FLEXNET: A Reconfigurable Network in the WAN

Jeremy Bogle and Tim Kralj

May 15th, 2019

Abstract

Fiber optic cables connecting data centers are an expensive resource acquired by large organizations that require significant monetary investment and oversight. Their importance has driven a conservative deployment approach with redundancy and reliability taken into account. Over-provisioning these networks to account for potential spikes in demand is a common practice. In this work, we take a more aggressive approach and argue for instead *dynamically* changing the capacity of fiber optic links based on the capacity requirements of the network and the current demands. We propose FLEXNET, a traffic engineering system that allows optical links to adapt their capacity based on the observed demands on the network to achieve higher throughput. Each node in the network can allocate its given total capacity, or node pool, across its outgoing links in an optimal fashion. We evaluate FLEXNET using real world networks to see potential network improvements, evaluate speed, and compare to existing approaches. Our results show a dynamic network like FLEXNET can outperform state-of-the-art algorithms for networking used today.

1. INTRODUCTION

Internet users are consistently demanding higher speeds as many services and applications are being run in datacenters hosted by cloud providers. Today, fiber optic cables are being used to get these blazing fast internet speeds; however, all the network infrastructure is currently static. These cables are very expensive to build and maintain and therefore should be used to their maximum efficiency in order to reduce costs. We plan to explore a novel way to reduce the costs of the in the optical topology of the WAN while increasing efficiency without laying new fiber. Flexible-grid capable ROADM (Re-configurable Optical Add-Drop Multiplexers) are considered one of the most significant advances in Dense Wavelength Division Multiplexing (DWDM) systems technology over the last decade. ROADMs have the ability to redirect wavelengths from an input port to any output port, enabling the network operator to program the allocation of wavelengths across connected fibers. In theory, a software defined network could have a centralized controller that programs the ROADMs to change its wavelengths in an optimal way thus effectively adjusting the capacity on different fibers – we will define capacity to be the maximum traffic a fiber can carry, which is a function of the number wavelengths being sent along that fiber. In traffic engineering (TE), previous approaches attempt to find optimal routing strategies that maximize flow using the well know max-flow optimization and various other schemes. All these schemes are constrained by the capacity on each link, and the static topology of the network. We plan to devise a routing algorithm that is aware of a re-configurable network and can reroute traffic and change capacity on each link. Initial studies in this field have assumed that a re-configurable network could create a fiber link between two switches when there wasnt one before [1]. We plan to consider a real topology where the constraints do not allow for the creation of links, but instead allow for changing capacities to 0 (essentially just an empty link) and then dynamically expanding them when needed. This provides a realistic framework for implementing a TE scheme into a network with ROADMs at each node. In order to evaluate how such a system would work in practice, we will also need to consider cost of re-configuration. We propose our system, FLEXNET, that casts this idea of a network with dynamic capacities as a Linear Program (LP). In section 4, we run a series of evaluations on real topologies, and evaluate possible cost models.

2. Motivation

Today's WAN networks are often considered static entities. The capacities on each link, defines as in the amount of traffic a link can carry, is currently bounded by the number of wavelengths that propagate through a given fiber. These capacities are static, and rarely changed. When these links are adjusted, it requires a large amount of time and money to increase their capacities. In this paper, we imagine a solution where a network can be built such that its capacities are dynamic, and can be adjusted to incoming demands. To do this, we must consider node has a re configurable multiplexer, (ROADM) that can take its incoming wavelengths, and direct them along any of its out-going edges. Therefore, the capacities on all links are actually a function of the wavelengths, or total capacity that can be sent from the outgoing node of that edge. We call this total capacity the *node pool*. This *pool* can then be allocated on the outgoing edges from each node in whatever way is most optimal. We see this manifest itself later as decision variables in the Linear Program we call FLEXNET.

