performance of different patterns in detail. NStride has near zero throughput when t=10ms and 20ms, as all the wavelengths must break down and reconfigure for each period. Basemesh does not help much for NStride: when the wavelengths are not yet available, basemesh can serve only $1/192$ of the demand. HStride reaches $\sim$75% full bisection bandwidth when the stability period is 10ms, because on average $3/4$ of its demands stay the same between consecutive periods, and demands in the new period reuse some of the previous wavelengths. Random pattern is wide-spread and requires more reconfigurations in each period than HStride. Thus it also suffers from unstable traffic, with 18.5% full-bisection bandwidth for t=10ms without basemesh. However, it benefits from basemesh the most, achieving 34.1% full-bisection bandwidth for t=10ms, because the flows between two nodes need not to wait for reconfigurations.

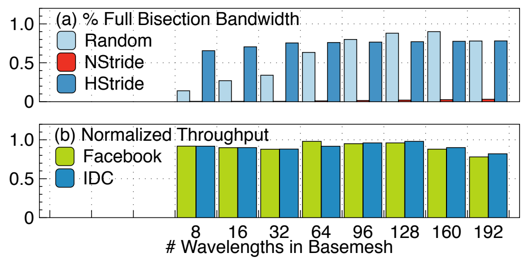


Figure 17: Handling unstable traffic with basemesh

In summary, MegaSwitch provides near full-bisection bandwidth for stable traffic of all patterns, but cannot achieve high throughput for concentrated, unstable traffic (NStride with stability period smaller than reconfiguration delay. We note that such pattern is designed to test the worst-case behavior of MegaSwitch, and is unlikely to occur in production DCNs.).

Impact of reconfiguration frequency: We vary the reconfiguration frequency to study its impact on throughput using realistic traces. In Figure 16, we collect the throughput of the replayed traffic traces[12], and then normalize them to the throughput of the same traces replayed on a non-blocking fabric with the same port count. The normalized throughput shows how MegaSwitch approaches non-blocking.

From the results, we find that MegaSwitch achieves 93.21% and 90.84% normalized throughput on average with and without basemesh respectively. This indicates that our traces collected from the production DCNs are very stable, thus can take advantage of the high bandwidth of optical circuits. This aligns well with traffic statistics in another study [42], which suggests, with good load balancing, the traffic is stable on sub-second scale, thus is suitable for MegaSwitch. We also observe that the traffic is wide-spread for realistic traces: for every period, a rack in Facebook, IDC-HDFS & IDC-Shuffle, talks with 12.1, 4.5 & 14.6 racks on average, respectively. Limited by rack-to-rack connectivity, other optical structures are less effective for such patterns.

Impact of adjusting basemesh capacity: For rapidly changing traffic, MegaSwitch can improve its throughput with more wavelengths to basemesh. In Figure 17, we measure the throughput of synthetic patterns (stability period is 10ms) and production traces. The production traces feature dynamic arrival/departure of tasks.

For NStride and HStride (Figure 17 (a)), since the traffic of each node is destined to one or two other nodes, increasing basemesh does not benefit them much. In contrast, for Random, each node sends to many nodes (i.e., wide-spread), increasing the capacity of basemesh, b, from 8 wavelengths to 32 wavelengths increases the throughput by 20.6%; Further increasing b to 160 can

[12]The wavelength schedules are obtain in the same way as Spark experiments in §5.1.2.

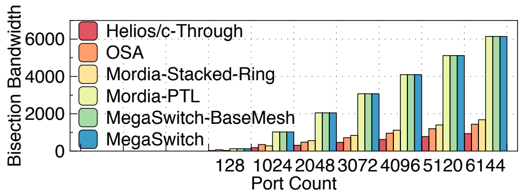


Figure 18: Average bisection throughput.

