B4: Experience with a Globally-Deployed Software Defined WAN

Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh,
Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla,
Urs Hölzle, Stephen Stuart and Amin Vahdat
Google, Inc.
b4-sigcomm@google.com

ABSTRACT

We present the design, implementation, and evaluation of B4, a private WAN connecting Google's data centers across the planet. B4 has a number of unique characteristics: i) massive bandwidth requirements deployed to a modest number of sites, ii) elastic traffic demand that seeks to maximize average bandwidth, and iii) full control over the edge servers and network, which enables rate limiting and demand measurement at the edge. These characteristics led to a Software Defined Networking architecture using OpenFlow to control relatively simple switches built from merchant silicon. B4's centralized traffic engineering service drives links to near 100% utilization, while splitting application flows among multiple paths to balance capacity against application priority/demands. We describe experience with three years of B4 production deployment, lessons learned, and areas for future work.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Routing Protocols

Keywords

Centralized Traffic Engineering; Wide-Area Networks; Software-Defined Networking; Routing; OpenFlow

1. INTRODUCTION

Modern wide area networks (WANs) are critical to Internet performance and reliability, delivering terabits/sec of aggregate bandwidth across thousands of individual links. Because individual WAN links are expensive and because WAN packet loss is typically thought unacceptable, WAN routers consist of high-end, specialized equipment that place a premium on high availability. Finally, WANs typically treat all bits the same. While this has many benefits, when the inevitable failure does take place, all applications are typically treated equally, despite their highly variable sensitivity to available capacity.

Given these considerations, WAN links are typically provisioned to 30-40% average utilization. This allows the network service provider to mask virtually all link or router failures from clients. Such overprovisioning delivers admirable reliability at the very real costs of 2-3x bandwidth over-provisioning and high-end routing gear.

We were faced with these overheads for building a WAN connecting multiple data centers with substantial bandwidth requirements. However, Google's data center WAN exhibits a number of unique characteristics. First, we control the applications, servers, and the LANs all the way to the edge of the network. Second, our most bandwidth-intensive applications perform large-scale data copies from one site to another. These applications benefit most from high levels of average bandwidth and can adapt their transmission rate based on available capacity. They could similarly defer to higher priority interactive applications during periods of failure or resource constraint. Third, we anticipated no more than a few dozen data center deployments, making central control of bandwidth feasible.

We exploited these properties to adopt a software defined networking (SDN) architecture for our data center WAN interconnect. We were most motivated by deploying routing and traffic engineering protocols customized to our unique requirements. Our design centers around: i) accepting failures as inevitable and common events, whose effects should be exposed to end applications, and ii) switch hardware that exports a simple interface to program forwarding table entries under central control. Network protocols could then run on servers housing a variety of standard and custom protocols. Our hope was that deploying novel routing, scheduling, monitoring, and management functionality and protocols would be both simpler and result in a more efficient network.

We present our experience deploying Google's WAN, B4, using Software Defined Networking (SDN) principles and OpenFlow [31] to manage individual switches. In particular, we discuss how we simultaneously support standard routing protocols and centralized Traffic Engineering (TE) as our first SDN application. With TE, we: i) leverage control at our network edge to adjudicate among competing demands during resource constraint, ii) use multipath forwarding/tunneling to leverage available network capacity according to application priority, and iii) dynamically reallocate bandwidth in the face of link/switch failures or shifting application demands. These features allow many B4 links to run at near 100% utilization and all links to average 70% utilization over long time periods, corresponding to 2-3x efficiency improvements relative to standard practice.

B4 has been in deployment for three years, now carries more traffic than Google's public facing WAN, and has a higher growth rate. It is among the first and largest SDN/OpenFlow deployments. B4 scales to meet application bandwidth demands more efficiently than would otherwise be possible, supports rapid deployment and iteration of novel control functionality such as TE, and enables tight integration with end applications for adaptive behavior in response to failures or changing communication patterns. SDN is of course

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
Copyright 2013 ACM 978-1-4503-2056-6/13/08 ...$15.00.

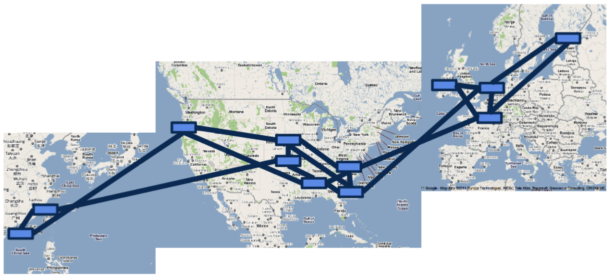


Figure 1: B4 worldwide deployment (2011).

not a panacea; we summarize our experience with a large-scale B4 outage, pointing to challenges in both SDN and large-scale network management. While our approach does not generalize to all WANs or SDNs, we hope that our experience will inform future design in both domains.

2. BACKGROUND

Before describing the architecture of our software-defined WAN, we provide an overview of our deployment environment and target applications. Google's WAN is among the largest in the Internet, delivering a range of search, video, cloud computing, and enterprise applications to users across the planet. These services run across a combination of data centers spread across the world, and edge deployments for cacheable content.

Architecturally, we operate two distinct WANs. Our user-facing network peers with and exchanges traffic with other Internet domains. End user requests and responses are delivered to our data centers and edge caches across this network. The second network, B4, provides connectivity among data centers (see Fig. 1), e.g., for asynchronous data copies, index pushes for interactive serving systems, and end user data replication for availability. Well over 90% of internal application traffic runs across this network.

We maintain two separate networks because they have different requirements. For example, our user-facing networking connects with a range of gear and providers, and hence must support a wide range of protocols. Further, its physical topology will necessarily be more dense than a network connecting a modest number of data centers. Finally, in delivering content to end users, it must support the highest levels of availability.

Thousands of individual applications run across B4; here, we categorize them into three classes: i) user data copies (e.g., email, documents, audio/video files) to remote data centers for availability/-durability, ii) remote storage access for computation over inherently distributed data sources, and iii) large-scale data push synchronizing state across multiple data centers. These three traffic classes are ordered in increasing volume, decreasing latency sensitivity, and decreasing overall priority. For example, user-data represents the lowest volume on B4, is the most latency sensitive, and is of the highest priority.

The scale of our network deployment strains both the capacity of commodity network hardware and the scalability, fault tolerance, and granularity of control available from network software. Internet bandwidth as a whole continues to grow rapidly [25]. However, our own WAN traffic has been growing at an even faster rate.

Our decision to build B4 around Software Defined Networking and OpenFlow [31] was driven by the observation that we could not achieve the level of scale, fault tolerance, cost efficiency, and control required for our network using traditional WAN architectures. A number of B4's characteristics led to our design approach:

- *Elastic bandwidth demands:* The majority of our data center traffic involves synchronizing large data sets across sites. These applications benefit from as much bandwidth as they can get but can tolerate periodic failures with temporary bandwidth reductions.
- *Moderate number of sites:* While B4 must scale among multiple dimensions, targeting our data center deployments meant that the total number of WAN sites would be a few dozen.
- *End application control:* We control both the applications and the site networks connected to B4. Hence, we can enforce relative application priorities and control bursts at the network edge, rather than through overprovisioning or complex functionality in B4.
- *Cost sensitivity:* B4's capacity targets and growth rate led to unsustainable cost projections. The traditional approach of provisioning WAN links at 30-40% (or 2-3x the cost of a fully-utilized WAN) to protect against failures and packet loss, combined with prevailing per-port router cost, would make our network prohibitively expensive.

These considerations led to particular design decisions for B4, which we summarize in Table 1. In particular, SDN gives us a dedicated, software-based control plane running on commodity servers, and the opportunity to reason about global state, yielding vastly simplified coordination and orchestration for both planned and unplanned network changes. SDN also allows us to leverage the raw speed of commodity servers; latest-generation servers are much faster than the embedded-class processor in most switches, and we can upgrade servers independently from the switch hardware. OpenFlow gives us an early investment in an SDN ecosystem that can leverage a variety of switch/data plane elements. Critically, SDN/OpenFlow decouples software and hardware evolution: control plane software becomes simpler and evolves more quickly; data plane hardware evolves based on programmability and performance.

We had several additional motivations for our software defined architecture, including: i) rapid iteration on novel protocols, ii) simplified testing environments (e.g., we emulate our entire software stack running across the WAN in a local cluster), iii) improved capacity planning available from simulating a deterministic central TE server rather than trying to capture the asynchronous routing behavior of distributed protocols, and iv) simplified management through a fabric-centric rather than router-centric WAN view. However, we leave a description of these aspects to separate work.

3. DESIGN

In this section, we describe the details of our Software Defined WAN architecture.

3.1 Overview

Our SDN architecture can be logically viewed in three layers, depicted in Fig. 2. B4 serves multiple WAN sites, each with a number of server clusters. Within each B4 site, the *switch hardware layer* primarily forwards traffic and does not run complex control software, and the *site controller layer* consists of Network Control Servers (NCS) hosting both OpenFlow controllers (OFC) and Network Control Applications (NCAs).

