Optimizing Bulk Transfers with Software-Defined Optical WAN

Xin Jin†, Yiran Li⋆, Da Wei⋆, Siming Li^, Jie Gao^,
Lei Xu◦, Guangzhi Li×, Wei Xu⋆, Jennifer Rexford†
†Princeton, ⋆Tsinghua, ^Stony Brook, ◦Sodero Networks, ×AT&T Labs

Abstract

Bulk transfer on the wide-area network (WAN) is a fundamental service to many globally-distributed applications. It is challenging to efficiently utilize expensive WAN bandwidth to achieve short transfer completion time and meet mission-critical deadlines. Advancements in software-defined networking (SDN) and optical hardware make it feasible and beneficial to quickly reconfigure optical devices in the optical layer, which brings a new opportunity for traffic management on the WAN.

We present Owan, a novel traffic management system that optimizes wide-area bulk transfers with centralized joint control of the optical and network layers. Owan can dynamically change the network-layer topology by reconfiguring the optical devices. We develop efficient algorithms to jointly optimize optical circuit setup, routing and rate allocation, and dynamically adapt them to traffic demand changes. We have built a prototype of Owan with commodity optical and electrical hardware. Testbed experiments and large-scale simulations on two ISP topologies and one inter-DC topology show that Owan completes transfers up to $4.45\times$ faster on average, and up to $1.36\times$ more transfers meet their deadlines, as compared to prior methods that only control the network layer.

CCS Concepts

•Networks → Layering; Network resources allocation; Network control algorithms; Traffic engineering algorithms; Network management;

Keywords

Software-defined networking; wide area networks; optical networks; bulk transfers; cross-layer network management

1. INTRODUCTION

Many globally-distributed applications have bulk data to transfer over the wide-area network (WAN). For example, search engines need to synchronize search indexes between data centers; financial institutions need to backup everyday transactions over remote sites; media companies need to deliver high-definition video content to multiple distribution areas. Bulk transfers have large size (terabytes to petabytes) and account for a big proportion of traffic, e.g., 85–95% for some inter-datacenter (inter-DC) WANs [1, 2, 3, 4].

Optimizing bulk transfers is important to network operators. Although bulk transfers are not as delay-sensitive as interactive traffic like web queries, it is beneficial and sometimes necessary to finish them quickly, as it is essential for service quality. For instance, the time to finish search index synchronization directly impacts the search quality [3]. Furthermore, some bulk transfers are associated with deadlines, e.g., timely delivery of high-definition video content to some cities by a certain time is the key for business success [3, 4]. It requires network operators to carefully schedule these transfers in order to meet their deadlines.

Existing practice performs traffic engineering (TE) in the network layer. Traditional WAN designs over-provision the network with 30–40% average network utilization, in order to handle traffic demand changes and failures [1]. Recent designs like Google B4 and Microsoft SWAN leverage software-defined networking (SDN) to directly control the network with a global view [1, 2, 3, 4]. They use a global TE to dynamically change routing and rate allocation, so that they can accommodate more traffic and meet more deadlines. They all assume a fixed network-layer topology.

In a modern WAN, the network-layer topology is constructed over an intelligent optical layer.[1] By reconfiguring the optical devices, the operator can dynamically change the network-layer topology. Figure 1 shows an example of a modern WAN infrastructure—the Internet2 network [5]. The network-layer link between SEA and LA in Figure 1(b) is implemented by an optical circuit that traverses multiple optical switches in the optical layer in Figure 1(a). In practice, a WAN router is connected to an optical switch called Reconfigurable Optical Add-Drop Multiplexer (ROADM) via short-reach wavelength. To connect two WAN router ports, the operator needs to properly configure the ROADMs along the path to establish an optical circuit. By changing the circuits in the optical layer, operators can change which two router ports are connected.

[1]A WAN network is a packet-switched network, which is usually built on top of an optical network. In this paper, the WAN network is referred as the network layer, and the optical network is referred as the optical layer.

SIGCOMM '16, August 22-26, 2016, Florianopolis, Brazil
 ISBN 978-1-4503-4193-6/16/08…$15.00
DOI: http://dx.doi.org/10.1145/2934872.2934904

(a) Internet2 physical infrastructure. (b) Internet2 IP layer topology.

Figure 1: WAN infrastructure example (Internet2 [5]).

Traditionally, the optical layer is reconfigured on a long time scale, e.g., weeks to months, or even years. The major reason is the labor and risk involved in the reconfiguration: operators need to deal with sophisticated configurations, including IP, BGP and access control list (ACL), and they have to perform operations on many routers without consistent configuration interfaces, which is tedious and error-prone. Also, after a optical layer reconfiguration, traditional distributed routing protocols may be slow to converge.

In this paper, we present Owan, a new traffic management system that optimizes wide-area bulk transfers with centralized joint control of the optical and network layers. We leverage two technology trends. The first is SDN that allows direct control of network devices and simplifies network management; the second is modern ROADM devices that allow fast remote reconfigurations (e.g., provisioning a circuit in tens to hundreds of milliseconds [6]). Owan orchestrates bulk transfers in a centralized manner. It computes and implements the optical circuit configuration (the optical circuits that implement the network-layer topology) and the routing configuration (the paths and rate allocation for each transfer) to optimize the transfer completion time or the number of transfers that meet their deadlines.

A major technical challenge for Owan is that the optimization problem includes a large number of constraints, some of which are integral. Most TE algorithms assume a given topology and only compute the network-layer configuration [1, 2, 3, 4]. While there is research on reconfigurable optics, these projects focus on data center networks under the assumption of specific optical devices (e.g., MEMS switches) and certain topologies [7, 8, 9, 10, 11]. However, there are three unique constraints on WANs that do not present in data centers: ROADMs, regenerators and arbitrary topology. We accommodate ROADMs in our formulation which are typically used as building blocks for WANs. We take into account *regenerators*, which regenerates optical signals after certain distance. Also, we do not make any assumptions of the optical-layer topology, allowing it to be irregular.

The key idea to solve the optimization problem is to do a probabilistic search in the search space with simulated annealing. At each time slot, we use the current topology as the starting point, and use simulated annealing to find a topology with the highest total throughput. There are two major benefits. First, searching for a topology, instead of the entire optical and routing configurations, substantially reduces the search space. Second, using the current topology as the starting point in simulated annealing allows us to find a target topology that is close to the current topology but has higher throughput. This significantly reduces the number of changes we need to make in the optical layer, in order to update the topology.

We build a Owan prototype using commodity optical and electronic hardware. The prototype has nine sites and emulates the Internet2 topology in Figure 1. We conduct extensive evaluations through both testbed experiments with our prototype that emulates the Internet2 network and large-scale simulations with data from an ISP network and an inter-DC network. Our results show that Owan improves the transfer completion time by up to $4.45\times$ on average and $3.84\times$ at the 95th percentile, as compared to prior methods that only control the network layer. Furthermore, Owan allows up to $1.36\times$ more transfers to meet their deadlines and up to $2.03\times$ more bytes to finish before their deadlines.

2. BACKGROUND AND MOTIVATION

We focus on bulk transfers on the WAN. Our design applies to both private WANs (e.g., inter-DC WANs) and public WANs (e.g., provided by ISPs). Large ISPs usually own both the public WAN and the underlying optical network. They can directly use Owan to manage their networks. Small ISPs and private WANs usually lease optical circuits from optical-network providers. In such case, they would need an interface with the optical-network operator to change the optical configurations together with Owan. Furthermore, Owan requires to know the traffic demand and to control the rate of each transfer, which can be assumed for inter-DC networks but not for ISP networks. To use Owan in ISP networks, ISPs can provide a bulk transfer service to their clients. This service has an interface for clients to submit transfer requests that contain traffic demand information and inform clients data rates they can use for their transfers. Before we introduce Owan, we give some background on WAN infrastructure and a motivating example to show the benefits of joint optimization of the optical and network layers.

2.1 Background on WAN Infrastructure

A typical WAN infrastructure consists of network routers, optical devices, and fibers. A bulk transfer enters a WAN on a router from an access network (e.g., a data-center network or a metro network) or other autonomous systems, passes through intermediate routers to the destination router, and leaves the network. Since a WAN link is a circuit in the optical layer, packets over any WAN link actually traverse multiple optical switches in the form of optical signals.

