Traffic Engineering with Forward Fault Correction

Hongqiang Harry Liu (Yale University) Srikanth Kandula Ratul Mahajan Ming Zhang David Gelernter (Yale University)

Microsoft Research

Abstract— Network faults such as link failures and high switch configuration delays can cause heavy congestion and packet loss. Because it takes time for the traffic engineering systems to detect and react to such faults, these conditions can last long—even tens of seconds. We propose forward fault correction (FFC), a proactive approach for handling faults. FFC spreads network traffic such that freedom from congestion is guaranteed under arbitrary combinations of up to k faults. We show how FFC can be practically realized by compactly encoding the constraints that arise from this large number of possible faults and solving them efficiently using sorting networks. Experiments with data from real networks show that, with negligible loss in overall network throughput, FFC can reduce data loss by a factor of 7–130 in well-provisioned networks, and reduce the loss of high-priority traffic to almost zero in well-utilized networks.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design–Network communications. C.2.3 [**Computer Communication Networks**]: Network Operations.

Keywords: Traffic Engineering; Congestion-Free; Fault Tolerance

1. Introduction

Centralized traffic engineering reduces network congestion and increases efficiency [1, 2, 3, 4, 5, 6]. In such systems, a TE controller frequently reconfigures the network to match current traffic demand. These reconfigurations enable the network to carry more traffic, thus enabling the operators to extract more value from the expensive infrastructure investment.

While centralized TE can be highly effective, it is unable to quickly react to faults in both the data and control planes. Data plane faults occur when a link or switch fails and stops forwarding packets. Another class of faults, which we call control plane faults, is where the controller fails to reconfigure a switch in a timely manner, even though the switch continues to forward packets (as previously configured). This fault can occur due to a host of factors, such as RPC (remote procedure call) failures, bugs in switch firmware or software, shortage of memory in the switch, and so forth.

Both classes of faults are common. Several studies have reported frequent link and switch failures in large networks [7, 8, 9]; in a wide area network that we study, a link fails every 30 minutes on average. Google reports both heavy delays and outright failures in configuring switches [1]. The percentage of configuration updates that failed was 0.1–1%. For switches from two different vendors, we observe a similar failure rate in our own experiments. In a network with a hundred switches, this failure rate implies the controller will commonly fail to configure at least some of them.

Data and control plane faults can cause heavy congestion and packet loss. When a data plane fault occurs, switches quickly move traffic to other available paths, but because this movement does not account for link capacity constraints, it can severely congest some links. Similarly, control plane faults cause congestion when a switch continues sending traffic on a link as per old configuration, while it was expected to move the traffic away. Our experiments show that data and control plane faults frequently lead to links getting 10–20% more traffic than their capacity. The resulting packet loss will hurt congestion-sensitive applications.

Today, relieving congestion due to these faults requires intervention by the TE controller. The downside of this approach is that interventions happen *after* congestion has occurred. Worse, they can take a long time because of the delay inherent to updating a large network, which stems from factors such as RPC delays, control load on switches, and the time to change forwarding rules. Updating a single switch rule can take a few seconds [1], and network-wide updates typically require updating many rules per switch.

We thus argue for proactively handling common fault scenarios. In particular, TE should spread traffic in the network such that no congestion occurs as long as the total number of faults is at most k (a configurable bound). This guarantee should hold for arbitrary combinations of faults. Our view is inspired by forward error correction (FEC), in which a transmitted packet stream is modified such that all original packets can be recovered, without any reaction (e.g., retransmission) as long as the number of losses is up to k. Analogously, we call our approach forward fault correction (FFC).

Practically realizing FFC requires addressing two intertwined challenges—minimizing throughput loss and computational scalability. Just as FEC has overhead in terms of redundant information that is transmitted, FFC will have overhead in terms of link capacity set aside to tolerate faults. This overhead must be low for FFC to be acceptable. The computational challenge is that with n potential faults, the number of combinations of up to k faults is $\sum_{i=1}^{k} \binom{n}{i}$. With n=1000, a plausible number of links in a large network, and k=3, this number over 10^9. Thus, enumerating all possible faults and considering their impact is intractable. Approximations may help reduce computational effort, but unless one is careful, they may result in high capacity overhead.