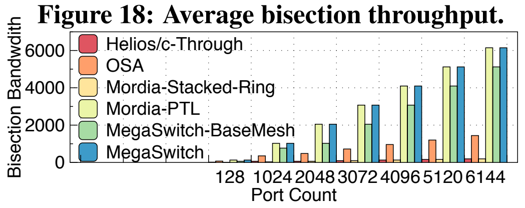


Figure 19: Worst-case bisection throughput.

increase the throughput by 56.1%. But this trend is not monotonic: if all the wavelengths are in basemesh (b=192), and the demands between nodes larger than 6 wavelengths cannot be supported. This is also observed for the realistic traces in Figure 17 (b): when >4 wavelengths are allocated to basemesh, the normalized throughput decreases.

In summary, for rapidly changing traffic, basemesh is an effective and adjustable tool for MegaSwitch to handle wide-spread demand fluctuation. As a comparison, hybrid structures [12, 24, 50] cannot dynamically adjust the capacity of the parallel electrical fabrics.

Bisection bandwidth: To compare MegaSwitch with other optical proposals, we calculate the average and worst-case bisection bandwidths of MegaSwitch and related proposals in Figure 18 & 19, respectively. The unit of y-axis is bandwidth per wavelength. We generate random permutation of port pairing for 10^4 times to compute the average bisection bandwidth for each structure, and design the worst case scenario for them: for Mordia, OSA, and Helios/c-Through, the worst-case scenario is when every node talks to all other nodes[13], because a node can only directly talk to a limited number of other nodes in these structures. We calculate the total throughput of simultaneous connections between ports. We rigorously follow respective papers to scale port counts from 128 to 6144. For example, OSA at 1024 port uses a 16-port OCS for 16 racks each with 64 hosts; Mordia at 6144 ports uses a 32-port OCS to connect 32 rings each with 192 hosts.

We observe: 1) MegaSwitch and Mordia-PTL can achieve full bisection bandwidth at high port count in both average and the worst case, while other designs, including Mordia-Stacked-Ring, cannot. Both structures are expected to maintain full bisection bandwidth with future scaling in optical devices with larger WSS radix and wavelengths per-fiber; 2) Basemesh reduces the worst-case bisection bandwidth by 16.7% for b=32, but full bisection bandwidth is still achieved on average.

[13]A node refers to a ring in Mordia, a rack in OSA/Helios/c-Through

6 Cost of MegaSwitch

Complexity analysis: The absolute costs of optical proposals are difficult to calculate, as it depends on multiple factors, such as market availability, manufacturing costs, etc. For example, our PRF module can be printed as a planar lightwave circuit, and mass-production can push its cost to that of common printed circuit boards [10]. Thus, instead of directly calculating the costs based on price assumptions, we take a comparative approach and analyze the structural complexity of MegaSwitch and closely related proposals.

In Table 1, we compare the complexity of different optical structures at the same port count (3072). For OSA, it translates to 32 racks, 96 DWDM wavelengths, and a 128-port OCS (Optical Circuit Switch). ToR degree is set to 4 [6]. Quartz's port count is limited to the wavelength per fiber (k) [26]: so a Quartz element with k=96 is essentially a 96-port switch, and we scale it to 3072 ports by composing 160 Quartz elements[14] in a FatTree [1] topology (with 32 pods and 32 cores). For Mordia, we use its microsecond 1×4 WSS and stack 32 rings via a 32 port OCS to reach 3072 ports. Mordia-PTL [11] is configured in the same way as Mordia, with the only difference that each EPS is directly connected to 32 rings in parallel without using OCS. Both Mordia and Mordia-PTL have $96/4$=24 stations on each ring. MegaSwitch is configured with 32 nodes and 96 wavelengths per node. Finally, we list a FatTree constructed with 24-port EPS, with totally $24^3/4$=3456 ports and 720 EPSes. We assume that, within the FatTree fabric, all the EPSes are connected via transceivers.