These servers enable distributed routing and central traffic engineering as a routing overlay. OFCs maintain network state based on NCA directives and switch events and instruct switches to set forwarding table entries based on this changing network state. For fault tolerance of individual servers and control processes, a per-site in-

Design Decision	Rationale/Benefits	Challenges
B4 routers built from merchant switch silicon	B4 apps are willing to trade more average bandwidth for fault tolerance. Edge application control limits need for large buffers. Limited number of B4 sites means large forwarding tables are not required. Relatively low router cost allows us to scale network capacity.	Sacrifice hardware fault tolerance, deep buffering, and support for large routing tables.
Drive links to 100% utilization	Allows efficient use of expensive long haul transport. Many applications willing to trade higher average bandwidth for predictability. Largest bandwidth consumers adapt dynamically to available bandwidth.	Packet loss becomes inevitable with substantial capacity loss during link/switch failure.
Centralized traffic engineering	Use multipath forwarding to balance application demands across available capacity in response to failures and changing application demands. Leverage application classification and priority for scheduling in cooperation with edge rate limiting. Traffic engineering with traditional distributed routing protocols (e.g. link-state) is known to be sub-optimal [17, 16] except in special cases [39]. Faster, deterministic global convergence for failures.	No existing protocols for functionality. Requires knowledge about site to site demand and importance.
Separate hardware from software	Customize routing and monitoring protocols to B4 requirements. Rapid iteration on software protocols. Easier to protect against common case software failures through external replication. Agnostic to range of hardware deployments exporting the same programming interface.	Previously untested development model. Breaks fate sharing between hardware and software.

Table 1: Summary of design decisions in B4.

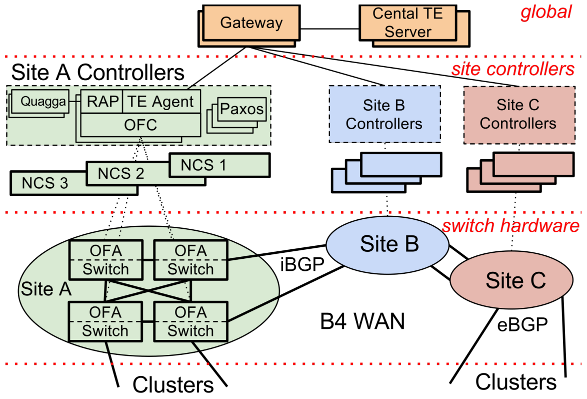


Figure 2: B4 architecture overview.

stance of Paxos [9] elects one of multiple available software replicas (placed on different physical servers) as the primary instance.

The *global layer* consists of logically centralized applications (e.g. an SDN Gateway and a central TE server) that enable the central control of the entire network via the site-level NCAs. The SDN Gateway abstracts details of OpenFlow and switch hardware from the central TE server. We replicate global layer applications across multiple WAN sites with separate leader election to set the primary.

Each server cluster in our network is a logical "Autonomous System" (AS) with a set of IP prefixes. Each cluster contains a set of BGP routers (not shown in Fig. 2) that peer with B4 switches at each WAN site. Even before introducing SDN, we ran B4 as a single AS providing transit among clusters running traditional BGP/ISIS network protocols. We chose BGP because of its isolation properties between domains and operator familiarity with the protocol. The SDN-based B4 then had to support existing distributed routing protocols, both for interoperability with our non-SDN WAN implementation, and to enable a gradual rollout.

We considered a number of options for integrating existing routing protocols with centralized traffic engineering. In an aggressive approach, we would have built one integrated, centralized service combining routing (e.g., ISIS functionality) and traffic engineering. We instead chose to deploy routing and traffic engineering as independent services, with the standard routing service deployed initially and central TE subsequently deployed as an overlay. This separation delivers a number of benefits. It allowed us to focus initial work on building SDN infrastructure, e.g., the OFC and agent, routing, etc. Moreover, since we initially deployed our network with no new externally visible functionality such as TE, it gave time to develop and debug the SDN architecture before trying to implement new features such as TE.

Perhaps most importantly, we layered traffic engineering on top of baseline routing protocols using prioritized switch forwarding table entries (§ 5). This isolation gave our network a "big red button"; faced with any critical issues in traffic engineering, we could disable the service and fall back to shortest path forwarding. This fault recovery mechanism has proven invaluable (§ 6).

Each B4 site consists of multiple switches with potentially hundreds of individual ports linking to remote sites. To scale, the TE abstracts each site into a single node with a single edge of given capacity to each remote site. To achieve this *topology abstraction*, all traffic crossing a site-to-site edge must be evenly distributed across all its constituent links. B4 routers employ a custom variant of ECMP hashing [37] to achieve the necessary load balancing.

In the rest of this section, we describe how we integrate existing routing protocols running on separate control servers with OpenFlow-enabled hardware switches. § 4 then describes how we layer TE on top of this baseline routing implementation.

3.2 Switch Design

Conventional wisdom dictates that wide area routing equipment must have deep buffers, very large forwarding tables, and hardware support for high availability. All of this functionality adds to hardware cost and complexity. We posited that with careful endpoint management, we could adjust transmission rates to avoid the need for deep buffers while avoiding expensive packet drops. Further, our switches run across a relatively small set of data centers, so we did not require large forwarding tables. Finally, we found that switch failures typically result from software rather than hardware issues. By moving most software functionality off the switch hardware, we can manage software fault tolerance through known techniques widely available for existing distributed systems.

Even so, the main reason we chose to build our own hardware was that no existing platform could support an SDN deployment, i.e., one that could export low-level control over switch forwarding behavior. Any extra costs from using custom switch hardware are more than repaid by the efficiency gains available from supporting novel services such as centralized TE. Given the bandwidth required

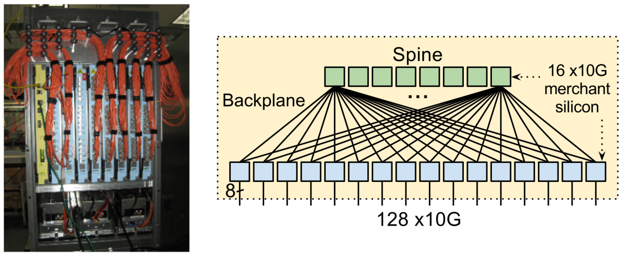


Figure 3: A custom-built switch and its topology.

at individual sites, we needed a high-radix switch; deploying fewer, larger switches yields management and software-scalability benefits.

To scale beyond the capacity available from individual switch chips, we built B4 switches from multiple merchant silicon switch chips in a two-stage Clos topology with a copper backplane [15]. Fig. 3 shows a 128-port 10GE switch built from 24 individual 16x10GE non-blocking switch chips. We configure each ingress chip to bounce incoming packets to the spine layer, unless the destination is on the same ingress chip. The spine chips forward packets to the appropriate output chip depending on the packet's destination.

The switch contains an embedded processor running Linux. Initially, we ran all routing protocols directly on the switch. This allowed us to drop the switch into a range of existing deployments to gain experience with both the hardware and software. Next, we developed an OpenFlow Agent (OFA), a user-level process running on our switch hardware implementing a slightly extended version of the Open Flow protocol to take advantage of the hardware pipeline of our switches. The OFA connects to a remote OFC, accepting OpenFlow (OF) commands and forwarding appropriate packets and link/switch events to the OFC. For example, we configure the hardware switch to forward routing protocol packets to the software path. The OFA receives, e.g., BGP packets and forwards them to the OFC, which in turn delivers them to our BGP stack (§3.4).

The OFA translates OF messages into driver commands to set chip forwarding table entries. There are two main challenges here. First, we must bridge between OpenFlow's architecture-neutral version of forwarding table entries and modern merchant switch silicon's sophisticated packet processing pipeline, which has many linked forwarding tables of various size and semantics. The OFA translates the high level view of forwarding state into an efficient mapping specific to the underlying hardware. Second, the OFA exports an abstraction of a single non-blocking switch with hundreds of 10Gb/s ports. However, the underlying switch consists of multiple physical switch chips, each with individually-managed forwarding table entries.

3.3 Network Control Functionality

Most B4 functionality runs on NCS in the site controller layer colocated with the switch hardware; NCS and switches share a dedicated out-of-band control-plane network.

Paxos handles leader election for all control functionality. Paxos instances at each site perform application-level failure detection among a preconfigured set of available replicas for a given piece of control functionality. When a majority of the Paxos servers detect a failure, they elect a new leader among the remaining set of available servers. Paxos then delivers a callback to the elected leader with a monotonically increasing generation ID. Leaders use this generation ID to unambiguously identify themselves to clients.