Optical layer: An optical network consists of ROADMs connected by fiber pairs. Today's commercial ROADM technology is able to support 80 or more wavelengths per fiber pair and 40 Gbps (100 Gbps, and higher with high-order modulations and digital coherent receivers) per wavelength, which leads to 3.2 Tbps (8 Tbps, and even higher capacity) per fiber pair. A router port can connect to a ROADM port with a tunable optical transponder via standard short-reach wavelength. The tunable optical transponder is able to tune the standard wavelength to another specific wavelength. The ROADM can switch the wavelength to an output port or an

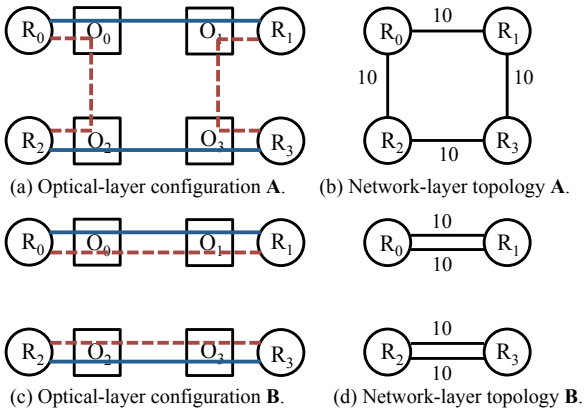


Figure 2: Example of topology reconfiguration. Different line types/colors in (a) and (c) denote different wavelengths. A router port or a wavelength carries 10 bandwidth units. By reconfiguring how wavelengths are switched in ROADMs (rectangle nodes), we can change how routers (circle nodes) are connected. (b) and (d) show the resulting network-layer topologies.

add/drop port connected to another router port. Commercial ROADMs can be reconfigured in hundreds of milliseconds and future ROADMs can reduce the time to tens of milliseconds and even lower [6, 12, 13].

Due to optical signal loss and some non-linear impacts on optical signals, a wavelength normally has limited transmission range, which is called *optical reach*. When an optical signal transmits beyond the optical reach, a regenerator device is required to regenerate the signal. In order to dynamically establish optical circuits on demand, operators usually pre-deploy some regenerators at certain concentration sites such that between any two ROADMs, there is at least one path using those limited regenerator concentration sites to satisfy the optical reach constraint [14, 15].

Network layer: A router is usually co-located with a ROADM. Customer-facing router ports are connected to customer equipment, such as data-center routers or metro-network routers; network-facing router ports are connected to ROADM ports via standard short-reach wavelength. A network-layer link is implemented by an optical circuit. By reconfiguring the optical layer, we can change the connectivity of router ports in the network layer, i.e., the network-layer topology.

Topology reconfiguration example: We use the example in Figure 2 to illustrate how the network-layer topology can be reconfigured with optical devices. In the network, we have four routers R_0-R_3 and four ROADMs O_0-O_3. Each router has two WAN-facing ports. In configuration **A**, each ROADM converts electrical packets from two router ports to different wavelengths and sends them to different ROADMs. For example, O_0 sends the solid/blue wavelength to O_1 and the dashed/red wavelength to O_2 (Figure 2(a)). In the resulting network-layer topology, each router is connected to two other routers (Figure 2(b)). In configuration **B**, a ROADM multiplexes two wavelengths on to the same fiber and is connected to only one other ROADM. For example, both wavelengths at O_0 are multiplexed to the fiber to O_1, with the fiber between O_0 and O_2 carrying no wavelengths (Figure 2(c)). In the network-layer topology, a router has both ports connected to another router (Figure 2(d)), doubling the capacity between R_0 and R_1 from configuration **A** (Figure 2(b)).

2.2 Motivating Example

Topology reconfiguration opens a new opportunity for optimizing bulk transfers. Existing approaches assume a given and fixed network-layer topology, and optimizes bulk transfers by controlling the routing and/or the rate of each transfer [1, 2, 3, 4]. We provide a motivating example to show that by reconfiguring the topology we can significantly reduce average transfer completion time (Figure 3).

In the example, we have four routers R_0-R_3 similar to Figure 2. We only show the network-layer topology and omit the ROADMs for brevity. We have two transfers, F_0 and F_1. Each transfer has 10 units of traffic to send. Plan A only controls routing (Figure 3(a)). It uses the shortest paths and the two transfers are transmitted simultaneously. The average transfer completion time is 1 time unit.

We can do better if we can control the sending rates too. Plan B (Figure 3(b-c)) schedules F_0 first with two paths, R_0-R_1 and R_0-R_2-R_3-R_1, and let F_0 wait. It takes 0.5 time unit for F_0 to finish. Then F_1 starts and takes another 0.5 time unit to finish. The average transfer completion time is $\frac{0.5+1}{2} = 0.75$ time unit, or $1.33\times$ faster than Plan A.

Note that both Plan A and B waste available network capacity, in different ways. Plan A leaves bandwidth unused while Plan B has two-hop routing paths. We can do better if we control the network-layer topology. Plan C reconfigures the topology (Figure 3(d)). Two router ports on R_0 are all connected to R_1. Now both F_0 and F_1 can enjoy a bandwidth of 20 units and finish within 0.5 time unit. Plan C is $2\times$ faster than Plan A, and $1.5\times$ faster than Plan B.

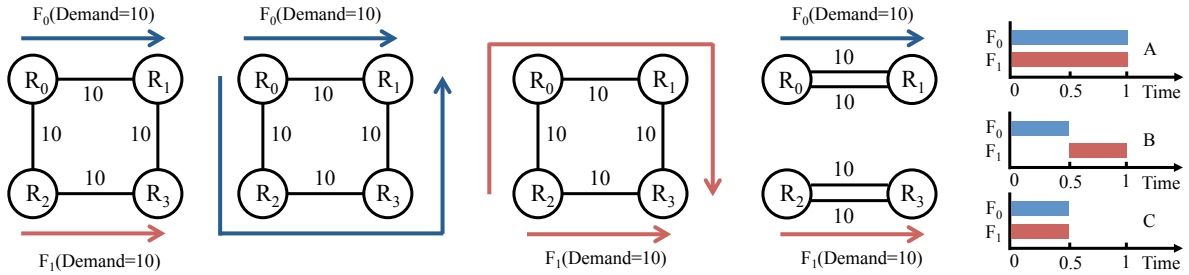
3. OWAN DESIGN

In this section, we first provide an overview of Owan. Then, we present the algorithms to compute the optical and routing configurations to optimize bulk transfers. Finally, we describe how to deal with updates and some practical issues.

3.1 Owan Overview

Owan is a centralized system that orchestrates bulk transfers on the WAN. Figure 4 shows the system architecture. Abstractly, Owan works as follows.

1. Clients submit bulk transfer requests to the controller. A request is a tuple $(src_i, dst_i, size_i, deadline_i)$ that denotes the source, destination, size, and deadline (optional) of transfer request i.
2. The controller has a global view of the physical topology and transfer requests. It computes the optical circuits that build the network-layer topology, the paths and the sending rates for transfers.
3. The controller sends the rate allocation to each client and clients enforce rates on their applications. The controller directly programs routers and ROADMs to set up rout-



(a) Plan A. (b) Plan B-1. (c) Plan B-2. (d) Plan C. (e) Time series.

Figure 3: Example of optimizing bulk transfers. (a) Plan A transmits F_0 and F_1 simultaneously. (b-c) Plan B first transmits F_0 and then F_1. (d) Plan C reconfigures the topology and has the lowest average transfer completion time. (e) Time series to show the transfer completions of these three plans.

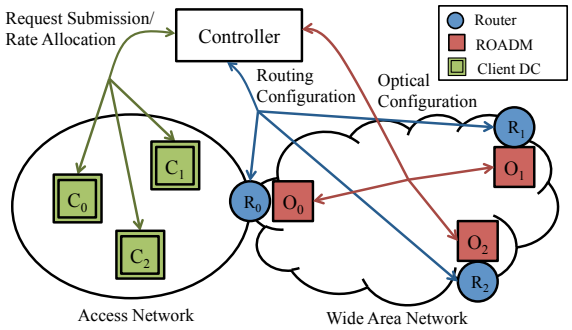


Figure 4: System architecture.

ing paths and optical circuits. On public WANs, the controller also needs to enforce rates with rate limiters on routers in case clients do not properly enforce these rate limits on their applications.

The above process is an online process. We divide time into time slots. A time slot is much longer than the time to reconfigure the network and adjust sending rates, i.e., a few minutes vs. hundreds or thousands of milliseconds. The major job for the controller is to compute the configurations at each time slot to optimize bulk transfers.

3.2 Computing Network State

All the configurations are denoted as network state. We precisely define the network state and formulate the problem as follows. Table 1 summarizes the key notations.

Network state: A WAN is represented as a graph $G = (V, E)$ where V is the set of all sites and E is the set of links in the network-layer topology. A site v consists of one ROADM, a set of pre-deployed regenerators (could be zero), and zero or one router.

A network state NS is a configuration of the devices in the WAN. It includes the optical configuration OC and the routing configuration RC. OC is the set of optical circuits to be configured on the optical devices, which builds the network-layer topology. A network-layer link between u and v is implemented by a circuit oc_{uv} in the optical layer. RC is the set of routing configurations to be installed on routers (and end hosts if rates are enforced by clients). Specifically, the routing configuration of a transfer f, denoted by rc_f, includes its routing paths and rate limits for each path.

