Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking

Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, Amin Vahdat
Google
sigcomm-jupiter-evolving@google.com

ABSTRACT

We present a decade of evolution and production experience with Jupiter datacenter network fabrics. In this period Jupiter has delivered 5x higher speed and capacity, 30% reduction in capex, 41% reduction in power, incremental deployment and technology refresh all while serving live production traffic. A key enabler for these improvements is *evolving Jupiter from a Clos to a direct-connect topology among the machine aggregation blocks*. Critical architectural changes for this include: A datacenter interconnection layer employing Micro-Electro-Mechanical Systems (MEMS) based Optical Circuit Switches (OCSes) to enable dynamic topology reconfiguration, centralized Software-Defined Networking (SDN) control for traffic engineering, and automated network operations for incremental capacity delivery and topology engineering. We show that the combination of traffic and topology engineering on direct-connect fabrics achieves similar throughput as Clos fabrics for our production traffic patterns. We also optimize for path lengths: 60% of the traffic takes direct path from source to destination aggregation blocks, while the remaining transits one additional block, achieving an average block-level path length of 1.4 in our fleet today. OCS also achieves 3x faster fabric reconfiguration compared to pre-evolution Clos fabrics that used a patch panel based interconnect.

CCS CONCEPTS

• **Networks → Data center networks**; **Traffic engineering algorithms**; **Network manageability**;

SIGCOMM'22, August 22-26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08.
https://doi.org/10.1145/3544216.3544265

KEYWORDS

Datacenter network, Software-defined networking, Traffic engineering, Topology engineering, Optical circuit switches.

ACM Reference Format:
Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, Amin Vahdat Google sigcomm-jupiter-evolving@google.com . 2022. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of ACM Conference (SIGCOMM'22).* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3544216.3544265

1 INTRODUCTION

Software-Defined Networking and Clos topologies [1, 2, 14, 24, 33] built with merchant silicon have enabled cost effective, reliable building-scale datacenter networks as the basis for Cloud infrastructure. A range of networked services, machine learning workloads, and storage infrastructure leverage uniform, high bandwidth connectivity among tens of thousands of servers to great effect.

While there is tremendous progress, managing the heterogeneity and incremental evolution of a building-scale network has received comparatively little attention. Cloud infrastructure grows incrementally, often one rack or even one server at a time. Hence, filling an initially empty building takes months to years. Once initially full, the infrastructure evolves incrementally, again often one rack at a time with the latest generation of server hardware. Typically there is no in advance blueprint for the types of servers, storage, accelerators, or services that will move in or out over the lifetime of the network. The realities of exponential growth and changing business requirements mean that the best laid plans quickly become outdated and inefficient, making incremental and adaptive evolution a necessity.

Incremental refresh of compute and storage infrastructure is relatively straightforward: drain perhaps one rack's worth of capacity among hundreds or thousands in a datacenter and replace it with a newer generation of hardware. Incremental refresh of the network infrastructure is more challenging as Clos fabrics require pre-building at least the spine layer for the entire network. Doing so unfortunately restricts the datacenter bandwidth available to the speed of the network technology available at the time of spine deployment.

Consider a generic 3-tier Clos network comprising machine racks with top-of-the-rack switches (ToRs), aggregation blocks connecting the racks and spine blocks connecting the aggregation blocks (Fig 1). A traditional approach to Clos will require pre-building spine at the maximum-scale (e.g., 64 aggregation blocks with Jupiter [33]) using the technology of the day. With 40Gbps technology, each spine would support 20Tbps burst bandwidth. As the next generation of 100Gbps becomes available, the newer aggregation blocks can support 51.2Tbps of burst bandwidth, however, these blocks would be limited to the 40Gbps link speed of the pre-existing spine blocks, reducing the capacity to 20Tbps per aggregation block. Ultimately, individual server and storage capacity would be derated because of insufficient datacenter network bandwidth. Increasing compute power without corresponding network bandwidth increase leads to system imbalance and stranding of expensive server capacity. Unfortunately, the nature of Clos topologies is such that incremental refresh of the spine results in only incremental improvement in the capacity of new-generation aggregation blocks. Refreshing the entire building-scale spine is also undesirable as it would be expensive, time consuming, and operationally disruptive given the need for fabric-wide rewiring.

We present a new end-to-end design that incorporates Optical Circuit Switches (OCSes) [31][1] to move Jupiter from a Clos to a block-level *direct-connect* topology that eliminates the spine switching layer and its associated challenges altogether, and enables Jupiter to incrementally incorporate 40Gbps, 100Gbps, 200Gbps, and beyond network speeds. The direct-connect architecture is coupled with network management, traffic and topology engineering techniques that allow Jupiter to cope with the traffic uncertainty, substantial fabric heterogeneity, and evolve without requiring any downtime or service drains. Along with 5x higher speed, capacity, and additional flexibility relative to the static Clos fabrics, these changes have enabled *architectural and incremental* 30% reduction in cost and a 41% reduction in power.

This work does not raise any ethical issues.

2 JUPITER'S APPROACH TO EVOLUTION

This section provides an overview of changes in Jupiter's architecture to address the problems introduced in §1.

[1]Appendix (§F, [41]) details the hardware design of the OCS platform and WDM transceiver technology used in Jupiter.

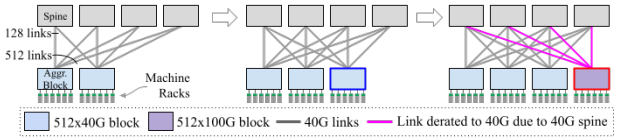

Figure 1: A 3-tier Clos network comprising machine racks, their aggregation blocks, and 40Gbs spine blocks connecting the aggregation blocks. All spines are deployed on Day 1. Blocks deployed on Day 2 are outlined in blue while those deployed on Day N are highlighted in red. Links from a 100Gbps aggregation block are derated to 40Gbps due to the 40Gbps spine.

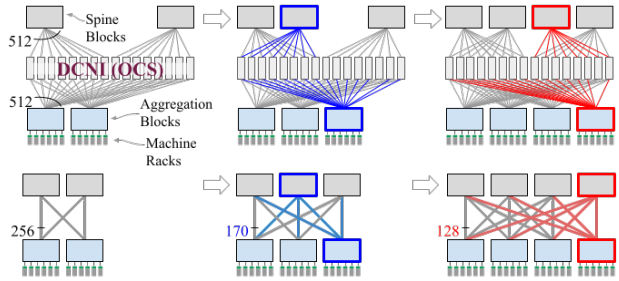

Figure 2: An optical switching layer, DCNI (top), enables incremental expansion in Jupiter while achieving full burst bandwidth among the aggregation blocks (logical topology at the bottom). The DCNI layer is implemented using Optical Circuit Switches (OCS), and allows incremental rewiring of the fabric as new blocks are added. See §5 for details.

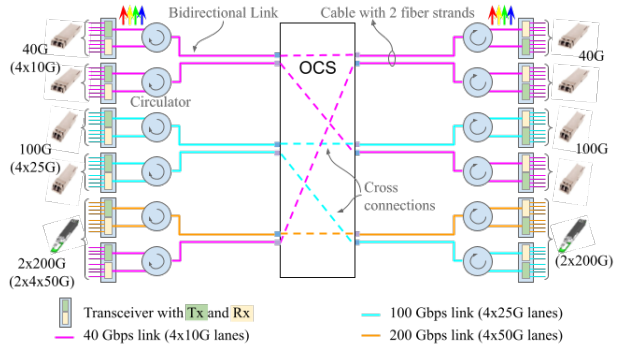

Figure 3: Tx/Rx diplexing using circulators and interoperation of coarse Wavelength Division Multiplexing (cWDM) Optics across various generations with OCS.

The datacenter interconnection layer. Jupiter fabrics [33] employ merchant silicon as the basis for aggregation and spine blocks, which in turn form a Clos fabric for building-scale datacenters. We introduced *an optical switched datacenter network interconnection layer* (***DCNI***) to connect the blocks. This layer uses MEMS-based Optical Circuit Switches (OCS) to enable fast, reliable and efficient restriping of links among the blocks (Fig. 2). With restriping, we can maintain full burst bandwidth among all aggregation blocks while incrementally adding aggregation

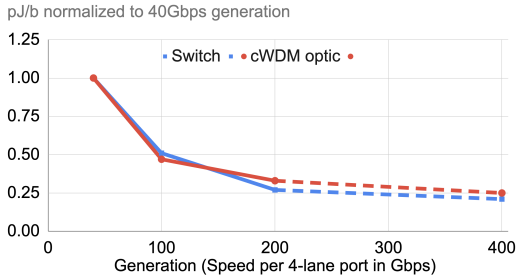

Figure 4: Diminishing returns in power consumption normalized to bandwidth (pJ/b) for successive generations of switches and optics (normalized to the 40Gbps generation).

and spine blocks, removing the challenge of upfront deployment of spines observed in Fig. 1. As we describe later, the OCSes also enable software driven traffic-aware topology reconfiguration (topology engineering). §3.1 describes the DCNI layer architecture in detail.

Multi-generational interoperability. The aggregation and spine blocks are units of deployment and typically employ the latest cost and performance competitive network technologies of the day. However, as the lifetime of a datacenter far exceeds the competitive span of a technology, deployment of multiple generations in a single fabric is inevitable. For this, Jupiter needs to allow multi-generational switching silicon and link speeds in modular blocks to coexist and inter-operate. Using Coarse Wavelength Division Multiplexing 4-lane (CWDM4) optical modules at the aggregation block interfaces is key to enabling simpler interoperability across link generations in a heterogeneous fabric (Fig. 3). Thanks to this interoperability, Jupiter is able to evolve incrementally and support heterogeneity as a norm: approximately 2/3rd of the fabrics in the fleet have aggregation blocks of at least two generations.

Streamlining the cost of optical switching. We employed several techniques to streamline the cost of the optical switching layer.

- *Halving the needed OCS ports using circulators.* We use optical circulators (Fig. 3, § F.3) to diplex the Tx and Rx into a single fiber strand, halving the number of OCS ports and fiber strands needed. This introduces a minor constraint of requiring bi-directional circuits vs. unidirectional ones - a cost-flexibility tradeoff we chose given the traffic patterns observed in our datacenters.
- *Incremental radix upgrades.* The total traffic from aggregation blocks depends on the level of compute and storage capacity deployed in a block, the network bandwidth intensity of the applications, and the level of intra-block locality in traffic. It is common that the inter-block traffic needs can be met with much less than the maximum inter-block capacity of an aggregation block. Jupiter initially deploys most blocks populating only half of optics for DCNI-facing ports and supports radix upgrade on the live fabric later, deferring the costs of optics and corresponding OCS ports until needed (§3.1, [33]).
- *Incremental deployment of DCNI.* The number of required DCNI ports grow incrementally as more aggregation and spine blocks are added to the fabric. So instead of deploying DCNI for the maximum Jupiter scale upfront, Jupiter defers the OCS costs by supporting deployment and expansion of the DCNI layer on the live fabric in three increments: 1/8⟶1/4⟶1/2⟶full size.

§5 describes the process to enable these incremental deployment changes and loss-free reconfiguration on live fabrics.

Direct-connect architecture. Fig. 1 shows that the link speed derating makes spines a dominant bottleneck as new technologies with higher link speeds are introduced. While it is possible to mitigate some of these issues by incrementally updating spine blocks or their line cards [2], such approaches induce cost, fabric-wide operational toil and production risk. Our multi-tenant and building-scale fabrics have relatively predictable traffic patterns with uncertainty that is far from worst-case permutation (§6.1), removing the need for non-blocking forwarding of worst-case permutation traffic that's enabled by Clos topology [10]. With these observations, *we removed the spine blocks from our topology for a direct-connect fabric (§3) enabled by traffic and topology engineering (§4) that jointly optimizes the inter-block forwarding and topology to achieve short and efficient paths, while simultaneously accounting for estimated uncertainty in the traffic.*

Direct connect also eliminated the cost and power associated with spine blocks (§6.5). This structural opex reduction is particularly important because upgrading to the latest generation of hardware has diminishing returns on performance and normalized cost of power with each successive generation of switches and optics speed (Fig. 4).