Problem. Imagining a small network of 4 nodes, we can illustrate this problem and how our solution would work. If we take a look at Figure 1, we see an example where these node pools can be maintained and the total throughput can be increased by re-configuring the link capacities. In the example, there are four nodes and 8 uni-directional edges that can carry traffic in between nodes. Node 1 is trying to send traffic to node 4. In the example we only provide capacities for the 4 edges that matter. Originally, node 1 can only send 2 units along the upper path and 3 along the lower path for a total admissible traffic of 5 units. In a re-configurable network like FLEXNET, edge (1,3) can borrow a unit of capacity from the edge (1,2) and

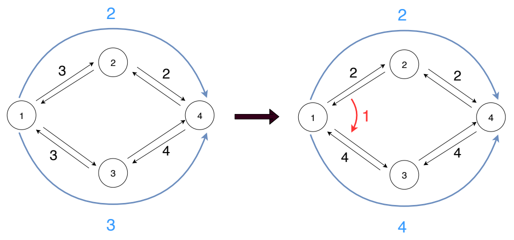

Figure 1: Here we show an example of a capacity constrained network demanding 6 units of traffic from node 1 to node 4. We see the highest possible throughput is carrying 2 on the upper path and 3 on the lower path. However, on the right, we show what a re-configurable network could do. If node 1 has a pool of 6, it could allocate 2 on the upper edge and 4 on its lower edge, allowing the network to carry 2 units of traffic on the upper path and 4 on the lower path for a total throughput of 6 units.

maintains the pool on node 1 (6 units). This small change allows for a throughput increase from 5 to 6. To achieve results like this, we present our optimization framework, FLEXNET. Then, in section 4 we run a series of experiments comparing FLEXNET to existing TE algorithms on larger networks.

3. Optimization Framework

We now describe the FLEXNET optimization framework in detail. We first formalize the model and delineate the goals of WAN TE [2], [3], [4], [5] (§3.1). Then, we introduce FLEXNET's novel approach to TE, showing that it enables finding optimal reconfigurations of network capacities.

3.1. WAN TE

Input. Like other WAN TE studies, we model the WAN as a directed graph $G = (V, E)$, where the vertex set V represents switches and edge set E represents links between switches. Link capacities are usually given by $C = (c_1, \dots, c_{|E|})$ (e.g., in bytes per second, bps) and as in any TE formulation, the total flow on each link should not exceed its capacity. However, in FLEXNET, these link capacities are no longer inputs. Instead, we

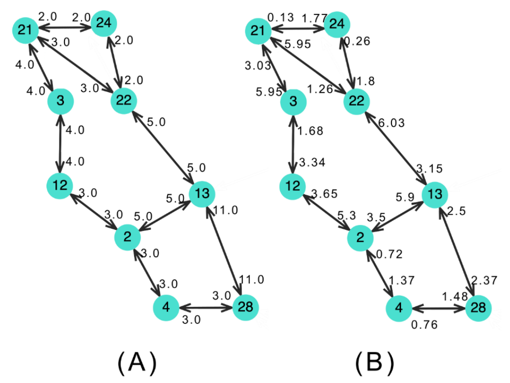

Figure 2: This figure shows the reconfiguration output from our linear program. (A) shows the original capacities for a subsection of one of our topologies. (B) shows the same graph with optimal capacities for one demand matrix while adhering to the node pools.

take *node pool* as an input represented as p_n where $p_n = \sum_{e \in E_n} c_e$ and E_n represents the set of outgoing edges for node n. TE decisions are made at fixed time intervals (say, every 5 minutes [3]), based on the estimated user traffic demands for that interval. In each time epoch, there is a set of source-destination switch-pairs ("commodities" or "flows"), where each such pair f is associated with a demand d_f, and a fixed set of paths (or "tunnels") $T_f \in T$ on which its traffic should be routed. FLEXNET assumes the tunnels are part of the input. In section 4, we use a generic tunnel selection scheme (e.g., k-shortest paths). We also experiment with edge-disjoint paths and varying k. In reality, any preferred path selection scheme, potentially one that takes latency, or distance into account could be used instead.

Output. The output of TE schemes often results in the bandwidth granted for each flow, and a specification for how traffic should be split across its tunnels. The output of FLEXNET, consists of two parts: (1) the specification for splitting traffic for each flow f, across its tunnels T_f, and (2) the *total* capacity c_e for each edge e.

Optimization goal. Previous studies of TE considered optimization goals such as maximizing total concurrent flow [3], [6], [5], max-min fairness [7], [2], [8], minimizing link over-utilization [4], minimizing hop count [9], and accounting for hierarchical bandwidth allocations [10]. As formalized below, an appropriate choice for our context is selecting the $w_{f,t}$'s (per-tunnel bandwidth weights) in a manner that maximizes the the sum of all flows. This choice of objective will enable us to maximize network throughput, but will not allow for fairness across flows. In §3.4 we discuss ways to extend our framework to other optimization objectives that achieve better fairness.