We address these challenges by first formulating FFC requirements as a linear program. This formulation is precise, which minimizes overhead, but has an intractably large number of constraints because of possible fault combinations. We then observe that these constraints can be transformed, with minimal loss in precision, to a "bounded M-sum" problem: the sum of any M out of N variables is bounded. Finally, we develop a method based on sorting networks [10], to efficiently encode this problem as $O(kn)$ linear constraints. For control plane FFC, our approach is optimal with

SIGCOMM'14, August 17–22, 2014, Chicago, Illinois, USA.

http://dx.doi.org/10.1145/2619239.2626314.

respect to overhead; for data plane FFC, it is optimal if the multiple paths that carry traffic between two switches are disjoint.

Our approach is flexible and applies to many TE scenarios. We show how it accommodates multiple traffic priorities, multi-step network updates [2], different TE objectives (*e.g.* fairness, maximizing throughput, and minimizing maximum link utilization), and handles control faults in traffic rate limiters [11, 12].

We evaluate FFC in a testbed and using simulations based on traffic and fault data from real networks. We find that it is valuable in a range of scenarios. In well-provisioned networks, as is common for ISPs today, FFC has negligible throughput overhead and reduces data loss by a factor of 7–130. In well-utilized networks that use multiple traffic priorities, as is common for inter-datacenter networks [1, 2], FFC protects high-priority traffic from almost all loss, again with negligible loss in total network throughput.

Fundamentally, FFC provides a novel control knob to network operators. Today, operators must either conservatively over-provision the network to guarantee the absence of congestion under faults or aggressively utilize the network [1, 2] at the risk of severe fault-induced congestion. FFC enables operating points that trade-off provisioning and congestion-risk in an informed manner, based on network characteristics and desired protection level.

2. Motivation

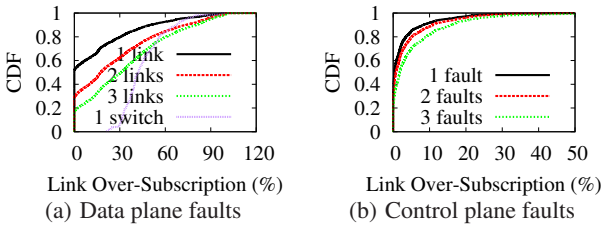
Many applications (e.g., Web search queries, online retail, multi-player games) are sensitive to congestion, and packet drops or long queuing delays hurt the user experience of these applications. FFC is motivated by the observations that data and control plane faults cause congestion and that reacting to these faults is slow. We illustrate these observations below. As is prevalent [1, 2, 3, 4] in networks with TE, we assume tunnel-based forwarding. One or more tunnels carry traffic between each ingress-egress switch pair; we call this traffic a *flow*. Relative weights configured at the ingress switch determine how the flow's traffic is split across tunnels.

2.1 Impact of data plane faults

When a link or switch fails, it impacts all tunnels traversing it. Upon detecting tunnel failures, ingress switches *rescale* traffic to the remaining tunnels in proportion to configured weights. Suppose a flow has 3 tunnels with splitting weights $(0.5, 0.3, 0.2)$. When the third tunnel fails, weights of $(\frac{0.5}{0.8}, \frac{0.3}{0.8}, 0)$ are used to split traffic. OpenFlow group tables can implement such rescaling [13].

Rescaling quickly restores connectivity but can leave the network in a congested state. For example, Figure 2(a) shows an initial traffic distribution with two flows: $\{s2, s3\}\rightarrow s4$. Dashed curves represent tunnels and numbers represent traffic volume they carry. When link $s2$-$s4$ fails and $s2$ rescales, the traffic distribution of Figure 2(b) emerges, in which link $s1$-$s4$ is heavily congested. Such congestion will persist until the TE controller can compute a new solution and configure the network, which can take tens of seconds (see below).