3 THE DIRECT-CONNECT JUPITER

Jupiter's new architecture directly connects the aggregation blocks with each other via the DCNI layer. Fig. 5 shows incremental deployment, traffic and topology engineering in action in such a fabric. The initial fabric can be built with just two blocks and then expanded (Fig. 5-①,②). The direct logical links between blocks comprise three parts: physical block-to-OCS links from each of the blocks and an OCS cross-connect (Fig. 3). Thanks to the OCS, the logical links can be programmatically and dynamically formed (§ 5).

For homogeneous blocks, we allocate logical links equally among all pairs of blocks. If demand perfectly matched the logical topology, all traffic could take the direct path between source and destination blocks. Practically, the demand is variable and not perfectly matched to the logical topology. Jupiter employs *traffic engineering* (§4.4) with a combination of direct and 1-hop indirect paths to balance performance and robustness against traffic uncertainty (Fig. 5 ③).

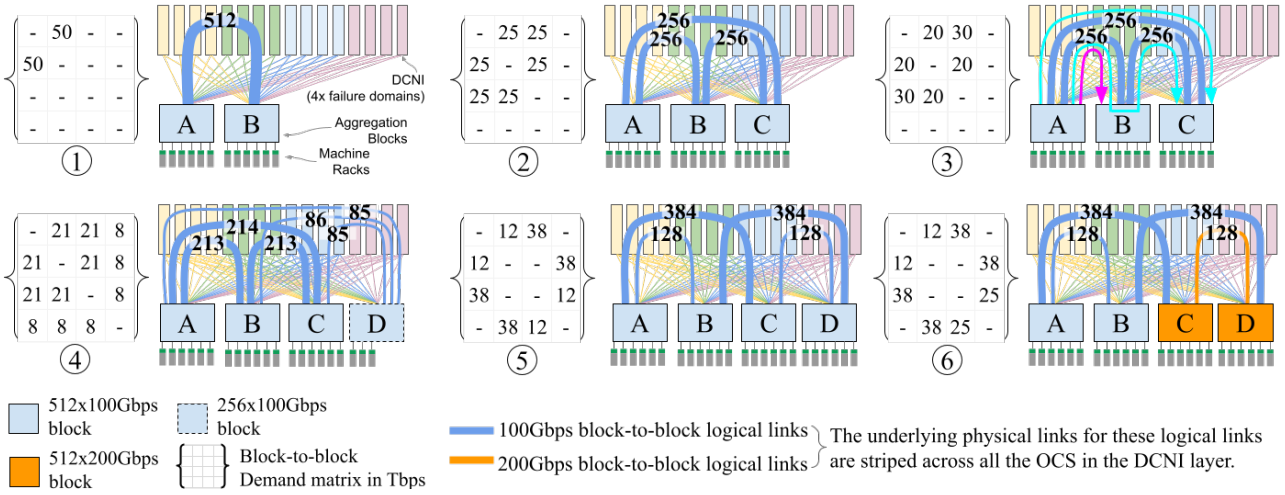

Figure 5: An incrementally deployable Jupiter fabric with a direct-connect topology enhanced by dynamic traffic and topology engineering. ①: Initially, Aggregation Blocks A, B are added with 512 uplinks each. ②: Block C is added with 512 uplinks. Each block has 50T outgoing demand, uniformly distributed across other blocks. Topology engineering forms a uniform mesh topology to match the uniform steady-state demand matrix. ③: Traffic Engineering (TE) adjusts Weighted Cost Multi Path (WCMP) weights based on finer-grained version of demand in ②: A sends all traffic (20T) to B directly (pink) and splits traffic to C (cyan) 5:1 (25T:5T) between direct and indirect paths (via B). ④: Block D is added with 256 uplinks (only a subset of machine racks in D are populated). ⑤: Block D is augmented to 512 uplinks. ⑥: Blocks C, D are refreshed to 200G link speed.

We use *topology engineering* (§4.5) to further better match the topology to demand. In Fig. 5 ④, with lower demand to/from blocks A/B/C to block D, topology engineering allocates more direct links among blocks A, B, C as compared to links to block D. We also adjust the topology based on non-uniform traffic demand (Fig. 5 ⑤), and link speeds of heterogeneous blocks (Fig. 5 ⑥).

3.1 Datacenter Interconnection Layer

OCSes for the DCNI layer are deployed in dedicated racks. The number of racks vary, but set on day 1 of deployment based on the maximum projected fabric capacity. The maximum size is 32 racks, with up to 8 OCS devices per rack. A fabric can start with one OCS per rack (1/8 populated), and later expand the DCNI capacity by doubling OCS devices in each rack. These expansions require manual fiber moves but our fiber design layout constrains such moves to stay within a rack, reducing the disruption and human effort.

We fan out the links of each superblock equally to all OCS. This allows us to create arbitrary logical topologies [46]. Due to the use of circulators, each block needs to have even number of ports attached to each OCS. These constraints ultimately guide the connectivity, as well as trigger expansions of the DCNI layer as the size of the fabric grows. This design also enables physical diversity such that a OCS rack failure impacts each Jupiter block uniformly. For example, failure of a rack in a 32 OCS rack deployment uniformly reduces capacity by 1/32, irrespective of the overall size of the fabric.

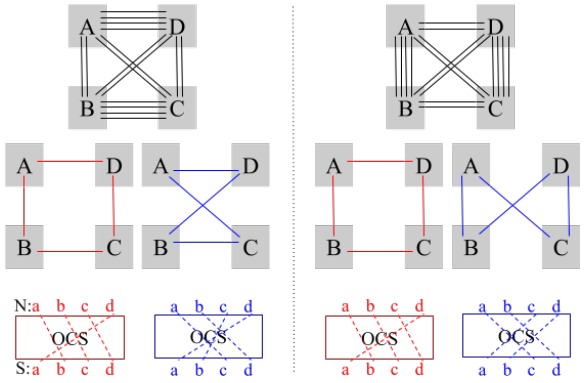

Figure 6: An illustrative multi-level logical topology factorization. Left: From top to bottom, we start with the block-level graph, factorize it into 4 factors each corresponding to a 25% failure domain (only two are shown), and finally map each factor to an OCS (in reality there are multiple OCSs). a, b, ... are ports from block A, B, ..., respectively. Each block has an even number of ports to each OCS. The two ports are illustratively placed on the two sides (N and S) of the OCS. The OCS can only cross-connect a N side port to a S side port. Right: When the block-level graph changes (e.g., topology engineering in §4.5), at each level of factorization, we minimize the difference between the new factors and the current factors. In this example, the red factor is unchanged, the blue factor has two logical links changed.

3.2 Logical Topology

Initially, we adopted a static and demand-oblivious topology. For homogeneous block speed and radix, we employ a

uniform mesh to interconnect blocks; every block pair has equal (within one) number of direct logical links. Uniform topology allows any block pair to burst up to a block's full egress bandwidth with only direct and single-transit indirect paths (§4.3). For homogeneous blocks with different radices, we set the number of links between the blocks to be proportional to the product of their radices. For example, we assign 4x as many links between two radix-512 blocks as between two radix-256 blocks. §6.1 and §C provide more details. In the common case however, a fabric would comprise aggregation blocks of different generations and supported speeds. In these cases, we rely on traffic-aware topology engineering to optimize the throughput and pathing for the observed demand matrix (§4.5).

After determining the top-level logical topology, we need to factor the block-level graph to the port-level connectivity for each OCS (Fig. 6 presents an example). We partition a block's ports into four failure domains, each comprising 25% of the ports (Fig. 7). Ideally failure domains should be equal in impact, i.e., the residual topology after loss of a single failure domain should retain $\geq 75\%$ of the throughput of the original. We achieve this by imposing a *balance* constraint that requires subgraphs corresponding to different failure domains are roughly identical. This way, if the original topology is non-uniform, the residual topology would retain the same proportionality as the original.

For reconfiguring logical topologies, we minimize the delta between the new and the current port-level connectivity to in turn minimize: i) the logical links that need to be reconfigured, and ii) the capacity that must be drained during the topology mutation (§5). A similar factorization problem for spine-full topology turns out to be NP-hard [49]. We employ a scalable approximation with a multi-level factorization using integer programming [21]. This technique allows us to solve any block-level topology for our largest fabric in minutes, while keeping the number of reconfigured links within 3% of the optimal.

4 TRAFFIC AND TOPOLOGY ENGINEERING

We discuss two levels of network adaptations in the following sections. The first level, Traffic Engineering (TE), operates on top of the logical topology and optimizes traffic forwarding across different paths based on real-time demand matrix representing communication patterns. It must manage any imbalances in inter-block connectivity resulting from striping, failures, or management operations. The second level, Topology Engineering (ToE), adapts the topology itself to increase bandwidth efficiency (§4.5).

There are two principal objectives when performing these optimizations: throughput and efficiency. First, we want to satisfy the varying traffic demand matrix while leaving enough headroom to accommodate traffic bursts, network failures and maintenance. Second, we wish to utilize direct paths more than indirect paths. We call the number of block-level edges traversed by inter-block traffic *stretch*. A direct block-level path has stretch=1.0. An indirect path via a spineblock or another aggregation block has stretch=2.0 (see an illustration in §A). Indirect paths consume more capacity and incur higher round-trip time (RTT), hurting flow-completion time (FCT) especially for small flows (§6.4). Consequently, we want to optimize for throughput and stretch, but we must simultaneously account for traffic uncertainty so that the network is able to sustain good performance between successive optimization runs. In the rest of the section, we describe the control plane elements, specifically the OCS controller. Next, we dive into the designs of traffic engineering and topology engineering, respectively. Last, we describe their interactions.

4.1 Control Plane Design

Orion, Jupiter's SDN control plane [12] programs the data-plane switches to achieve the desired traffic flow, including for traffic engineering. The network operations layer is used for the topology reconfiguration, including for topology engineering (§5).

Orion achieves high availability by partitioning the routing function in two levels (Fig. 7). At the first level, each Aggregation block is a single Orion domain. Routing Engine (RE), Orion's *intra-domain* routing app, provides connectivity within the block, and serves as an interface for external connectivity to other domains. Orion also programs the OCS (§4.2): we group OCS devices into four separate Orion domains (DCNI domains) each containing 25% of OCSes to limit the blast radius in case of an OCS control plane failure.

The second level of the control hierarchy is responsible for the links among the aggregation blocks. We partition these links into four mutually exclusive *c*olors, each color controlled by an independent Orion domain. Inter-Block Router-Central (IBR-C), the inter-block routing app in these domains, establishes reachability between blocks by computing the inter-block forwarding paths and coordinating routing advertisements for the inter-block links.

This design limits the impact of a single traffic engineering domain to 25% of the DCNI. However, this risk reduction comes at expense of some available bandwidth optimization opportunity as each domain optimizes based on its view of the topology, particularly as it relates to imbalances due to planned (e.g. drained capacity for re-stripes) or unplanned (e.g. device failures) events.

4.2 Optical Engine Controller

The Optical Engine establishes logical connectivity among the aggregation blocks by programming the OCS based on cross-connect intent from the network operations layer (Fig. 7). For uniformity with our packet switches, we implemented an OpenFlow [25] interface to the OCSes, where each OCS cross-connect is programmed as two flows that match on an input port and forward to an output port:

```
match {IN_PORT 1} instructions {APPLY: OUT_PORT 2}
match {IN_PORT 2} instructions {APPLY: OUT_PORT 1}
```

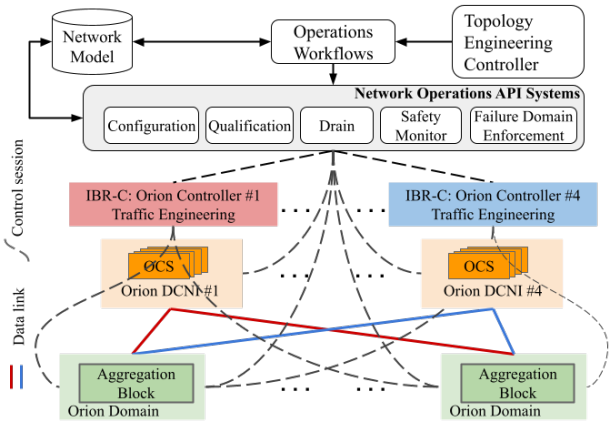

Figure 7: Orion control and network management systems.