From the table, we find that compared to other optical structures, MegaSwitch supports the same port count with less optical components. Compared to all the optical solutions, FatTree uses $2\times$ optical transceivers (non-DWDM) at the same scale.

Transceiver sost: Our prototype uses commercial 10Gb-ER DWDM SFP+ transceivers, which are $\sim10\times$ more expensive per bit per second than the non-DWDM modules using PSM4. This is because ER optics is usually used for long-range optical networks, rather than short-range networks within a DCN. Our choice of using ER optics is solely due to its ability to use DWDM wavelengths, not its power output or 40KM reach. If MegaSwitch or similar optical interconnects are widely adopted in future DCNs, we expect that DWDM transceivers customized for DCNs will be used instead of current PSM4 modules. Such transceivers are being developed [22], and due to relaxed requirements of short-range DCN, the cost is expectedly lower. Even if the customized DWDM transceivers are still more expensive

Components	OSA	Quartz	Mordia	Mordia-PTL	MegaSwitch	FatTree
Amplifier	0	6144	768	768	32	0
WSS	32 (1×4)	960	768 (1×4)	768 ($1\times$ 4)	32 (32×1)	0
WDM Filter	0	0	768	768	0	0
OCS	1×128-port	0	1×32-port	0	0	0
Circulator	3072	0	0	0	0	0
Transceivers	3072	3072	3072	3072	3072	6912

Table 1: Comparison of optical complexity at the same scale (3072 ports) in optical components used.

than non-DWDM ones, since FatTree needs many more EPSes (and thus transceivers), and the cost difference will grow as the network scales [38].

Amplifier cost: With only one stage of amplification on each fiber, MegaSwitch can maintain good OSNR at large scale (Figure 3). Therefore, MegaSwitch needs much fewer OAs (optical amplifier) at the same scale. The tradeoff is that they must be more powerful, as the total power to reach the same number of ports cannot be reduced significantly. Although unit cost of a powerful OA is higher[15], we expect the total cost to be similar or lower, as MegaSwitch requires much fewer OAs ($24\times$ fewer than Mordia and $192\times$ fewer than Quartz).

WSS cost: In Table 1, MegaSwitch uses 32×1 WSS, and one may wonder its cost versus 1×4 WSS. In fact, our design allows for low cost WSS, as transceiver link budget design does not need to consider optical hopping, thus the key specifications can be relaxed (e.g., bandwidth, insertion loss, and polarization dependent loss requirements). Liquid Crystal (LC) technology is used in our WSS as it is easier to cost-effectively scale to more ports. As the majority of the components in a 32×1 WSS are same as a 1×4 WSS, the per-port cost of 32×1 WSS is about 4 times lower than that of a current 1×4 WSS [6, 38]. In the future, silicon photonics (e.g., matrix switch by ring resonators [13]) can improve the integration level [52] and further reduce the cost.

7 Conclusion

We presented MegaSwitch, an optical interconnect that delivers rearrangeably non-blocking communication to 30+ racks and 6000+ servers. We have implemented a working 5-rack 40-server prototype, and with experiments on this prototype as well as large-scale simulations, we demonstrated the potential of MegaSwitch in supporting wide-spread, high-bandwidth demands workloads among many racks in production DCNs.

Acknowledgements: This work is supported in part by Hong Kong RGC ECS-26200014, GRF-16203715, GRF-613113, CRF-C703615G, China 973 Program No.2014CB340303, NSF CNS-1453662, CNS-1423505, CNS-1314721, and NSFC-61432009. We thank the anonymous NSDI reviewers and our shepherd Ratul Mahajan for their constructive feedback and suggestions.

[14]We assume a Quartz ring of 6 wavelength add/drop multiplexers [26] in each element, thus 6×160=960 in total.

[15]For example, powerful OA can be constructed by a series of less powerful ones.

References

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

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

[3] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 63–74.

[4] Annexstein, F., and Baumslag, M. A unified approach to off-line permutation routing on parallel networks. In *ACM SPAA* (1990).