3.2. Linear Program

With FLEXNET, we use all the same inputs from regular WAN TE algorithm, except we do not take capacities as an input. Instead, we take *node pool* as input represented as p_n. Then, as output, we return our decision variables c_e and $w_{f,t}$. Outlined in the table below, we have represented these variables in a linear program. We represent the objective as a maximization function where we simply maximize throughput, or total satisfied demand. We discuss different objective functions that achieve better fairness in section 3.4. We solve this objective function subject to 3 constraints. The first of these constrains states that all overlapping flows on a given edge must not exceed the capacity of that edge. We cannot have this because the hardware cannot exceed that threshold. To assure this, we multiply the weight of a given flows tunnels ($w_{f,t}$) by its demanded traffic (d_f), and by a binary variable ($L_{t,e}$). We use the binary variable to test if a tunnel t uses edge e so we do not count tunnels that are unused. Unlike other TE scheme, in this first constraint, the capacity on each edge will be a decision variable in the formulation and be determined by the other constraints as well instead of an input. This is because we have the ability to adjust our capacities that are going through each tunnel in each edge. In other two constraints, we guarantee that outgoing and incoming capacities do not exceed the total pool for that node. The first of these two constrains is incoming flows and the second constrains outgoing flows. We discuss the choice to use weights as an input, $w_{f,t}$ and how it translates to a granted bandwidth for each flow b_f in the next section. Figure 2 shows an example optimal capacity output from a subsection of one of our graphs.

Input	$G(N,E)$	Network graph with switches N and links E.
	$p_n \in P$	The pool that node n can allocate to its incoming and outgoing edges.
	$d_f \in D$	The bandwidth demand of flow f.
	$T_f \in T$	Set of tunnels for flow f.
Auxiliary variables	$L_{t,e}$	1 if tunnel t uses edge e, 0 otherwise.
	$Z_{n,e}$	1 if edge e is an outgoing edge of node n, 0 otherwise.
	$X_{n,e}$	1 if edge e is an incoming edge of node n, 0 otherwise.
Output	c_e	The capacity on each edge; $c_e \geq 0$
	$w_{f,t}$	The weight of traffic for flow f on tunnel $t \in T_f$; $0 \leq w_{f,t} \leq 1$

$$\begin{aligned}
\text{maximize} \quad & \sum_{f \in F, t \in T_f} w_{f,t} d_f \\
\text{s.t} \quad & \sum_{f \in F, t \in T_f} w_{f,t} d_f L_{t,e} \leq c_e \quad && \forall e \in E \\
& \sum_{e \in E} c_e Z_{n,e} \leq p_n && \forall n \in N \\
& \sum_{e \in E} c_e X_{n,e} \leq p_n && \forall n \in N
\end{aligned}$$

3.3. Routing with FLEXNET

A common practice in today's TE schemes is to output a granted bandwidth per flow and more specifically an allocation for each flow on each tunnel [5], [2]. In certain schemes they use weights [4] instead of exact allocations. When actually routing traffic, a weight assignment rule must be used where each tunnel is allocated a proportional amount of bandwidth on its available tunnels. For schemes that give allocations in bps, $a_{f,t}$, weights can be given as $w_{f,t}$, where $w_{f,t} = \frac{a_{f,t}}{\sum_{t \in T_f} a_{f,t}}$. However, in FLEXNET, we instead use the weights, $w_{f,t}$ explicitly in the optimization so that they are optimized as outputs. This allows for ease of use when routing as this proportional assignment is not necessary. We also allow for the weights to sum to less than 1, so $\sum_{t \in Tf} w_{f,t} \leq 1$ which allows for natural bandwidth limitation by the formulation. Therefore the granted bandwidth for a flow b_f can be computed simply as multiplying this sum of weights by the original demand for that flow, $b_f = \sum_{t \in T_f} w_{f,t} d_f$. Finally, in the event of failures, weights can be split among surviving tunnels proportionally. During failures, when $\bar{T_f} \subseteq T_f$ is the subset of f's tunnels that are still available, each tunnel $t \in \bar{T_f}$ can be reassigned a weight as $w_{f,t} \leftarrow \frac{w_{f,t}}{\sum_{t \in \bar{T_f}} w_{f,t}}$ to route around the failure.