The OCS *fails static*, maintaining the last programmed cross connect, similar to the behavior of packet switches. This keeps the dataplane available even if the control plane is disconnected. Upon re-establishing the control connection, the Optical Engine reconciles the flows with the OCS and then programs based on the latest intent. While network performance can degrade in the fail-static period, such degradations are incremental. As a result the degradation risk is far more acceptable than the risk of entirely losing the data plane due to mere control plane disconnections.

OCSes do not maintain the cross-connects on power loss, breaking the logical links. Inter-block routing can manage the sudden loss of a modest fraction of capacity (e.g. each OCS rack impacts only 1/32 of total DCNI links for a 32 rack deployment). However, synchronized power loss to many OCS devices would be problematic. We align OCS power domains with control domains so that even a significant building-level power issue only affects a maximum of 25% of OCSes. While extremely rare, more catastrophic power events are possible, but in those cases, the DCNI layer shares fate with server, storage, and data plane switches. For these cases, we rely on separate load balancing systems to divert application level traffic away from the affected fabrics.

Outside of the Optical Engine, Orion control systems, including routing, management, LLDP, and ARP, operate over the logical topology. A benefit of our layering approach is that the Layer-1 OCS devices are transparent to most of the control software, which is only aware of the logical Layer-2 connectivity.

4.3 Non-Shortest Path Routing

Traditional Clos topologies naturally support load balancing with up-down routing through the spine. However, direct-connect topologies are oversubscribed n:1 for worst case permutation traffic among n blocks with shortest path forwarding. Reducing the oversubscription ratio to 2:1 for worst case traffic requires non-shortest path forwarding. For uniform random traffic, it is possible to approach 1:1 or non-blocking communication, but requires careful management of available paths and some available slack in bandwidth demand for some blocks as demonstrated in Fig. 5. While transiting other blocks appears less bandwidth efficient, we identify four scenarios where it is beneficial.

#1: Demand between two blocks may exceed the direct path capacity between the blocks (Fig. 5③).

#2: Block pairwise traffic may be asymmetric. Forcing all traffic over direct paths means building the pairwise capacity to the larger of the two directions (due to symmetric pairwise capacity, see the constraints from circulators in §2).

#3: Datacenter traffic usually exhibits different degrees of unpredictable variability. Amortizing this uncertainty over multiple paths, including transit paths, reduces the likelihood of any path being overloaded (§4.4).

#4: In a heterogeneous network, we may choose to transit the traffic from a high-speed block to a low-speed block via another high-speed block (Fig. 9). It allows us to preserve the high-speed ports on the (first) high-speed block at the expense of reduced aggregate bandwidth on the transit block. But as we will observe in §6.1, each fabric usually contains some blocks with substantial bandwidth slack that can be exploited for transit.

Fortunately, a large block radix (256 or 512) means we can construct a mesh topology that gives each block pair high burst bandwidth using a combination of direct and transit paths. We limit traffic engineering to 1-hop (single-transit) paths as having a bounded path length is important for delay-based congestion control like Swift [19]. Longer paths also reduce bandwidth efficiency, complicate loop-free routing and change sequencing. It is worth noting that single-transit does not automatically avoid routing loops. Consider two paths A $\rightarrow$B $\rightarrow$C and B $\rightarrow$A $\rightarrow$C. If traffic is simply matched on the destination IP of C, we will create a loop between A and B. We eliminate loops by isolating the source and transit traffic into two virtual routing and forwarding (VRF) tables. Packets arriving on DCNI-facing ports but not destined for a local destination are annotated to the transit VRF. Here we match on the destination IP and forward the packet over the direct links to the destination block.

4.4 Traffic-Aware Routing

Initially, we adopted demand-oblivious routing similar to Valiant Load Balancing (VLB) [48]. It splits traffic across all available paths (direct and transit) based on the path capacity. However, under this scheme, each block operates with a 2:1 oversubscription ratio, too large for highly-utilized blocks (§6.3). To prevent over-subscription for such blocks and higher aggregate fabric throughput, we optimize the weights across different paths based on real-time traffic data.

We collect flow measurements (through flow counter diffing or packet sampling) from every server. These

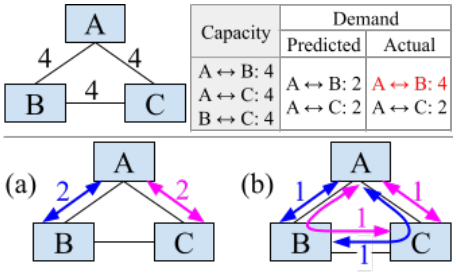

Figure 8: Comparing two routing schemes: (a) Predicted MLU of 0.5 with demand exclusively placed on direct paths. (b) Predicted MLU of 0.5, but with demand split equally between direct and transit paths. (b) is more robust achieving MLU of 0.75 vs. (a) achieving MLU of 1.0 if the actual demand between A and B turns out to be 4 units.

fine-grained measurements are aggregated to form the block-level traffic matrix every 30s. Each entry (i, j) specifies the number of bytes sent from block i to block j in the last 30s. Based on this traffic matrix stream, we maintain a *predicted* traffic matrix for WCMP optimization. The predicted traffic matrix is composed from the peak sending rate for each (i, j) pair in the last one hour. We update the predicted traffic matrix 1) upon detecting a large change in the observed traffic stream, and also 2) periodically to keep it fresh. Based on simulation, we have found an hourly refresh to be sufficient[46].

We formulate the path weight optimization as a multi-commodity flow problem with the objective to minimize the *maximum link utilization* (MLU) for the predicted traffic matrix. We found MLU to be a reasonable proxy metric for throughput as well as for resilience against traffic pattern variation. In our experience, high MLU is indicative of many links being in danger of getting overloaded, causing packet losses, increasing flow-completion time [19], and reducing fabric throughput. Our MLU formulation does not account for potential load balancing issues across parallel links connecting two blocks due to either poor hashing or skewed flow sizes. In practice we find that these simplifications have little impact on performance (see §D).

The predicted matrix should ideally tightly bound the traffic demand for each block pair (a commodity). However, the 30s traffic trace 1) shows high variability over time and, as a result, past peaks often fail to predict future peaks, 2) does not capture bursts shorter than 30s. As a result, a naive formulation runs the risk of overfitting the path weights to the predicted traffic matrix, reducing its robustness to traffic variations. Fig. 8 presents an illustrative example: Two WCMP solutions may have the same MLU for the predicted traffic, but the second solution leverages more path diversity and hence is more robust to traffic matrix uncertainty (aforementioned reason #3 for transit).

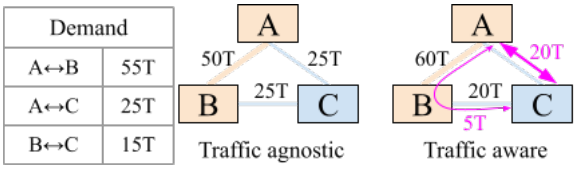

Figure 9: A and B are 200Gbps blocks. C is 100Gbps. All blocks have 500 ports. We assign 250 links between each pair of blocks for a traffic-agnostic topology. This topology cannot support the demand: aggregate demand out of A is 80Tbps while the aggregate bandwidth out of A is only 75Tbps. A traffic-aware topology can assign more links (300) between the 200Gbps blocks to boost the aggregate bandwidth out of A, and enabling transit of part of the A↔C demand via B.

We see this divergence between predicted and actual MLU consistently in both simulation and production, but generally the divergence for commodities is independent of each other. As a result, a link that carries more commodities, but a smaller fraction of each, sees a smaller load increase when one commodity bursts relative to a link that carries fewer commodities but a larger fraction of each. Therefore, while transit generally creates more load in the network, we have found it more robust to traffic misprediction.

We can thus imagine a continuum of solutions bookended by an optimization fitting the the predicted traffic with minimal MLU and stretch and a VLB-like solution that is demand agnostic and splits traffic equally across all available paths. The points along the continuum would have different tradeoffs between optimality under correct prediction and robustness under misprediction. We observe that while all datacenter traffic generally exhibits some uncertainty, different fabrics have different degrees of unpredictability due possibly to their different workload mix. Therefore, we devise a scheme, called *variable hedging*, that allows us to operate at different points along this solution continuum. In our experience, while different fabrics tend to have different optimal hedging due to difference in traffic uncertainty, the optimum for a fabric seems stable enough (§6.3) to be configured quasi-statically. The stability also allows us to search for the optimal hedging offline and infrequently by evaluating against traffic traces in the recent past. A detailed formulation for variable hedging is presented in in §B.

4.5 Topology Engineering

In a homogeneous-speed fabric, a uniform mesh topology is sufficient to support the traffic patterns we see in production thanks to the substantial bandwidth slack in the network (§6.1, §6.2). However, we still desire to align topology with the traffic matrix to reduce stretch (Fig. 12) and to protect against tail burst behavior.

In a heterogeneous speed fabric, a uniform mesh topology can have too many links derated due to being paired with lower speed peers, reducing the overall throughput in the

fabric. The example in Fig. 9 shows that a traffic-agnostic, uniform topology fails to provision enough aggregate bandwidth for traffic demand from block *A*, but a traffic-aware topology can easily accommodate the load. In addition, a traffic-aware topology may exploit a lower-utilized high-speed block to serve as a "de-multiplexer" to convert a high-speed link into multiple low-speed links. This optimization is becoming more important (Fig. 12) as our fabrics grow more heterogeneous and as high-speed (newer) blocks become the dominant offered load contributor.

To optimize the traffic-aware topology, we use a joint formulation with both link capacity and path weights as decision variables, MLU and stretch as objectives. This alignment in the formulation ensures that the traffic-engineering optimization continues to produce good outcomes for the topology. We also minimize the delta from a uniform topology – this produces networks that are unsurprising from an operations point of view, uniform-like but have either higher throughput or lower stretch or both (§6.3). Some other techniques to avoid overfit have been explored in [46]. We think more research is needed in this area.

4.6 Traffic and Topology Engineering Cadence

Today, traffic engineering and topology engineering operate at distinct time scales. Traffic engineering is the inner control loop that responds to topology and traffic changes at $O(\text{seconds})$ to $O(\text{minutes})$ granularity, depending on the urgency of the change. We require this level of optimization to take no more than a few tens of seconds even for our largest fabric. The outer loop of topology engineering is much slower and does not respond to failures or drains. Effecting a new logical topology currently relies on the same mechanism designed to handle physical topology changes, and takes $O(\text{hour})$ to finish (§5).

Thanks to OCS, topology engineering can be optimized to be on-par with (or faster than) routing changes. However, based on simulation results, we find that block-level topology reconfiguration more frequent than every few weeks yields limited benefits [46]. This is because of two main reasons: 1) concerning throughput, most traffic changes can be adequately handled by routing. The kind of traffic change that requires assistance from topology tends to happen rather slowly. 2) For a more frequent topology reconfiguration to be beneficial, a) we need to be able to generate a more accurate short-term traffic prediction than a long-term one, and b) the short-term prediction has to be sufficiently different from the long-term one to warrant a topology change. Neither has been the case historically because much of the uncertainity arises from short-term variations in traffic that's stable on longer horizons. We believe that large scale ML workloads are going to change this with their high bandwidth requirements and relative predictability of their traffic patterns.