[5] Ceglowski, M. The website obesity crisis. "http://idlewords.com/talks/website_obesity.htm", 2015.

[6] Chen, K., Singla, A., Singh, A., Ramachandran, K., Xu, L., Zhang, Y., Wen, X., and Chen, Y. Osa: An optical switching architecture for data center networks with unprecedented flexibility. In *USENIX NSDI* (2012).

[7] Chen, K., Wen, X., Ma, X., Chen, Y., Xia, Y., Hu, C., and Dong, Q. Wavecube: A scalable, fault-tolerant, high-performance optical data center architecture. In *IEEE INFOCOM* (2015).

[8] Chowdhury, M., Zhong, Y., and Stoica, I. Efficient coflow scheduling with varys. In *ACM SIGCOMM* (2014).

[9] Cole, R., and Hopcroft, J. On edge coloring bipartite graphs. *SIAM Journal on Computing 11*, 3 (1982), 540–546.

[10] Doerr, C. R., and Okamoto, K. Advances in silica planar lightwave circuits. *Journal of lightwave technology 24*, 12 (2006), 4763–4789.

[11] Farrington, N., Forencich, A., Porter, G., Sun, P.-C., Ford, J., Fainman, Y., Papen, G., and Vahdat, A. A multiport microsecond optical circuit switch for data center networking. In *IEEE Photonics Technology Letters* (2013).

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

[13] Gaeta, A. L. All-optical switching in silicon ring resonators. In *Photonics in Switching* (2014).

[14] Geis, M., Lyszczarz, T., Osgood, R., and Kimball, B. 30 to 50 ns liquid-crystal optical switches. In *Optics express* (2010).

[15] Ghobadi, M., Mahajan, R., Phanishayee, A., Devanur, N., Kulkarni, J., Ranade, G., Blanche, P.-A., Rastegarfar, H., Glick, M., and Kilper, D. Projector: Agile reconfigurable data center interconnect. In *ACM SIGCOMM* (2016).

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

[17] Halperin, D., Kandula, S., Padhye, J., Bahl, P., and Wetherall, D. Augmenting data center networks with multi-gigabit wireless links. In *SIGCOMM* (2011).

[18] Hamedazimi, N., Qazi, Z., Gupta, H., Sekar, V., Das, S. R., Longtin, J. P., Shah, H., and Tanwery, A. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM* (2014).

[19] Hedlund, B. Understanding hadoop clusters and the network. "http://bradhedlund.com/", 2011.

[20] Hinden, R. Virtual router redundancy protocol (vrrp). *RFC 5798* (2004).

[21] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. The nature of datacenter traffic: Measurements and analysis. In *ACM IMC* (2009).

[22] Lightwave. Ranovus unveils optical engine, 200-gbps pam4 optical transceiver. "https://goo.gl/QbRUd4", 2016.

[23] Lin, X.-W., Hu, W., Hu, X.-K., Liang, X., Chen, Y., Cui, H.-Q., Zhu, G., Li, J.-N., Chigrinov, V., and Lu, Y.-Q. Fast response dual-frequency liquid crystal switch with photo-patterned alignments. In *Optics letters* (2012).

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

[25] Liu, V., Halperin, D., Krishnamurthy, A., and Anderson, T. F10: A fault-tolerant engineered network. In *NSDI* (2013).

[26] Liu, Y. J., Gao, P. X., Wong, B., and Keshav, S. Quartz: a new design element for low-latency dcns. In *ACM SIGCOMM* (2014).

[27] Lovász, L., and Plummer, M. D. *Matching theory*, vol. 367. American Mathematical Soc., 2009.

[28] Malkhi, D., Naor, M., and Ratajczak, D. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing* (2002), ACM, pp. 183–192.

[29] Manku, G. S., Bawa, M., Raghavan, P., et al. Symphony: Distributed hashing in a small world. In *USENIX USITS* (2003).