While this example was illustrative, Figure 1(a) characterizes congestion due to data plane faults for `L-Net`, a real network on which we provide more information in §8. This experiment uses topology and traffic data from the network, and it uses a standard TE algorithm (§4.1) to spread traffic every interval (5 minutes) across six tunnels per-flow. We fail randomly selected links or switches in each time interval and measure the maximum link oversubscription rate, i.e., the amount of traffic above capacity that arrives at the link. The graph plots the CDF of the oversubscription rate for the cases of 1–3 link failures and 1 switch failure. Even with a single link failure—which occurs every 30 minutes on average in this network—the link oversubscription rate is over 20% a quarter of the time (75th %-ile). For a 100 Gbps link, 20% over-



(a) Data plane faults (b) Control plane faults

Figure 1: Congestion due to faults in `L-Net`.

subscription means that 120 Gbps traffic will arrive soon after a failure. Even with 100 MB buffer capacity, the switch will be unable to buffer even 50 ms of this traffic and applications will suffer a burst of high loss rate.

2.2 Impact of control plane faults

We now illustrate how a delay or failure in configuring a switch can cause congestion. Figure 3(a) shows a simple network with 4 flows: $s1\rightarrow\{s2, s3\}$ and $\{s2, s3\}\rightarrow s4$. Assume that the controller wants to change the network configuration to Figure 3(b), to accommodate a new flow $s1\rightarrow s4$. This change requires updating $s2$ and $s3$, to modify traffic split weights for their flows. If $s2$ does not update and continues to split traffic as before, link $s1$-$s4$ will be congested, as in Figure 3(c).

Figure 1(b) characterizes congestion in `L-Net` due to control plane faults. This experiment simulates a network update every interval based on the same TE algorithm as above. For each update, we inject control plane faults at randomly selected switches and measure the maximum link oversubscription rate. We see from the graph that, though control plane faults are less damaging than data plane faults, even a single fault—which can occur every 5 minutes if the fault rate is 1% and the network has 100 switches—can lead to an oversubscription of 10% a tenth of the time.

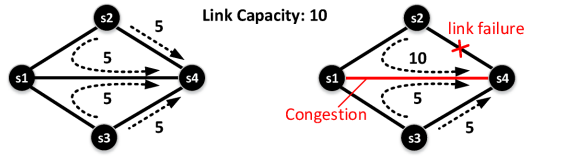
The example above assumes that all updates are sent to the switches in one shot; in another method, updates are sent in multiple steps to avoid transient congestion due to switches applying updates at different times [2]. To transition from Figure 3(a) to 3(b), possible steps are: 1) update traffic splitting ratios at $s2$ and $s3$; and 2) if that succeeds, update the rate of flow $s1\rightarrow s4$. This way, no congestion will occur if $s2$ (or $s3$) fails to update. However, configuration failures will stall network updates because Step 2 cannot proceed until Step 1 finishes. They will also lower throughput since flow $s1\rightarrow s4$ cannot start if Step 1 fails. FFC handles control plane faults for multi-step updates as well.

While this section focuses on switch configuration faults, in networks that control traffic rate [1, 2], a similar problem arises for rate limiter configuration as well. Congestion will occur if a rate limiter continues to send traffic at the older, higher rate. We show that FFC handles such faults as well.

2.3 Slow reaction to faults

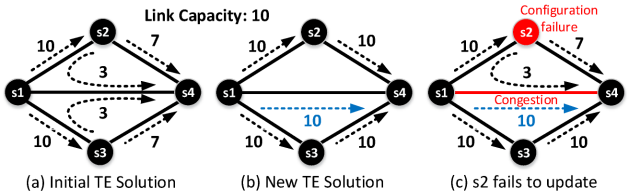
Reactive approaches suffer from the fact that they start *after* congestion and loss has already started, and they can take a long time in large production networks. Figure 6(a) shows the distribution of rule update times in B4, based on Figure 12 and Table 4 of the paper [1]. It excludes switches for which configuration completely fails. It shows both RPC delays and the time to update a single forwarding rule. Typically, many rules per switch are updated during a network update; this number is commonly over 100 for `L-Net`.