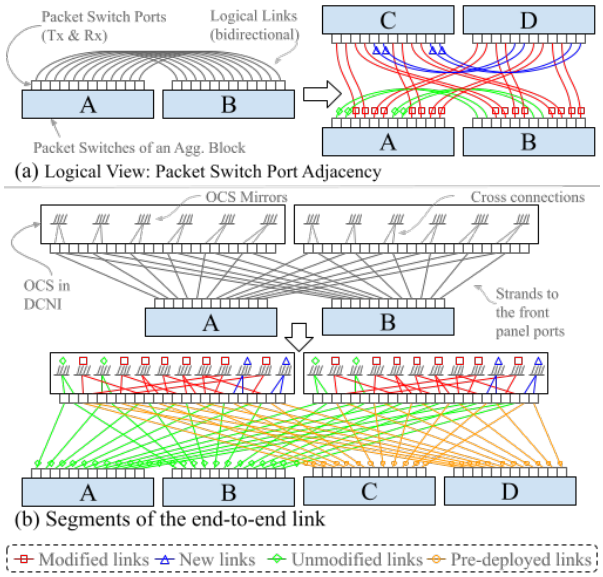

Figure 10: Illustrative rewiring to add two aggregation blocks. (a) Changes to the logical topology. (b) Corresponding changes in cross connections.

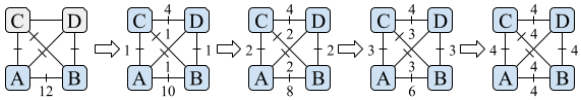

Figure 11: Incremental rewiring to achieve the change of Fig. 10. Edges labels show the number of bidirectional links.

5 LIVE FABRIC REWIRING

Fig. 10 illustrates a fabric initially comprising two aggregation blocks connected via the OCSes of the DCNI layer and its reconfiguration to add two more blocks. The optical path underlying the inter-block logical links has two types of segments: the two fiber strands from the transceiver ports on the blocks to the *front panel* of the OCS switches, and the *cross connections* within the OCS that connect these links (Please see Fig. 3). Adding two blocks requires modifying many logical links (Fig 10(a)-right), but the change to the underlying optical path is achieved by only reprogramming the cross connections – none of the strands connected to the front panel are moved or modified during these operations (Fig. 10(b)-bottom).

Common network operations, connecting/disconnecting entire blocks or their additional radix with the fabric, adapting the proportional direct connectivity among blocks for topology engineering (§4.5), and even converting a fabric from a Clos to direct connect, follow this pattern. Thanks to the OCS, the cross connections can be programmed quickly and reliably using a software configuration, enabling huge operational benefits. Jupiter also supports rewiring of strands connected to the front panel, but such manual rewiring is needed for either relatively infrequent operations, or done

when the front-panel facing strands are not part of end-to-end links. Appendix (§E.2) lists use cases for such operations.

Safety considerations during rewiring: While OCSes significantly simplify fabric rewiring, performing this operation on a live fabric requires two key safety considerations: maintaining availability and performance SLOs, and reducing risk for production best practices.

- **Maintaining production traffic SLOs** relies heavily on maintaining sufficient capacity at all cuts in the network to avoid significant congestion and packet loss. A single shot rewiring for Fig 10(a)-right would create a substantial disruption in capacity as 2/3rd of the links go offline during rewiring and routing systems adjust to the new topology. However, an incremental rewiring can keep more capacity online. Fig. 11 shows such a incremental sequence for the change in Fig. 10 – it preserves at least 10 units of bidirectional capacity (approx. 83%) between blocks A and B at all steps, including accounting for links being temporarily unavailable during rewiring. Using the same principle, in production settings we can support increments as small as reprogramming/rewiring a single OCS chassis at a time, allowing us to maintain safety even in highly utilized fabrics. Each incremental rewiring step is bookended by traffic drain/undrain, which makes the rewiring loss-free.
- **Avoiding correlated failures** across independent failure domains and limiting the impact blast radius of operations is critical as software and operational bugs are inevitable at scale. To achieve this, the rewiring has to additionally avoid concurrent operations on multiple failure domains (DCNI and IBR), and ensure that operations on a failure domain have successfully completed before proceeding to the next one to avoid cascading failures in a run-away train scenario.

§E.1 describes the workflow for rewiring in our production fabrics, including details for finding suitable reconfiguration increments, draining the links before modification to achieve loss-free reconfiguration at each step, and enforcement of other safety considerations.

6 EVALUATION

6.1 Traffic Characteristics

Our production traffic matrix has two salient points for network design: i) inter-block traffic is best described by a *gravity model* and ii) offered load varies significantly across different blocks. The gravity model states that the traffic demand between a pair of aggregation blocks is proportional to the product of their total demands (§C). This trend for block-level traffic arises due to uniform random communication pattern for fabric-wide machine-to-machine traffic [10]. Gravity model allows us to make baseline informed choices for the number of links to allocate between block pairs in a heterogeneous network fabric. For example, the ratio of the capacity between a pair of 20Tbps blocks to the capacity between a pair of 50Tbps in the same fabric would be 4:25.

We find that typically a small number of blocks contribute the majority of the offered load for a fabric. To quantify this, we define the *normalized peak offered load* (NPOL) for an aggregation block as the peak (99th percentile) offered load normalized to the capacity of that block. Next, we consider the NPOL for all aggregation blocks in ten heavily loaded fabrics with a mix of Search, Ads, Logs, Youtube and Cloud. Each fabric has a distribution of NPOLs for its set of aggregation blocks. We observe that there is large variation in all these distributions. Indeed, the coefficient of variation (i.e., the ratio of standard deviation to mean) of NPOL ranges from 32% to 56% across the ten fabrics. Over 10% of blocks in each fabric have NPOL below one standard deviation from the mean NPOL in that fabric, and the least-loaded blocks have NPOL <10%, indicating the substantial slack bandwidth present in all fabrics that can be exploited for transit traffic.

6.2 Optimal throughput and stretch for direct connect fabrics

We evaluate the optimal throughput and stretch for direct connect topologies using the ten heavily loaded fabrics from §6.1. Fabric throughput is the maximum scaling of its traffic matrix before any part of the network is saturated [17]. A higher throughput corresponds to lower MLU and hence larger headroom for traffic growth. For simplicity, we consider a single traffic matrix $T^{\max}$ where $T^{\max}_{ij}$ is the peak traffic from block i to block j over one week. We pick one week to capture daily and weekly recurring peaks.

Fig. 12 shows that a uniform direct connect topology achieves maximum throughput in most fabrics. Traffic-aware topology further improves the throughput to the upper bound in two heterogeneous-speed fabrics. Only fabric A fails to achieve the upper bound. The bottom figure illustrates the minimum stretch without degrading the throughput for $T^{\max}$. In uniform direct-connect topologies, the stretch is higher because traffic demand can substantially exceed the direct path capacity (reason #1 for transit in §4.3). In contrast, traffic-aware topology engineering admits most traffic on direct paths and delivers stretch closer to the lower bound of 1.0. The residual transit traffic is attributable to asymmetric and higher variability in the traffic (§4.3. For comparison, a Clos topology has stretch of 2.0 since all inter-block traffic transits spine blocks. These results assume perfect traffic knowledge to characterize the performance limits. Next, we present more detailed simulation results as well as production measurements where the traffic is not known a priori.

6.3 Simulation Results

In this section, we use fabric D to illustrate the tradeoff between different traffic and topology engineering designs under direct connect. Fabric D is one of the most loaded in the fleet and its speed heterogeneity is growing due to a high

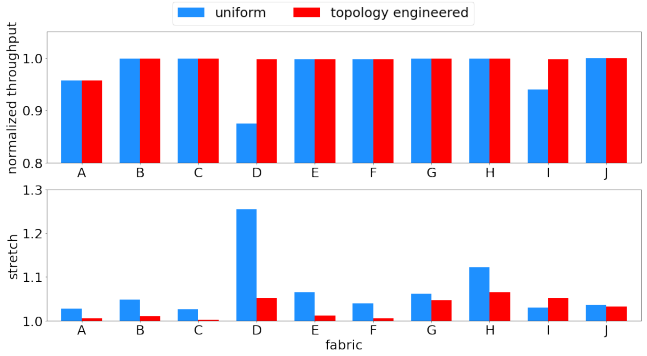

Figure 12: Top: optimal throughput in 10 fabrics with uniform direct connect vs. topology-engineered direct connect. Throughput is normalized by an upper bound that assumes a perfect, high-speed spine layer that eliminates link speed derating and perfectly balances its traffic across links. Bottom: Optimal stretch under the same throughput.

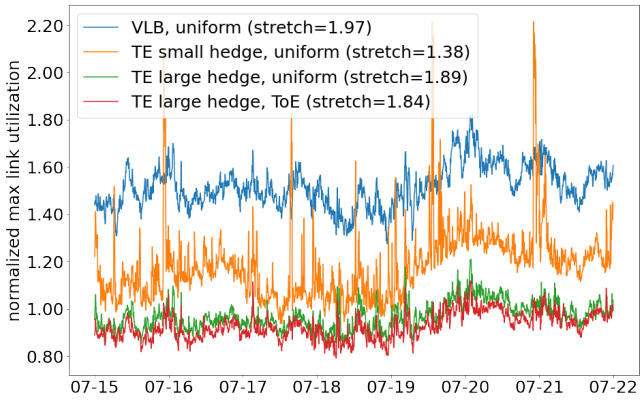

Figure 13: MLU time series under different traffic and topology engineering configurations, normalized by the peak MLU under routing and topology with perfect traffic knowledge. Stretch is stable and the average value is in the legend.

ratio of low-speed to high-speed blocks and the growing traffic of the latter. We use time series simulation, verified to accurately match production (§D), to study the following configurations: 1) demand-oblivious routing (VLB) under uniform topology, 2) traffic engineering (TE) with smaller or 3) larger hedge under uniform topology, and 4) TE with larger hedge under topology engineering (ToE). Fig. 13 shows the MLU time series and stretch under the four configurations. VLB cannot support the traffic most of the time as the high MLU indicates (we also validate this result in a less-utilized production fabric in the next section). Under traffic-aware routing, larger hedging reduces average MLU and eliminates most spikes, at the cost of higher stretch. Topology engineering can reduce both MLU and stretch. At the aggregate level, the 99th percentile MLU under traffic and topology engineering (red time series) is within 15% of the 99th percentile optimal MLU. Optimal assumes perfect routing and topology where traffic is known at each time snapshot.

The trade-off between MLU and stretch varies from fabric to fabric so we must consider a fabric-specific hedge factor. For example, in fabric G, the 99th percentile MLU is 5% lower and the average stretch is 21% lower for a small hedge compared with a larger hedge. This is because G's traffic is more stable and hence predictable and the small hedge favors optimality for correct prediction as opposed to robustness to misprediction. There are differences among fabrics, but within a fabric the MLU and stretch under different hedging levels have stable ranking over time, and thus we only need to reconfigure/re-optimize infrequently (§4.4).

6.4 Production Experience

We examine an instance of Clos to direct-connect conversion and another of uniform to traffic-aware topology conversion. We focus on transport layer measurements of min RTT, flow completion time (FCT), and delivery rate, and compare these metrics before and after the conversion. For each metric, we compute the daily median and 99th percentile for two weeks before and after the conversion. We then used the Student's t-test to determine whether the changes are statistically significant and report the differences where p-value$\leq$ 0.05 in Table 1. The min RTT and FCT of small flows are sensitive to path length, and are reduced after the conversion from Clos to uniform direct-connect topology (stretch is reduced from 2 to 1.72). Min RTT and FCT are also reduced after enabling topology engineering (stretch is reduced from 1.64 to 1.04). Lower min RTT contributes to higher delivery rate. After the conversion from Clos to uniform direct-connect topology, total DCN-facing capacity of aggregation blocks increased by 57% since removing the lower speed spine removed the derating effect and allowed more links to operate at a higher speed. The 99th percentile FCT, which is dominated by queuing delay, is either reduced or statistically unchanged, indicating that network congestion is either reduced or unchanged by conversions.

To validate the advantage of TE, we conducted an experiment on a moderately-utilized uniform direct-connect fabric where we turned off TE and ran VLB for one day. During the experiment, stretch increased from 1.41 to 1.96, total load increased by 29% even though the demand incidentally decreased by 8% (within the typical variation over time). Min RTT increased by $6-14$% due to larger stretch and congestion, 99-percentile FCT increased by up to 29% due to heavier congestion, and average discard rate increased by 89%.