Symbol	Description
V	The set of sites.
E	The set of network-layer links.
G	The network-layer topology.
F	The set of transfers.
NS	The network state $NS = (OC, RC)$.
OC	The set of optical circuits.
RC	The routing configuration $RC = \{rc_f \mid f \in F\}$.
p	A routing path.
$r_{f,p}$	The rate of transfer f on routing path p.
rc_f	The routing configuration of f: $\{r_{f,p} \mid p \in P_f\}$.
θ	The capacity of a wavelength.

Table 1: Key notations in problem formulation.

Problem formulation: The problem of finding the optimal network state is an online optimization problem. There are a stream of new transfers arriving at the system. At each time slot, we need to compute a network state NS that optimizes the average transfer completion time or the number of transfers that meet their deadlines. The problem has the following constraints.

1. The number of router ports connected to ROADM ports at each site v is limited, denoted by fp_v. This constrains the total ingress and egress capacity of the router in the network-layer topology.
2. A wavelength can traverse at most distance η before it needs to be regenerated. If an optical circuit is longer than η, it has to use regenerators on its path to regenerate the signal.
3. The number of regenerators at each site v is limited, denoted by rg_v. A regenerator can regenerate one optical circuit and transform the circuit to a different wavelength if needed.
4. The optical circuits in the same fiber have to use different wavelengths. A fiber can carry at most ϕ different wavelengths and each wavelength can support a capacity of θ.
5. The total rate of transfers on a network-layer link cannot exceed its capacity θ.

As an additional consideration, we want to keep the changes to the network *incremental*, i.e., only updating a small number of optical links when we perform an update. This minimizes the disturbance during the network update process.

Algorithm overview: The problem has a large number of constraints and variables. Some constraints, like the num-

Algorithm 1 Compute Next Network State (Main Routine)

```
1: function ComputeNetworkState(G)
2:     s_current ← G
3:     e_current ← ComputeEnergy(s)
4:     T ← e_current
5:     s* ← s_current
6:     e* ← e_current
7:     while T > ϵ do
8:         s_neighbor ← ComputeNeighbor(s_current)
9:         e_neighbor ← ComputeEnergy(s_neighbor)
10:        if e_neighbor > e* then
11:            s* ← s_neighbor
12:            e* ← e_neighbor
13:        if P(e_current, e_neighbor, T) > Rand(0, 1) then
14:            s_current ← s_neighbor
15:            e_current ← e_neighbor
16:        T ← T × α
17:    return s*
```

ber of router ports at each site, the number of regenerators at each site, and the number of wavelengths on each fiber, are integral. Even if the network-layer topology is given, optimizing for average transfer completion time is NP-hard [16].

A naive approach is to separately optimize the optical layer and the network layer. However, as the routing decisions are highly coupled with the underlying optical configuration, this greedy approach does not yield good performance results, as we will show in §5.4.

Instead, we use simulated annealing [17] to search for an approximate solution. The motivation for using simulated annealing is that we have a huge search space with integral variables. Simulated annealing is effective in finding acceptable local optimums in a reasonable amount of time while finding the global optimum is computationally expensive. Furthermore, the potential loss of using local optimums is compensated by the fact that the traffic demand changes over time and we frequently reconfigures the network to adapt to the traffic demand changes.

At a high level, we use the network-layer topology G as the state in simulated annealing. We use the current topology as the initial state and probabilistically jump to a neighbor state in each iteration, aiming to find a topology with the highest total throughput. To minimize changes to the network, we construct neighbor states by randomly changing four links of the current state, which is the minimal number of links to change to satisfy the port number constraints.

Our approach has two benefits. First, using the network-layer topology G as the state in simulated annealing, instead of the entire network state NS, significantly reduces the search space. If we search for NS, we have to decide both the optical circuits, the routing paths, and the rate assignments for the network. Instead, if we search for G, we only need to decide the links in the network-layer topology. Second, by using the current topology as the initial state, we are likely to end up with a topology that is not very different from the current one. This reduces the number of changes we need to make for network updates. Now we describe the algorithms in more details.

Algorithm 2 Generate A Random Neighbor State

```
1: function ComputeNeighbor(s)
2:     l_uv, l_pq ← RandomlySelectTwoEdges(E_l)
3:     l_uv.capacity ← l_uv.capacity − θ
4:     l_pq.capacity ← l_pq.capacity − θ
5:     l_up.capacity ← l_up.capacity + θ
6:     l_vq.cacacity ← l_vq.capacity + θ
7:     return s
```

Simulated Annealing (Algorithm 1): The algorithm uses the current topology G as the initial state, the current throughput as the initial temperature (line 2-3). s^* is used to store the topology with the highest throughput and e^* is the energy (throughput) of s^*. The algorithm searches in the search space (line 7-16) until temperature T is less than an epsilon value. T is decreased by a factor of α in every iteration. At each iteration, it uses $ComputeNeighbor$ subroutine to find a neighbor state of the current state and uses $ComputeEnergy$ to compute the energy of the neighbor state. If the neighbor state has a higher energy than s^*, it updates s^* (line 10-12). The algorithm uses a probabilistic function P to decide whether to transition from the current state to the neighbor state. The probabilistic function P is defined as follows: if the neighbor state has a higher energy than the current state, the probability is 1; otherwise, the probability is $e^{(e_{current}-e_{neighbor})/T}$.

ComputeNeighbor (Algorithm 2): This subroutine finds a neighbor state of the current state. It first randomly selects two links from E, say e_{uv}, e_{pq}. Then it decreases the capacity of the selected two links by θ while increasing the capacity of e_{up}, e_{vq} by θ. In other words, it moves the capacity from e_{pq} and e_{uv} to e_{up} and e_{vq} by reconfiguring the optical links. This procedure ensures the total number of ports used on each router remains unchanged.

ComputeEnergy (Algorithm 3): This function computes the total throughput that can be achieved on the given state s, where s is a network-layer topology. The computation is divided into two steps. The first step is to establish multiple optical circuits for each link (line 2-14) based on its desired capacity, and the second step is to assign routing paths and rates to the flows based on the topology (line 15-25).

In the first step, we have constraints 2-4 in the problem formulation to affect whether an optical circuit can be established for a link. We use a regenerator graph to help us compute an optical circuit under these constraints. The nodes in a regenerator graph contain the source site, the destination site, and the sites that have remaining regenerators. We create an edge in the regenerator graph if the shortest paths between two sites is no longer than η. Figure 5(a) shows a regenerator graph. If the source and the destination are directly connected in the graph, we can directly establish an optical circuit; otherwise, they have to use regenerators in the intermediate sites. We want to balance the consumption of regenerators in different sites to improve the possibility that a later optical circuits can find an available one to use. To do this, we assign a weight to each node with *the inverse of their remaining regenerators*; the source and the destina-

Algorithm 3 Compute Energy

```
1: function ComputeEnergy(s)
       // build optical circuits for each link
2:     for network link l ∈ s.links do
3:         Build regenerator graph RG
4:         Build transformed graph TG
5:         P ← TG.sortedPathsByLength(l.src, l.dst)
6:         c ← l.capacity
7:         for path p in P do
8:             if p.canBeBuilt() then
9:                 Build circuit p for l
10:                c ← c − θ
11:                if c <= 0 then
12:                    break
13:        if c > 0 then
14:            Decrease the cacacity of l by c
       // assign routing paths and rates
15:    throughput ← 0
16:    Sort transfers F by policy // e.g., SJF, EDF
17:    l ← 1
18:    while (there exists unsatisfied demand
19:    and there exists free network capacity) do
20:        for transfer f ∈ F do
21:            for path p ∈ paths of f with length l do
22:                min_c ← min_{e∈p} e.remain_capacity
23:                r_{f,p} ← min(f.demand, min_c)
24:                throughput+ = r_{f,p}
25:        l ← l + 1
26:    return throughput
```

tion nodes are assigned with weight zero. Then the problem is to find a path with smallest total node weight in the regenerator graph. This problem can be transformed to the shortest path problem in a directed graph. The transformation first builds a transformed graph from the regenerator graph. The transformed graph has the same nodes as the regenerator graph. An undirected edge in the regenerator graph is replaced by two directed edges; the weight of an edge is set to be the weight of the node the edge points to. It is easy to prove that the shortest path (the path with the smallest total edge weight) in the transformed graph corresponds to the path with smallest total node weight in the regenerator graph. Figure 5(b) illustrates the transformed graph of Figure 5(a). After we have the transformed graph, we iterate the paths based on path length to find enough number of paths we need that can be built as optical circuits (line 7-12). Line 8-12 check whether there are available wavelengths on the path to use, and build the circuit if so. If there are not enough possible optical circuits to satisfy all the desired capacity, we have to decrease the link capacity (line 13-14).

