vFiber: A System for Virtualizing Optical Fibers

Matthew Hall
University of Oregon
Computer and Information Science
Directed Research Project Report

1. INTRODUCTION

No technological advancement in computing has offered more significant benefits than virtualization. Benefits include increased utilization of the virtualized resource, reduced operational costs, and improved agility via rapid disaster discovery. For example, Software Defined everything Infrastructures (SDxI), enabled by virtualization, allow programmability to achieve speed, agility, and cost-effectiveness in modern cloud platforms [32]. In short, virtualization is a crucial cornerstone for many domains (*e.g.*, compute, storage, and networking).

While progress has been made in virtualizing different resources ranging from processors to storage devices to batteries, there has been one under-utilized resource that has not been virtualized: the unused optical fiber strands, known as *dark fiber*. Given the diversity of the physical Internet [14] and the fiber glut [10, 7], we posit that optical fiber strands connecting different endpoints (*e.g.*, switches) in an enterprise setting are a prime candidate for the next wave of virtualization. Users who require temporary or limited-time access to fiber, and enterprises (*i.e.,* cloud providers) with over-provisioned links between their facilities stand to gain from virtualization of optical fibers. Unfortunately, the interfaces between fiber and users today are telephone calls and conversations between lawyers and network engineers. Accessing fiber can take months to years. The process requires the owner of the fiber and the client to agree on the terms of the client's Indefeasible Rights of Use (IRU)[3, 4, 2, 5]. In this work, we provide an alternative means for a fiber owner to provide access to their network infrastructure via virtualization.

Motivation. There are many use-cases to virtualizing optical fibers. These include geographic diversity (GD) eliminating shared risks [14], improved network resiliency, and bandwidth on demand.

(1) GD refers to the ability for a sender to choose the path which his/her data travels to reach its destination. Such a capability is not available in today's Internet and would have implications for more secure routing of traffic[1] in the case where a sender wishes for their traffic to avoid specific intermediate hops.

(2) A virtualization system for optical fiber can improve network resiliency. In the face of network outages or fiber cuts the virtualization platform will quickly respond to the event by provisioning higher capacity on alternative links. Typically fiber cuts can take hours to days to recover from, but a virtualization system for fiber could reduce this time to minutes or seconds. The network experiencing the outage will be able to use the virtualization system to see a significant reduction in the traffic loss during the outage.

(3) Finally, bandwidth on demand will enable networks which experience brief bursts in demand beyond their available capacity to access the underutilized fiber of another network. Wide Area Network (WAN) administrators and traffic engineers have been studying how to achieve the highest utilization out of expensive networks consisting of fiber spanning the globe. A novel virtualization system would give them a new tool to profit from underutilized fiber strands or to access third party strands when demand is exceptionally high.

Scope and Context. The aim of this work is to spin-up unused optical fibers between adjacent layer-2 network devices (endpoints) and to show how this capability can be used to virtualize multi-link optical paths between these endpoints. We envision this system to be useful for clients of an enterprise data center network with high bandwidth demands. Similar to the way in which people use virtual compute resources in the cloud today, we see an opportunity to virtualize the connectivity between cloud hosting locations. This system will provide greater flexibility for fiber assets and their users.

Cloud service providers (CSPs) are in a choice position to virtualize fiber. Consider Microsoft, which owns a global data-center network and thousands of miles of fiber assets. Microsoft has more than 25,000 enterprise clients in North America, many of whom operate mission-critical systems, applications, and products (SAP) in the cloud [16]. Another possible purveyor of virtualized fiber is Amazon Web Services (AWS) who has approximately 100,000 enterprise clients worldwide [9]. These enterprise clients trust CSPs with their critical SAP infrastructure. These clients could also benefit

[1]We are not concerned with traffic on the fiber provider's network. Instead, we accommodate alien wavelengths within the network. Alien wavelengths are optical frequencies that travel along the same strands as the provider's, but that terminate at different layer-2 hardware. The benefit of adopting alien wavelengths has been discussed by network engineers in recent years [24, 29].

by having dynamic access to fiber between the cloud's many data centers around the world. CSPs are proficient in packing client virtual machines into available servers and rack-space. vFiber will allow them to pack client data transmission requests into unused optical fibers.

Regarding the pricing and determination of which client receives which strand by the virtualization system, we assume that there is a method in place to handle this decision. Similar models that might be appropriate are Amazon's Spot instance marketplace [1], or auctions for web-based advertisements [15, 35]. We leave the analysis of different economic models for fiber allocation to future work and assume that the network-provider has full control and discretion to allocate links to the first client who offers the price determined by the network operator.

Our Solution. In this work, we propose a set of techniques and the design of a system called vFiber. This system is intended to virtualize available optical fiber assets in an enterprise data center network, *e.g.* a regional cloud service provider's wide area network (WAN). We implement this system and demonstrate its ability to scale bandwidth on a 100-meter strand of optical fiber in the lab. In section 3.2.1 we benchmark the responsiveness of vFiber and the underlying network hardware to provision an idle circuit between two switches. In section 3.2.2 we measure the time to scale up bandwidth on an existing link. We also developed a monitoring capability for vFiber, which reports to the vFiber controller whenever it detects a fiber-cut. In section 3.2.3 we leverage this monitor to demonstrate vFiber's ability to respond to a fiber-cut incident quickly. The control plane for vFiber is implemented by a set of distributed servers running the Raft consensus algorithm [30]. In section 3.2.4 we demonstrate the robustness of our Raft implementation by simulating server failures in the face of network demand. Finally, in section 3.2.5 we model the scalability of this distributed implementation by gauging the time to allocate paths on different network topologies with server clusters of various sizes.

2. vFiber SYSTEM DESIGN

Overview. vFiber is an intra-domain virtualization system designed for enterprises to effectively orchestrate and operate their physical infrastructural assets—specifically, the unused (dark) fiber optic link assets. vFiber consists of two main components; namely the distributed controller (DC) and the underlying physical infrastructure substrate. We explain these two components below.

Physical Infrastructure. The physical infrastructure is composed of the traditional layer-1 network substrate including (dark and lit) fiber optic strands, switches, routers, and multiplexers. An enterprise using vFiber should expose its infrastructure information as an *infrastructure graph* (IG)—a new graph-based abstraction—to the DC. An IG encodes address/interface information of devices along with the configuration parameters, the number of lambdas (or wavelengths) between devices, cost per lambda (if any), and device/fiber status information. In vFiber, we create IGs using the NetworkX [19] library.

Distributed Controller (DC). The DC is a cluster of (redundant) servers hosted within an enterprise to virtualize or *light a path on-demand* between two endpoints based on the information embedded in its IG. Our distributed design uses an implementation of the RAFT consensus algorithm [30] called PySyncObj [6]. Each server in the DC cluster consists of a public-facing request handler, knowledge of the underlying physical infrastructure (from the IG), and abstractions to manage, configure and orchestrate the infrastructure assets.