Fabric rewiring with OCS: Table 2 shows the speed up in the rewiring of the DCNI layer for a 10-month period in fabrics with OCS relative to our earlier work where the DCNI layer comprised manual patch-panel (PP) [49]. OCS provides a 9.6x speedup at the median, and 3x speedup at the mean. Due to the faster rewiring speeds in OCS fabrics, there is a several folds larger contribution of operational

	Clos to uniform direct connect	Uniform to ToE direct connect
Min RTT 50p	-6.89%	-11.02%
Min RTT 99p	-7.00%	-16.01%
FCT (small flow) 50p	-5.77%	-12.37%
FCT (small flow) 99p	-24.22%	p>0.05
FCT (large flow) 50p	-10.29%	p>0.05
FCT (large flow) 99p	p>0.05	p>0.05
Delivery rate 50p	13.59%	p>0.05
Delivery rate 99p	36.35%	13.76%
Discard rate	p>0.05	N/A

Table 1: Transport metrics and discard percentage changes for 1) Clos to uniform direct connect topology conversion in a fabric, where stretch is reduced from 2 to 1.72, and 2) uniform to topology engineered (ToE) direct connect in a different fabric, where stretch is reduced from 1.64 to 1.04.

	Speedup w/ OCS	Operations workflow on critical path	
		OCS	PP
Median	9.58 x	37.7%	4.7%
Average	3.31 x	31.1%	8.4%
90^{th}-%	2.41 x	27.0%	10.9%

Table 2: Comparison of fabric rewiring performance between fabrics with OCS and PP based DCNI.

workflow software (§E.1) on the critical path for rewiring for OCS based fabrics, highlighting an obvious avenue for even further improvements. We are investing in integration of fabric reconfiguration capabilities within the SDN controller which allows us to make fabric-wide rewiring safely but at much higher rates, unlocking the potential of OCS to a fuller extent.

6.5 Cost Model

We now examine the costs of our current Plan of Record (PoR) deployed architecture (with direct-connect topology, OCSes and circulators) and an equivalently sized baseline conventional datacenter design (with Clos topology and PP based DCNI, but without circulators).

Fig. 14 presents the components in the cost model for a Clos-based or direct-connect network architectures. The machine rack cost (Fig. 14 ⓪) is not included in the fabric cost but everything above it (①-⑤) comprising the aggregation block switches (assuming the same radix for blocks in both architectures), optics, copper cables, fiber, rack enclosures, OCS and spines is included as applicable. We also consider the normalized cost of power for each architecture.

Our current Jupiter PoR architecture has 70% capex cost of the baseline. The savings from direct connect, which eliminated ④ and ⑤, and from using circulators outweigh the expense of using OCSes over patch panels in ③.

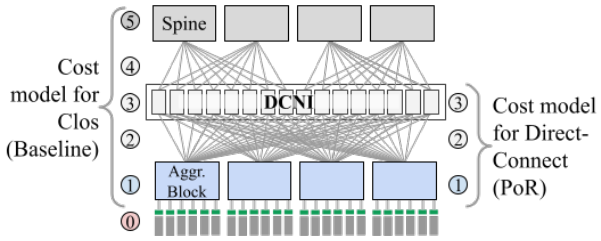

Figure 14: Layered components considered in the network fabric cost model for Clos and direct-connect variants.

Using PP instead of OCSes in ③ could further reduce the capex. However, it would neither reduce toil during expansions nor unlock the ability to promptly engineer the topology based on traffic patterns. Our use of direct-connect topology and circulators each separately halves the OCS ports required in the architecture. Furthermore, the cost of the OCS is amortized over multiple generations of aggregation blocks. Consequently, the true cost of the PoR architecture is between 62% and 70% capex of baseline in practice, depending on the datacenter service lifetime.

The normalized cost of power for the PoR architecture is 59% of baseline. Most of the power reduction comes from removal of spine switches ⑤ and associated optics in the direct-connect architecture. Circulators are passive with no power consumption, and the OCSes consume negligible power.

These estimates provide a lower bound on the savings achieved in production relative to the baseline. The labor cost of OCS-based expansions is expected to be lower than PP-based ones in both time and toil. The network efficiency due to topology engineering with the OCS results in both building less network capacity and deferring network augments, which reduce deployment costs.

6.6 System Complexity

The combination of direct-connect topology and traffic engineering has delivered significant cost reductions and similar or better performance than Clos fabrics, but they have also substantially increased the system complexity. We mitigate this by investing in analysis and debugging tools. For instance, we rely on record-replay tools based on the network state and the routing solution to debug reachability and congestion issues. Radix planning similarly needs to account for the dynamic transit traffic. We have eased the planning difficulty using automated analysis.

7 RELATED WORK

Network topologies: Clos-based topologies [1, 2, 14, 24, 33], while highly scalable, are not the most effective when it comes to incremental network expansions [37], and path length. Jellyfish [36, 37] adopts an unstructured topology to facilitate incremental expansions in exchange for increase in cabling complexity, management, and routing design. Direct-connect topologies [6, 18, 32, 45], primarily in High-Performance Computing (HPC), optimize the long

reach links in the network by grouping high-radix routers as virtual routers to improve the effective radix of the network and reduce the network diameter. Dragonfly+ [32] takes a hybrid approach where routers in each group are connected in a Clos topology instead of a fully connected mesh, while FatClique [45] has a hierarchy of directly connected cliques. Slim Fly [6] gets closer to the theoretical optimal network diameter by solving for graphs that approximate the solution to the degree-diameter problem. The direct-connect Jupiter topology adopts a similar hybrid approach as in Dragonfly+ [32] where aggregation blocks with Clos-based topology are directly connected. Compared to hardware-based adaptive routing in Dragonfly+, Jupiter provides software-based traffic engineering that can work on legacy commodity silicon switches. With the OCS layer, Jupiter provides SDN-based topology engineering to match the topology to the traffic pattern, a unique feature compared to other direct connect networks. Moreover, Jupiter supports heterogeneous speed topologies whereas the HPC networks are typically uniform deployments, and heterogeneous networks are only explored theoretically in a network with at most two types of switches [36], to the best of our knowledge.

Reconfigurable networks: There is extensive work on reconfigurable interconnects [5, 9, 11, 13, 16, 22, 26, 27, 29, 43]. Helios [11] takes a similar approach to the OCS layer presented in this paper by leveraging hybrid electrical/optical switches for a dynamic topology, but lacks fine-grained, demand-based traffic engineering. RotorNet [27] takes a different design point by alternating the OCS connectivity through a static set of configurations that provide uniform bandwidth between all endpoints. This design completely eliminates the need for a centralized control plane at the cost of sub-optimal VLB-like bandwidth utilization when traffic pattern is not uniform.

Traffic and topology engineering: Some prior work on demand-aware routing [44, 47] reconfigure based on multiple traffic matrices, but focus on routing for fixed wide-area networks. More similar to Jupiter, recent works propose various algorithms for fine-grained, demand-based topology and routing optimizations in networks with OCS interconnect [8, 40], or take a more coarse-grained approach based on the high level flow types [15]. Jupiter's traffic engineering and topology engineering algorithms and update cadences have evolved over time to scale to large datacenters with highly variable traffic with proven success in production. Previous works develop robust routing algorithms under traffic uncertainty by formulating traffic variation as a penalty [30] or supporting a given range of traffic with performance guarantee [4]. While our designs share similar considerations, our formulation is simpler and addresses different levels of uncertainty. There are also other routing techniques that either focus only on robustness to failure [20, 38], or take a demand-oblivious approach [3] which perform less effectively relative to demand-aware approaches. There is also prior work in reconfiguring topology in long-haul WAN optical networks to adjust links rates based on SNR [35], or optimize the number of hops by evaluating the signal strength [34]. Jupiter provides DCNI topology engineering based on traffic patterns primarily. We assume that the links are maintained to support the intended quality for signal strength and bit error rates (§ E.1).

8 CONCLUSION

This paper presents a retrospective on the evolution of Google's datacenter network, Jupiter, spanning nearly a decade. Central to this journey are MEMS-based Optical Circuit Switches, Software-Defined Networking and Automated Safe Operation as key enabling technologies. With these underlying technologies, Jupiter transformed into an incrementally deployable fabric where heterogeneous blocks of networking co-exist and are refreshed modularly.

Jupiter started with a standard Clos topology, which delivers optimum throughput for arbitrary admissible traffic patterns. To exploit capacity slack in our production networks and to address technology refresh challenges with spine blocks in Clos fabrics, we evolved Jupiter into a direct-connect topology and enabled both dynamic traffic and topology engineering for the observed block-level traffic patterns. Doing so achieves comparable throughput and shorter paths on average relative to the conventional Clos approach, in addition to substantial Capex and Opex savings.

Several future directions remain active areas of research. These include i) co-optimizing workload scheduling with network traffic and topology engineering to enable predictable end-to-end performance, which is important for emerging high bandwidth Machine Learning workloads, ii) reducing bandwidth stranding and other optimizations due to the inter-operation among disparate heterogeneous technologies, and iii) scaling and extending out to the campus and inter-datacenter networking. Our vision is to enable a continuously evolving virtual datacenter-level computer that jointly optimizes compute, storage and ML with dynamic traffic and topology engineered networking.

9 ACKNOWLEDGEMENTS

Many teams contributed to the evolution of the datacenter network technologies within Google. In particular, we would like to acknowledge the Networking Infrastructure, Platforms (Hardware and Optics), Software Quality Assurance (SQA), Global Capacity Delivery (GCD), Datacenter Operations, Site Reliability Engineering (SRE), and Program Management Organization (PMO) teams, to name a few. We would also like to thank our shepherd, Rachee Singh, as well as the anonymous SIGCOMM reviewers for their useful feedback.

REFERENCES

[1] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.* 38, 4 (August 2008), 63–74. https://doi.org/10.1145/1402946.1402967

[2] Alexey Andreyev, Xu Wang, and Alex Eckert. 2019. Reinventing Facebook's data center network. https://engineering.fb.com/2019/03/14/data-center-engineering/f16-minipack/. *Facebook Engineering* (2019).

[3] David Applegate, Lee Breslau, and Edith Cohen. 2004. Coping with Network Failures: Routing Strategies for Optimal Demand Oblivious Restoration. In *Proc. ACM SIGMETRICS*.

[4] David Applegate and Edith Cohen. 2003. Making Intra-domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs. In *Proc. ACM SIGCOMM*.

[5] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. 2020. Sirius: A Flat Datacenter Network with Nanosecond Optical Switching. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*.

[6] Maciej Besta and Torsten Hoefler. 2014. Slim fly: A cost effective low-diameter network topology. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 348–359.

[7] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Murphy. 2016. Site Reliability Engineering. https://sre.google/books/. *Google Engineering* (2016).

[8] Peirui Cao, Shizhen Zhao, Min Yee Teh, Yunzhuo Liu, and Xinbing Wang. 2021. TROD: Evolving From Electrical Data Center to Optical Data Center. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 1–11.

[9] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2013. OSA: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking* 22, 2 (2013), 498–511.

[10] William James Dally and Brian Patrick Towles. 2004. *Principles and practices of interconnection networks*. Elsevier.

[11] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. 2010. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *Proc. ACM SIGCOMM*.

[12] Andrew D Ferguson, Steve Gribble, Chi-Yao Hong, Charles Edwin Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, et al. 2021. Orion: Google's Software-Defined Networking Control Plane.. In *NSDI*. 83–98.

[13] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 216–229.

[14] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM '09)*. Association for Computing Machinery, New York, NY, USA, 51–62. https://doi.org/10.1145/1592568.1592576

[15] Chen Griner, Johannes Zerwas, Andreas Blenk, Manya Ghobadi, Stefan Schmid, and Chen Avin. 2021. Cerberus: The Power of Choices in Datacenter Topology Design-A Throughput Perspective. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–33.