For the second step, we assign paths and rates to each transfer based on the topology to optimize their completion times or deadlines met. The problem is known to be hard. Even if the topology is non-blocking and only the ingress and egress ports are bottlenecks, it is NP-hard to compute rate allocations to achieve the minimum average transfer completion time [16]. It is also NP-hard to maximize the number of transfers that can be finished before the deadlines, when the network is fixed and three or more distinct deadlines are present [18]. A good approximation algorithm is to route transfers based on the order of the remaining transfer size or the deadline. However, in our scenario, the network is not ideal and we need multi-path routing to route some transfers. We approximate the optimal result by using the same ordering of transfers and prioritizing transfers to use shorter paths first. We order transfers with classic scheduling policies like shortest job first (SJF) and earliest deadline first (EDF) (line 16). At each iteration, we only schedule transfers to use paths with length l (line 18-25). At each iteration, we assign rates to each transfer based on its demand and network capacity (line 22-23). Line 24 updates the total throughput. To avoid starvation, we schedule a transfer if it is not scheduled for $\hat{t}$ (configurable) time slots, which we omit in the algorithm for brevity.

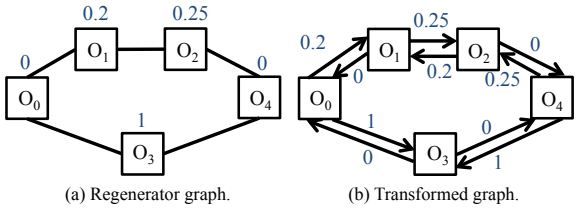


Figure 5: Example of regenerator graph.

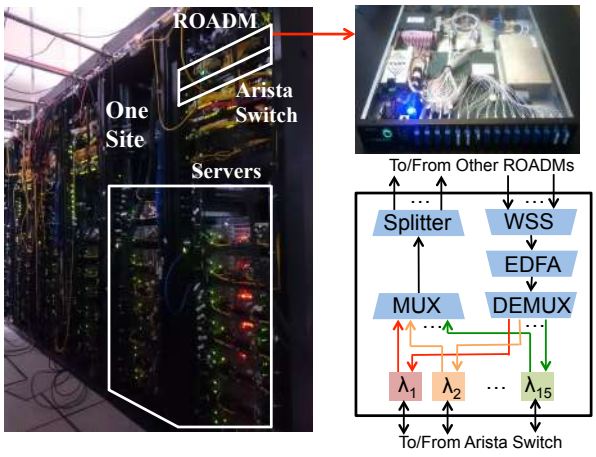
3.3 Updating Network State

After we compute the network state, we need to update the device configurations to the new state. Without careful update scheduling, there can be loops and routing blackholes during the update process. For example, if some packets were sent over a link with the underlying circuit being updated, these packets would be dropped. We need to be especially careful when updating the optical links as it can take several seconds. Dionysus is a recent solution on consistent network updates [19]. Dionysus builds a dependency graph to capture the dependencies between individual update operations and carefully schedules them to make the update fast and consistent. But Dionysus only handles network-layer updates and is not sufficient to handle cross-layer updates. To solve this problem, we extend Dionysus by introducing *circuit nodes* into its dependency graph. Circuit nodes have dependencies on fibers as creating a circuit consumes a wavelength and removing a circuit frees a wavelength; circuit nodes also have dependencies on routing paths as a routing path cannot be used until circuits for all links on the path are established. After we build the dependency graph, we use the same scheduling algorithm as Dionysus to schedule the update operations.

3.4 Handling Practical Issues

Network failures: Link and switch failures are detected and sent to the controller. The controller removes these links and switches from the physical network, and recomputes the network state with the updated physical network. As our algorithm minimizes the amount of updates needed, it is likely to converge to a new feasible schedule with only incremental updates to avoid the failed links.

Controller Failures: Since the scheduling algorithm is stateless, we only need to store the physical network and the set



(a) Testbed. (b) ROADM.

Figure 6: Owan **testbed implementation.**

of all transfers with a reliable distributed storage. When the controller fails, we spawn a new instance, which starts to compute and reconfigure the network state at the next time slot. During the controller failover, the network still carries traffic for the current time slot and is not affected.

Group of transfers: Some applications may need to send traffic to multiple locations and the important metric is the last completion time of all transfers in the group. This is similar to the coflow concept in big data applications in data centers [20, 21]. We can either treat them as single transfers or use better heuristics (like Smallest-Effective-Bottleneck-First [20]) to optimize for groups. A full exploration is our future work.

4. OWAN IMPLEMENTATION

We have built a prototype of Owan. We describe the testbed hardware implementation in §4.1 and the controller software implementation in §4.2.

4.1 Owan Hardware Implementation

Our testbed has nine routers and ROADMs, and emulates the Internet2 topology in Figure 1. We use Arista 7050S-64 as the routers. Since commercial ROADMs are expensive, we use commodity optical components to emulate ROADMs that have the features needed to evaluate the system.

Figure 6 shows the prototype and the optical hardware design to emulate a ROADM. The optical elements for each ROADM is arranged in a 1U box. We have a Freescale i.MX53 micro controller in the box to control the optical elements. At the bottom of a ROADM, it has n(=15) ports that interface with the router. Each interface is an optical transceiver that can convert between electrical packets and optical signals at different wavelengths. The fifteen transceivers are at wavelengths from 1553.33nm to 1542.14nm, which are defined at standard ITU 100GHZ channel spacing.

In order to emulate any possible network-layer topology, we structure the nine ROADMs as a full mesh, i.e., each ROADM has a fiber to connect to every other ROADM. In this way, a ROADM can allocate the n wavelengths among the nine fibers arbitrarily as long as the total number of wavelengths in the nine fibers sum up to n. This means that in the network-layer topology, each router can have any number of links to any other router as long as the total number of links adjacent to a router satisfy the router port constraint. Therefore, our testbed can faithfully emulate the Internet2 network since the testbed is able to construct any network-layer topology that the Internet2 network is able to construct.

Figure 6 depicts the internal structure of our ROADM. For the outward direction of a ROADM, the n wavelengths from n transceivers are multiplexed by a multiplexer (MUX) on to a single fiber. Then the splitter replicates them and sends them to eight other ROADMs. For the inward direction, a Wavelength Selective Switch (WSS) receives n wavelengths from each neighbor and selects up to n different wavelengths from the input wavelengths. Then an Erbium-Doped Fiber Amplifier (EDFA) is used to amplify the wavelengths selected by the WSS, in order to compensate signal loss. Finally, a demultiplexer (DEMUX) demultiplexes the selected wavelengths and send them to corresponding ports. The MUXes and DEMUXes are the same type of device (Oplink AAWG) with different configurations.

To transmit packets from one router to another, the optical signal passes through multiple optical elements, including MUX, splitter, fiber, WSS and DEMUX. These five elements introduce typical optical power loss of 5 dB, 10.5 dB, 0.5 dB, 7 dB, and 5 dB, respectively. The total optical power loss is $\sim$28 dB, which is higher than the optical power budget ($\sim$16 dB) of the transceivers. That is the reason to put an EDFA between WSS and DEMUX. The EDFA is set to operate at fixed gain mode, and has a default setting of gain parameter of 18 dB to compensate the optical power loss.

4.2 Owan Software Implementation

The Owan controller is implemented with 5000+ lines of Java code and uses several third-party software and libraries. It has the following four modules.

Core module: The core module implements the algorithms in §3. We have implemented the blossom algorithm [22] for maximum matching in general graphs and used JGraphT library [23] for other graph algorithms.

Router module: We configure the Arista switches to work in OpenFlow 1.0 mode. We use the Floodlight controller [24] to handle the details of the OpenFlow protocol and interface with the switches. The router module uses the RESTful API exposed by the Floodlight controller to install routing rules.

ROADM module: The Freescale i.MX53 micro controller in each ROADM handles the low level configurations, monitors the optical elements, and exposes a simple API for remote configuration. The ROADM module uses this API to configure each ROADM.

Client module: The client module sends the rate allocation of each transfer to the end hosts. Since a transfer may use multiple paths, we break a transfer into multiple flows and use prefix splitting to implement multi-path routing. The client module configures Linux Traffic Control on each end host to enforce rates.

5. EVALUATION

In this section, we present the evaluation results. Besides a testbed that emulates the Internet2 topology, we have also built a flow-based simulator to evaluate Owan in a large scale with topologies and traffic from an ISP network as well as an inter-DC network from an Internet service company.