Each DC member has an interface to receive and process physical reconfiguration requests specified in CSV format. Each request includes two endpoints (*e.g.* a source and destination for a data transfer) managed by vFiber, the number of required lambdas, and the name of the client who requested the path. Upon receipt of a request, the DC queries the IG to check if there are sufficient network resources along a requested path. The DC then modifies the IG by making those resources unavailable to subsequent requests. Then, the DC used a physical layer reconfiguration API—called Torchbearer—to push the configuration changes reflected in the updated IG to the physical hardware interfaces for the lambdas along the requested path.

Torchbearer uses credentials specified by a network administrator to reconfigure network hardware. These credentials and stored in the IG and used to access a command-line-interface (CLI) at each switch along the requested path. From the CLI, Torchbearer can activate or deactivate interfaces to light a strand of fiber with one or more lambdas. The API can be expanded to perform more complex tasks, like creating virtual local area networks (VLANs) and optical tunnels for clients.

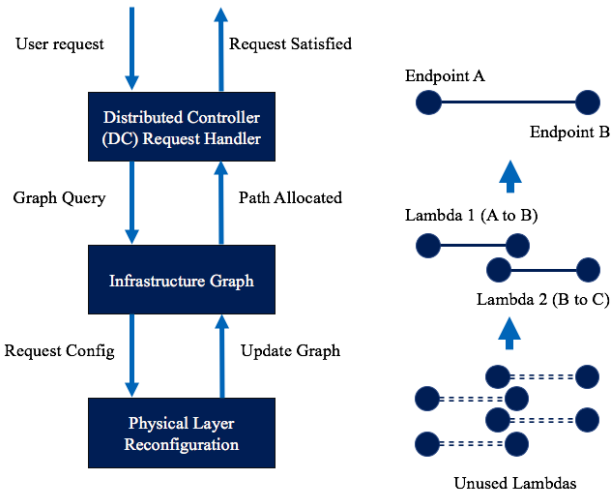


Figure 1: vFiber flow diagram.

The DC is aware of the physical infrastructure and accounts for (in)active links with the IG. Precisely, each server tracks link-specific details (*e.g.,* Is a link lit, or dark? If it is lit,

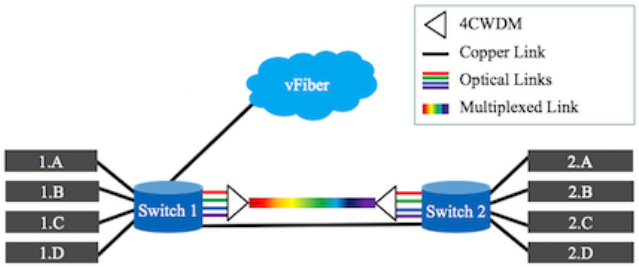


Figure 2: Lab-based test-bed using which we evaluate vFiber. IP addresses of network hosts are shown as 1.A, 1.B, 2.A, 2.B, etc.

how many lambdas are in use? How many dark-lambdas are available to new customers? Which physical port interfaces correspond to lambdas on the links?). The IG stores these details as attributes of an edge whose nodes are layer-2 hardware devices. One such attribute for an edge is an *availability list*. Before vFiber allocates a lambda, the corresponding physical interfaces are obtained from the availability list and pushed into an *unavailability list*. In our ongoing work, we intend to add more functionalities to the DC to monitor active link statistics such SNR similar to RADWAN [33].

Figure 1 depicts the process flow of a request through vFiber. When vFiber receives a request the DC will find the desired dark fiber strands in the IG and check for available interfaces on those edges based on the monitored statistics. If there is a pair of available interfaces for all edges of a path, then the DC will first make those interfaces unavailable to other requests (*e.g.*, from other enterprise administrators) and then send configuration commands to the routers, switches, and multiplexers to light the fiber (and end interfaces) along the requested path. While these configuration changes are taking place, PySyncObj ensures the consistency of the IG across the enterprise users/administrators. We store the IGs as replicated objects in our system. Thus, changes to this replicated object are always reflected across a majority of the servers in the DC, and can only be referenced by the leader of the DC [30].

3. vFiber IMPLEMENTATION AND EVALUATION

In this section, we start by describing the implementation details of vFiber (on a lab-based test-bed shown in figure 2) and then discuss the testing capabilities that we use in our evaluations.

3.1 vFiber Implementation

Our physical network topology is composed of two Cisco 3560-E switches. Each switch connects to a 4-Channel Coarse-Wavelength Division Multiplexer/DeMultiplexer (4-CWDM) and two physical machines. Each switch also connects to two or more network hosts on each physical machine, and each network host has a unique physical interface and IP address. Enterprise administrators orchestrate/manage resources across these network hosts.

Both switches have two twin-gig adapters (CRV-X2-SFPs). We plug two CWDM-SFPs into each CRV-X2-SFP. Thus, each switch has four CWDM-SFP transceivers. The transceivers operate at different wavelengths (*lambdas*), namely 1470, 1490, 1510, and 1530nm. The CWDM SFPs operate in full duplex mode and are each connected to a 1-meter long strand of duplex single-mode 2mm fiber. Each of these four strands then connects to a 4-CWDM. The 4-CWDMs are passive optical devices which are directly connected via 100m of SMF-28 fiber. Each strand of fiber between a switch and a 4-CWDM has a capacity of 1 Gbps based on the Cisco 3560-E's Gigabit Ethernet interface. Thus, the multiplexed link has a capacity of 4 Gbps.

We built the vFiber's DC in Python using PySyncObj [6] for consensus. The DC consists of a cluster of servers running Ubuntu 16.04 virtual machines (VMs). The VMs each have one processor and 2 GB of memory. They run on a MacBook Pro with a 2.9GHz Intel i7 CPU and 16 GB of memory. The VMs have two network adapters that they use throughout all experiments: a bridged-adaptor for communicating with non-VM machines and a host-only adaptor for coordinating with other peer vFiber servers.

We wrote our testing and benchmarking tool (discussed in section 3.2.1) in Python and Bash. This tool leverages vFiber's physical layer reconfiguration API to activate and deactivate physical interfaces on the switches while probing for an available host at the remote end of a network. We use this tool to measure the baseline responsiveness of the physical layer to reconfiguration change and compare this against the responsiveness to reconfiguration via a network administrators request sent to vFiber.

Concurrency Control. It is crucial for the implementation of a distributed system to facilitate effective concurrency control of a shared resource. In our case, vFiber servers are responsible for modifying physical hardware, and therefore the access to this hardware must be orchestrated logically and efficiently. The primary objective is to ensure that the state of the physical infrastructure is identical to the physical topology at any time. To achieve this, we leverage the fact that our infrastructure graph (IG) has one edge for every strand of fiber. When the members of the cluster wish to change the IG, they must obtain a lock for the edges they wish to modify and then release them after all of the edges associated with that path have been updated in the IG.