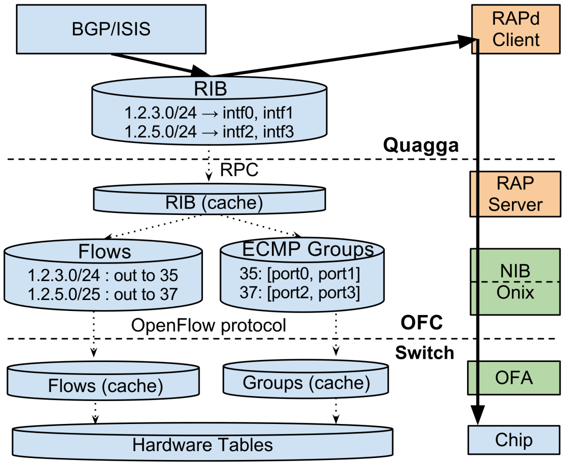


Figure 4: Integrating Routing with OpenFlow Control.

We use a modified version of Onix [26] for OpenFlow Control. From the perspective of this work, the most interesting aspect of the OFC is the Network Information Base (NIB). The NIB contains the current state of the network with respect to topology, trunk configurations, and link status (operational, drained, etc.). OFC replicas are warm standbys. While OFAs maintain active connections to multiple OFCs, communication is active to only one OFC at a time and only a single OFC maintains state for a given set of switches. Upon startup or new leader election, the OFC reads the expected static state of the network from local configuration, and then synchronizes with individual switches for dynamic network state.

3.4 Routing

One of the main challenges in B4 was integrating OpenFlow-based switch control with existing routing protocols to support hybrid network deployments. To focus on core OpenFlow/SDN functionality, we chose the open source Quagga stack for BGP/ISIS on NCS. We wrote a Routing Application Proxy (RAP) as an SDN application, to provide connectivity between Quagga and OF switches for: (i) BGP/ISIS route updates, (ii) routing-protocol packets flowing between switches and Quagga, and (iii) interface updates from the switches to Quagga.

Fig. 4 depicts this integration in more detail, highlighting the interaction between hardware switches, the OFC, and the control applications. A RAPd process subscribes to updates from Quagga's RIB and proxies any changes to a RAP component running in the OFC via RPC. The RIB maps address prefixes to one or more named hardware interfaces. RAP caches the Quagga RIB and translates RIB entries into NIB entries for use by Onix.

At a high level, RAP translates from RIB entries forming a network-level view of global connectivity to the low-level hardware tables used by the OpenFlow data plane. B4 switches employ ECMP hashing (for topology abstraction) to select an output port among these next hops. Therefore, RAP translates each RIB entry into two OpenFlow tables, a *Flow* table which maps prefixes to entries into a *ECMP Group* table. Multiple flows can share entries in the ECMP Group Table. The ECMP Group table entries identify the next-hop physical interfaces for a set of flow prefixes.

BGP and ISIS sessions run across the data plane using B4 hardware ports. However, Quagga runs on an NCS with no data-plane connectivity. Thus, in addition to route processing, RAP must proxy routing-protocol packets between the Quagga control plane and the

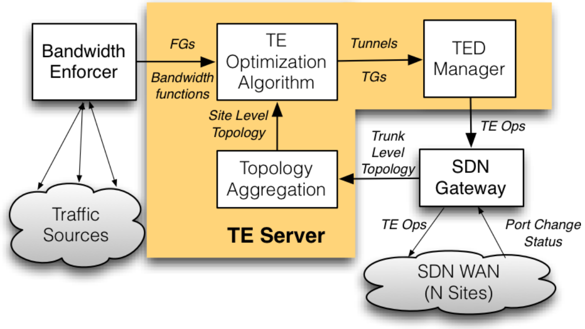


Figure 5: Traffic Engineering Overview.

corresponding switch data plane. We modified Quagga to create `tuntap` interfaces corresponding to each physical switch port it manages. Starting at the NCS kernel, these protocol packets are forwarded through RAPd, the OFC, and the OFA, which finally places the packet on the data plane. We use the reverse path for incoming packets. While this model for transmitting and receiving protocol packets was the most expedient, it is complex and somewhat brittle. Optimizing the path between the switch and the routing application is an important consideration for future work.

Finally, RAP informs Quagga about switch interface and port state changes. Upon detecting a port state change, the switch OFA sends an OpenFlow message to OFC. The OFC then updates its local NIB, which in turn propagates to RAPd. We also modified Quagga to create `netdev` virtual interfaces for each physical switch port. RAPd changes the netdev state for each interface change, which propagates to Quagga for routing protocol updates. Once again, shortening the path between switch interface changes and the consequent protocol processing is part of our ongoing work.

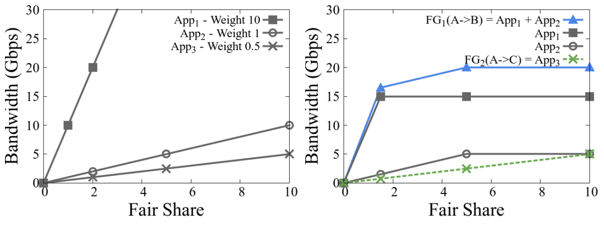
4. TRAFFIC ENGINEERING

The goal of TE is to share bandwidth among competing applications possibly using multiple paths. The objective function of our system is to deliver max-min fair allocation[12] to applications. A max-min fair solution maximizes utilization as long as further gain in utilization is not achieved by penalizing fair share of applications.

4.1 Centralized TE Architecture

Fig. 5 shows an overview of our TE architecture. The *TE Server* operates over the following state:

- The **Network Topology** graph represents sites as vertices and site to site connectivity as edges. The *SDN Gateway* consolidates topology events from multiple sites and individual switches to TE. TE aggregates trunks to compute site-site edges. This abstraction significantly reduces the size of the graph input to the *TE Optimization Algorithm* (§4.3).
- **Flow Group (FG)**: For scalability, TE cannot operate at the granularity of individual applications. Therefore, we aggregate applications to a Flow Group defined as $\{source\ site, dest\ site, QoS\}$ tuple.
- A **Tunnel (T)** represents a site-level path in the network, e.g., a sequence of sites ($A \rightarrow B \rightarrow C$). B4 implements tunnels using IP in IP encapsulation (see §5).
- A **Tunnel Group (TG)** maps FGs to a set of tunnels and corresponding weights. The weight specifies the fraction of FG traffic to be forwarded along each tunnel.



(a) Per-application.

(b) FG-level composition.

Figure 6: Example bandwidth functions.

TE Server outputs the Tunnel Groups and, by reference, Tunnels and Flow Groups to the SDN Gateway. The Gateway forwards these Tunnels and Flow Groups to OFCs that in turn install them in switches using OpenFlow (§5).

4.2 Bandwidth functions

To capture relative priority, we associate a *bandwidth function* with every application (e.g., Fig. 6(a)), effectively a contract between an application and B4. This function specifies the bandwidth allocation to an application given the flow's relative priority on an arbitrary, dimensionless scale, which we call its *fair share*. We derive these functions from administrator-specified static weights (the slope of the function) specifying relative application priority. In this example, App_1, App_2, and App_3 have weights 10, 1, and 0.5, respectively. *Bandwidth functions* are configured, measured and provided to TE via *Bandwidth Enforcer* (see Fig. 5).

Each Flow Group multiplexes multiple application demands from one site to another. Hence, an FG's *bandwidth function* is a piecewise linear additive composition of per-application *bandwidth functions*. The max-min objective function of TE is on this per-FG *fair share* dimension (§4.3.) *Bandwidth Enforcer* also aggregates bandwidth functions across multiple applications.

For example, given the topology of Fig. 7(a), Bandwidth Enforcer measures 15Gbps of demand for App_1 and 5Gbps of demand for App_2 between sites A and B, yielding the composed *bandwidth function* for FG_1 in Fig. 6(b). The *bandwidth function* for FG_2 consists only of 10Gbps of demand for App_3. We flatten the configured per-application *bandwidth functions* at measured demand because allocating that measured demand is equivalent to a FG receiving infinite *fair share*.

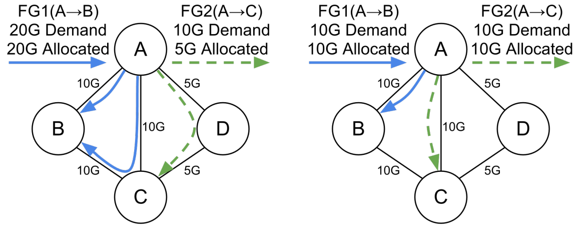
Bandwidth Enforcer also calculates bandwidth limits to be enforced at the edge. Details on Bandwidth Enforcer are beyond the scope of this paper. For simplicity, we do not discuss the QoS aspect of FGs further.

4.3 TE Optimization Algorithm

The LP [13] optimal solution for allocating *fair share* among all FGs is expensive and does not scale well. Hence, we designed an algorithm that achieves similar fairness and at least 99% of the bandwidth utilization with 25x faster performance relative to LP [13] for our deployment.