[30] Martinez, A., Calò, C., Rosales, R., Watts, R., Merghem, K., Accard, A., Lelarge, F., Barry, L., and Ramdane, A. Quantum dot mode locked lasers for coherent frequency comb generation. In *SPIE OPTO* (2013).

[31] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *ACM Computer Communication Review* (2008).

[32] Metaweb-Technologies. Freebase wikipedia extraction (wex). http://download.freebase.com/wex/, 2010.

[33] Misra, J., and Gries, D. A constructive proof of vizing's theorem. *Information Processing Letters 41*, 3 (1992), 131–133.

[34] OSRG. Ryu sdn framework. "https://osrg.github.io/ryu/", 2017.

[35] Pasquale, F. D., Meli, F., Griseri, E., Sguazzotti, A., Tosetti, C., and Forghieri, F. All-raman transmission of 192 25-ghz spaced wdm channels at 10.66 gb/s over 30 x 22 db of tw-rs fiber. In *IEEE Photonics Technology Letters* (2003).

[36] Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., and Shenker, S. Extending networking into the virtualization layer. In *ACM Hotnets* (2009).

[37] Pi, R. An arm gnu/linux box for $ 25. *Take a byte* (2012).

[38] Porter, G., Strong, R., Farrington, N., Forencich, A., Sun, P.-C., Rosing, T., Fainman, Y., Papen, G., and Vahdat, A. Integrating microsecond circuit switching into the data center. In *ACM SIGCOMM* (2013).

[39] Qiao, C., and Mei, Y. Off-line permutation embedding and scheduling in multiplexed optical networks with regular topologies. *IEEE/ACM Transactions on Networking 7*, 2 (1999), 241–250.

[40] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. *A scalable content-addressable network*, vol. 31. ACM, 2001.

[41] Rowstron, A., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing* (2001), Springer, pp. 329–350.

[42] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the social network's (datacenter) network. In *ACM SIGCOMM* (2015).

[43] Sanfilippo, S., and Noordhuis, P. Redis. "http://redis.io", 2010.

[44] Schrijver, A. Bipartite edge-colouring in o(δm) time. *SIAM Journal on Computing* (1998).

[45] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *IEEE MSST* (2010).

[46] Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P., et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM* (2015).

[47] Skomoroch, P. Wikipedia traffic statistics dataset. http://aws.amazon.com/datasets/2596, 2009.

[48] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for

internet applications. *ACM SIGCOMM Computer Communication Review 31*, 4 (2001), 149–160.

[49] VASSEUR, J.-P., PICKAVET, M., AND DEMEESTER, P. *Network recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Elsevier, 2004.

[50] WANG, G., ANDERSEN, D., KAMINSKY, M., PAPAGIANNAKI, K., NG, T., KOZUCH, M., AND RYAN, M. c-Through: Part-time optics in data centers. In *ACM SIGCOMM* (2010).

[51] WANG, H., XIA, Y., BERGMAN, K., NG, T., SAHU, S., AND SRIPANIDKULCHAI, K. Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity. In *ACM SIGCOMM Computer Communication Review* (2013).

[52] WON, R., AND PANICCIA, M. Integrating silicon photonics, 2010.

[53] YAMAMOTO, N., AKAHANE, K., KAWANISHI, T., KATOUF, R., AND SOTOBAYASHI, H. Quantum dot optical frequency comb laser with mode-selection technique for 1-μm waveband photonic transport system. In *Japanese Journal of Applied Physics* (2010).

[54] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX NSDI* (2012).

[55] ZHU, Z. Scalable and topology adaptive intra-data center networking enabled by wavelength selective switching. In *OSA/OFC/NFOFC* (2014).

[56] ZHU, Z., ZHONG, S., CHEN, L., AND CHEN, K. A fully programmable and scalable optical switching fabric for petabyte data center. In *Optics Express* (2015).