Since we are implementing a locking scheme, we must consider the four necessary and sufficient conditions for deadlock. These conditions are mutual exclusion, hold and wait, no preemption, and circular wait. Lock ordering is an efficient way to avoid deadlock when one has the luxury of a priori knowledge about the locks that a process will require. Fortunately, this condition is satisfied in our system because we can determine all of the links that will be required by a client's request. Thus, we create an ordering of edges in our

system and require that a client must acquire locks in edge-order. Edge-order is the order in which vFiber learns about an edge. The first edge created receives the edge number 0. The next edge added to the IG is 1, and so on.

While we have implemented a two-phase-commit locking protocol for edges in the IG, we leave it to future work to test the efficiency of this protocol against other protocols such as optimistic concurrency control. In section 3.2.5 we see that requests for multi-link paths are completed in time proportional to the number of links in the path.

3.2 vFiber Evaluation

We seek to evaluate our implementation of vFiber, a novel system for virtualizing optical fibers. There are many pieces to consider when evaluating a distributed system such as vFiber. The system itself relies on a symphony of parts operating correctly together. The distributed controller must replicate state across its member servers. The infrastructure graph must always reflect the state of the underlying hardware. The system must be robust enough to handle internet link failures, and server crashes. Lastly, the system must operate on diverse network topologies.

Our analyses and evaluations focus on three key characteristics of the system. These are (1) scalability, (2) availability, and (3) performance. We developed and ran five experiments to glean insight into these critical aspects. To evaluate performance, we run a benchmarking suite in section 3.2.1. This test quantifies the provisioning time for an idle optical link via vFiber, as well as the overhead related directly to the hardware in the physical network substrate. In section 3.2.2 we evaluate the scalability of vFiber by using it to scale bandwidth up and down on link connecting two remote network host locations. We demonstrate that vFiber can be used to increase or decrease the bandwidth for a network, and show how to use functionality provided by the switch (LACP) to achieve this. We also benchmark the time to scale bandwidth on an active link. In sections 3.2.3 and 3.2.4 we evaluate the availability of the distributed controller (DC) and physical infrastructure respectively. Section 3.2.3 introduces a monitoring capability to use in tandem with vFiber. The monitor studies the available links in a portion of the network and reports to vFiber if a link is cut or otherwise disconnected. We show how vFiber can use this information to reallocate networking resources that it had provisioned to a client in response to such outages. Section 3.2.4 demonstrates the availability of the DC in the face of random server failures. We show that vFiber is capable of fulfilling client request despite limited periodic random server failures, and show its ability to recover after a majority of the servers has crashed for more than five minutes. Finally, putting it all together, we study the three dimensions of the system simultaneously in section 3.2.5, where we look at the performance of vFiber on different topologies while varying the number of servers active in the DC, and stress testing the system with a variable number of requests. Together, these experiments aim to present a holistic picture of vFiber, a complex and powerful tool for virtualizing optical fibers.

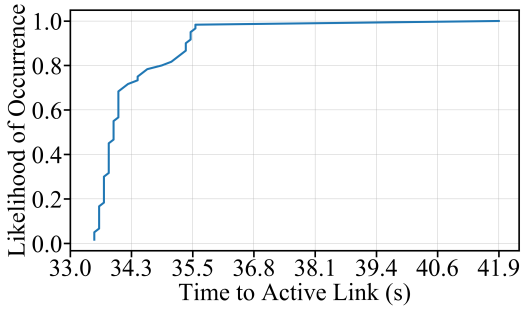


Figure 3: After 50 trials, 95% of all link activation times were less than 35.5 seconds. 5% completed within 33.6 seconds.

3.2.1 Responsiveness of vFiber

Goal: First and foremost, the resources (*i.e.*, fiber-optic links) virtualized by the system should enable and support five-nines availability. Unfortunately, due to the many ways in which failure can arise, satisfying this requirement is challenging. We posit that this challenge can be addressed by enabling infrastructure to be provisioned in a fast and flexible fashion. In particular, vFiber must be able to support *reactive* reliability and put paths into service quickly when needed by an enterprise to recover from an unexpected failure. Naturally, the ability to rapidly virtualize fiber infrastructure simplifies the process of activating backup resources during network maintenance or outage events.

To this end, we measure the responsiveness of our system. Specifically, we seek to understand the time taken by different components in vFiber to virtualize a fiber-optic link between two endpoints and quantify the overheads in the vFiber system versus the underlying physical infrastructure substrate. Using our testing and benchmarking capability, we measure the time spent to generate the configuration files, virtualize and provision the actual link between two switches, the hardware processing time and limitation (if any), and total response time to process a physical link reconfiguration request.

To understand the hardware limitations of the switches in our test-bed, we employ two functions—*light* and *extinguish*—from the testing capability. *Light* enables a pair of interfaces at the ends of a link, thus adding a new lambda. *Extinguish* works similarly to light but disables the interfaces rather than enabling them. We leverage these functions while sending pings to a remote host (*, e.g.,* a local network address that is only accessible via the optical link) every tenth of a second to measure the availability of a newly provisioned path from one network host to another.

Parameters: We issue a command at one network host to send 600 pings per minute, for 50 minutes, to a remote host. Simultaneously, we call *extinguish* to shutdown interfaces

at both ends of the link. We then wait 10 seconds for the switch to clear its cache about the presence of a link (if there was one). Then we call *light* to activate the link and note the number of pings that timed out while the link was inactive. We divide this by ten pings-per-second to arrive at the number of seconds that the link was down and then subtract ten seconds to account for the time we waited before activating the link. We calculate the time taken for a lit link to become available for end-to-end communication using Equation 1. The δ in Equation 1 is the time in which the connection is purposefully left idle between extinguishing and lighting an end-to-end path.

$$T = \frac{|PingTimeouts|}{PingsPerSecond} - \delta \quad (1)$$

Results: We run our benchmark 50 times and analyze the distribution of activation times over these runs. We can see from figure x that for the 95% of the results were less than 36 seconds, with one outlier of ∼41 seconds. This outlier could be caused by queueing delay at any of the host interfaces or within the switches. We omit this outlier and infer that the link activation time is between 33 and 36 seconds 95% of the time.

During these experiments, spanning tree was active on the switches per best-practices advise from network administrators. The protocol is designed to prevent broadcast loops between switches. Much of the delay that we observed between activating a lambda and using it was due to the spanning tree protocol which has a run time of ∼30 seconds [12].