[16] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer. 2014. FireFly: A Reconfigurable Wireless Data Center Fabric Using Free-space Optics. In *Proc. ACM SIGCOMM*.

[17] S. A. Jyothi, A. Singla, P. B. Godfrey, and A. Kolla. 2016. Measuring and Understanding Throughput of Network Topologies. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[18] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*. IEEE, 77–88.

[19] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[20] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-Oblivious Traffic Engineering: The Road Not Taken. In *Proc. NSDI*.

[21] Weiqiang Li, Rui Wang, and Jianan Zhang. 2022. Configuring data center network wiring. US Patent 11,223,527.

[22] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. 2015. Scheduling Techniques for Hybrid Circuit/Packet Networks. In *Proc. ACM CoNEXT*.

[23] Hong Liu, Ryohei Urata, Xiang Zhou, and Amin Vahdat. 2020. *Evolving Requirements and Trends in Datacenter Networks*. Springer handbook of optical networks.

[24] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-Tolerant Engineered Network. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. USENIX Association, Lombard, IL, 399–412. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/liu_vincent

[25] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM Computer Communication Review* 38 (2008), 69–74. Issue 2. https://doi.org/10.1145/1355734.1355746

[26] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *Proc. NSDI*.

[27] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 267–280.

[28] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. 2020. Experiences with Modeling Network Topologies at Multiple Levels of Abstraction. In *17th Symposium on Networked Systems Design and Implementation (NSDI)*. https://www.usenix.org/conference/nsdi20/presentation/mogul

[29] George Porter, Richard D. Strong, Nathan Farrington, Alex Forencich, Pang-Chen Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. In *Proc. ACM SIGCOMM*.

[30] Matthew Roughan, Mikkel Thorup, and Yin Zhang. 2003. Traffic engineering with estimated traffic matrices. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*. 248–258.

[31] R Ryf, J Kim, JP Hickey, A Gnauck, D Carr, F Pardo, C Bolle, R Frahm, N Basavanhally, C Yoh, et al. 2001. 1296-port MEMS transparent optical crossconnect with 2.07 petabit/s switch capacity. In *Optical Fiber Communication Conference*. Optical Society of America, PD28.

[32] Alexander Shpiner, Zachy Haramaty, Saar Eliad, Vladimir Zdornov, Barak Gafni, and Eitan Zahavi. 2017. Dragonfly+: Low cost topology for scaling datacenters. In *2017 IEEE 3rd International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*. IEEE, 1–8.

[33] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hanying Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM '15*.

[34] Rachee Singh, Nikolaj Bjorner, Sharon Shoham, Yawei Yin, John Arnold, and Jamie Gaudette. 2021. Cost-effective capacity provisioning in wide area networks with Shoofly. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 534–546.

[35] Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. 2018. RADWAN: rate adaptive wide area network. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 547–560.

[36] Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. 2014. High throughput data center topology design. In *11th USENIX Symposium on Networked Systems Design and Implementation NSDI 14)*. 29–41.

[37] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In *Proc. USENIX NSDI*.

[38] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. 2011. Network Architecture for Joint Failure Recovery and Traffic Engineering. In *Proc. ACM SIGMETRICS*.

[39] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky H.Y. Wong, and Hongyi Zeng. 2016. Robotron: Top-down Network Management at Facebook Scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 426–439. https://doi.org/10.1145/2934872.2934874

[40] Min Yee Teh, Shizhen Zhao, Peirui Cao, and Keren Bergman. 2020. COUDER: robust topology engineering for optical circuit switched data center networks. *arXiv preprint arXiv:2010.00090* (2020).

[41] Ryohei Urata, Hong Liu, Kevin Yasumura, Erji Mao, Jill Berger, Xiang Zhou, Cedric Lam, Roy Bannon, Darren Hutchinson, Daniel Nelson, Leon Poutievski, Arjun Singh, Joon Ong, and Amin Vahdat. 2022. Mission Apollo: Landing Optical Circuit Switching at Datacenter Scale. *arXiv*.

[42] Ryohei Urata, Hong Liu, Xiang Zhou, and Amin Vahdat. 2017. Datacenter interconnect and networking: From evolution to holistic revolution. In *Proceedings of Optical Fiber Communication (OFC) 2017 Conference*.

[43] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 327–338.

[44] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. 2006. COPE: Traffic Engineering in Dynamic Networks. In *Proc. ACM SIGCOMM*.

[45] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. 2019. Understanding Lifecycle Management Complexity of Datacenter Topologies. In *Proc. NSDI*.

[46] Mingyang Zhang, Jianan Zhang, Rui Wang, Ramesh Govindan, Jeffrey C. Mogul, and Amin Vahdat. 2021. Gemini: Practical Reconfigurable Datacenter Networks with Topology and Traffic Engineering. arXiv:cs.NI/2110.08374

[47] Y. Zhang and Z. Ge. 2005. Finding critical traffic matrices. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*.

[48] R. Zhang-Shen and N. McKeown. 2005. Designing a Predictable Internet Backbone with Valiant Load-balancing. In *Proc. IEEE IWQoS*.

[49] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C Mogul, and Amin Vahdat. 2019. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 221–234.

[50] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*. Association for Computing Machinery, New York, NY, USA, Article 5, 14 pages. https://doi.org/10.1145/2592798.2592803

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A AGGREGATION BLOCK DESIGN CHOICE

The first aggregation block in Jupiter [33] was a 3 switch stage design with stage 1 comprising ToRs and stages 2 and 3 arranged in 8 independent blocks (called Middle Blocks or MBs). We have since settled on a generic 4 MB, 3 switch stage design for subsequent generations of Jupiter aggregation blocks.

Fig. 15 (top) shows the general design for an aggregation block. There are 4 separate MBs for redundancy, each housed in a physical rack with 2 stages of switches interconnected within each MB in a blocking topology.

The MBs collectively expose up to 512 links towards ToRs and up to 512 links towards the DCNI layer. Links within the aggregation block can run at 40G (4x10G lanes), 100G (4x25G lanes), or 200G (4x50G lanes) depending on the generation of the packet switches, with a roadmap to run at 400G (4x100G) and 800G (4x200G) in the future.

Fig. 15 (bottom) shows the network path of an intra-block flow (purple), an inter-block flow via a direct inter-block path (cyan) as well as via an intermediate aggregation block (pink). Traffic engineering and topology engineering combine to maximize the direct cyan paths and minimize the pink paths for cross-block traffic by optimizing the topology and routing (forwarding paths) for the cross-block traffic pattern.

There are two reasons for choosing a 3-stage aggregation block vs. a flat 2-stage topology. First, having 4 fully connected MBs allows the ToR uplinks to be deployed in multiples of 4 enabling flexibility in bandwidth provisioning based on the compute/memory/storage under the ToR in the machine rack. A flat 2-stage aggregation block topology would have required each ToR to connect to all the aggregation switches (rather than just 4 MBs) affording less flexibility in ToR uplink provisioning.

Second, the 2 stages within each MB allow each MB to connect to all the corresponding MBs in other aggregation blocks in the fabric. This enables "transit traffic" in an intermediate aggregation block to reach any destination aggregation block by *bouncing* within the MB and not all the way to the ToRs as depicted by the pink path in Fig. 15. It is

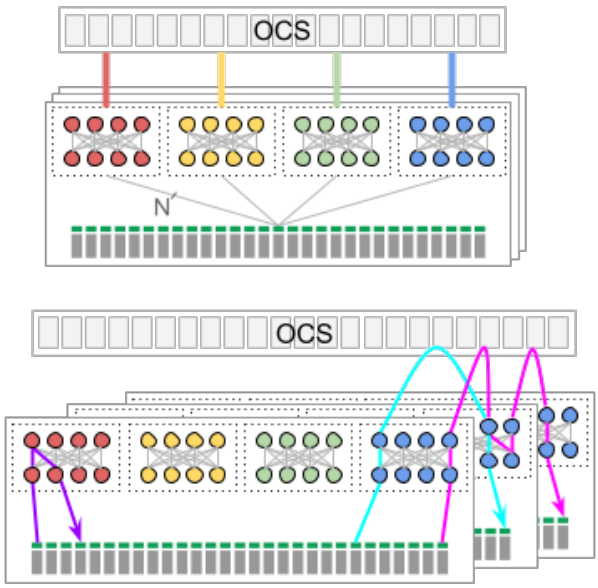

Figure 15: The top figure shows the generic Aggregation Block design which follows a 4 post (Middle Blocks or MBs) and a 3 switch stage architecture (ToR, stage 2, stage 3). Each ToR switch connects to every MB with N uplinks (N=1,2,4,...). The bottom figure shows the network path of an intra-block flow (purple), an inter-block flow via a direct inter block path (cyan) as well as via an intermediate aggregation block (pink).

undesirable to require transit traffic to bounce via the ToRs because ToRs are deployed incrementally on demand and ToR-to-MB links are less evenly balanced compared to stage 2 to stage 3 links which are at a higher level of the network hierarchy. The Traffic engineering controller monitors the residual bandwidth in each MB and optimally uses the most idle aggregation blocks for transit traffic.

B VARIABLE HEDGING FORMULATION

At a high level, we implement hedging by introducing additional constraints to the basic multi-commodity flow formulation to force a commodity to go over multiple block-level paths. A set of link-disjoint paths (direct and single-transit paths) is given for each commodity. A path's capacity is denoted by C_p; it's the minimum capacity across the one or two block-level edges comprising the path. Summing up the capacity across all paths for a commodity gives the *burst bandwidth*: $B = \sum_p C_p$. Let the offered load of a commodity be D, and the decision variable on how much offered load to place on a path be x_p, we have $\sum_p x_p = D$, i.e., all offered load must be allocated. We use a parameter *Spread*, denoted by S where $S \in (0, 1]$, to indicate the fraction of burst bandwidth a commodity must be spread over. Specifically, the hedging constraint is formulated as $x_p \leq D \cdot \frac{C_p}{B \cdot S}$. Note that when $S = 1$, $x_p = D \cdot \frac{C_p}{B}$ if offered load is to be fully satisfied. It degenerates to a demand-oblivious allocation (similar to VLB) in which each commodity (regardless of offered load) is allocated to available paths proportional to each path's capacity. When S approaches 0, the hedging constraint becomes free, letting the formulation degenerate to the classic multi-commodity flow. These are desired properties allowing us to operate anywhere between the two extreme points by adjusting S.

Other formulations we tried to avoid overfitting either scaled poorly or did not allow us to explicitly trade-off MLU and stretch for different workload and fabrics or both. For example, a formulation that maximizes the headroom for each commodity will often prefer high stretch even when traffic uncertainty is low, and the solve time was too long even for small fabric size.

C GRAVITY TRAFFIC MODEL VALIDATION AND NETWORK THROUGHPUT

If communications between machines are uniformly random, then the aggregate inter-block traffic follows the gravity model. We verify that inter-block traffic can be approximated by the gravity model, by comparing the inter-block traffic demand in the measured traffic matrix D and the traffic matrix D' generated based on the gravity model. Recall that $D'_{ij} = E_i \cdot I_j / L$, where E_i is the egress traffic demand at block i and I_j is the ingress traffic demand at block j. In Fig. 16, we compare D_{ij} and D'_{ij}, normalized by the largest entry in D and D', respectively, and the RMSE of demand estimation is 0.05. The measured traffic matrices $\{D\}$ include 100 30s-granularity traffic matrices for each of ten fabrics.

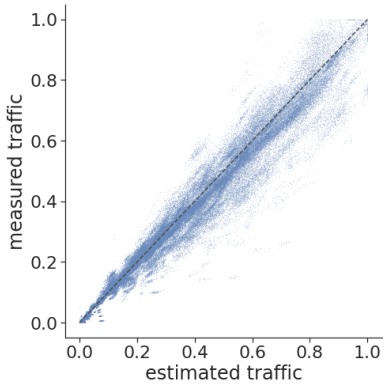

Figure 16: Each point represents the estimated traffic demand (x-axis) and the measured demand (y-axis) between a pair of blocks. The dashed line corresponds to perfect estimation.