Appendix

Proof for non-blocking connectivity

We first formulate the wavelength assignment problem:

Problem formulation: Denote $G{=}(V{,}E)$ as a MegaSwitch logical graph, where V/E are node/edge sets, and each edge $e{\in}E$ is directed (a pair of fiber channels between two nodes, one in each direction). Given all nodes are connected via one hop, G is a complete digraph. The bandwidth demand between nodes can be represented by the number of wavelengths needed, and each node has k available wavelengths. Suppose $\gamma{=}\{k_e \mid e{\in}E\}$ is a bandwidth demand of a communication, where k_e is the number of wavelengths (i.e., bandwidth) needed on edge $e{=}(u{,}v)$ from node u to v. A feasible demand and wavelength efficiency are defined as:

Definition 2. *A demand γ is feasible iff $\sum_v k_{(u,v)} \leq k, \forall u{\in}V$ and $\sum_u k_{(u,v)} {\leq} k, \forall v{\in}V$, i.e. each node cannot send/receive more than k wavelengths.*

Definition 3. *Given a feasible bandwidth demand γ, an assignment is wavelength efficient iff it is non-interfering, satisfies γ, and uses at most k wavelengths.*

Centralized optimal algorithm: To prove the wavelength efficiency, we first show that non-interfering wavelength assignment in MegaSwitch can be recast as an edge-coloring problem on a bipartite multigraph. We first transform G into a directed bipartite graph by decomposing each node in G into two logical nodes s_i (sources) and d_i (destinations). Each edge points from a node in $\{s_i\}$ to a node in $\{d_i\}$. Then, we expand G to a multigraph G' by duplicating each edge e between two nodes k_e times. On this multigraph, the requirement of non-interference is that no two adjacent edges share same wavelength. Suppose each wavelength has a unique color, this equivalently transforms to the edge-coloring problem.

While the edge-coloring problem is $\mathcal{NP}$-hard for general graph [33], polynomial-time optimal solutions exist for bipartite multigraph. Chen et al. [7] have presented such a centralized algorithm, which we leverage to prove our wavelength efficiency for any feasible demands.

Algorithm 1: DECOMPOSE($\cdot$) Decompose G'_r into $\Delta(G'_r)$ perfect matchings

```
Input: $G'_r$
Output: $\pi{=}\{m_1{,}m_2{,}...m_{\Delta(G'_r)}\}$
if $\Delta(G'_r){\leq}1$ then
    return $G'_r$;
else
    $m{\leftarrow}$ Perfect_Matching($G'_r$);
    return $m \bigcup$ Decompose($G'_r \backslash m$);
```

Satisfying arbitrary feasible γ with wavelength efficiency: We extend G' to a $\Delta(G')$-regular multigraph[16], G'_r, by adding dummy edges, where $\Delta(G')$ is the largest vertex degree of G', *i.e.* the largest k_e in γ. For a bipartite graph G' with maximum degree $\Delta(G')$, at least $\Delta(G')$ wavelengths are needed to satisfy γ. This optimality can be achieved by showing that we only need the smallest value possible, $\Delta(G')$, to satisfy γ, which also proves wavelength efficiency since $\Delta(G'){\leq}k$.

Theorem 4. *A feasible demand γ only needs $\Delta(G')$, i.e., the minimum number of colors, for edge-coloring. Any feasible MegaSwitch graph G' can be satisfied with $\Delta(G'){\leq}k$ wavelengths using Algorithm 1.*

Proof. DECOMPOSE($\cdot$) recursively finds $\Delta(G^r)$ perfect matchings in G'_r, because "any k-regular bipartite graph has a perfect matching" [44]. Thus, we can extract one perfect matching from the original graph G'_r, and the residual graph becomes $(\Delta(G'){-}1)$-regular; we continue to extract one by one until we have $\Delta(G')$ perfect matchings[17]. Thus, any demand described by G' can be satisfied with $\Delta(G')$ wavelengths. Note that $\Delta(G'){\leq}k$, as the out-degree and in-degree of any node must be $\leq k$ in any feasible demand due to the physical limitation of k ports. Therefore any feasible demand γ can be satisfied using at most k wavelengths. □

The MegaSwitch controller runs Algorithm 1 to decompose the demands, and assigns a wavelength to the matching generated in each iteration.