3.4. Achieving Fairness Across Flows

As we mentioned in the previous section, the formulation presented in section 3.2 doesn't have any notion of fairness and only tries to maximize total bandwidth. In this section we describe an approach for achieving max-min fairness, where we maximize the minimum bandwidth achieved across all flows.

To do this we would like to apply an objective maximizes the function

$$F(w) = \min_f \left[\sum_{t \in T_f} w_{f,t} d_f \right] \tag{1}$$

While this objective function is nonlinear, we are able to transform this into a linear objective by applying an additional constraint. We represent this objective as a constraint where b_{min} is the minimum bandwidth among all flows

$$b_{min} \geq \sum_{t \in T_f} w_{f,t} d_f \quad \forall f \in F \tag{2}$$

Then with this constraint, we can simply maximize b_{min} which will be the minimum b_f of all flows.

3.5. Path Selection

We use the k-shortest path algorithm to find the paths between source and destination nodes in FLEXNET. This algorithm, also known as Yen's algorithm [11] uses an iterative approach to Dijkstra's and returns only the k shortest (or less if that many do not exist). This gives us the benefit of not having to use excess amounts of compute power to find all the paths—an unscaleable solution for very large graphs. In addition, we do not want to have too many long paths between our nodes being used since there is a correlation between path length and latency. We want to maintain fast speeds for users of the network so latency is important. The parameter k is user definable to fit the specific instance of the system you are using with FLEXNET. We talk more about runtimes of different k's in 4.4. In addition, since path selection is completely agnostic to the formulation, any preferred path selection scheme that may take into account latency, or distance or another goal could be used instead.

3.6. Cost Model

Currently, the cost for reconfiguration is not represented in this formulation. This means that, if a setup, or configuration of capacities exists that is more optimal, then the formulation will find it. While this makes it most optimal in theory, it is not realistic with actual hardware as there could be costs associated with changing the network capacities (i.e dropping traffic, energy consumption, and hardware malfunctions). Because of this, we must think about how different configuration speeds could affect the results, and how we can quantify the *cost* of re-configuring this network. This would allow us to see the trade offs that would exist between a static network and a dynamic one, and at what point the dynamic hardware becomes useful. In section 4.5 we discuss different cost models in depth and how one can determine whether or not it is beneficial to switch to the optimal configuration.

3.7. Implementation

We implement the linear program from this section in Julia using the JuMP framework. We are able to successfully represent the example in Figure 1 and see the optimizer choose to reallocate the node pool on node 1 to allow for the 6 units of traffic to flow. In section 4, we run a series of experiments on larger networks to prove the effectiveness of FLEXNET.

4. Evaluations

In this section we present the results from our evaluations of FLEXNET. We compare FLEXNET to state of the art TE schemes and show how it can outperform these approaches under high traffic situations. Other approaches are constrained to using the static capacity of the network while FLEXNET can adjust or reconfigure capacities according to the current demand, while still adhering to the same *node pool*. We also provide analysis for cost of reconfiguration and how that would affect effectiveness of FLEXNET.

4.1. Experimental setting

Topologies. We evaluate FLEXNET on four network topologies: B4, IBM, ATT, and X-Net. The first three topologies (and their traffic matrices) were obtained from the authors of SMORE [12]. X-Net is the network topology of a large cloud provider in North America. See Table 1 for a specification of network sizes. Our empirical data from the X-Net network consists of the following: the capacity of all links (in Gbps), the traffic matrices (source, destination, amount of data in Mbps) over four months at a resolution of one sample per hour. For the ATT, B4, and IBM topologies we obtained a set

Topology Name	#Nodes	#Edges
B4	12	38
IBM	18	48
ATT	25	112
X-Net	≈ 30	≈ 100

TABLE 1: Network topologies used in the evaluations. For confidentiality reasons, we do not report exact numbers for the X-Net topology.

of at least 24 demand matrices and link capacities. For the sake of FLEXNET, we look at the total capacities on each outgoing edge of a given node set this as the *node pool* for that node.