5.1 Methodology

Topologies: The testbed topology has nine sites as described in §4. We use Figure 1(b) as the network-layer topology to evaluate TE methods with only network-layer control. The simulations use a topology from an ISP backbone that contains about 40 sites. These sites are connected into an irregular mesh. The inter-DC network has about 25 sites. There are several sites called "super cores" that are connected to many smaller sites, and the super cores are connected in a ring topology.

Workloads: We obtain traces from both the ISP network and the inter-DC network. The traces are traffic counters collected from routers. From the traces, we can get site-to-site traffic demand, but not transfer-level details like transfer sizes and deadlines. Similar to previous work [3, 4], we use synthetic models to generate transfer-level information as follows. First, we sum up all the incoming and outgoing traffic demand for each site. Then we generate transfers for two hours. The transfers for the ISP network follow an exponential distribution with a mean of 500GB/5TB for testbed/simulation experiments. For each transfer, we randomly select a pair of sites whose total traffic demand has not exceeded the sum obtained from the traces. We multiply the sum of traffic demand at each site by a traffic load factor λ to evaluate the system under different loads. For deadline-constrained traffic, we choose deadlines from a uniform distribution between $[T, \sigma T]$ where T is the time slot length and σ is deadline factor that is used to change the tightness of deadlines. The inter-DC traffic follows roughly a similar distribution (with different λ values), except for that it has some "hotspots" in the network that generate lots of transfers for a period of time, and these hotspots can move from site to site.

Traffic engineering approaches: We compare the following approaches. Only Owan has optical-layer control. Tempus [3] and Amoeba [4] only work with transfers with deadlines, so we only compare them on deadline-constrained traffic in §5.3.

- Owan: The approach described in this paper. It jointly controls the optical layer and the network layer.
- MaxFlow: This approach uses linear programming to maximize the total throughput for each time slot.
- MaxMinFract: This approach uses linear programming to maximize the minimal fraction that a transfer can be served at each time slot.
- SWAN [2]: It uses linear programming to maximize the throughput while achieving approximate max-min fairness for each time slot.
- Tempus [3]: It deals with deadline-constrained traffic. It first maximizes the minimal fraction a transfer can be served across all time slots and then maximizes the total number of bytes that can be satisfied.
- Amoeba [4]: This is another approach that deals with deadline-constrained traffic. It uses graph algorithms to compute routing and rate allocation for multiple time slots and adjust previous allocation when new transfers arrive.

Performance metrics: For deadline-unconstrained traffic, the primary metric is *transfer completion time*. We use *factor of improvement* to show the improvements of Owan over other approaches, which is the transfer completion time of the alternative approach divided by that of Owan. We also show *makespan*, which is the total time to complete a series of transfers.

For deadline-constrained traffic, we use *the percentage of transfers that meet deadlines* and *the amount of bytes that finish before deadlines*.

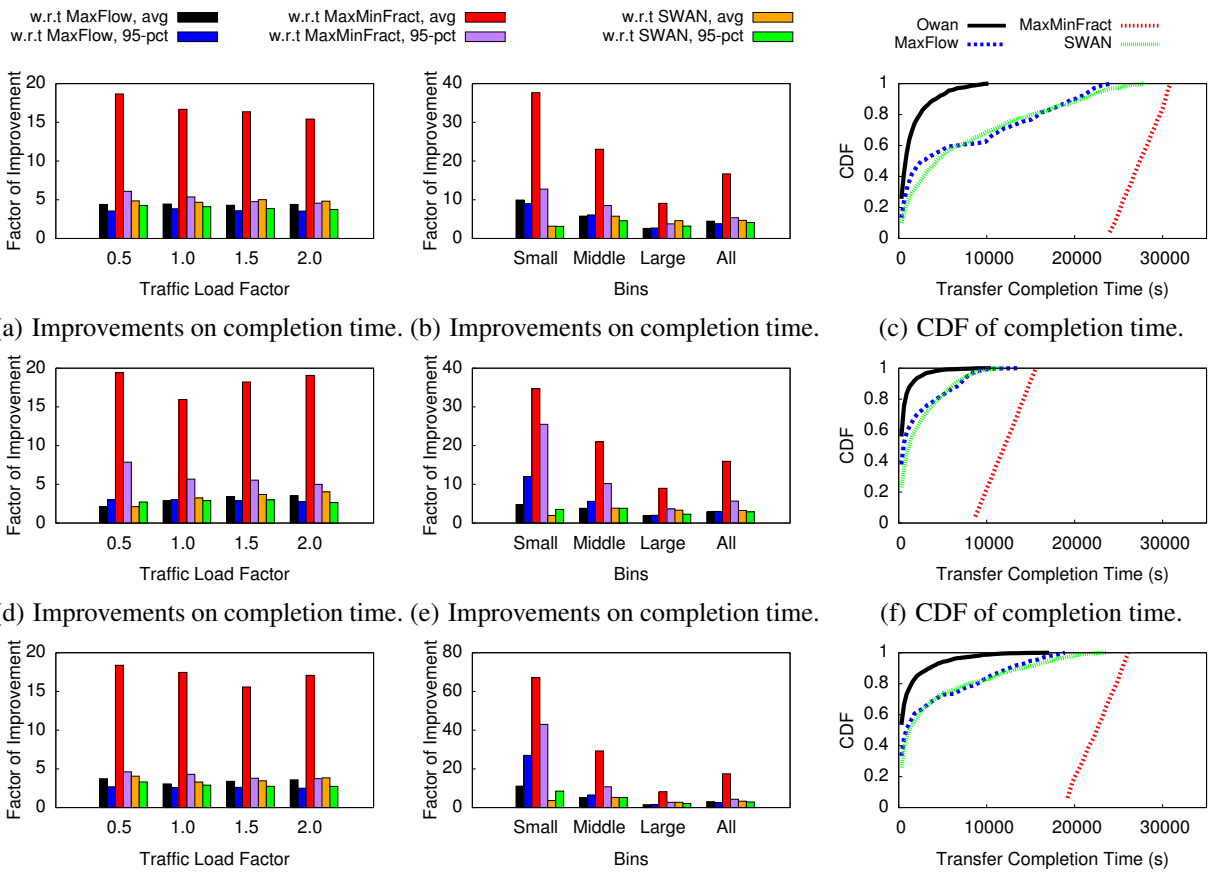
Performance validation: We have validated the results of our flow-based simulator with our testbed results on the Internet2 topology. The difference on the performance metrics is within 10%, which is mainly from the imperfect rate limiting and prefix splitting for multi-path routing on the testbed.

Testbed configurations: We run the controller on a commodity 2U server with two six-core Intel Xeon E5-2620v2 processors running at 2GHz. As we will show later, even this modest configuration is sufficient to run the Owan core module. All the test clients run on servers with the same configuration, and they connect to the network with 10GE. We use both *iperf* and a custom multi-threaded traffic generator to send emulated traffic, and we have verified that each client is able to saturate a 10Gbps link. Both generators have TCP enabled. We perform reconfigurations every five minutes.

5.2 Deadline-Unconstrained Traffic

In this experiment, the transfer requests submitted to the system do not have deadlines. The key metric is to optimize the transfer completion time. Figure 7(a-c) shows the results of the testbed experiments on the Internet2 topology. Figure 7(a) shows the factor of improvement on the average and the 95th percentile transfer completion time under different traffic loads. Compared to MaxFlow, Owan improves the average (95th percentile) transfer completion time by up to $4.45\times$ ($3.84\times$); compared to MaxMinFract, Owan improves the average (95th percentile) transfer completion time by up to $18.66\times$ ($6.09\times$); compared to SWAN, Owan improves the average (95th percentile) transfer completion time by $5.01\times$ ($4.27\times$). The results show that by dynamically reconfiguring the optical layer, Owan can significantly improve the transfer completion time for bulk transfers on the WAN. Moreover, we observe that Owan has bigger improvements over MaxMinFract than MaxFlow and SWAN. This is because MaxMinFract optimizes for the minimal fraction served by each transfer for each time slot, which causes most transfers to take several time slots to finish.

To further zoom in on the results, we divide the transfers into three bins (small, middle, large) based on transfer size, i.e., the smallest $1/3$ transfers are in the small bin, the middle $1/3$ in the middle bin, and the largest $1/3$ in the large



(a) Improvements on completion time. (b) Improvements on completion time. (c) CDF of completion time.

(d) Improvements on completion time. (e) Improvements on completion time. (f) CDF of completion time.

(g) Improvements on completion time. (h) Improvements on completion time. (i) CDF of completion time.

Figure 7: Results for deadline-unconstrained traffic. (a-c), (d-f), and (g-i) are results of the Internet2 network, ISP network, and inter-DC network, respectively.

bin. Figure 7(b) shows the factor of improvement in different bins when the traffic load factor is 1. Owan consistently improves the average and 95th percentile transfer completion time over MaxFlow, MaxMinFract and SWAN in different bins. We observe the most improvement is in the small bin. This is because Owan adjusts the network-layer topology based on traffic demand and small transfers are prioritized to take the most benefits of the topology.