Considerations for Basemesh Topology

As described in §3.2.2, basemesh is an overlay DHT network on the multi-fiber ring of MegaSwitch. DHT literature is vast, and there are many potential choices for basemesh topology. The following are the representative ones: Chord [48], Pastry [41], Symphony [29], Viceroy [28], and CAN [40]. Since we have compared Chord [48] & Symphony [29] in §3.2.2, we next look at the remaining ones.

- Pastry suffers from average path lengths when many wavelengths are used in the basemesh. Its average path length is $log_2(l{\cdot}n)$ [41], where n is the number of nodes on a ring, and l is the length of node ID in bits. For MegaSwitch, both parameter is fixed (like Chord), and we cannot reduce the path length by adding more wavelengths to basemesh.
- Viceroy emulates the butterfly network on a DHT ring. For MegaSwitch, its main issue is the difficulty of updating the routing tables and wavelength assignments

[16]A graph is regular when every vertex has the same degree.

[17]Perfect_Matching($\cdot$) on regular bipartite graph is well studied, we leverage existing algorithms in literature [27].

when the capacity of basemesh is increased and decreased. For Symphony, we can just pick a new wavelength in random, and configure accordingly without affecting the other wavelengths. In contrast, to maintain butterfly topology, adjusting capacity of basemesh using Viceroy algorithm affects all wavelength but one in the worst case.

- CAN is a d-dimensional Cartesian coordinate system on a d-torus, which is not suitable for MegaSwitch ring where every node is connected directly to every other node. However, CAN will become useful when we expand MegaSwitch into a 2-D torus topology, and we will explore this as future work.

Reconfiguration Latency Measurements on MegaSwitch Prototype

In §5.1.1, we measured ~20ms reconfiguration delay for MegaSwitch prototype, and here we break down this delay into EPS configuration delay (t_e), WSS switching delay (t_o), and control plane delay (t_c). The total reconfiguration delay for MegaSwitch is t_r=max(t_e,t_o)+t_c, as EPS and WSS configurations can be done in parallel.

EPS configuration delay: To setup a route in MegaSwitch, the source/destination EPSes must be configured. We measure the EPS configuration delay as follows (all servers and the controller are synchronized by NTP): we first setup a wavelength between 2 nodes and leave EPSes unconfigured. We let a host in one of the nodes keep generating UDP packets using `netcat` to a host in the other node. Then, the controller sends OpenFlow control message to both EPSes to setup the route, and we collect the packets at the receiver with `tcpdump`. Finally we can calculate the delay: the average and 99th percentile are 5.12ms and 12.87ms, respectively.

WSS switching delay: We measure the switching speed of the 8×1 WSS used in our testbed by switching the optical signal from one port to another port. Figure 12 shows the power readings from the original port and from the destination port, and the measured switching delay is 3.23ms (subject to temperature, drive voltage, etc.).

Control plane delay: With our wavelength assignment algorithm (see Appendix) running on a server (the centralized controller) with Intel E5-1410 2.8Ghz CPU, we measured an average computation time of 0.53ms for 40-port (40×40 demand matrix as input), and 3.28ms for 6336-port MegaSwitch (n=33,k=192). For basemesh routing tables, our greedy algorithm runs 0.45ms for 40 ports and 1.78ms for 6336 ports. For the OWS controller processing, we measured 4.31ms from receiving a wavelength assignment to set GPIO outputs. Round-trip time in control plane network (using 1GbE switch) is 0.078ms.