Without the overhead of waiting for the link to become available, the system's average processing time was 2.023 seconds with a standard deviation of 0.278 milliseconds over 50 runs—enabling rapid enterprise fiber infrastructure virtualization. Later, in section 3.2.5 we see responsiveness of vFiber in completing multiple requests with the overhead of lambda activation reported in figure 3.

3.2.2 Scalability of vFiber

Goal: One of the key objectives of any enterprise is to maximize revenue. Maximizing revenue is directly tied to maximizing the utilization of network resources. This objective motivates the second required feature for vFiber. Various strategies include: (1) providing resource elasticity, *i.e.*, by creating an illusion of unlimited fiber optic links in the network; (2) efficiently packing the fiber conduits with maximum amounts of traffic without introducing network congestion (*e.g.,* RADWAN [33]); (3) and reclaiming/reusing all the underutilized dark fiber resources (*e.g.,* GreyFiber [13] and our vFiber approach).

Parameters: To achieve this objective, we present a case study on how the virtualization of fiber optic links using vFiber can increase the total capacity of the network on demand. Using our test-bed depicted in figure 2, with four hosts at each end of the dumbbell topology, we virtualize the link between the 4-CWDMs.

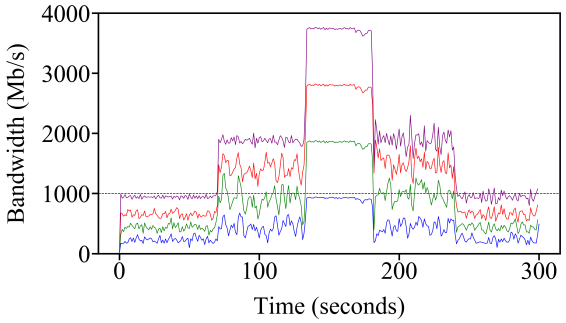


Figure 4: Cumulative bandwidth between 4 pairs of hosts on a network, using vFiber to scale bandwidth up and down between the host locations (network diagram in figure 2).

We use iPerf3 [34] to saturate the 1 Gbps link (with only one lambda) between the two switches by running iPerf servers on hosts connected to switch 1 and iPerf3 clients on hosts connected to switch 2, making all four hosts at each endpoint compete for the bandwidth.

Results: After one minute of saturating the link, we send a physical reconfiguration request from a host for more bandwidth to vFiber, and ∼9 seconds later, the new network capacity of 1,885 Mbps[2] is available. We repeat this process another minute later, this time sending two additional requests for increased bandwidth. As seen in figure 4, vFiber-based fiber optic virtualization can scale the total network capacity from 942 Mbps to 3,763 Mbps. Again, we observed this shift ∼9 seconds after sending the requests.

The client terminates its lease on the lambda after sending the second requests. At this time, bandwidth on the link drops back down to ∼2 Gbps. Another minute later, the first request terminates, and vFiber repossesses that link as well. At this time, the bandwidth on the link returns to ∼1 Gbps.

In figure 4, the dashed line at 1 Gbps shows the initial capacity of the link. The blue line (bottom) shows the bandwidth from an iPerf3 client through one iPerf3 server. The green line shows bandwidth between two iPerf3 clients and two iPerf3 servers. The red line shows bandwidth through three iPerf3 client/server pairs. Finally, the purple line (top) shows the observed bandwidth through all four iPerf3 client/server pairs simultaneously.

Discussion on LACP. In our experiment above, the switches use Cisco's Link Aggregation Control Protocol (LACP) to aggregate a 4-lambda link; we consider the implementation of the protocol as a black box in our experiment. Although LACP is useful in virtualizing optical fibers, it presents unique challenges for flexible use in our system. Specifically, LACP cannot guarantee fair load balancing—or exclusive lambda access—for links whose cardinality is not a power of two

[2]Bandwidth calculated based on the average throughput before and after scalability on the network is realized for each physical layer reconfiguration request sent.

(*i.e.*, aggregates of 3, 5, 6, and 7 lambdas will be unfairly balanced) [8]. In what follows, we outline two API calls based on Cisco's Internetwork Operating System (IOS) functions to address this issue but leave the implementation details for future work.

First, we acknowledge that we can make use of an IOS source-address based load-balancing call in our API to ensure that packets hash to an optical fiber interface based on the packet's source address. The IOS command is:

```
port-channel load-balance src-ip
```

Next, we will add a layer in our API call to test which IP addresses are mapped to which ports. Thus we can assign IP addresses to only those pairs of hosts at opposite ends of the network such that each pair's traffic will map to the same switch interface. We can repeatedly call the following function, giving different parameters for the source address until we find two addresses that hash to the same physical port or lambda:

```
test ethernet load-balance interface \
port 1 ip x.y.z.A x.y.z.B
```

In this way, we can virtualize fiber optic strands in a network topology where select pairs of hosts will have distinct lambdas and capacity to utilize. It is important to note if one or more links from the link-aggregation fail, traffic which would otherwise be destined for one dead interface will migrate to an available link. In the next section (3.2.3), we propose and evaluate a solution to this problem.

3.2.3 Robustness to Link Outages

Goal: One use case for vFiber is to provide geographic diversity (GD), thus enabling more robust connectivity between a client's host locations. We understand that the robustness of the physical layer is critical for achieving this goal. Therefore, vFiber must respond to physical layer disruptions like fiber-cuts proactively. If it can perform this function, then users can expect a relatively high quality of service from the system. Since the physical layer cannot watch itself and inform the higher layers of the networking stack when a fiber is cut, we build a physical layer monitoring capability for vFiber. This monitor allows the system to reallocate these links to the client in the case of a link outage.

Parameters: The monitor pings a switch every second, scanning the interfaces that are available in vFiber to see whether or not any are disconnected. If a link is registered as disconnected for more than five seconds the monitor alerts vFiber. vFiber needs to translate the IP-address and port information that it receives from the monitor into an IG graph query to update the lambda on an edge corresponding to the fiber-cut. After vFiber finds the edge in the IG associated with the disrupted link, it updates its infrastructure graph (IG) and records the interfaces as unavailable and broken. In case a client was using the interface at the time of the fiber cut, vFiber allocates a new lambda on an alternative interface for that client (if a redundant link is available).

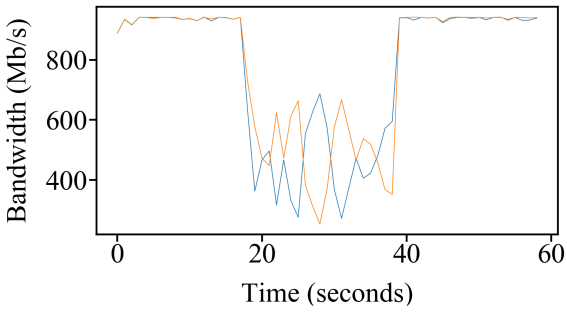


Figure 5: Bandwidth between two pairs of hosts on a network during a fiber-cut, using vFiber.