4.2. Satisfied Demand Across All Topologies

Our first experiments seeks to show total satisfied demand by different TE schemes on multiple topologies and many demand matrices.

Setup. We begin by loading in all four topologies listed above. We iterate over these topologies and compute the sum of allowed demand for each flow as $\sum_{f \in F, t \in T_f} w_{f,t} d_f$. Then we divide by the total requested demand, $\sum_{f \in F} d_f$ and report this as the satisfied demand for this flow. For each point, we average the satisfied demands across 10 demand matrices. Demand is scaled up for each topology to a scale where FLEXNET can achieve 99% satisfaction. This allows us to normalize results across topologies. The results are shown in Figure 3 based on the demand satisfied for each topology and different TE schemes.

From this experiment, we see that FLEXNET outperforms other schemes, especially with larger networks like X-Net and ATT. ECMP, or *Equal-Cost-MultiPath* [13], simply splits weights equally across all tunnels. This approach has no notion of capacity and must drop traffic if it breaks the capacity on a given link. FFC [5], is a more recent approach that takes into account failures and optimizes total bandwidth. However, FFC$_k$ limits bandwidth to ensure total satisfaction up to k-failures. So we see that with these networks FFC$_2$ must severely limit bandwidth to be robust up to 2 concurrent failures anywhere in the network. FLEXNET, on the other hand, can reconfigure the capacities on the

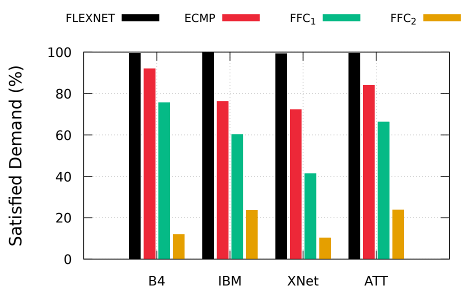

Figure 3: In this experiment, we show the results of satisfied demand as a percentage of total demand across 4 different topologies. Demand is scaled up for each topology until FLEXNET can achieve 99% satisfaction to provide normalization across topologies. We see that FLEXNET consistently outperforms other schemes.

links according to the node pools at each node to be most optimal for the current demand in the network. FLEXNET achieves higher satisfied demand across all topologies and demand matrices.

4.3. Scaling Demands

Networks are often significantly over-provisioned, and thus under-utilized, in order to handle unexpected shifts in traffic. In this experiment, we put the network in a constrained state and by scaling demand up and examine its performance.

Setup. For this experiment, we run FLEXNET with many demand matrices and for each one, we compute satisfied demand using the same procedure in the previous experiment. Initially, we scale demand up beforehand by applying a multiplicative factor to all flows, and then from there slowly increase the scale and plot the satisfied demand on the y-axis. For each point, we average the satisfied demands across 10 demand matrices.

From looking at the results in Figure 4, we can see that FLEXNET outperforms other approaches. The biggest contributing factor for this is that FLEXNET is able to reconfigure capacities while other approaches are limited by the existing capacity configuration on the links.

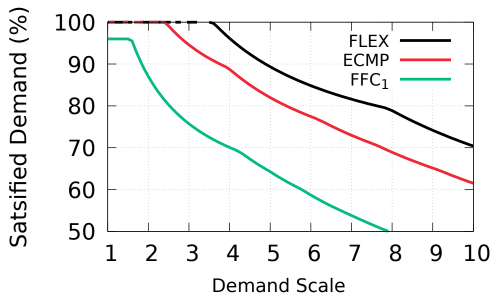

Figure 4: This figure shows the percentage of satisfied demand as demand is scaled up. We see that with FLEXNET we can achieve more satisfied demand at higher scales. This is because FLEXNET is able to configure capacities accordingly where as other approaches like ECMP or FFC are constrained by the static capacities of the network. We also scaled demand initially to put the network in a constrained state so scale 1.0 actually corresponds to 3x demand. Results are averaged over 10 demand matrices and one topology. FFC_2 was not shown here because it achieves significantly lower satisfied demand.