The TE Optimization Algorithm has two main components: (1) *Tunnel Group Generation* allocates bandwidth to FGs using *bandwidth functions* to prioritize at bottleneck edges, and (2) *Tunnel Group Quantization* changes split ratios in each TG to match the granularity supported by switch hardware tables.

We describe the operation of the algorithm through a concrete example. Fig. 7(a) shows an example topology with four sites. Cost is an abstract quantity attached to an edge which typically represents



(a) (b)

Figure 7: Two examples of TE Allocation with two FGs.

the edge latency. The cost of a tunnel is the sum of cost of its edges. The cost of each edge in Fig. 7(a) is 1 except edge $A \rightarrow D$, which is 10. There are two FGs, $FG_1(A \rightarrow B)$ with demand of 20Gbps and $FG_2(A \rightarrow C)$ with demand of 10Gbps. Fig. 6(b) shows the *bandwidth functions* for these FGs as a function of currently measured demand and configured priorities.

Tunnel Group Generation allocates bandwidth to FGs based on demand and priority. It allocates edge capacity among FGs according to their *bandwidth function* such that all competing FGs on an edge either receive equal *fair share* or fully satisfy their demand. It iterates by finding the bottleneck edge (with minimum *fair share* at its capacity) when filling all FGs together by increasing their *fair share* on their preferred tunnel. A preferred tunnel for a FG is the minimum cost path that does not include a bottleneck edge.

A bottleneck edge is not further used for TG generation. We thus freeze all tunnels that cross it. For all FGs, we move to the next preferred tunnel and continue by increasing *fair share* of FGs and locating the next bottleneck edge. The algorithm terminates when each FG is either satisfied or we cannot find a preferred tunnel for it.

We use the notation T_x^y to refer to the y^{th}-most preferred tunnel for FG_x. In our example, we start by filling both FG_1 and FG_2 on their most preferred tunnels: $T_1^1 = A \rightarrow B$ and $T_2^1 = A \rightarrow C$ respectively. We allocate bandwidth among FGs by giving equal *fair share* to each FG. At a *fair share* of 0.9, FG_1 is allocated 10Gbps and FG_2 is allocated 0.45Gbps according to their *bandwidth functions*. At this point, edge $A \rightarrow B$ becomes full and hence, bottlenecked. This freezes tunnel T_1^1. The algorithm continues allocating bandwidth to FG_1 on its next preferred tunnel $T_1^2 = A \rightarrow C \rightarrow B$. At *fair share* of 3.33, FG_1 receives 8.33Gbps more and FG_2 receives 1.22Gbps more making edge $A \rightarrow C$ the next bottleneck. FG_1 is now forced to its third preferred tunnel $T_1^3 = A \rightarrow D \rightarrow C \rightarrow B$. FG_2 is also forced to its second preferred tunnel $T_2^2 = A \rightarrow D \rightarrow C$. FG_1 receives 1.67Gbps more and becomes fully satisfied. FG_2 receives the remaining 3.33Gbps.

The allocation of FG_2 to its two tunnels is in the ratio 1.67:3.33 ($\doteq$ 0.3:0.7, normalized so that the ratios sum to 1.0) and allocation of FG_1 to its three tunnels is in the ratio 10:8.33:1.67 ($\doteq$ 0.5:0.4:0.1). FG_2 is allocated a *fair share* of 10 while FG_1 is allocated infinite *fair share* as its demand is fully satisfied.

Tunnel Group Quantization adjusts splits to the granularity supported by the underlying hardware, equivalent to solving an integer linear programming problem. Given the complexity of determining the optimal split quantization, we once again use a greedy approach. Our algorithm uses heuristics to maintain fairness and throughput efficiency comparable to the ideal unquantized tunnel groups.

Returning to our example, we split the above allocation in multiples of 0.5. Starting with FG_2, we down-quantize its split ratios to 0.0:0.5. We need to add 0.5 to one of the two tunnels to complete the quantization. Adding 0.5 to T_2^1 reduces the *fair share* for FG_1 below 5, making the solution less max-min fair[12][1]. However, adding 0.5 to T_2^2 fully satisfies FG_1 while maintaining FG_2's *fair share* at 10. Therefore, we set the quantized split ratios for FG_2 to 0.0:1.0. Similarly, we calculate the quantized split ratios for FG_1 to 0.5:0.5:0.0. These TGs are the final output of TE algorithm (Fig. 7(a)). Note how an FG with a higher *bandwidth function* pushes an FG with a lower *bandwidth function* to longer and lower capacity tunnels.

Fig. 7(b) shows the dynamic operation of the TE algorithm. In this example, App_1 demand falls from 15Gbps to 5Gbps and the aggregate demand for FG_1 drops from 20Gbps to 10Gbps, changing the *bandwidth function* and the resulting tunnel allocation.

TE Construct	Switch	OpenFlow Message	Hardware Table
Tunnel	Transit	FLOW_MOD	LPM Table
Tunnel	Transit	GROUP_MOD	Multipath Table
Tunnel	Decap	FLOW_MOD	Decap Tunnel Table
Tunnel Group	Encap	GROUP_MOD	Multipath table, Encap Tunnel table
Flow Group	Encap	FLOW_MOD	ACL Table

Table 2: Mapping TE constructs to hardware via OpenFlow.

5. TE PROTOCOL AND OPENFLOW

We next describe how we convert Tunnel Groups, Tunnels, and Flow Groups to OpenFlow state in a distributed, failure-prone environment.

5.1 TE State and OpenFlow

B4 switches operate in three roles: i) an encapsulating switch initiates tunnels and splits traffic between them, ii) a transit switch forwards packets based on the outer header, and iii) a decapsulating switch terminates tunnels and then forwards packets using regular routes. Table 2 summarizes the mapping of TE constructs to OpenFlow and hardware table entries.

Source site switches implement FGs. A switch maps packets to an FG when their destination IP address matches one of the prefixes associated with the FG. Incoming packets matching an FG are forwarded via the corresponding TG. Each incoming packet hashes to one of the Tunnels associated with the TG in the desired ratio. Each site in the tunnel path maintains per-tunnel forwarding rules. Source site switches encapsulate the packet with an outer IP header whose destination IP address uniquely identifies the tunnel. The outer destination-IP address is a tunnel-ID rather than an actual destination. TE pre-configures tables in encapsulating-site switches to create the correct encapsulation, tables in transit-site switches to properly forward packets based on their tunnel-ID, and descapsulating-site switches to recognize which tunnel-IDs should be terminated. Therefore, installing a tunnel requires configuring switches at multiple sites.

5.2 Example

Fig. 8 shows an example where an encapsulating switch splits flows across two paths based on a hash of the packet header. The switch encapsulates packets with a fixed source IP address and a per-tunnel destination IP address. Half the flows are encapsulated with outer IP src/dest IP addresses `2.0.0.1, 4.0.0.1` and forwarded along the shortest path while the remaining flows are encapsulated with the label `2.0.0.1, 3.0.0.1` and forwarded through a transit site. The destination site switch recognizes that it must decapsulate

[1] S_1 is less max-min fair than S_2 if ordered allocated *fair share* of all FGs in S_1 is lexicographically less than ordered allocated *fair share* of all FGs in S_2

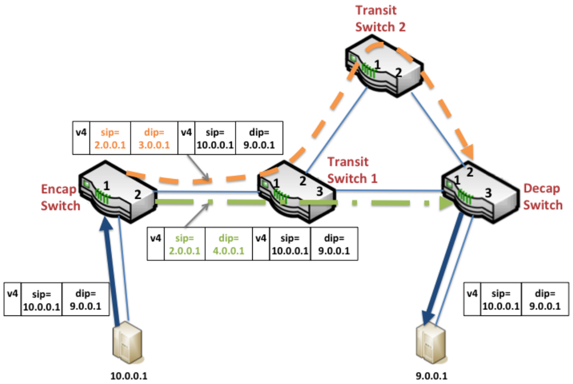
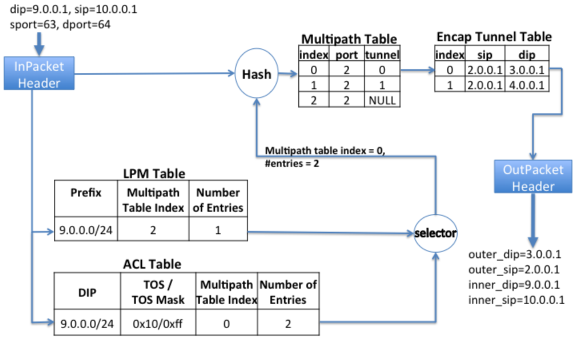


Figure 8: Multipath WAN Forwarding Example.