We chose five seconds as a threshold for classifying a link outage event because of the way that new links are activated, that is—when a switch interface is enabled, its state changes from "administratively down" to "disconnected" until an interface at the other end of the link can successfully communicate with it. At that time, the interface's state changes to "connected." This protocol takes roughly five seconds; therefore if the monitoring threshold were any less than five seconds, we would misclassify newly connected links as fiber-cuts.

Results: We test this capability by disconnecting a link between one switch and a CWDM after vFiber has provisioned it for a client. We show that vFiber can detect this link outage in real-time and reallocate the client's request to a new lambda. Figure 5 shows this process in action.

Two hosts (orange and blue) from a network have been allocated independent lambdas on a strand of fiber, and are using $\sim$1 Gbps each. One of the strands is cut, forcing the two hosts to share a lambda. Thus their bandwidth drops dramatically. vFiber's monitor notices the event and automatically allocates a new lambda for the network users.

The discovery and mitigation of the fiber-cut took $\sim$20 seconds. This time accounts for the 5 seconds that it took vFiber to discover and classify the outage, and the 9 seconds that it took to scale the link back up (as seen in section 3.2.2). Some time is also taken for the monitor to establish a connection with vFiber after the fiber-cut is detected, and for the monitor to relay the relevant information to vFiber. The remainder of the time is spent by vFiber searching the IG for the affected link and updating the edge data. Only after updating the IG, and finding a suitable lambda, does vFiber activate a new interface for the client.

We measured the bandwidth demand of the monitoring capability to be 2.9 kbps using Wireshark [31]. This overhead is less than $\%0.003$ of the available bandwidth to the switch (1 Gbps). Thus we are confident that the cost of monitoring will not adversely affect the performance of vFiber. Thus, we have demonstrated vFiber's ability to detect fiber-cuts, and to provide an alternate link if one is available.

3.2.4 Robustness to Server Failures

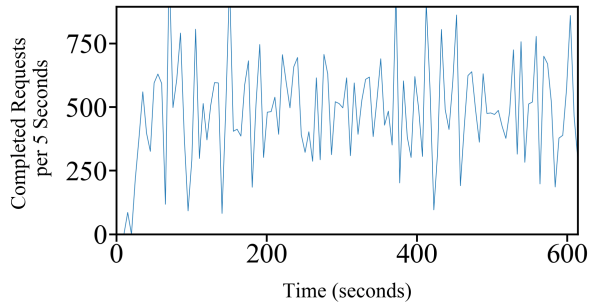


Figure 6: DC handling ∼100 requests per second for 10 minutes under no failure scenario.

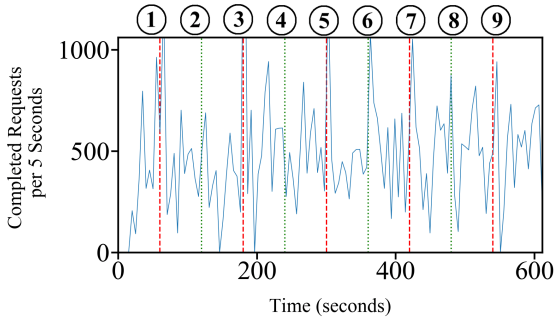


Figure 7: Requests completed per 5-seconds with random server failures. (1) Server 2 fails. (2) Server 2 recovers. (3) Server 1 fails. (4) Server 1 recovers. (5) Server 2 fails again. (6) Server 2 recovers. (7) Server 1 fails. (8) Server 1 recovers. (9) Server 3 fails.

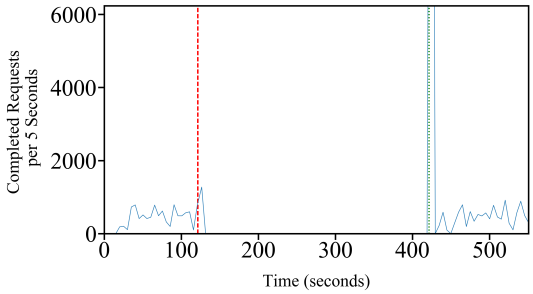


Figure 8: Request completed per 5-seconds before an after a majority of the cluster has been offline for 5 minutes. Two of the three servers are shut down at 120 seconds (red line) and then recovered at 420 seconds (green line).

Goal: A primary requirement for service providers is to guarantee five-nines availability to their customers [18] (*i.e.*, available 99.999% of the time). Likewise, the vFiber system must be highly available to virtualize fiber optic strands flexibly and for providing *reactive* reliability for enterprise networks. Two positive consequences of such a highly-available system are that it enables the treating of failures as a normal situation to handle [28] and that a high level of service can be guaranteed through service level agreements (SLA), if needed, with low risk to the enterprise.

Parameters: To this end, we evaluate the availability of the DC in vFiber. Using our testing capability, we send ∼100 physical (re)configuration requests per second from a single network administrator. The exact number of requests per second is generated based on the Poisson distribution, following the prior effort on modeling Internet traffic [23]. A host machine makes requests to vFiber by running the capability as a process. The host sends the requests to a random server in the DC, then waits for a response from vFiber before terminating. If no servers are available, then the capability will also end. However, we assume that there is always at least one server active in our experiments. Furthermore, to ensure that no process terminates via a timeout, we set the request timeout in the capability to be longer than the length of the experiment.

When a server in the DC receives a request, it can process that request as long as the majority of the servers are up. PySyncObj handles the consensus protocol on a server-only address space; thus, any server which is not the leader of the RAFT-based DC will always know the identity of the leader and will forward any requests to that leader. We test the availability of the servers in the DC under three scenarios (described below). Subsequently, we measure the request throughput as perceived by the network administrator. We calculate the request throughput based on the number of requests sent and responses received during five minutes. Next, we parse the number of completed requests at each time t into mutually exclusive 5-second bins and report measurements based on these bins.

No Failures. First, to establish a baseline, we measure the number of requests processed when all servers in the DC are available. Figure 6 depicts the fact that the number of requests processed per five-second bin oscillates around 500. We note that it takes ∼8 seconds for a server in the DC to process a request; therefore we see zero completed requests initially. After 8 seconds, the DC consistently returns responses with no more than 1 second of delay between newly completed requests and an average throughput of 95.7 requests per second.

Random Periodic Failures. Given that we tested the DC under the baseline scenario (where all servers are available), we next introduce periodic failure and recovery of a single random server (figure 7). In this test we run the cluster for 10 minutes, periodically failing a randomly chosen server every 2 minutes. We then introduce the recovery of the failed server after it has been down for one minute. In figure 7 red lines underneath odd-number annotations show the instants that a server terminates, while green lines underneath even-number annotations show the instants at which the server recover. From figure 7, we observe that request throughput stays roughly the same when compared with the baseline scenario. Altogether, vFiber completed a total of 58,588

requests in 614 seconds with an average throughput of 95.4 requests per second.