4.4. Speed Analysis

To perform runtime analysis on the scalability of FLEXNET, we ran experiments with our implementation and different path selection mechanisms. The results are shown in figure 3.5. To test how the runtimes changed with different values of k and an increasing number of edges, we run this experiment with values of k of 3, 5, 10, and 20. We do average this across 10 different demand matrices. We only use FLEXNET in this experiment to illustrate how the algorithm performs in an increasing load. This is used to assess if FLEXNET can scale to larger graphs with ease.

As we can see in 4, we see an increase in runtime as the number of edges increase. In addition, as the value of k increases, we see that the runtimes increase. Here we can see the tradeoff of increasing your search space vs the runtime of the algorithm. On the on hand, one may not want to search the whole space for speed but also may want to give each flow more options for splitting its traffic. This also shows that our formulation runs in about 100 milliseconds, with as many as 20 paths per flow on large networks.

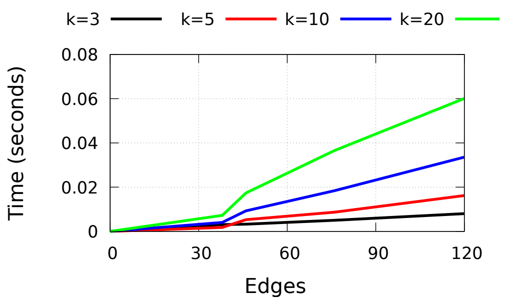

Figure 5: This figure shows how FLEXNET scales with increasing the number of edges with respect to the value of k (in k-shortest neighbors) used in the search. The increase in runtime comes from the increase of search space from both the number of edges in the graph and the value of k. The times increase at a linear rate with respect to edges as they increase for each individual k value.

4.5. Cost Tradeoff Analysis

In our previous experiments, we have assumed that you can reconfigure a network instantly, with 0 cost. Potential costs for reconfiguration include network downtime, energy consumption, and hardware malfunctions. Prior work has been done to show how to include a cost into linear programs with maximization objectives [14]. We provide some analysis of how FLEXNET would be affected in the worst case, where reconfiguration results in entire network outage until reconfiguration has finished. We show that there eventually is a break even point where it might still be beneficial to switch to the new reconfiguration. Here, reconfiguration time is assumed to be a variable. Other work has shown that reconfiguration can take on the order of tens of hundreds of milliseconds [15].

Considering cost is important for systems designers to consider when implementing FLEXNET. There are multiple ways that reconfiguration can be done in the network. We show this in Figure 6 and Figure 7. In both of these figures, we begin with an initial network configuration, and initial demand d_0. At time d_1 we receive the next incoming demand matrix and must decide whether to reconfigure. The first of these shows how a short but total network outage would effect satisfied demand. This figure represents the worst case

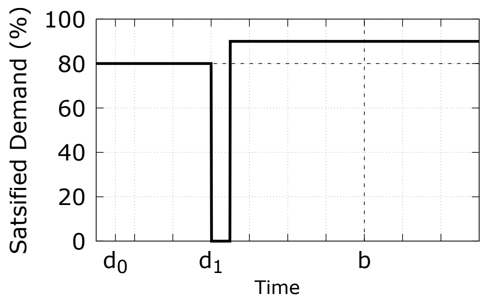

Figure 6: A sample graph showing the satisfied demand over time during a reconfiguration. The demand d_1 causes FLEXNET to perform a configuration switch in the hardware. The drop in satisfied demand happens for a short time, but it goes to 0% satisfied. At time b, the drop in satisfied demand during reconfiguration hits a break-even point. After time $t > b$, the total satisfied demand is greater than it would have been without reconfiguring.

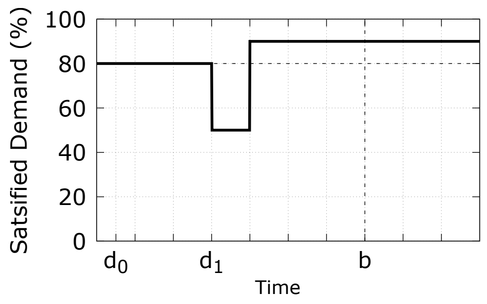

Figure 7: A graph that shows the demand satisfied over time during a different reconfiguration scheme than Figure 6 for FLEXNET. This case has the demand drop to a smaller percentage (50% instead of 0%) but this drop lasts for a longer time. The breakpoint for the drop in satisfied demand is at time b. Similar to Figure 6, after time $t > b$, the total satisfied demand it greater than it would have been with the original configuration from time d_0.