We prove that a uniform direct connect topology can support the same throughput as a Clos topology if all blocks have the same speed and radix and the traffic matrix follows the gravity model and is symmetric. We first prove that, if a

network supports a symmetric gravity-model traffic matrix, then it also supports the traffic if the aggregate traffic at a node decreases. Notice that commodities not from or to node u increase as the aggregate traffic at u decreases under the gravity model.

Lemma 1. *If a network $G(V, E)$ can support a gravity-model traffic matrix where block egress and ingress aggregate traffic demands are the same and given by $\{D_i, \forall i \in V\}$, then $G(V, E)$ can support the demand after the aggregate demand at one node decreases, i.e., the block aggregate demands are given by $\{D_i, \forall i \in V \setminus u\} \bigcup \{D'_u : D'_u < D_u\}$.*

Proof. Given block aggregate traffic demands $\{D_i, \forall i \in V\}$, the traffic demand from i to j is $D_{ij} = D_i D_j / \sum_{k \in V} D_k$ based on the gravity model. If the aggregate demand at u decreases from D_u to D'_u, the demand from i to u reduces by

$$c_{iv} = \frac{D_i D_v}{\sum_{k \in V} D_k} - \frac{D_i D'_v}{\sum_{k \in V} D_k + D'_v - D_v}.$$

The demand from i to j increases by

$$b_{ij} = \frac{D_i D_j}{\sum_{k \in V} D_k + D'_v - D_v} - \frac{D_i D_j}{\sum_{k \in V} D_k}.$$

If $c_{iv} > \sum_{j \in V \setminus \{i,v\}} b_{ij}$, then the increased demands from i to all other nodes except v can transit through v, since the reduced demand from i to v frees up sufficient capacity. The remaining work is to prove that $c_{iv} > \sum_{j \in V \setminus \{i,v\}} b_{ij}$ holds for all $i \in V \setminus \{v\}$.

$$\begin{aligned}
& c_{iv} - \sum_{j \in V \setminus \{i,v\}} b_{ij} \\
&= \frac{D_i D_v}{\sum_{k \in V} D_k} - \frac{D_i D'_v}{\sum_{k \in V} D_k + D'_v - D_v} \\
&- \sum_{j \in V \setminus \{i,v\}} \Big(\frac{D_i D_j}{\sum_{k \in V} D_k + D'_v - D_v} - \frac{D_i D_j}{\sum_{k \in V} D_k} \Big) \\
&= \frac{D_i}{\sum_{k \in V} D_k} \Big(D_v + \sum_{j \in V \setminus \{i,v\}} D_j \Big) \\
&- \frac{D_i}{\sum_{k \in V} D_k + D'_v - D_v} \Big(D'_v + \sum_{j \in V \setminus \{i,v\}} D_j \Big) \\
&= \alpha \Big[\sum_{j \in V \setminus \{i\}} D_j (\sum_{k \in V} D_k + D'_v - D_v) \\
&- (\sum_{k \in V} D_k)(D'_v + \sum_{j \in V \setminus \{i,v\}} D_j) \Big] \\
&= \alpha D_i (D_v - D'_v) \\
&\geq 0,
\end{aligned}$$

where

$$\alpha = \frac{D_i}{(\sum_{k \in V} D_k)(\sum_{k \in V} D_k + D'_v - D_v)}.$$

□

Consider a set of symmetric traffic matrices $\mathcal{D} = \{D(t)\}$, where $D(t)$ represents the traffic demands between nodes at time t and follows the gravity model. Let $D_i(t)$ denote the aggregate traffic at node i at time t. Let $D_i = \max_t D_i(t)$. We prove that a static mesh topology is able to support all traffic using dynamic traffic-aware routing.

Theorem 2. *A static mesh topology G can support all traffic matrices in $\mathcal{D}$. In G, the link capacity between i and j is given by $u_{ij} = D_i D_j / \sum_{k \in V} D_k$.*

Proof. Consider $D^{\max}$ where all nodes have the maximum aggregate traffic demand $D_i, \forall i \in V$. A mesh topology G where link capacity $u_{ij} = D_i D_j / \sum_{k \in V} D_k, \forall i, j \in V$ supports all commodity traffic over direct paths. We prove that G is able to support $D(t), \forall t$ by induction.

For time t, consider traffic matrix $D^n(t)$ that follows the gravity model and $D^n_x(t) = D_x(t), x = 1, 2, \ldots, n$ and $D^n_x(t) = D_x, x = n+1, n+2, \ldots, |V|$. We prove that G is able to support $D^n(t), \forall n \in 1, 2, \ldots, |V|$ by induction. By Lemma 1, if G supports $D^n(t)$, then G supports $D^{n+1}(t)$, since the aggregate traffic demand at a single node reduces from D_{n+1} to $D_{n+1}(t)$. Given that G supports $D^0(t) = D^{\max}$, G supports $D^n(t), n = 1, 2, \ldots, |V|$, including $D(t)$. □

The total capacity of links adjacent to i is

$$\sum_j u_{ij} = \sum_j (D_i D_j / \sum_{k \in V} D_k) = D_i,$$

which is the maximum aggregate traffic at node i. In other words, a static mesh topology supports all symmetric gravity-model traffic matrices if all node aggregate traffic demands do not exceeds their capacities. To compare, a Clos topology supports all traffic matrices if all node aggregate traffic demands do not exceeds their capacities.

In the special case where all blocks are identical, a uniform mesh topology supports a uniform traffic matrix where the aggregate traffic of each node equals its capacity. By Theorem 2, a uniform mesh topology supports all traffic if all node aggregate traffic demands do not exceed their capacities and the traffic matrix is symmetric and follows the gravity model.

D SIMULATION METHODOLOGY

We rely on simulation to guide our traffic and topology engineering designs, reproduce production network state for diagnosis and debugging, and run what-if analysis for production changes. The simulation infrastructure is important since 1) building physical testbeds to match the scale and diversity of our fabrics and workloads is impractical, 2) exploring different designs or answering what-if questions directly in production is risky and slow, 3) network state is too complex to model mathematically.

To make simulation tractable we make the following simplifications:

- *Network topology.* We abstract the Jupiter topology to a block-level simple graph with each vertex representing a

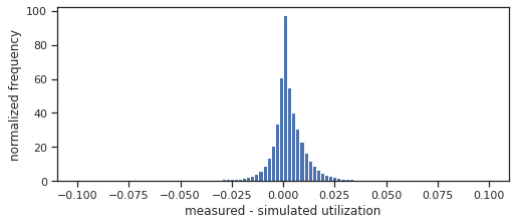

Figure 17: Histogram of the error between measured link utilization and simulated link utilization.

block and each edge representing all links spanning the two blocks. Each block is thus reduced to an abstract switch with 256 or 512 ports. The traffic on an edge is assumed to be perfectly load-balanced across the edge's constituent links (even when those links span different source and destination physical switches).

- *Traffic.* We simulate the network over the 30s-granularity traffic trace. As mentioned before, a traffic matrix entry captures the number of bytes sent from the source block to the destination block over a 30s interval. This traffic matrix is considered the offered load at the time.
- *Ideal load balance and steady state assumptions.* We assume that the offered load is split over multiple paths exactly to the WCMP weights. We thus omit the errors from 1) different flow sizes, 2) imperfect hashing, and 3) WCMP weight reduction [50]. While we run the traffic prediction, routing optimization, and topology optimization loops exactly as configured in production, we do not simulate the transient state when routes are being programmed in the network (we do simulate topology transition as that takes longer). In other words, we assume zero delay changing from one forwarding state to the next.

Despite these simplifications, the simulated link load matches the measured link load well. The Root Mean Square Error (RMSE) between simulated and measured link utilization is less than 0.02. This precision is more than sufficient for the aforementioned use cases. By these simplifications, we can simulate each traffic matrix independently and in parallel, which allows us to simulate the entire fleet over multiple months in a few hours of simulation time.

Fig. 17 shows an error histogram of the measured link utilization vs. the simulated link utilization. The samples consist of over one million data points, corresponding to links from six fabrics across different times spanning a month. The errors are concentrated around zero and have small magnitude, which demonstrates the accuracy of the simulation.

E OPERATIONALIZING FABRIC REWIRING

We describe the automated workflows that operationalize the high level approach to fabric rewiring laid out in §5. We also highlight the uses cases that require rewiring of the fiber strands at the front of the OCS.

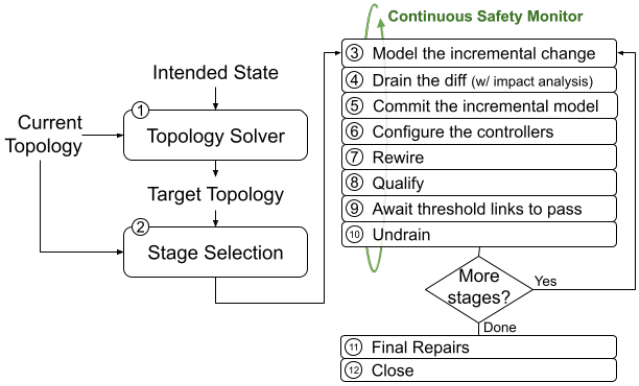

Figure 18: Workflow for topology changes.

E.1 Rewiring workflow

Fig. 18 shows the workflow used for achieving the rewiring process. In Step ① a solver uses the intended fabric state (such as the set of blocks, their platform type, radix, expressed in a proprietary intent expression language) and traffic matrices (for topology engineering §4.5) to produce a target topology that meets the physical constraints (e.g., radix of the DCNI and aggregation block switches) and logical objectives such as balancing the links across failure domains and desired pair-wise block connectivity. The solver also uses the current topology[2] to minimize the diff while achieving the intended state [49].

Depending on the scale of the fabric, change in the intended state, and the feasibility of minimization of diff, the difference between current and target topology can vary from a few hundred links to tens of thousands of links.

Step ②, stage selection, identifies the number of increments needed to safely transition to the target topology by subtracting progressively smaller divisions (1, 1/2, 1/4, 1/8, etc.) of the difference from the current topology, and simulate routing and utilization levels based on recent traffic levels on the residual network to assess whether the network will be able to maintain SLOs for the various traffic classes. We align the divisions with DCNI's domains and their sub divisions at the level of racks – this provides natural alignment with the failure domains, reducing risk.

The workflow executes a set of steps for each increment, starting with modeling the post-increment topology ③ and hitlessly draining the traffic from the links[3] in the diff from the current topology ④ – we perform additional safety checks using analysis similar to the stage selection step to ensure that the post-drain network can carry the traffic while meeting the SLOs. The validation and draining of the links is critical for loss-free reconfiguration. Once the

[2]The current and target topology is represented in an expressive machine readable format (e.g., [28, 39]) to facilitate reasoning about the topology in the later stages of the workflow, as well as to configure and manage the resulting fabric.

[3]Hitless draining is an SDN function that programs alternative paths before atomically diverting packets away from the affected network element.

affected links are safely drained, we commit the modeled post-increment topology ⑤ and dispatch the configuration to the SDN controllers ⑥. Using the post-increment topology, the controllers appropriately configure link speeds and dispatch LLDP packets. This helps detect any miscabling during the rewiring steps ⑦.

The cross-connection rewiring is done using a software configuration to the SDN controllers in the DCNI domains (§4.2), which then program the OCS switches. As new end-to-end links are formed, the workflow performs link qualification including ascertaining correct logical adjacency, optical levels, and link quality using bit-error rate tests ⑧. The workflow requires that 90+% of the links of a stage are successfully qualified before proceeding to the next step[4]. We undrain the links as they are formed and move to the next increment once a threshold number of links of successfully formed and undrained. As a performance optimization, steps related to modeling ③ and drain impact analysis ④ for the next stage can proceed in parallel to steps ⑥ - ⑨ of the previous stage. Once all the stages are completed, we await for final repairs that fix any remaining broken links left over from the iterations, and then conclude the operation.