Multipath Table

index	port	tunnel
0	2	0
1	2	1
2	2	NULL

Encap Tunnel Table

index	sip	dip
0	2.0.0.1	3.0.0.1
1	2.0.0.1	4.0.0.1

LPM Table

Prefix	Multipath Table Index	Number of Entries
9.0.0.0/24	2	1

ACL Table

DIP	TOS / TOS Mask	Multipath Table Index	Number of Entries
9.0.0.0/24	0x10/0xff	0	2

Figure 9: Layering traffic engineering on top of shortest path forwarding in an encap switch.

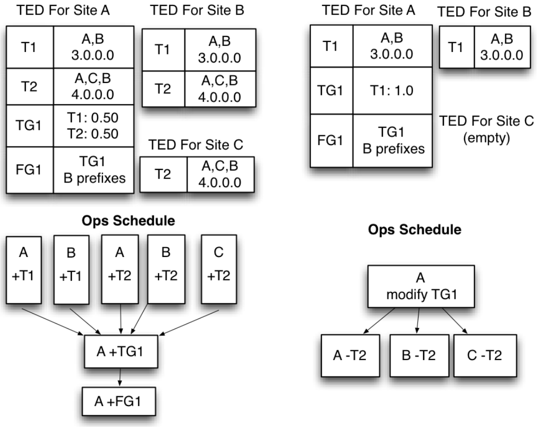
the packet based on a table entry pre-configured by TE. After decapsulation, the switch forwards to the destination based on the inner packet header, using Longest Prefix Match (LPM) entries (from BGP) on the same router.

5.3 Composing routing and TE

B4 supports both shortest-path routing and TE so that it can continue to operate even if TE is disabled. To support the coexistence of the two routing services, we leverage the support for multiple forwarding tables in commodity switch silicon.

Based on the OpenFlow flow-entry priority and the hardware table capability, we map different flows and groups to appropriate hardware tables. Routing/BGP populates the LPM table with appropriate entries, based on the protocol exchange described in §3.4. TE uses the Access Control List (ACL) table to set its desired forwarding behavior. Incoming packets match against both tables in parallel. ACL rules take strict precedence over LPM entries.

In Fig. 9, for example, an incoming packet destined to `9.0.0.1` has entries in both the LPM and ACL tables. The LPM entry indicates that the packet should be forwarded through output port 2 without tunneling. However, the ACL entry takes precedence and indexes into a third table, the Multipath Table, at index 0 with 2 entries. Also in parallel, the switch hashes the packet header contents, modulo the number of entries output by the ACL entry. This implements ECMP hashing [37], distributing flows destined to `9.0.0.0/24` evenly between two tunnels. Both tunnels are forwarded through output port 2, but encapsulated with different src/dest IP addresses, based on the contents of a fourth table, the Encap Tunnel table.



(a) TED For Site A

T1	A,B 3.0.0.0
T2	A,C,B 4.0.0.0
TG1	T1: 0.50 T2: 0.50
FG1	TG1 B prefixes

(a) TED For Site B

T1	A,B 3.0.0.0
T2	A,C,B 4.0.0.0

(a) TED For Site C

T2	A,C,B 4.0.0.0

(b) TED For Site A

T1	A,B 3.0.0.0
TG1	T1: 1.0
FG1	TG1 B prefixes

(b) TED For Site B

T1	A,B 3.0.0.0

(a) (b)

Figure 10: System transition from one path assignment (a) to another (b).

5.4 Coordinating TE State Across Sites

TE server coordinates T/TG/FG rule installation across multiple OFCs. We translate TE optimization output to a per-site Traffic Engineering Database (TED), capturing the state needed to forward packets along multiple paths. Each OFC uses the TED to set the necessary forwarding state at individual switches. This abstraction insulates the TE Server from issues such as hardware table management, hashing, and programming individual switches.

TED maintains a key-value datastore for global Tunnels, Tunnel Groups, and Flow Groups. Fig. 10(a) shows sample TED state corresponding to three of the four sites in Fig. 7(a).

We compute a per-site TED based on the TGs, FGs, and Tunnels output by the TE algorithm. We identify entries requiring modification by diffing the desired TED state with the current state and generate a single TE op for each difference. Hence, by definition, a single TE operation (TE op) can add/delete/modify exactly one TED entry at one OFC. The OFC converts the TE op to flow-programming instructions at all devices in that site. The OFC waits for ACKs from all devices before responding to the TE op. When appropriate, the TE server may issue multiple simultaneous ops to a single site.

5.5 Dependencies and Failures

Dependencies among Ops: To avoid packet drops, not all ops can be issued simultaneously. For example, we must configure a Tunnel at all affected sites before configuring the corresponding TG and FG. Similarly, a Tunnel cannot be deleted before first removing all referencing entries. Fig. 10 shows two example dependencies ("schedules"), one (Fig. 10(a)) for creating TG_1 with two associated Tunnels T_1 and T_2 for the $A \rightarrow B$ FG_1 and a second (Fig. 10(b)) for the case where we remove T_2 from TG_1.

Synchronizing TED between TE and OFC: Computing diffs requires a common TED view between the TE master and the OFC. A TE Session between the master TE server and the master OFC supports this synchronization. We generate a unique identifier for the TE session based on mastership and process IDs for both endpoints. At the start of the session, both endpoints sync their TED view. This functionality also allows one source to recover the TED from the other in case of restarts. TE also periodically synchronizes TED state to a persistent store to handle simultaneous failures. The Session ID allows us to reject any op not part of the current session, e.g., during a TE mastership flap.

Ordering issues: Consider the scenario where TE issues a TG op (TG_1) to use two tunnels with T1:T2 split 0.5:0.5. A few milliseconds later, it creates TG_2 with a 1:0 split as a result of failure in T2. Network delays/reordering means that the TG_1 op can arrive at the OFC after the TG_2 op. We attach site-specific sequence IDs to TE ops to enforce ordering among operations. The OFC maintains the highest session sequence ID and rejects ops with smaller sequence IDs. TE Server retries any rejected ops after a timeout.

TE op failures: A TE op can fail because of RPC failures, OFC rejection, or failure to program a hardware device. Hence, we track a (**D**irty/**C**lean) bit for each TED entry. Upon issuing a TE op, TE marks the corresponding TED entry dirty. We clean dirty entries upon receiving acknowledgment from the OFC. Otherwise, we retry the operation after a timeout. The dirty bit persists across restarts and is part of TED. When computing diffs, we automatically replay any dirty TED entry. This is safe because TE ops are idempotent by design.

There are some additional challenges when a TE Session cannot be established, e.g., because of control plane or software failure. In such situations, TE may not have an accurate view of the TED for that site. In our current design, we continue to assume the last known state for that site and *force fail* new ops to this site. Force fail ensures that we do not issue any additional dependent ops.

6. EVALUATION

6.1 Deployment and Evolution

In this section, we evaluate our deployment and operational experience with B4. Fig. 11 shows the growth of B4 traffic and the rollout of new functionality since its first deployment. Network traffic has roughly doubled in year 2012. Of note is our ability to quickly deploy new functionality such as centralized TE on the baseline SDN framework. Other TE evolutions include caching of recently used paths to reduce tunnel ops load and mechanisms to adapt TE to unresponsive OFCs (§7).

We run 5 geographically distributed TE servers that participate in master election. Secondary TE servers are hot standbys and can assume mastership in less than 10 seconds. The master is typically stable, retaining its status for 11 days on average.

Table 3(d) shows statistics about B4 topology changes in the three months from Sept. to Nov. 2012. In that time, we averaged 286 topology changes per day. Because the TE Server operates on an aggregated topology view, we can divide these remaining topology changes into two classes: those that change the capacity of an edge in the TE Server's topology view, and those that add or remove an edge from the topology. We found that we average only 7 such additions or removals per day. When the capacity on an edge changes, the TE server may send operations to optimize use of the new capacity, but the OFC is able to recover from any traffic drops without TE involvement. However, when an edge is removed or added, the TE server must create or tear down tunnels crossing that edge, which increases the number of operations sent to OFCs and therefore load on the system.

Our main takeaways are: i) topology aggregation significantly reduces path churn and system load; ii) even with topology aggregation, edge removals happen multiple times a day; iii) WAN links are susceptible to frequent port flaps and benefit from dynamic centralized management.

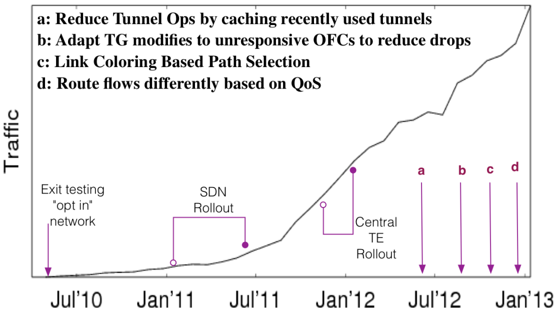


Figure 11: Evolution of B4 features and traffic.