While B4 is a complex system with many sources of delay and variability, we also quantify update delays in a controlled, lab environment using commodity switches. We issue rule update commands, while the switches have moderate background control load such as reading counters and running tunnel liveness detection pro-



(a) Initial traffic distribution (b) Traffic distribution after local rescaling

Figure 2: Congestion due to a data plane fault.



(a) Initial TE Solution (b) New TE Solution (c) s2 fails to update

Figure 3: Congestion due to a control plane fault.

tocol. (Forwarding load has no visible impact on rule update time.) The number and type of updates are drawn from those issued in `L-Net`. Figure 6(b) shows that the rule update time is still substantial. The median is 10 ms and the worst case is over 200 ms. For updating R rules, the total update delay will be RPC delay + $R\times$ per-rule delay; we have confirmed this behavior experimentally. Ignoring RPC delay, for updating 100 rules, the median update delay for a switch will be 1 second and the worst case over 20 seconds. We will show that FFC provides significant benefit even when switch update characteristics mimic this simplified setting.

It may be possible to make reaction times faster, but it is fundamentally limited by factors such as network path latencies, overloaded switch CPUs, time to update forwarding tables,[1] and noise inherent to any production environment. We thus advocate an approach where common faults are handled proactively, and only big, rare faults are handle reactively.

3. FFC Overview and Challenges

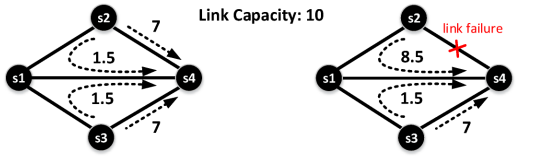
Our goal is to develop proactive methods to handle data and control plane faults. Inspired by FEC, we develop the concept of FFC, which guarantees that no congestion will occur as long as the number of faults is at most (a configurable bound) k. Explicit fault detection or any reaction from the TE controller is not needed.

As the primary controls in FEC are the number of packets sent and their encoding, the primary controls in FFC are the amount of traffic entering a network and its spread. (Assume for now traffic injection is controlled; we consider networks without rate control later.) We illustrate below how these controls, with some overhead in terms of lower throughput, can proactively protect against faults.

3.1 FFC for control plane faults

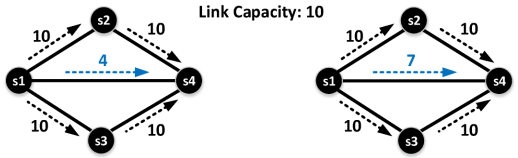
We start with control plane faults because they are unique to centralized TE and have not been studied before. Control plane FFC guarantees that no congestion occurs as long as the number of switches that experience a configuration fault is up to k. To see how this guarantee may be achieved, let us revisit Figure 3, in which we wanted to update switches $s2$ and $s3$ to accommodate a new flow. If we try to update the network from Figure 3(a) to 3(b), in which flow $s1\rightarrow s4$ sends 10 units of traffic, it is impossible to be robust against configuration failure of $s2$ or $s3$. However, the network configuration of Figure 5(a), in which $s1\rightarrow s4$ sends 4 units of traffic, is robust to either one or both switches failing to configure. Thus, this traffic distribution is an example of FFC with k=2, where no congestion will occur if up to two switches fail to update.

[1]Being based on TCAMs (ternary content addressable memory), forwarding tables are optimized for fast lookups, rather than fast updates [4]



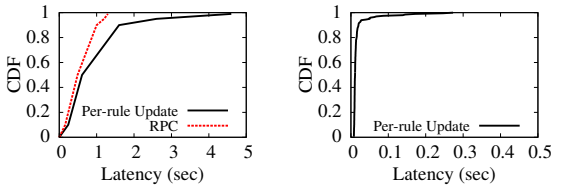
(a) Planned traffic distribution (b) Traffic distribution after a link fails

Figure 4: FFC for link failures ($k = 1$)



(a) k=2 (b) k=1

Figure 5: FFC for control plane faults.



(a) Measurement in B4 (b) Controlled experiments

Figure 6: Switch update latencies.