All workflow steps are shadowed by a continuous loop monitoring the traffic, fabric, Orion controller health and other "big-red-button" signals. Upon detecting anomalies, it can preempt the ongoing step, and even initiate an automated rollback. We also enforce pacing of operations across the failure domains within the fabric, and across the fleet - this ensures that all the telemetry has had a chance to catch up to the change and the safety loop can intervene preventing a cascading failure. The network management API systems layer (Fig. 7) creates a separation between the workflows and the SDN control and dataplane and enforces the safety constraints. The network management services are sharded for zonal and regional failure domains, and use progressive rollouts and shadow launch techniques [7] to prevent bugs and correlated failures in this critical layer.

In Table 2, we include steps ①-⑤ as workflow overhead, whereas the other steps are considered part of the core rewiring steps. The end-to-end speedup in rewiring excludes the final repair steps ⑪.

E.2 Front panel rewiring

The following operations require connectivity changes at the front panel with the assistance of operations staff on the datacenter floor.

- *Addition of new blocks or radix upgrades.* Jupiter reserves space and power for future blocks, which additionally allows pre-installation of the the fiber from reserved spots to the DCNI switches. However, to minimize the inventory, the aggregation blocks (and optics) are deployed later when they are needed. The newly added links are physically connected to the DCNI switches prior to the logical rewiring steps (shown as "pre-deployed" in Fig 10(b)-bottom). Removal of blocks follows the reverse order – blocks are first disconnected from the fabric using logical rewiring, and then physically disconnected from the DCNI layer.
- *DCNI expansion.* The DCNI layer follows the incremental deployment model of Jupiter aggregation blocks (§ 3.1). Adding new OCS chassis requires re-balancing the links from the packet switches to the entire DCNI layer, which naturally requires rewiring at the front panel.
- *Repairs and Technology Refresh.* Repairs of miscabled links, bad optics, fiber strands, OCS ports, and chassis all require changes at the front panel. These changes are often made in-place, i.e., the point of incidence of links on the DCNI chassis does not change.

It is desirable to maximize the spatial locality of incremental rewiring steps for manual operations (e.g., all front-panel rewiring). The locality allows operations staff to complete the work for a specific step without having to move vast distances across the datacenter, maximizing productivity and reducing the chances of mistakes. This locality is achieved by sequencing the workflow to process OCS chassis that are physically adjacent to each other. When the the rewiring step (⑦ in Fig. 18) involves manual work, the workflows offer a web-based user interface that provides real time assistance and diagnostics to the operations staff. This further helps with productivity and reduces the chances of errors.

F HARDWARE COMPONENTS

We developed three critical hardware components, the OCS, WDM transceivers, and optical circulators, to realize a cost-effective, large-scale optical switching layer/DCNI [41].

F.1 Palomar OCS

For the first several years of deployment, a vendor-based OCS solution was adopted in the DCNI layer. However, due to the difficulties in maintaining reliability and quality of this solution at scale, the decision was made to internally develop an OCS system.

Figure 19 shows the high-level optical design and operation principles of the Palomar OCS. The input/output optical signals enter the optical core through two-dimensional (2D) fiber collimator arrays. Each collimator array consists of an NxN fiber array and 2D lens array. The optical core consists of two sets of 2D MEMS mirror arrays. Each inband optical signal traverses through a port in each collimator array and two MEMS mirrors, as indicated by the green line of Figure 19. Mirrors are actuated and tilted to switch the signal to a corresponding input/output collimator fiber. The entire end-to-end optical path is broadband and reciprocal, for data

[4]Links may fail qualification tests due to incorrect cabling, unseated plugs, dust, or deterioration of cables or optics that fail the more stringent tests performed during the tests. We prefer to perform the repair as part of the workflow because it is necessary to return the capacity to the fabric to proceed to the next step, and the datacenter technicians are on-hand during these operations making it more convenient from an operations perspective.

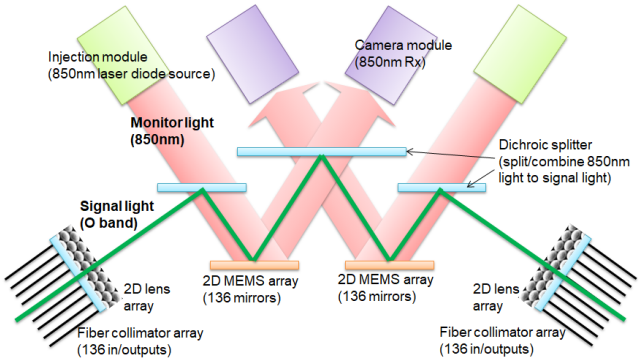

Figure 19: Illustration of design and optical path of Palomar OCS optical core. The inband optical signal path is indicated by the green line. Superposed with the inband signal path, a monitoring channel at the 850nm wavelength (red arrows) assists with tuning of the mirrors.

rate agnostic and bidirectional communication across the OCS. Superposed with the inband signal path, a monitoring channel (red bars, Figure 19) assists with tuning of the mirrors.

Each MEMS array is injected with this 850nm light. The reflected monitor signals are then received at a matching camera module. A servo utilizes the camera image feedback to optimize MEMS actuation for minimum loss of the optical signal path. A pair of injection/camera modules controls each 2D MEMS array. By implementing mirror controls based on image processing of a single camera image per MEMS array, the control scheme is significantly simplified in comparison to conventional approaches which require individual monitoring and/or photodetector hardware per mirror. This design choice was critical to realizing a low-cost, manufacturable OCS solution. The above design yields a non-blocking, 136x136 OCS with bijective, any-to-any input to output port connectivity.

Figure 20 shows some representative insertion loss and return loss data for the Palomar OCS. Insertion losses are typically <2dB for all NxN permutations of connectivity. The tail in the distributions is nominally due to splice and connector loss variation. Return loss is typically -46dB, with a nominal spec of <-38dB. The stringent return loss requirement stems from the use of bidirectional communication along each optical path, as any single reflection superposes directly on top of the main optical signal to degrade signal-to-noise ratio. Multilevel PAM-based communication further increases sensitivity to these reflections. not clear if figure 20 is essential to the section, since the text describes all the key properties well. this is actual data, so would prefer to keep. cut if space does not allow

F.2 WDM Optical Transceivers

Figure 21 shows the corresponding WDM single mode interconnect roadmap developed by Google over the past decade

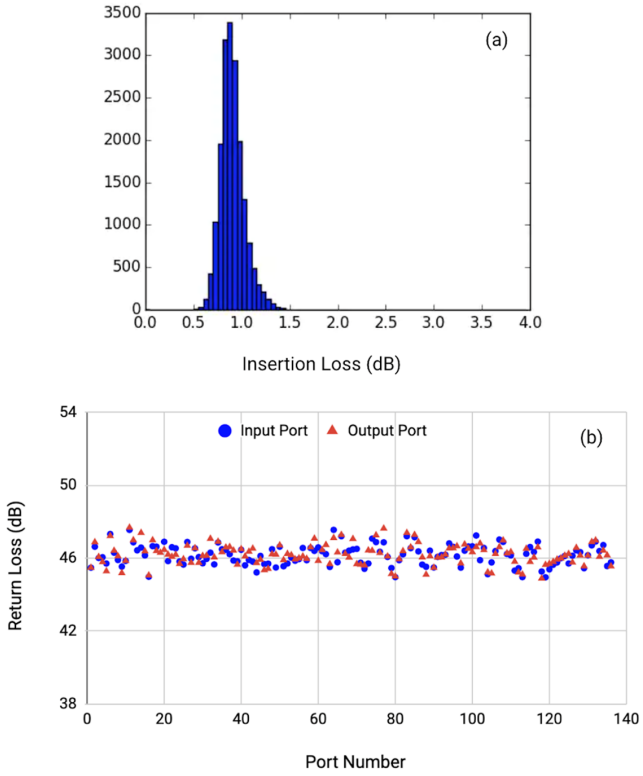

Figure 20: a) Representative Palomar OCS insertion loss histogram for 136x136 (18,496) cross-connections. b) Return loss versus port number for 136 input/output ports, with the OCS configured for 1:1 connections (input 1-output 1, input 2-output 2, ...).

[23]. To support the higher loss budget due to the OCS and circulators, transceiver design emphasized low optical component and packaging losses. Thereafter, a variety of technology directions and trends are notable. First, in order to support direct interop between various generation aggregation blocks, it was critical to maintain the same CWDM4 wavelength grid. This necessitated development of key, critical component technologies starting at the 25G per optical lane generation well in advance, most importantly uncooled CWDM directly modulated lasers (DMLs). This eventually led to after-the-fact creation of the CWDM4 MSA.

Second, in order to keep reducing the cost, power, and density per Gb/s and enable the use of the switch ASIC/Moore's Law improvements [23], continuous speed up of each lane was essential. This was achieved through development of a variety of key technologies: optical, electrical, and signal processing. For the optics, we worked with the industry to develop faster optical components (lasers/photo-detectors), migrating from DMLs to more recently the use of externally modulated lasers (EMLs), due to higher speed and extinction

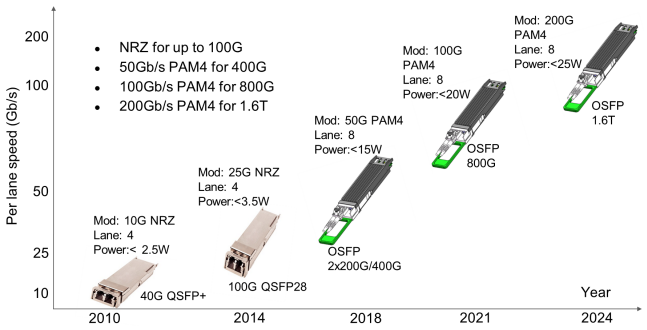

Figure 21: WDM single mode interconnect review and roadmap.

ratio requirements (critical for mitigating multi-path interference (MPI) effects enhanced by bidirectional communication). For high-speed IC/electrical technologies, we migrated from analog-based clock-and-data recovery (CDR) solutions to digital signal processing (DSP)-based ASICs. The DSP ASIC provided a more robust, scalable solution by relaxing the requirement on optical and analog electrical components at the expense of increased power consumption and latency. In addition, we leveraged the new found digital capabilities to develop algorithms that mitigate MPI impairments inherent in bidirectional links as well as forward error correction (FEC) techniques [42] in order to support the higher link budgets needed for the OCS-based links.

One additional and crucial element of the interconnect development was the requirement for backward compatibility of the optical transceiver technology. In addition to wavelength grid compatibility, this meant the optical performance specifications and high speed data path needed to support various line rates. The current generation transceiver must also support a superset of all transmitter and receiver dynamic ranges of the previous generation transceivers.

F.3 Optical Circulators

As stated earlier, the optical circulator enables bidirectional operation of DCNI links to halve the number of required OCS ports and fibers. The optical circulator is a three-port non-reciprocal device that has a cyclic connectivity. Input into port 1 is directed to port 2, input into port 2 is directed to port 3. This functionality can be realized through the use of birefringent crystals, magneto-optical Faraday rotators (typically made of Garnet), and polarizers. Figure 22 illustrates an example implementation. Although the optical signals running in opposing directions do superpose on top of each other along the fiber and OCS, they do not directly interact with each other due to the bosonic nature of photons. Similar to the OCS, the broadband nature of the circulator allows its reuse across multiple generations of CWDM4-based optical transceiver technologies, with costs amortized accordingly.

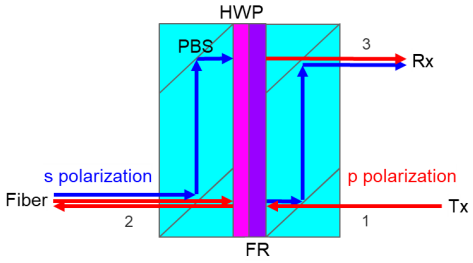

Figure 22: Example integrated circulator implementation. Blue lines indicate s-polarization, red lines indicate p-polarization. PBS - polarizing beam splitter, FR - faraday rotator, HWP - half wave plate.