Dead-Stop Failures. Finally, we would like to see how quickly vFiber can recover after a majority of the servers have gone offline. Furthermore, we would like to determine if there is a correlation between the duration of the failure and the recovery time. To this end, we introduce a dead-stop failure scenario in which a majority of the servers (*i.e.*, 2 out of 3) in the DC fail. Then, we observe vFiber's ability to recover. We repeated this experiment for downtimes of 1 to 5 minutes and noticed that after the servers recovered, vFiber was easily able to process the back-logged requests, and then proceed to handle new requests. We show results for the 5-minute failure case in figure 8. The choice of a 5-minute failure period is consistent with the requirement of five-nines availability, which corresponds to roughly five minutes of system downtime in a year.

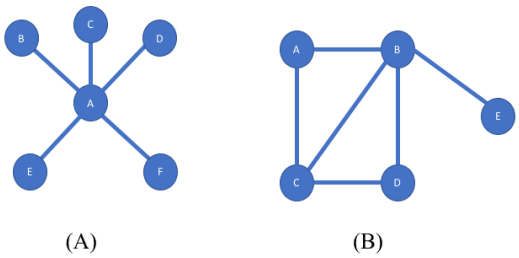
Results: To complement figures 6, 7, and 8, we summarize the aggregate statistics from the availability experiments in Table 1. From this table, we see that the median number of requests completed per-second in our No Failure and Random Periodic scenarios differ by only three requests. We also see that the average completion rate for No Failure and Random Periodic are within one request-per-second of each other. This insight strengthens our confidence in the availability of the system. We also see that the maximum number of requests processed per-second was highest in the Dead-stop failure scenario because requests *accumulate* at the last living server until another peer comes back online. Once enough peers are available to hold quorum, they immediately proccess the backlogged requests. Overall, we see that the implementation of RAFT that we use in our DC is highly available under different failure scenarios.

	Median	Average	S.D.	Max
No Failures	101.0	98.96	72.0	329
Random Periodic	98.0	98.8	91.4	906
DS pre-failure	89.0	93.2	145.0	1273
DS post-failure	93.0	313.6	2137.1	24092

Table 1: Aggregate statistics for each availability experiment based on request completion rate (requests per second). DS refers to the Dead-stop scenario with a 5-minute failure.

3.2.5 Diverse Topologies, Demands, and Cluster Sizes

Goal: Until now, we have studied various aspects of vFiber in isolation (responsiveness, availability, scalability). Now it is time to see how the system operates as a whole. To this end, we seek a multidimensional analysis of the system. This analysis spans three characteristics of interest: (1) How efficiently does vFiber perform across different topologies? (2) How does the performance of vFiber vary with demand? (3) How does the performance change while varying the size



(A) (B)

Figure 9: (A) A star graph with six nodes and five edges. (B) A random graph with five nodes and six edges.

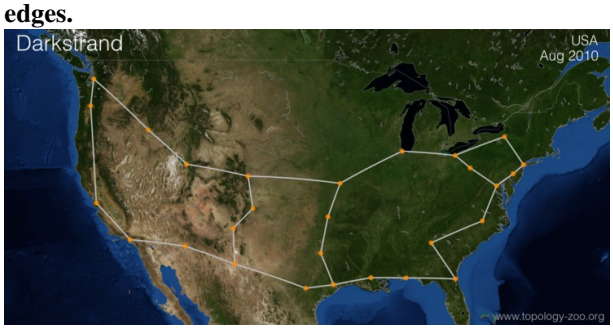


Figure 10: Darkstrand, a real fiber deployment from the Internet Topology Zoo. This graph has twenty-seven nodes and thirty edges.

of the DC?

Throughout the previous experiments, we considered vFiber was running on dumbbell topology. We will now relax this constraint by simulating several topologies: a star topology with six nodes (figure 9 A), a random topology with six nodes and five edges (figure 9 B), and a real fiber deployment topology with twenty-seven nodes and thirty edges called Darkstrand (figure 10). We chose to simulate vFiber on the Darkstrand because it is a real-world fiber deployment that was publicly available through the Internet topology zoo [25]. We stress test vFiber on these topologies by sending the DC different numbers of client requests, up to the maximum number of requests that could be satisfied based on the limited resources in the underlying topology. Furthermore, we evaluate the performance of vFiber—across different topologies, and with variable demand—while running on a variable number of servers.

Parameters: We simulate the time to allocate a single link along a path using the 95th percentile link activation time of 35.5 seconds (from the benchmark experiments in section 3.2.1). The number of lambdas available at each link is 40. This parameter is consistent with the availability 40-Channel Dense Wavelength Division Multiplexers (DWDMs) used in Internet backbone links [11]. We consider DC clusters of size three to nine. We do not look at DC groups of a size larger than nine because we assume it is unlikely that five

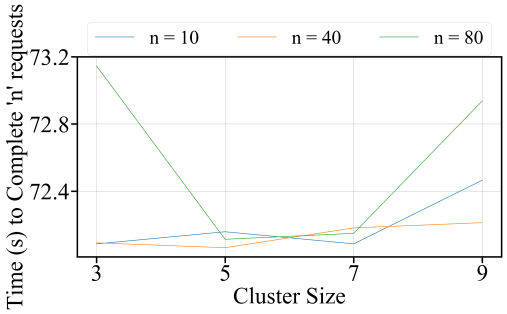


Figure 11: Evaluation of vFiber on the star topology considering 10, 40, and 80 requests sent to DC clusters of various sizes. The trend for the graph is an increase in completion time with respect to cluster size, with an exception of 80 requests processed by a cluster of three severs. This is likely due to non-deterministic request processing by the cluster.

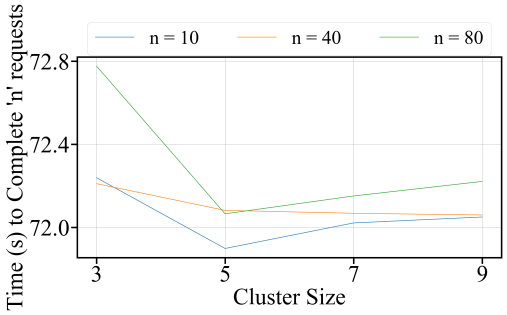


Figure 12: Evaluation of vFiber on the random topology considering 10, 40, and 80 requests sent to DC clusters of various sizes. The trend for this graph is a gradual increase in completion time with respect to cluster size with the exception of processing time from three to five servers. This could indicate that the servers are balancing requests more efficiently on five servers than three.

out of the nine servers will ever fail simultaneously. The probability of this event decreases by hosting the servers in physically diverse locations. Thus, we host the servers on one of three distinct physical machines. These machines can be thought of as racks in a data center. They are all connected to the same local area network (LAN) switch.