- no traffic can be satisfied during reconfiguration. We see that there is 0 demand met for a period of time once the demand is analyzed at point d_1 for the total time it takes to reconfigure all switches at once. After this point, however, the network is back up and has been reconfigured to be more optimal for d_1 and can therefore satisfy a higher percentage of traffic. Eventually it comes to a break-even point b where the demand loss of configuration is balanced out. For all time $t > b$ we have satisfied more demand than we would have had we not switched configurations.

Similarly, in Figure 7, we start with an initial demand d_0 and initial configuration and receive the next incoming demand at time d_1. The drop in satisfied demand is less than the previous figure but for a longer time. This represents a switching scheme where hardware switches sequentially in different parts of the network to allow some continual traffic through the network at all times during reconfiguration. When we receive the new demand d_1, We decide to change the hardware, and after a longer period of time than Figure 6, we see the new configuration is complete. In both of these cases, we still achieve the same break-even point in the system at time b. Similarly, for all time when $t > b$, demand satisfied is higher and we determine that it was worth switching to the new configuration.

These two examples show how cost can play a role in configuration and have different schemes to achieve the needs of the specific system — an important note mentioned above is these machines have been shown to achieve reconfiguration in the order of tens of hundreds of milliseconds [15]; therefore, the second strategy employed in Figure 7 would likely be superior. The shift is still very fast, but the system would have a much smaller drop in satisfied demand. We can also imagine more complex switching schemes that can do some configurations in paralell and other sequentially, to maximize satisfied demand throughout the entire switching process. For all of these cost models, when actually using FLEXNET, we would have to do these post processing calculations to decide whether it was worth switching configurations or not. In the future, we plan to model this cost in the optimization so that the optimization can tell us whether the configuration outweighs the costs, or if there is a middle-ground such that *some* but not all reconfiguration steps can be made to minimize cost and maximize satisfied demand.

5. Contributions

On this project we have had even contributions from both Jeremy and Tim. We met and discussed previous work in this field with Prof. Manya Ghobadi to help us understand the hardware involved, and begin to formulate a linear program. Together, we designed, implemented, and tested our optimization in Julia. We setup a github shared repo to work together on this project. We also both contributed to this paper and presentation.

6. Moving Forward

In this section, we provide background for both traffic engineering in general, and more specifically a few bodies of work that explore re configurable networks. We also describe in detail the future ideas and challenges encountered in the creation of FLEXNET.

6.1. Related Work

WAN traffic management. Optimizing WAN backbone traffic is a well-researched challenge. Most studies focus on optimizing bandwidth allocation with recent interest in centralized TE for WANs driven by software-defined approaches for running and optimizing networks at scale (such as SWAN [3], B4 [2], FFC [5], and BwE [10]). These schemes exploit a global view of the network and perform global updates to configure flow allocations.

Reconfigurable networks. Some recent work has been done leveraging re-configurable optical devices like ROADMs [16], [17] that would be used by FLEXNET. A recent paper called RADWAN [14] presents a similar idea where a SDN can be reconfigurable, but focuses on increasing certain capacities on links by a certain amount, instead of maintaining a given node pool. Another recent idea which provided some inspiration for this work explores dynamic networks more generally and discusses data structures that could be used to represent a dynamic network and makes a case for using dynamic networks in WANs [1]. Lastly, Jin *et al*. presented work that shows how networks could be changed on larger timescales to be used for bulk data transfers [16], but not specifically for TE. This work also provides a lot of ideas for reconfiguration costs, and scheduling updates during reconfiguration.

6.2. Challenges and Future Work

Edge directionality. One major challenge for FLEXNET is the issue of how to represent the graph network. In most previous representations, the graph has nodes and is connected by an edge to represent a link and these links have constant capacity. In addition, links are often represented as bidirectional. However, we are try to make a routing scheme that is able to change the amount the capacity a node can send and receive. Because of this, we decided to make our graph have two uni-directional links between each node. This gives us multiple benefits. Mainly, it allows FLEXNET to make constraints separate for the inputs of a node (how much each router in our network receives) and the outputs for each node (how much each router can send). Another benefit from this representation is that is it allows the system to have routers that are able to accept and receive different amounts of data. Currently, most fiber capacities are bidirectional, but our model has the ability to assign different capacities on different directions of a single fiber.