(a) TE Algorithm

Avg. Daily Runs	540
Avg. Runtime	0.3s
Max Runtime	0.8s

(b) Topology

Sites	16
Edges (Unidirectional)	46

(c) Flows

Tunnel Groups	240
Flow Groups	2700
Tunnels in Use	350
Tunnels Cached	1150

(d) Topology Changes

Change Events	286/day
Edge Add/Delete	7/day

Table 3: Key B4 attributes from Sept to Nov 2012.

6.2 TE Ops Performance

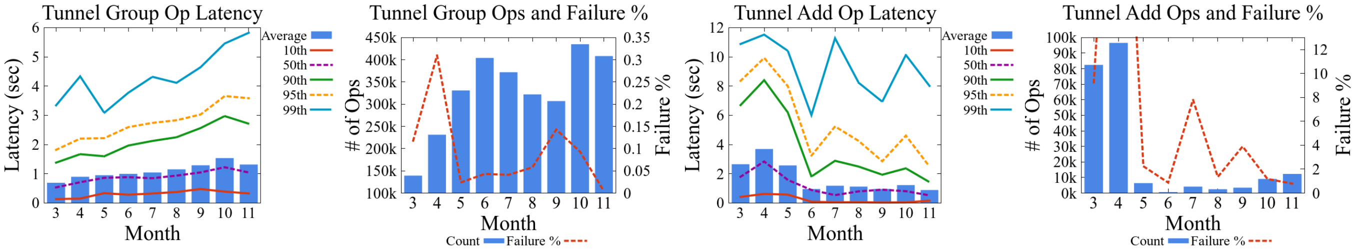
Table 3 summarizes aggregate B4 attributes and Fig. 12 shows a monthly distribution of ops issued, failure rate, and latency distribution for the two main TE operations: Tunnel addition and Tunnel Group mutation. We measure latency at the TE server between sending a TE-op RPC and receiving the acknowledgment. The nearly 100x reduction in tunnel operations came from an optimization to cache recently used tunnels (Fig. 12(d)). This also has an associated drop in failed operations.

We initiate TG ops after every algorithm iteration. We run our TE algorithm instantaneously for each topology change and periodically to account for demand changes. The growth in TG operations comes from adding new network sites. The drop in failures in May (Month 5) and Nov (Month 11) comes from the optimizations resulting from our outage experience (§ 7).

To quantify sources of network programming delay, we periodically measure latency for sending a NoOp TE-Op from TE Server to SDN Gateway to OFC and back. The 99th percentile time for this NoOp is one second (Max RTT in our network is 150 ms). High latency correlates closely with topology changes, expected since such changes require significant processing at all stack layers and delaying concurrent event processing.

For every TE op, we measure the switch time as the time between the start of operation processing at the OFC and the OFC receiving acks from all switches.

Table 4 depicts the switch time fraction ($STF = \frac{\text{Switch time}}{\text{Overall TE op time}}$) for three months (Sep-Nov 2012). A higher fraction indicates that there is promising potential for optimizations at lower layers of the stack. The switch fraction is substantial even for control across the WAN. This is symptomatic of OpenFlow-style control still being in its early stages; neither our software or switch SDKs are optimized for dynamic table programming. In particular, tunnel tables are typ-



(a) (b) (c) (d)

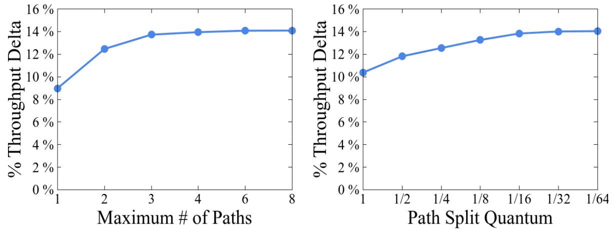
Figure 12: Stats for various TE operations for March-Nov 2012.

Op Latency Range (s)	Avg Daily Op Count	Avg STF	10th-perc STF
0-1	4835	0.40	0.02
1-3	6813	0.55	0.11
3-5	802	0.71	0.35
5-∞	164	0.77	0.37

Table 4: Fraction of TG latency from switch.

Failure Type	Packet Loss (ms)
Single link	4
Encap switch	10
Transit switch neighboring an encap switch	3300
OFC	0
TE Server	0
TE Disable/Enable	0

Table 5: Traffic loss time on failures.



(a) (b)

Figure 13: TE global throughput improvement relative to shortest-path routing.

ically assumed to be "set and forget" rather than targets for frequent reconfiguration.

6.3 Impact of Failures

We conducted experiments to evaluate the impact of failure events on network traffic. We observed traffic between two sites and measured the duration of any packet loss after six types of events: a single link failure, an encap switch failure and separately the failure of its neighboring transit router, an OFC failover, a TE server failover, and disabling/enabling TE.

Table 5 summarizes the results. A single link failure leads to traffic loss for only a few milliseconds, since the affected switches quickly prune their ECMP groups that include the impaired link. An encap switch failure results in multiple such ECMP pruning operations at the neighboring switches for convergence, thus taking a few milliseconds longer. In contrast, the failure of a transit router that is a neighbor to an encap router requires a much longer convergence time (3.3 seconds). This is primarily because the neighboring encap switch has to update its multipath table entries for potentially several tunnels that were traversing the failed switch, and each such operation is typically slow (currently 100ms).

By design, OFC and TE server failure/restart are all hitless. That is, absent concurrent additional failures during failover, failures of these software components do not cause any loss of data-plane traffic. Upon disabling TE, traffic falls back to the lower-priority forwarding rules established by the baseline routing protocol.

6.4 TE Algorithm Evaluation

Fig. 13(a) shows how global throughput improves as we vary maximum number of paths available to the TE algorithm. Fig. 13(b) shows how throughput varies with the various quantizations of path splits (as supported by our switch hardware) among available tunnels. Adding more paths and using finer-granularity traffic splitting both give more flexibility to TE but it consumes additional hardware table resources.

For these results, we compare TE's total bandwidth capacity with path allocation against a baseline where all flows follow the shortest path. We use production flow data for a day and compute average improvement across all points in the day (every 60 seconds).

For Fig. 13(a) we assume a $\frac{1}{64}$ path-split quantum, to focus on sensitivity to the number of available paths. We see significant improvement over shortest-path routing, even when restricted to a single path (which might not be the shortest). The throughput improvement flattens at around 4 paths.

For Fig. 13(b), we fix the maximum number of paths at 4, to show the impact of path-split quantum. Throughput improves with finer splits, flattening at $\frac{1}{16}$. Therefore, in our deployment, we use TE with a quantum of $\frac{1}{4}$ and 4 paths.

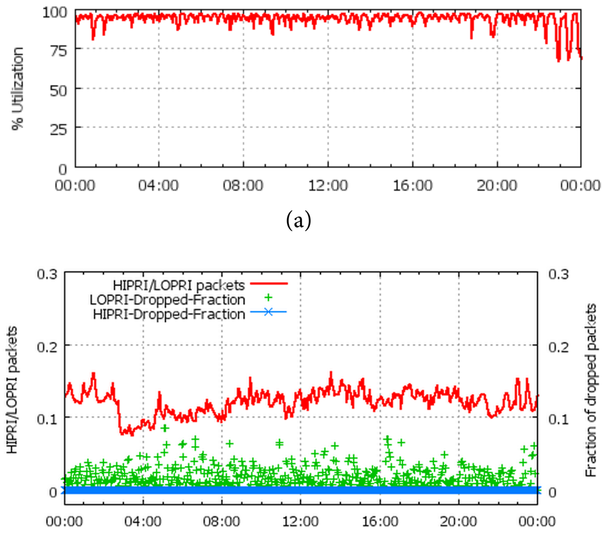
While 14% average throughput increase is substantial, the main benefits come during periods of failure or high demand. Consider a high-priority data copy that takes place once a week for 8 hours, requiring half the capacity of a shortest path. Moving that copy off the shortest path to an alternate route only improves average utilization by 5% over the week. However, this reduces our WAN's required deployed capacity by a factor of 2.

6.5 Link Utilization and Hashing

Next, we evaluate B4's ability to drive WAN links to near 100% utilization. Most WANs are designed to run at modest utilization (e.g., capped at 30-40% utilization for the busiest links), to avoid packet drops and to reserve dedicated backup capacity in the case of failure. The busiest B4 edges constantly run at near 100% utilization, while almost all links sustain full utilization during the course of each day.

We tolerate high utilization by differentiating among different traffic classes.

The two graphs in Fig. 14 show traffic on all links between two WAN sites. The top graph shows how we drive utilization close to 100% over a 24-hour period. The second graph shows the ratio of high priority to low priority packets, and packet-drop fractions for each priority. A key benefit of centralized TE is the ability to mix priority classes across all edges. By ensuring that heavily utilized edges carry substantial low priority traffic, local QoS schedulers can ensure that high priority traffic is insulated from loss despite shallow switch buffers, hashing imperfections and inherent traffic burstiness. Our low priority traffic tolerates loss by throttling transmission rate to available capacity at the application level.