To show the performance of Owan from another angle, we also plot the CDF of the transfer completion time. Figure 7(c) shows the CDF of the transfer completion time when the traffic load is 1. In the figure, the line of Owan stays at the leftmost, which means Owan achieves the smallest transfer completion time across all percentiles. MaxFlow, MaxMinFract and SWAN have longer tails than Owan. This means some transfers can have longer completion time than other transfers if MaxFlow, MaxMinFract or SWAN is used. The reason is also due to the fixed network-layer topology used by these approaches. The fixed topology causes many transfers to use multiple hops to reach their destinations and the total throughput of the network is lower than that in Owan. Overtime, some transfers are accumulated in the scheduling queue because of the limited total throughput and need to take a long time to complete.

To evaluate Owan on a topology larger than our 9-site testbed, we also perform simulations using the ISP topology and inter-DC topology. Figure 7(d-f) and (g-i) show the respective results. Similar to the Internet2 results, Owan significantly improves the transfer completion time. Specifically, Figure 7(d) shows that Owan improves the average (95th percentile) transfer completion by up to $3.52\times$ ($3.00\times$) as compared to MaxFlow, $19.42\times$ ($7.86\times$) as compared to MaxMinFract, and $4.03\times$ ($3.00\times$) as compared to SWAN. Also, Owan is better than the other three approaches across different bins (Figure 7(e)) and different percentiles (Figure 7(f)). Figure 7(g-i) shows similar improvement factors on the inter-DC topology.

Finally, we show the improvement on *makespan*. Makespan is the total time to finish a given number of requests. We inject traffic requests for two hours and measure the makespan of different approaches under different traffic loads. Figure 8 shows the improvement of Owan on makespan over the other three approaches. From the figure, we can see that Owan improves the makespan by up to $2.56\times$, $1.80\times$ and $1.60\times$ in the three topologies respectively. The improvement increases with the traffic load. This is because by reconfiguring the topology Owan can achieve higher total throughput and thus finish more requests in a certain time. When the load is higher, MaxFlow, MaxMinFract and SWAN have more transfers accumulated over time than Owan.

(a) Internet2 result. (b) ISP result. (c) Inter-DC result.

Figure 8: Improvements on makespan.

5.3 Deadline-Constrained Traffic

This experiment evaluates the performance of Owan for deadline-constrained traffic. In addition to MaxFlow, MaxMinFract and SWAN, we also compare Owan with another two approaches, Tempus and Amoeba, which are specifically designed for deadline-constrained traffic on the WAN. Figure 9(a-c) shows the results of testbed experiments on the Internet2 topology. Figure 9(a) shows the percentage of transfers that meet deadlines under different deadline factors. Owan enables the most number of transfers to meet deadlines. Amoeba is particularly designed for transfers to meet deadlines and performs the second best. The objective of Tempus is to maximize the minimal fraction served for each transfer across all time slots and then maximize the total bytes that finish before their deadlines. Thus it has relative poor performance to enable transfers to meet their deadlines. Overall, Owan increases the number of transfers that meet their deadlines by up to $1.36\times$, as compared to Amoeba, which performs the second best. The improvement is relatively small when the deadline factor is too small or too large. This is because when the deadline factor is too small, all the transfers have tight deadlines and there is little room for Owan to further increase the number of transfers that can meet their deadlines. When the deadline factor is too large, many transfers can easily meet their deadlines, and the absolute value of the percentage is already high. The benefit of reconfiguring the optical layer is most significant when the deadline factor is not at extreme values.

Besides the percentage of transfers that meet their deadlines, we also show the percentage of bytes that finish before the deadlines in Figure 9(b). Owan outperforms other approaches more significantly on this metric. It improves the bytes that finish before the deadlines by up to $2.03\times$ than the second best one (Amoeba). Also we can see that MaxMinFract and Tempus perform better on this metric than the percentage of transfers that meet their deadlines. This means they finish many bytes of a transfer though the entire transfer does not meet the deadline. This metric is important to applications that can use the available bytes as they arrive before the deadlines.

Similar to deadline-unconstrained traffic, we also show the breakdown of the percentage of transfers that meet deadlines in different bins with regard to transfer size. Figure 9(c) shows the result when the deadline factor is 20. Owan performs better than the other approaches across different bins.

Figure 9(d-f) and (g-i) shows the results of the simulation results on the ISP and inter-DC topology, respectively. Similarly, Owan consistently outperforms other approaches. It improves the number of transfers that meet their deadlines by up to $1.13\times$ and $1.08\times$ respectively, and the number of bytes that finish before their deadlines by up to $1.46\times$ and $1.33\times$ respectively, as compared to the second best alternative. Owan also performs well across different transfer sizes.

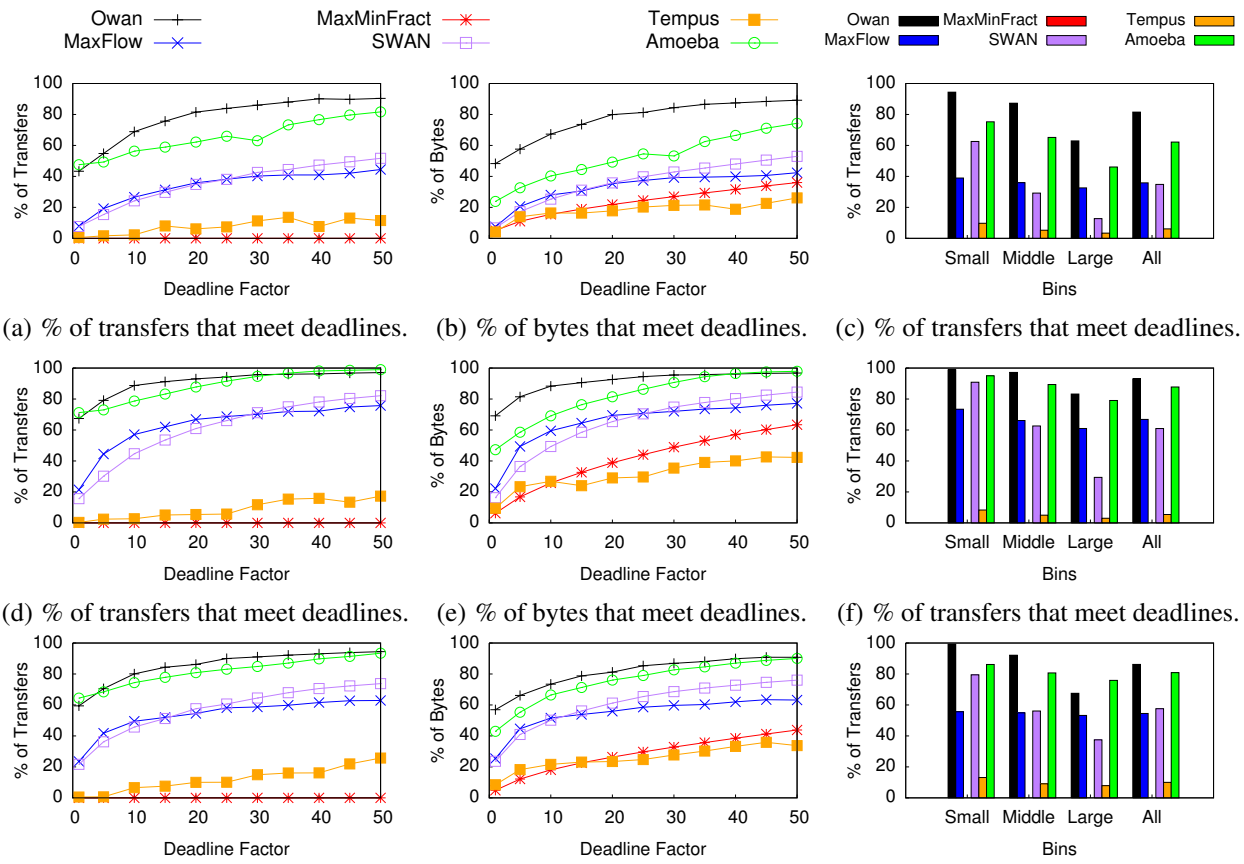
5.4 Microbenchmarks

We now show some microbenchmarks. All the experiments are performed on the inter-DC topology with deadline-unconstrained traffic and the load factor being 1 if not otherwise specified.