A client requests a path by choosing a source and destination within a given topology. The client chooses this request at random, by sampling from all of the possible source-destination pairs within the topology with equal probability. These paths could be single-hop or multi-hop. To assess the scalability of vFiber we send 10, 40, and 80 requests at a time. If we allowed bursts of more than 80 requests, then many of the requests would be denied due to limited lambdas on the links. We are looking for insight into how quickly vFiber clusters of different sizes can allocate network paths on different topologies. Therefore we need the majority of requests to succeed. We chose to send bursts of 10 and 40 requests as well to gain insight into what relationship existed might exist between request volume on different topologies and cluster sizes. Each request sent originates from distinct client processes. We measure and report the time for vFiber to complete the sum of client requests.

For any given topology, we include the smaller set of requests in every larger set of requests. We also send the same sets of requests to all of the DC clusters. For instance, if we send a burst of n requests to a cluster of size 3, then we will send the same n requests to clusters of size 5, 7, and 9 too. Furthermore, when we send a burst of 40 requests, we ensure that the first 10 were the same 10 that we sent in a previous experiment. Similarly, the first 40 requests in a group of 80 we send are those 40 that we used earlier. This staging of requests ensures that the variations we see between cluster sizes and groups of requests are not due to larger clusters receiving easier sets of requests (*e.g.,* requests for shorter paths), or to larger groups of processes sending easier requests than those groups of smaller numbers.

When the DC processes the requests, it first checks for availability of the resource within the infrastructure graph (IG). It handles concurrent access to the IG with the locking protocol described in section 3.1. When all locks are acquired, and sufficient capacity is available, the DC activates the links along the requested path. If there isn't enough capacity to satisfy the request, then the request is rejected.

Results: Figure 11 shows the various completion times for requests on the star topology. All of the requests for each cluster and request volume competed within 73.15 seconds. Interestingly, that maximum completion time for this topology occurred when we evaluated 80 requests on the smallest cluster size, three. We hypothesize that this could be an outlier that occurs due to the nondeterminism in processing all of these requests in parallel and where clients are choosing servers to send their requests to at random. The general trend of the graph for different request sizes is stable for clusters of size five and seven, with less than a tenth of a second of difference between any set of requests evaluated on these clusters. When we scale the cluster up to nine servers, however, the completion time for 80 requests goes from 72.1 to 72.9 seconds. This time is still less than the 73.14 seconds that a cluster of three servers completed the same 80 requests. Again, we attribute this observation to the nondeterminism in request processing.

In figure 12 we see the results evaluated on the random topology. vFiber clusters of different sizes completed all of the requests within 72.7 seconds for this topology. Again, we observed that the maximum completion time occurred for 80 requests on a cluster of three servers. When vFiber ran on five servers, the completion time for all of the different request batches dropped. This anomaly could indicate that with five servers, this set of vFiber clients might have been balancing their requests more equitably among the servers (recall that

servers can perform some actions, such as parsing requests, independently and only consult the leader of the cluster when making changes to the IG). From clusters of size five to nine, the general trend is a steady, gradual increase in completion times.

The request completion times were ∼72 seconds when evaluated on both the star and random topology. This similarity occurs because the longest path that was requested in on both topologies was a path of length two. This time is consistent with the time that it takes to activate the two links sequentially. There appears to be a more significant variation in completion times for nine servers on the star topology than similar requests and servers on the random topology. We hypothesize that this is due to more competing requests for overlapping resources in the star topology, where any two request requests will share at least one node (the center of the star).

Figure 13 shows the results from testing vFiber on a real fiber deployment topology. The completion times for requests, in this case, are much larger (∼358 seconds). This difference occurred because Darkstrand topology had many more edges than the previous topologies, and those links were activated sequentially for every request. In spite of the delay associated with link activation, the variation of completion time across different cluster sizes is relatively small (∼1 second). One interesting characteristic of the results evaluated on Darkstrand is that request batches with a volume of 40 are consistently completed as much as 0.2 seconds more quickly than request batches of volume 80 until the cluster reaches a size of nine. Another observation is that completion time for 10 requests drops from a high of 358.6 seconds for a cluster of size three to 357.7 seconds for a cluster of size seven. We attribute both of these observations to nondeterministic request handling. The trend for request processing times for 40 and 80 requests is a gradual increase with cluster size.

Overall, we noticed that there was no monotonically increasing behavior for completion time with respect to cluster size or request volume for any topology. We hypothesize the following explanation for this result. There are two sources of nondeterminism that arise in processing a set of requests. First, the servers process all of the client requests in parallel. Second, nondeterminism appears in the face of client requests generation, that is, each client sends its request to a random server. Therefore, the completion time depends on the scheduling of concurrent threads on the server, and the likelihood of a client choosing the leader of the cluster when sending its requests. For clusters of size nine or fewer, these effects can have a more significant impact on the completion time than the number of servers running in the cluster. These variations again, are consistently less than 1 second and therefore do not cause us great concern for the performance of the system in this setting. There is more room to improve on elsewhere in the system, by efficiently provisioning non-adjacent links of a path in parallel. We leave this optimization to future work.

4. RELATED WORK

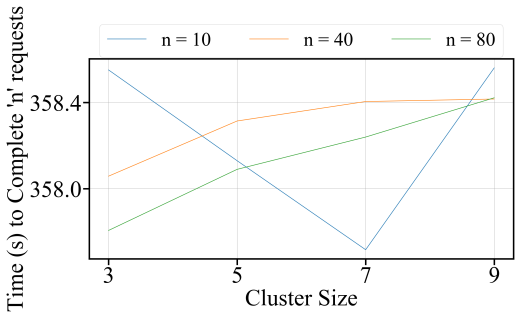


Figure 13: Evaluation of vFiber on the Darkstrand topology considering 10, 40, and 80 requests sent to DC clusters of various sizes. With the exception of the batch of 10 requests, the trend is a gradual increase in completion time with respect to cluster size. This exception is likely due to nondeterministic request handling by the cluster.

Optimizing WAN Traffic. There has been significant recent work in optimizing the utilization of expensive WAN resources for unpredictable workloads *e.g.,* between data centers, and increasing the total capacity of the network. Hong *et al.* describe SWAN to achieve high bandwidth utilization in Software-Defined Wide Area Networks [20]; B4 [21] describes a similar system for Google. Xu *et al.* define a diverse routing optimization algorithm that network planners can use to provision diverse paths thereby mitigating potential *choke-points* in a network [36]. BwE by Kumar *et al.* utilizes a fixed network capacity between data centers and optimizes the capacity allocation among different applications [26]. Jin *et al.* describe a joint optimization of optical and routing substrates using OWAN [22]. Our effort is complementary: prior works operate under the constraint of fixed network capacity between endpoints. With a system for virtualizing optical fibers, we posit a number of benefits to prior works including enhanced resilience to outages, greater flexibility for accommodating surges in bandwidth demand, and even an increase in revenue by leasing their fiber during times when the internal network demand is low. The notion of reconfigurability and fan-out agility in data centers are explored in [17].