(a)

(b)

Figure 14: Utilization and drops for a site-to-site edge.

Site-to-site edge utilization can also be studied at the granularity of the constituent links of the edge, to evaluate B4's ability to load-balance traffic across all links traversing a given edge. Such balancing is a prerequisite for topology abstraction in TE (§3.1). Fig. 15 shows the uniform link utilization of all links in the site-to-site edge of Fig. 14 over a period of 24 hours. In general, the results of our load-balancing scheme in the field have been very encouraging across the B4 network. For at least 75% of site-to-site edges, the max:min ratio in link utilization across constituent links is 1.05 without failures (i.e., 5% from optimal), and 2.0 with failures. More effective load balancing during failure conditions is a subject of our ongoing work.

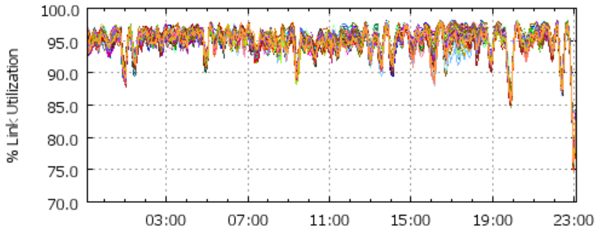


Figure 15: Per-link utilization in a trunk, demonstrating the effectiveness of hashing.

7. EXPERIENCE FROM AN OUTAGE

Overall, B4 system availability has exceeded our expectations. However, it has experienced one substantial outage that has been instructive both in managing a large WAN in general and in the context of SDN in particular. For reference, our public facing network has also suffered failures during this period.

The outage started during a planned maintenance operation, a fairly complex move of half the switching hardware for our biggest site from one location to another. One of the new switches was inadvertently manually configured with the same ID as an existing switch. This led to substantial link flaps. When switches received ISIS Link State Packets (LSPs) with the same ID containing different adjacencies, they immediately flooded new LSPs through all other interfaces. The switches with duplicate IDs would alternate responding to the LSPs with their own version of network topology, causing more protocol processing.

Recall that B4 forwards routing-protocols packets through software, from Quagga to the OFC and finally to the OFA. The OFC to OFA connection is the most constrained in our implementation, leading to substantial protocol packet queueing, growing to more than 400MB at its peak.

The queueing led to the next chain in the failure scenario: normal ISIS Hello messages were delayed in queues behind LSPs, well past their useful lifetime. This led switches to declare interfaces down, breaking BGP adjacencies with remote sites. TE Traffic transiting through the site continued to work because switches maintained their last known TE state. However, the TE server was unable to create new tunnels through this site. At this point, any concurrent physical failures would leave the network using old broken tunnels.

With perfect foresight, the solution was to drain all links from one of the switches with a duplicate ID. Instead, the very reasonable response was to reboot servers hosting the OFCs. Unfortunately, the high system load uncovered a latent OFC bug that prevented recovery during periods of high background load.

The system recovered after operators drained the entire site, disabled TE, and finally restarted the OFCs from scratch. The outage highlighted a number of important areas for SDN and WAN deployment that remain active areas of work:

1. Scalability and latency of the packet IO path between the OFC and OFA is critical and an important target for evolving OpenFlow and improving our implementation. For example, OpenFlow might support two communication channels, high priority for latency sensitive operations such as packet IO and low priority for throughput-oriented operations such as switch programming operations. Credit-based flow control would aid in bounding the queue buildup. Allowing certain duplicate messages to be dropped would help further, e.g., consider that the earlier of two untransmitted LSPs can simply be dropped.
2. OFA should be asynchronous and multi-threaded for more parallelism, specifically in a multi-linecard chassis where multiple switch chips may have to be programmed in parallel in response to a single OpenFlow directive.
3. We require additional performance profiling and reporting. There were a number of "warning signs" hidden in system logs during previous operations and it was no accident that the outage took place at our largest B4 site, as it was closest to its scalability limits.
4. Unlike traditional routing control systems, loss of a control session, e.g., TE-OFC connectivity, does not necessarily invalidate forwarding state. With TE, we do not automatically reroute existing traffic around an unresponsive OFC

(i.e., we *fail open*). However, this means that it is impossible for us to distinguish between physical failures of underlying switch hardware and the associated control plane. This is a reasonable compromise as, in our experience, hardware is more reliable than control software. We would require application-level signals of broken connectivity to effectively disambiguate between WAN hardware and software failures.

5. The TE server must be adaptive to failed/unresponsive OFCs when modifying TGs that depend on creating new Tunnels. We have since implemented a fix where the TE server avoids failed OFCs in calculating new configurations.
6. Most failures involve the inevitable human error that occurs in managing large, complex systems. SDN affords an opportunity to dramatically simplify system operation and management. Multiple, sequenced manual operations should not be involved for virtually any management operation.
7. It is critical to measure system performance to its breaking point with published envelopes regarding system scale; any system will break under sufficient load. Relatively rare system operations, such as OFC recovery, should be tested under stress.

8. RELATED WORK

There is a rich heritage of work in Software Defined Networking [7, 8, 19, 21, 27] and OpenFlow [28, 31] that informed and inspired our B4 design. We describe a subset of these related efforts in this section.

While there has been substantial focus on OpenFlow in the data center [1, 35, 40], there has been relatively little focus on the WAN. Our focus on the WAN stems from the criticality and expense of the WAN along with the projected growth rate. Other work has addressed evolution of OpenFlow [11, 35, 40]. For example, DevoFlow[11] reveals a number of OpenFlow scalability problems. We partially avoid these issues by proactively establishing flows, and pulling flow statistics both less frequently and for a smaller number of flows. There are opportunities to leverage a number of DevoFlow ideas to improve B4's scalability.

Route Control Platform (RCP)[6] describes a centralized approach for aggregating BGP computation from multiple routers in an autonomous system in a single logical place. Our work in some sense extends this idea to fine-grained traffic engineering and details an end-to-end SDN implementation. Separating the routing control plane from forwarding can also be found in the current generation of conventional routers, although the protocols were historically proprietary. Our work specifically contributes a description of the internal details of the control/routing separation, and techniques for stitching individual routing elements together with centralized traffic engineering.

RouteFlow's[30, 32] extension of RCP is similar to our integration of legacy routing protocols into B4. The main goal of our integration with legacy routing was to provide a gradual path for enabling OpenFlow in the production network. We view BGP integration as a step toward deploying new protocols customized to the requirements of, for instance, a private WAN setting.

Many existing production traffic engineering solutions use MPLS-TE [5]: MPLS for the data plane, OSFP/IS-IS/iBGP to distribute the state and RSVP-TE[4] to establish the paths. Since each site independently establishes paths with no central coordination, in practice, the resulting traffic distribution is both suboptimal and non-deterministic.

Many centralized TE solutions [3, 10, 14, 24, 34, 36, 38] and algorithms [29, 33] have been proposed. In practice, these systems operate at coarser granularity (hours) and do not target global optimization during each iteration. In general, we view B4 as a framework for rapidly deploying a variety of traffic engineering solutions; we anticipate future opportunities to implement a number of traffic engineering techniques, including these, within our framework.

It is possible to use linear programming (LP) to find a globally max-min fair solution, but is prohibitively expensive [13]. Approximating this solution can improve runtime [2], but initial work in this area did not address some of the requirements for our network, such as piecewise linear bandwidth functions for prioritizing flow groups and quantization of the final assignment. One recent effort explores improving performance of iterative LP by delivering fairness and bandwidth while sacrificing scalability to the larger networks [13]. Concurrent work [23] further improves the runtime of an iterative LP-based solution by reducing the number of LPs, while using heuristics to maintain similar fairness and throughput. It is unclear if this solution supports per-flow prioritization using bandwidth functions. Our approach delivers similar fairness and 99% of the bandwidth utilization compared to LP, but with sub-second runtime for our network and scales well for our future network.

Load balancing and multipath solutions have largely focused on data center architectures [1, 18, 20], though at least one effort recently targets the WAN [22]. These techniques employ flow hashing, measurement, and flow redistribution, directly applicable to our work.

9. CONCLUSIONS

This paper presents the motivation, design, and evaluation of B4, a Software Defined WAN for our data center to data center connectivity. We present our approach to separating the network's control plane from the data plane to enable rapid deployment of new network control services. Our first such service, centralized traffic engineering allocates bandwidth among competing services based on application priority, dynamically shifting communication patterns, and prevailing failure conditions.

Our Software Defined WAN has been in production for three years, now serves more traffic than our public facing WAN, and has a higher growth rate. B4 has enabled us to deploy substantial cost-effective WAN bandwidth, running many links at near 100% utilization for extended periods. At the same time, SDN is not a cure-all. Based on our experience, bottlenecks in bridging protocol packets from the control plane to the data plane and overheads in hardware programming are important areas for future work.