Scalability. We think that moving forward, a challenge FLEXNET may face is scalability. As networks get larger, we believe the number of constraints will grow dramatically and we want to ensure our linear program is still solveable in an efficient manner. We use matrices L, Z, and X to provide $O(1)$ look-ups for edges in each tunnel, and incoming and out-going edges for each node. The networks we tested with only increase to about 120 edges, but ROADM networks are rarely orders of magnitude larger. Because of this, FLEXNET should easily scale to these types of networks.

Cost of reconfiguration. Our formula works under the assumption that cost of reconfiguration is 0 and the network can reconfigure instantaneously with some hypothesis on the time it should take to reconfigure a ROADM chip. In the future, we want to test with hardware to get a sense of the order of time it takes to change. If the time to change a physical network is shown to be in the hundreds of milliseconds, FLEXNET provides a lot of opportunity to implement a dynamic network configuration. After testing on hardware, we could update our linear program to more accurately consider the cost of reconfiguration.

7. Conclusion

In this paper we present FLEXNET, a novel way for WAN networks to use ROADMS (Re-configurable Optical Add-Drop Multiplexers) in a reconfigurable system. We show how networks can be self-adjusting to better serve users by satisfying a greater demand than previous algorithms such as ECMP, FFC-1, and FFC-2. The hardware that enables FLEXNET is still not widely used, but it can greatly benefit network providers. Our main contributions are the linear program for network reconfigurations and evaluation on real world data sets.

References

[1] Chen Avin and Stefan Schmid. Toward demand-aware networking: A theory for self-adjusting networks. *CoRR*, abs/1807.02935, 2018.

[2] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, August 2013.

[3] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 15–26, 2013.

[4] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, Renton, WA, 2018. USENIX Association.

[5] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*, pages 527–538, 2014.

[6] Cynthia Barnhart, Niranjan Krishnan, and Pamela H. Vance. Multicommodity flow problems. In *Encyclopedia of Optimization*, pages 2354–2362. Springer, 2009.

[7] Dritan Nace and Michal Pióro. Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial. *IEEE Communications Surveys and Tutorials*, 10(1-4):5–17, 2008.

[8] E. Danna, S. Mandal, and A. Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *2012 Proceedings IEEE INFOCOM*, pages 846–854, March 2012.

[9] Youngseok Lee, Yongho Seok, Yanghee Choi, and Changhoon Kim. A constrained multipath traffic engineering scheme for MPLS networks. In *ICC*, pages 2431–2436. IEEE, 2002.

[10] Alok Kumar, Sushant Jain, Uday Naik, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Bjrn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Sigcomm '15*, 2015.

[11] Thomas A. Williams and Gregory P. White. A note on yen's algorithm for finding the length of all shortest paths in n-node nonnegative-distance networks. *J. ACM*, 20(3):389–390, July 1973.

[12] Praveen Kumar, Chris Yu, Yang Yuan, Nate Foster, Robert Kleinberg, and Robert Soulé. Yates: Rapid prototyping for traffic engineering systems. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 11:1–11:7, New York, NY, USA, 2018. ACM.

[13] M. Chiesa, G. Kindler, and M. Schapira. Traffic engineering with equal-cost-multipath: An algorithmic perspective. *IEEE/ACM Transactions on Networking*, 25(2):779–792, April 2017.

[14] Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. Radwan: Rate adaptive wide area network. August 2018.

[15] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. Optimizing bulk transfers with software-defined optical wan. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 87–100, New York, NY, USA, 2016. ACM.

[16] Xin Jin, Yiran Li, Da Wei, Siming Li, Jie Gao, Lei Xu, Guangzhi Li, Wei Xu, and Jennifer Rexford. Optimizing bulk transfers with software-defined optical wan. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 87–100, New York, NY, USA, 2016. ACM.

[17] Ajay Mahimkar, Angela Chiu, Robert Doverspike, Mark D. Feuer, Peter Magill, Emmanuil Mavrogiorgis, Jorge Pastor, Sheryl L. Woodward, and Jennifer Yates. Bandwidth on demand for inter-data center communication. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 24:1–24:6, New York, NY, USA, 2011. ACM.