Increasing Capacity. Other recent work has addressed the challenge of increasing network capacity. For instance, Laoutaris *et al.* propose NetStitcher which allows clients to purchase additional bandwidth from a network provider [27]. Singh *et al.* takes a rate-adaptive approach with RADWAN to increase capacity on fiber during events that would otherwise be outages [33]. Their system monitors the Signal-to-Noise (SNR) ratio of an optical link and reduces the link's capacity when the SNR drops below a threshold.

The only other system to use dark fiber to increase capacity is GreyFiber [13]. While GreyFiber made significant

progress towards providing Connectivity-as-a-Service, we differ from GreyFiber is several novel ways. First, we architect our system with scalability and reliability in mind. Hence, we distribute the components and infrastructure used by our system. Second, we operate at the scale of an enterprise/provider and hence vFiber is not a fiber marketplace. Third, we introduce new testing and benchmarking capabilities to test link installations by our system.

5. SUMMARY

vFiber seeks to virtualize the under-utilized dark fiber in today's enterprise networks. We posit that the virtualization of optical fibers will provide new means for efficiently orchestrating and operating physical infrastructure assets. We have addressed three critical challenges in building vFiber. (1) We distribute the components of vFiber using RAFT-based consensus and show that vFiber is resilient to different failure scenarios. (2) We have developed a capability to test and benchmark the responsiveness of network hardware to configuration changes in vFiber. (3) Finally, we have shown how to virtualize dark fiber in a lab testbed to scale bandwidth from 942 Mbps to 3,763 Mbps in 9 seconds between endpoints. We also developed a monitoring capability, which vFiber can use to respond to link failure events automatically. We also tested the scalability of our distributed server environment by simulating fulfillment of requests on several distinct topologies while varying the number of servers in the DC cluster.

6. REFERENCES

[1] Amazon EC2 Spot Pricing. https://aws.amazon.com/ec2/spot/pricing/.

[2] Dark Fiber IRU, 2017. https://kentuckywired.ky.gov/SiteCollectionDocuments/Bluegrass%20Network%20-%20IRU%20Agreement%20(Executed).pdf.

[3] IRU between Level3 and Comcast. https://corporate.comcast.com/news-information/news-feed/comcast-extends-national-fiber-infrastructure.

[4] IRU Swaps. http://chogendorn.web.wesleyan.edu/excessive.pdf.

[5] Media Advertisement. https://www.sec.gov/Archives/edgar/data/1111634/000114420411019057/v214475_10k.htm.

[6] PySyncObj. https://github.com/bakwc/PySyncObj.

[7] The Fiber-Optic "Glut" – in a New Light. http://www.bloomberg.com/news/articles/2001-08-30/the-fiber-optic-glut-in-a-new-light.

[8] Understanding EtherChannel Load Balancing and Redundancy on Catalyst Switches. https://www.cisco.com/c/en/us/support/docs/lan-switching/etherchannel/12023-4.html.

[9] Who's Using AWS. https://www.contino.io/insights/whos-using-aws.

[10] Why the Glut In Fiber Lines Remains Huge. http://www.wsj.com/articles/SB111584986236831034.

[11] F. Corporation. https://www.finisar.com/sites/default/files/downloads/100ghz_dwdm_40_chs_mux_demux_1ru_module_product_specification_rev_a.pdf.

[12] G. Donahue. *Network warrior*. "O'Reilly Media, Inc.", 2007.

[13] R. Durairajan, P. Barford, J. Sommers, and W. Walter. GreyFiber: A System for Providing Flexible Access to Wide-Area Connectivity. *1807.05242*, 2018.

[14] R. Durairajan, P. Barford, J. Sommers, and W. Willinger. InterTubes: A Study of the US Long-haul Fiber-optic Infrastructure. In *Proceedings of ACM SIGCOMM*, 2015.

[15] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet Advertising and the Generalized Second-Price Auction: Selling Billions of Dollars Worth of Keywords. *The American economic review*, 2007.

[16] B. Evans. Microsoft cloud hits superscale as huge customers migrate mission-critical sap workloads to azure. https://tinyurl.com/sapAzure, 2018.

[17] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 216–229. ACM, 2016.

[18] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *ACM SIGCOMM*, 2016.

[19] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[20] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving High Utilization with Software-driven WAN. In *ACM SIGCOMM*, 2013.

[21] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a Globally-deployed Software Defined WAN. In *ACM SIGCOMM*, 2013.

[22] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford. Optimizing Bulk Transfers with Software-Defined Optical WAN. In *ACM SIGCOMM*, 2016.

[23] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido. A nonstationary poisson view of internet traffic. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1558–1569. IEEE, 2004.

[24] S. Keck. I want to believe - alien wavelengths. https://forums.juniper.net/t5/Service-Provider-Transformation/I-Want-to-Believe-Alien-Wavelengths/ba-p/288728, 2016.

[25] S. Knight, H. X. Nguyen, N. Falkner, R. A. Bowden, and M. Roughan. The Internet Topology Zoo. In *IEEE JSAC*, 2011.

[26] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, et al. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *ACM SIGCOMM*, 2015.

[27] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter Bulk Transfers with NetStitcher. In *ACM SIGCOMM*, 2011.

[28] T. Limoncelli, J. Robbins, K. Krishnan, and J. Allspaw. Resilience Engineering: Learning to Embrace Failure. *Communications of the ACM*, 2012.

[29] R. Nuijts. Fom (figure of merit) for dark fiber links. Customer Empowered Fiber Networks workshop http://archiv.ces.net/events/2009/cef/p/nuijts.pdf, 2009.

[30] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, 2014.

[31] A. Orebaugh, G. Ramirez, and J. Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[32] SDxCentral. SDxCentral. Software Defined Everything Part 3: SDx infrastructure. https://www.sdxcentral.com/cloud/definitions/software-defined-everything-part-3-sdx-infrastructure, 2017.

[33] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill. Radwan: Rate adaptive wide area network. 2018.

[34] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. iperf: Tcp/udp bandwidth measurement tool. 01 2005.

[35] W. Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders. *The Journal of Finance*, 1961.

[36] D. Xu, G. Li, B. Ramamurthy, A. Chiu, D. Wang, and R. Doverspike. On Provisioning Diverse Circuits in Heterogeneous Multi-layer Optical Networks. In *Computer Communications*, 2013.