Joint optimization of the optical and network layers: We show the benefit of jointly optimizing the optical and network layers. For comparison, we develop a greedy algorithm, which first builds a network-layer topology based on traffic demand between every two sites, and then it tries to find a routing configuration that maximizes total throughput using a similar routine as described in Algorithm 3. In other words, the greedy algorithm optimizes the optical layer and the network layer *separately*. The greediness simplifies the computation by limiting the search space. Unfortunately as Figure 10(a) shows, the total throughput is 21% less than the joint optimization, even if the joint optimization is only an approximation using simulated annealing. This performance difference is not incidental: as we have multiple paths for each flow, the routing configuration is tightly coupled with the optical configuration. Also, the greedy algorithm does not try to minimize the number of optical links to change while the simulated annealing algorithm does.

Consistent network updates: It takes about three to five seconds on our testbed to reconfigure an optical circuit. During the update of an optical circuit, the circuit goes dark and cannot carry any traffic. To avoid traffic disruptions, we use a consistent update scheme in §3.3. To demonstrate its effectiveness, Figure 10(b) shows the comparison of with and without the consistent update scheme. Without consistent update, all links are updated simultaneously in one shot to minimize update completion time. The total throughput drops 10% during the update, as packets get lost on these links, affecting the overall TCP performance. With consistent update, we observe no throughput drop during the update process, and we do not observe changes in end-to-end packet drop rate either.

Breakdown of gains: Owan jointly optimizes network-layer topology, routing and rate allocation. We use an experiment to show a breakdown of gains from controlling the three



(a) % of transfers that meet deadlines. (b) % of bytes that meet deadlines. (c) % of transfers that meet deadlines.

(d) % of transfers that meet deadlines. (e) % of bytes that meet deadlines. (f) % of transfers that meet deadlines.

(g) % of transfers that meet deadlines. (h) % of bytes that meet deadlines. (i) % of transfers that meet deadlines.

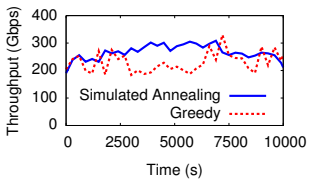
Figure 9: Results for deadline-constrained traffic. (a-c), (d-f), and (g-i) are results of the Internet2 network, ISP network, and inter-DC network, respectively.

parts. Figure 10(c) shows the result of the experiment. In the experiment, we compare the average transfer completion time when the system has different levels of control of the network. All times are normalized by the average transfer completion time when the traffic load factor is 0.5 and the system has controls of all three parts. In the most basic scheme, the "rate" line in the figure, the system only controls rate allocation. The system cannot reconfigure the network-layer topology, nor can it change routing. It can only adjusts the sending rates of the transfers. In the second scheme, the "+rout." line in the figure, the system has controls of both routing and rate allocation. It assigns routing paths and rates to transfers similar to line 15-25 in Algorithm 3. The third scheme, the "+topo." line in the figure, has controls of all three parts. As we can see from the figure, we have lower average transfer completion time when we have more control of the network.

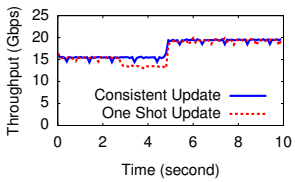
Running time and convergence: We use simulated annealing to find a good topology. Since simulated annealing is an approximation algorithm that performs probabilistic search for the optimum, the quality of the result is related to its running time. The longer the algorithm runs, the more states it can search in the search space and the better the result can be. In our solution, since we use the current topology as the initial state of the algorithm, instead of a random topology, the algorithm starts its search with a reasonable good state. Since our system runs the algorithm and reconfigures the network every a few minutes, the traffic on the network is unlikely to change dramatically. Therefore, the algorithm can quickly find a good new topology by starting from the current topology and only changing a few links, as compared to starting from a random topology and spending a lot of time on finding a reasonably good topology. Figure 10(d) shows the performance of our algorithm when we run simulated annealing for different amounts of time. The performance is measured by the average transfer completion time. From the figure, we can see that the algorithm performs very bad when the simulated annealing only runs for 20 ms. However, the algorithm converges quickly, and it only requires about 320 ms to find a good topology to significantly reduce the average transfer completion time.

6. RELATED WORK

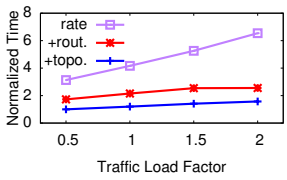
WAN Traffic Engineering: Traffic engineering is a classic topic in networking research. Early work focuses on avoiding congestions. Many algorithms are developed to minimize the maximum link utilization under different conditions, such as changing traffic demands and network failures [25, 26, 27]. There are also efforts on achieving different fairness metrics theoretically and practically [28, 29].



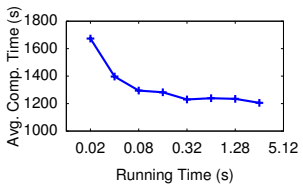
(a) Simulated annealing vs. the greedy algorithm.



(b) With and without consistent update scheme.



(c) Breakdown of gains.



(d) Impact of running time of simulated annealing.

Figure 10: Microbenchmark results.

With the emergence of SDN and the ability to direct program switches, researchers develop new centralized control systems, like Google B4 and Microsoft SWAN, to improve network utilization and its robustness in face of control plane and data plane failures [1, 2, 30, 31, 32]. Recent work goes beyond network-wide objectives like network utilization, to more fine-grained transfer-level objectives, like minimizing transfer completion time and meeting deadlines [3, 4, 33, 34, 35], and controls not only switches, but also proxies, load balancers, and DNS servers [36]. Owan follows the trend of centralized control for WANs. The key feature that differentiates Owan from previous solutions is the joint management of the optical and network layers, and we show that dynamically reconfiguring the optical layer can significantly.

The routing problem in overlay networks also concerns two layers [37, 38, 39]. The routing in the underlay network (the network layer in this paper) builds the topology for the overlay network. However, the overlay and underlay networks are usually managed by different parties, and an overlay network usually traverses multiple ASes and has unstable end-to-end network performance.

Data-Center Traffic Engineering: Data-center networks have massive scale in terms of number of switches and hosts. Most traffic engineering work in data-center networks focuses on routing elephant flows as it is impractical to deal with all flows in a centralized manner [40, 41, 42, 43, 44, 45]. To cope with the scalability problem, CONGA designs a distributed load balancing mechanism and implement it in switch hardware [46]. Most of these solutions tackle the routing problem, i.e., choosing a path for a flow or flowlet. To solve the rate allocation problem, i.e., deciding the rate for each flow to optimize the flow completion time or the number of flows that meet deadlines, researchers have developed a wide range of new flow scheduling and congestion control algorithms [16, 47, 48, 49, 50, 51, 52, 53]. Some of them are entirely host-based; others leverage both host and switch features. There are also works that optimize for a group of flows, which are important for many big data applications [20, 21]. Owan has similar objectives as these works, but the target of Owan is WANs. WANs do not have a structured topology as FatTree or CLOS in data center networks (which many algorithms for data-center networks rely on), and the topology can be changed by reconfiguring the underlying optical layer.

Besides these works, some solutions propose to provide bandwidth guarantee to cloud applications and tenants, in order to provide predictable performance and enforce isolation [54, 55, 56, 57]. In these solutions, requests are formulated as bandwidth reservations between ingress and egress points. For bulk data transfers, it is more appropriate to formulate requests as volumes of data as in Owan. On the other hand, bandwidth reservations are also a useful abstraction for some use cases on the WAN. It is an interesting area of future work to explore how the reconfigurability in the optical layer can improve bandwidth reservations.

Optical Networks: With the advancements in optical technology and centralized control, researchers have started to build centralized production systems to manage the optical layer on the WAN [14, 15]. Xu *et al.* [14] present an on-line system to reconfigure the optical circuits given a set of circuit demands with constraints. Bathula *et al.* [15] develop algorithms to compute the minimal set of regenerator concentration sites such that any two optical ROADMs have at least one path available by using the selected sites. In terms of cross-layer control, early studies present algorithms and analysis for the joint optimization of the optical and network layers [58, 59, 60]. They mainly focus on admissible traffic demand and attempt to optimize for objectives like network cost and routing hops. Recent work begins to explore building systems to jointly control the optical and network layers, such as the DARPA CORONET program [6]. Our work is built up these efforts and presents the design and implementation of Owan to jointly control the optical and network layers and optimize bulk transfers for transfer completion time and deadlines met.

In terms of data centers, many researchers have proposed to use optics to boost the network performance. For example, Helios, cThrough and OSA use MEMS switches [7, 8, 9]; FireFly uses free-space optics [10]; WaveCube uses WSS switches [11]. The major objective in these works is to improve the *network throughput*. By reconfiguring the topology, they can make the network be comparable to a non-blocking network, while saving on power, cost, and wiring complexity. Other works use optics to reduce latency [61]; use optics to support multicast [62, 63, 64, 65]; and design new optical hardware [66, 67, 68]. Differently, Owan reconfigures topologies in the WAN scenario, which uses ROADMs, regenerators and has the optical reach constraint, and Owan combines topology reconfiguration with routing and rate allocation to optimize transfer-level objectives.