While our architecture does not generalize to all SDNs or to all WANs, we believe there are a number of important lessons that can be applied to a range of deployments. In particular, we believe that our hybrid approach for simultaneous support of existing routing protocols and novel traffic engineering services demonstrates an effective technique for gradually introducing SDN infrastructure into existing deployments. Similarly, leveraging control at the edge to both measure demand and to adjudicate among competing services based on relative priority lays a path to increasing WAN utilization and improving failure tolerance.

Acknowledgements

Many teams within Google collaborated towards the success of the B4 SDN project. In particular, we would like to acknowledge the development, test, operations and deployment groups including Jing Ai, Rich Alimi, Kondapa Naidu Bollineni, Casey Barker, Seb Boving, Bob Buckholz, Vijay Chandramohan, Roshan Chepuri, Gaurav Desai, Barry Friedman, Denny Gentry, Paulie Germano, Paul Gyugyi, Anand Kanagala, Nikhil Kasinadhuni, Kostas Kassaras, Bikash Koley, Aamer Mahmood, Raleigh Mann, Waqar

Mohsin, Ashish Naik, Uday Naik, Steve Padgett, Anand Raghuraman, Rajiv Ramanathan, Faro Rabe, Paul Schultz, Eiichi Tanda, Arun Shankarnarayan, Aspi Siganporia, Ben Treynor, Lorenzo Vicisano, Jason Wold, Monika Zahn, Enrique Cauich Zermeno, to name a few. We would also like to thank Mohammad Al-Fares, Steve Gribble, Jeff Mogul, Jennifer Rexford, our shepherd Matt Caesar, and the anonymous SIGCOMM reviewers for their useful feedback.

10. REFERENCES

[1] Al-Fares, M., Loukissas, A., and Vahdat, A. A Scalable, Commodity Data Center Network Architecture. In *Proc. SIGCOMM* (New York, NY, USA, 2008), ACM.

[2] Allalouf, M., and Shavitt, Y. Centralized and Distributed Algorithms for Routing and Weighted Max-Min Fair Bandwidth Allocation. *IEEE/ACM Trans. Networking 16*, 5 (2008), 1015–1024.

[3] Aukia, P., Kodialam, M., Koppol, P. V., Lakshman, T. V., Sarin, H., and Suter, B. RATES: A Server for MPLS Traffic Engineering. *IEEE Network Magazine 14*, 2 (March 2000), 34–41.

[4] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and Swallow, G. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209, IETF, United States, 2001.

[5] Awduche, D., Malcolm, J., Agogbua, J., O'Dell, M., and McManus, J. Requirements for Traffic Engineering Over MPLS. RFC 2702, IETF, 1999.

[6] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, K. Design and Implementation of a Routing Control Platform. In *Proc. of NSDI* (April 2005).

[7] Casado, M., Freedman, M. J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. Ethane: Taking Control of the Enterprise. In *Proc. SIGCOMM* (August 2007).

[8] Casado, M., Garfinkel, T., Akella, A., Freedman, M. J., Boneh, D., McKeown, N., and Shenker, S. SANE: A Protection Architecture for Enterprise Networks. In *Proc. of Usenix Security* (August 2006).

[9] Chandra, T. D., Griesemer, R., and Redstone, J. Paxos Made Live: an Engineering Perspective. In *Proc. of the ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2007), ACM, pp. 398–407.

[10] Choi, T., Yoon, S., Chung, H., Kim, C., Park, J., Lee, B., and Jeong, T. Design and Implementation of Traffic Engineering Server for a Large-Scale MPLS-Based IP Network. In *Revised Papers from the International Conference on Information Networking, Wireless Communications Technologies and Network Applications-Part I* (London, UK, UK, 2002), ICOIN '02, Springer-Verlag, pp. 699–711.

[11] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. DevoFlow: Scaling Flow Management for High-Performance Networks. In *Proc. SIGCOMM* (2011), pp. 254–265.

[12] Danna, E., Hassidim, A., Kaplan, H., Kumar, A., Mansour, Y., Raz, D., and Segalov, M. Upward Max Min Fairness. In *INFOCOM* (2012), pp. 837–845.

[13] Danna, E., Mandal, S., and Singh, A. A Practical Algorithm for Balancing the Max-min Fairness and Throughput Objectives in Traffic Engineering. In *Proc. INFOCOM* (March 2012), pp. 846–854.

[14] Elwalid, A., Jin, C., Low, S., and Widjaja, I. MATE: MPLS Adaptive Traffic Engineering. In *Proc. IEEE INFOCOM* (2001), pp. 1300–1309.

[15] Farrington, N., Rubow, E., and Vahdat, A. Data Center Switch Architecture in the Age of Merchant Silicon. In *Proc. Hot Interconnects* (August 2009), IEEE, pp. 93–102.

[16] Fortz, B., Rexford, J., and Thorup, M. Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine 40* (2002), 118–124.

[17] Fortz, B., and Thorup, M. Increasing Internet Capacity Using Local Search. *Comput. Optim. Appl. 29*, 1 (October 2004), 13–48.

[18] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In *Proc. SIGCOMM* (August 2009).

[19] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM CCR 35*, 5 (2005), 41–54.

[20] Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In *Proc. ACM workshop on Programmable Routers for Extensible Services of Tomorrow* (2008), pp. 57–62.

[21] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: Towards an Operating System for Networks. In *SIGCOMM CCR* (July 2008).

[22] He, J., and Rexford, J. Toward Internet-wide Multipath Routing. *IEEE Network Magazine 22*, 2 (March 2008), 16–21.

[23] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Have Your Network and Use It Fully Too: Achieving High Utilization in Inter-Datacenter WANs. In *Proc. SIGCOMM* (August 2013).

[24] Kandula, S., Katabi, D., Davie, B., and Charny, A. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *Proc. SIGCOMM* (August 2005).

[25] Kipp, S. Bandwidth Growth and the Next Speed of Ethernet. *Proc. North American Network Operators Group* (October 2012).

[26] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. Onix: a Distributed Control Platform for Large-scale Production Networks. In *Proc. OSDI* (2010), pp. 1–6.

[27] Lakshman, T., Nandagopal, T., Ramjee, R., Sabnani, K., and Woo, T. The Softrouter Architecture. In *Proc. HotNets* (November 2004).

[28] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM CCR 38*, 2 (2008), 69–74.

[29] Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., and Diot, C. Traffic Matrix Estimation: Existing Techniques and New Directions. In *Proc. SIGCOMM* (New York, NY, USA, 2002), ACM, pp. 161–174.

[30] Nascimento, M. R., Rothenberg, C. E., Salvador, M. R., and Magalhães, M. F. QuagFlow: Partnering Quagga with OpenFlow (Poster). In *Proc. SIGCOMM* (2010), pp. 441–442.

[31] OpenFlow Specification. `http://www.openflow.org/wp/documents/`.

[32] Rothenberg, C. E., Nascimento, M. R., Salvador, M. R., Corrêa, C. N. A., Cunha de Lucena, S., and Raszuk, R. Revisiting Routing Control Platforms with the Eyes and Muscles of Software-defined Networking. In *Proc. HotSDN* (2012), pp. 13–18.

[33] Roughan, M., Thorup, M., and Zhang, Y. Traffic Engineering with Estimated Traffic Matrices. In *Proc. IMC* (2003), pp. 248–258.

[34] Scoglio, C., Anjali, T., de Oliveira, J. C., Akyildiz, I. F., and Uhl, G. TEAM: A Traffic Engineering Automated Manager for DiffServ-based MPLS Networks. *Comm. Mag. 42*, 10 (October 2004), 134–145.

[35] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. FlowVisor: A Network Virtualization Layer. Tech. Rep. OPENFLOW-TR-2009-1, OpenFlow, October 2009.

[36] Suchara, M., Xu, D., Doverspike, R., Johnson, D., and Rexford, J. Network Architecture for Joint Failure Recovery and Traffic Engineering. In *Proc. ACM SIGMETRICS* (2011), pp. 97–108.

[37] Thaler, D. Multipath Issues in Unicast and Multicast Next-Hop Selection. RFC 2991, IETF, 2000.

[38] Wang, H., Xie, H., Qiu, L., Yang, Y. R., Zhang, Y., and Greenberg, A. COPE: Traffic Engineering in Dynamic Networks. In *Proc. SIGCOMM* (2006), pp. 99–110.

[39] Xu, D., Chiang, M., and Rexford, J. Link-state Routing with Hop-by-hop Forwarding Can Achieve Optimal Traffic Engineering. *IEEE/ACM Trans. Netw. 19*, 6 (December 2011), 1717–1730.

[40] Yu, M., Rexford, J., Freedman, M. J., and Wang, J. Scalable flow-based networking with DIFANE. In *Proc. SIGCOMM* (2010), pp. 351–362.