7. CONCLUSION

We present Owan, a new traffic management system that optimizes bulk transfers on the WAN. Besides controlling routing and rate allocation, Owan goes one important step further than prior solutions into the optical layer. It reconfig-

ures the optical layer in the same time scale as routing and rate allocation in a centralized manner. We develop efficient algorithms to compute the optical and routing configurations to optimize bulk transfers. Testbed experiments and large-scale simulations show that Owan completes data transfers up to $4.45\times$ faster in average and up to $1.36\times$ more flows meet their deadlines than methods with only network-layer control. Owan is the first step towards software-defined optical WANs. We believe centralized control of the optical and network layers would have a far-reaching impact on the theory and practice of network management for WANs.

Acknowledgments We thank our shepherd Hitesh Ballani and the anonymous reviewers for their feedback. This work was supported in part by NSF grants CNS-1162112, DMS-1418255 and CCF-1535900, AFOSR grant FA9550-14-1-0193, National Natural Science Foundation of China grants 61361136003 and 61532001, 1000 Talent Plan grant, Tsinghua Initiative Research Program grant 20151080475, and a Google Faculty Research Award.

8. REFERENCES

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *ACM SIGCOMM*, August 2013.

[2] 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*, August 2013.

[3] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *ACM SIGCOMM*, August 2014.

[4] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-datacenter transfers," in *EuroSys*, April 2015.

[5] "Internet2." http://www.internet2.edu.

[6] A. L. Chiu, G. Choudhury, G. Clapp, R. Doverspike, M. Feuer, J. W. Gannett, J. Jackel, G. T. Kim, J. G. Klincewicz, T. J. Kwon, *et al.*, "Architectures and protocols for capacity efficient, highly dynamic and highly resilient core networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 4, pp. 1–14, January 2012.

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

[8] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," in *ACM SIGCOMM*, August 2010.

[9] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," in *USENIX NSDI*, April 2012.

[10] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "FireFly: A reconfigurable wireless data center fabric using free-space optics," in *ACM SIGCOMM*, August 2014.

[11] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, Q. Dong, and Y. Liu, "WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture," in *IEEE INFOCOM*, April 2015.

[12] "Infinera ROADM Specification." http://tinyurl.com/jjog6no.

[13] "Oclaro WSS." http://tinyurl.com/hotq4s3.

[14] D. Xu, G. Li, B. Ramamurthy, A. Chiu, D. Wang, and R. Doverspike, "On provisioning diverse circuits in heterogeneous multi-layer optical networks," *Computer Communications*, vol. 36, no. 6, pp. 689–697, 2013.

[15] B. G. Bathula, R. K. Sinha, A. L. Chiu, M. D. Feuer, G. Li, S. L. Woodward, W. Zhang, R. Doverspike, P. Magill, and K. Bergman, "Constraint routing and regenerator site concentration in roadm networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, pp. 1202–1214, November 2013.

[16] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," in *ACM SIGCOMM*, August 2013.

[17] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[18] M. A. Bonuccelli and M. C. Clò, "Scheduling of real-time messages in optical broadcast-and-select networks," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 541–552, October 2001.

[19] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *ACM SIGCOMM*, August 2014.

[20] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *ACM SIGCOMM*, August 2014.

[21] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM*, August 2015.

[22] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys*, vol. 18, pp. 23–38, March 1986.

[23] "JGraphT Graph Library." http://jgrapht.org.

[24] "Floodlight OpenFlow Controller." http://floodlight.openflowhub.org/.

[25] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *ACM SIGCOMM*, August 2003.

[26] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *IEEE INFOCOM*, March 2000.

[27] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," in *ACM SIGCOMM*, August 2005.

[28] E. Danna, A. Hassidim, H. Kaplan, A. Kumar, Y. Mansour, D. Raz, and M. Segalov, "Upward max min fairness," in *IEEE INFOCOM*, March 2012.

[29] E. Danna, S. Mandal, and A. Singh, "A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering," in *IEEE INFOCOM*, March 2012.

[30] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *ACM SIGCOMM*, 2014.

[31] A. Kumar, S. Jain, U. Naik, A. Raghuraman, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *ACM SIGCOMM*, August 2015.

[32] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and

expressive approach to control forwarding paths in carrier-grade networks," in *ACM SIGCOMM*, August 2015.

[33] B. B. Chen and P. V.-B. Primet, "Scheduling deadline-constrained bulk data transfers to minimize network congestion," in *IEEE CCGRID*, May 2007.

[34] K. Rajah, S. Ranka, and Y. Xia, "Advance reservations and scheduling for bulk transfers in research networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 1682–1697, November 2009.

[35] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with NetStitcher," in *ACM SIGCOMM*, August 2011.

[36] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang, "Efficiently delivering online services over integrated infrastructure," in *USENIX NSDI*, March 2016.

[37] D. G. Andersen, A. C. Snoeren, and H. Balakrishnan, "Best-path vs. multi-path overlay routing," in *ACM SIGCOMM Conference on Internet Measurement Conference*, October 2003.

[38] Z. Li and P. Mohapatra, "QRON: QoS-aware routing in overlay networks," vol. 22, no. 1, pp. 29–40, 2004.

[39] Y. Liu, H. Zhang, W. Gong, and D. Towsley, "On the interaction between overlay routing and underlay routing," in *IEEE INFOCOM*, March 2005.

[40] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks.," in *USENIX NSDI*, April 2010.

[41] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *ACM CoNEXT*, December 2011.

[42] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM*, August 2011.

[43] Z. Shao, X. Jin, W. Jiang, M. Chen, and M. Chiang, "Intra-data-center traffic engineering with ensemble routing," in *IEEE INFOCOM*, April 2013.

[44] X. Wu and X. Yang, "Dard: Distributed adaptive routing for datacenter networks," in *IEEE ICDCS*, June 2012.

[45] S. Radhakrishnan, M. Tewari, R. Kapoor, G. Porter, and A. Vahdat, "Dahu: Commodity switches for direct connect data center networks," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, October 2013.

[46] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, August 2014.

[47] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM*, August 2011.

[48] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *ACM SIGCOMM*, August 2011.

[49] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *ACM SIGCOMM*, August 2012.

[50] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *ACM SIGCOMM*, August 2012.

[51] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *ACM SIGCOMM*, August 2012.

[52] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," in *ACM SIGCOMM*, August 2014.

[53] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *USENIX NSDI*, May 2015.

[54] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM*, August 2011.

[55] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical network performance isolation at the edge," in *USENIX NSDI*, April 2013.

[56] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing," in *ACM SIGCOMM*, August 2013.

[57] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *ACM SIGCOMM*, August 2014.

[58] K. C. Guan, *Cost-effective optical network architecture: A joint optimization of topology, switching, routing and wavelength assignment*. PhD thesis, Massachusetts Institute of Technology, February 2007.

[59] A. Brzezinski and E. Modiano, "Dynamic reconfiguration and routing algorithms for IP-over-WDM networks with stochastic traffic," *Journal of Lightwave Technology*, vol. 23, pp. 3188–3205, October 2005.

[60] B. Ramamurthy and A. Ramakrishnan, "Virtual topology reconfiguration of wavelength-routed optical WDM networks," in *IEEE GLOBECOM*, November 2000.

[61] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A new design element for low-latency dcns," in *ACM SIGCOMM*, August 2014.

[62] H. Wang, Y. Xia, K. Bergman, T. Ng, S. Sahu, and K. Sripanidkulchai, "Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity," *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 52–58, July 2013.

[63] P. Samadi, D. Calhoun, H. Wang, and K. Bergman, "Accelerating cast traffic delivery in data centers leveraging physical layer optics and SDN," in *International Conference on Optical Network Design and Modeling*, May 2014.

[64] P. Samadi, H. Wang, D. Calhoun, Y. Xia, K. Sripanidkulchai, T. Ng, and K. Bergman, "An optical programmable network architecture supporting iterative multicast for data-intensive applications," in *IEEE Optical Interconnects Conference*, May 2014.

[65] Y. Xia and T. Ng, "A cross-layer sdn control plane for optical multicast-featured datacenters," in *ACM SIGCOMM HotSDN Workshop*, August 2014.

[66] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *ACM SIGCOMM*, August 2013.

[67] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with REACToR," in *USENIX NSDI*, April 2014.

[68] S. Liu, Q. Cheng, A. Wonfor, R. V. Penty, I. White, and P. M. Watts, "A low latency optical top of rack switch for data centre networks with minimized processor energy load," in *Optical Fiber Communication Conference*, March 2014.