The downside is that the network throughput is lower than what would have been in the absence of faults and FFC. However, this throughput overhead will be temporary if no further changes in traffic demand (or network topology) occur. In the example, once $s2$ and $s3$ are successfully configured, the flow $s1\rightarrow s4$ will be allowed to increase its rate to 10 units in the next period. Even if temporary, lowered throughput is an overhead of robustness provided by FFC.

As with FEC, FFC overhead is lower for lower protection levels. In the example above, if robustness to the configuration failure of up to one switch (k=1) is desired, we can safely install the configuration of Figure 5(b), which supports 7 units of flow $s1\rightarrow s4$. There will be no congestion if $s2$ or $s3$ (not both) fail to configure.

3.2 FFC for data plane faults

Data plane FFC guarantees no congestion occurs after rescaling when up to k links or switches fail. To see how to achieve this, we revisit Figure 2, where congestion occurs when link $s2$-$s4$ fails. However, if we spread traffic as in Figure 4(a), any single link failure ($k = 1$) will not cause congestion. For example, if link $s2$-$s4$ fails, the traffic distribution after rescaling is shown in Figure 4(b).

Like control plane FFC, data plane FFC can also lower throughput. However, while the overhead of control plane FFC is borne only when the network is updated, the overhead of data plane FFC is persistent. The difference in the two cases is that when a link or switch fails, network capacity is reduced. To not congest after a failure, we must leave spare capacity to absorb the traffic that was being carried by the failed element.

3.3 Applying FFC

Operators use FFC by choosing the value of k, which can be different for the two types of faults. This choice allows them to make an informed trade-off between the risk of congestion due to faults and network throughput. For a fixed network capacity, higher values of k may lower throughput but also lower the risk of congestion that can hurt congestion-sensitive applications.[2]

[2]Throughput is lowered by rate limiting at end hosts [1, 2], not by dropping packets in the network, which to applications is indistinguishable from congestion. Packets dropped in the network generally hurt congestion-sensitive applications more than lim-

Throughput loss with FFC is not a given. First, in well-provisioned networks with ample spare capacity and in networks that do not rate control traffic, network throughput with and without FFC will be similar. The difference will be that FFC will strategically spread traffic in a way that guarantees protection against faults. Second, in networks that carry multi-priority traffic, the spare capacity that is needed to protect high-priority traffic can be used to carry lowest-priority (congestion-tolerant) traffic. Doing so leads to negligible loss in total throughput while protecting high-priority traffic (§8.4).

Finally, our FFC techniques have another use case. For a given traffic demand, they can precisely determine the link capacities needed for a desired level of protection from fault-induced congestion. Today, operators that want to minimize the risk of congestion tend to heavily over-provision the network, and even that does not provide any guarantee. We do not explore this use case in this paper, but enabling it requires straightforward modifications to the FFC constraints that we outline in the following sections.

3.4 Challenges and overview of techniques

Practical realization of FFC for arbitrary topologies and traffic demands poses two challenges. The first is the scalability with which robust traffic distributions can be computed. If there are n network entities, and we want to be robust to up to k of them failing, FFC has to deal with $\sum_{j=1}^{k} \binom{n}{j}$ failure cases. Thus, naive, enumeration-based approaches are computationally intractable for large networks. We must meet the computational challenge while meeting the second challenge of minimizing the loss in network throughput. If network throughput were not a concern, a trivially robust solution is to not carry any traffic.

We address these challenges by first formulating the conditions on traffic quantity and spread as linear constraints. While this formulation is precise, it has a large number of constraints. We then reduce these large number of constraints to a much smaller number by observing that the constraints can be transformed to what we call the "bounded M-sum" problem and all constraints in such a problem can be reduced to a single constraint on the largest (or smallest) M variables. Finally, we encode these variables using efficient linear expressions with the aid of sorting networks [10]. The result is an FFC formulation with $O(kn)$ constraints.

Our techniques exploit two properties of our setting. First, the impact of a fault is easy to model. If switch configuration fails, it sticks to its old configuration; if a link fails, ingress switches deterministically rescale traffic. This simplicity allows us to capture the conditions imposed by FFC using efficient, linear constraints. Second, while faults are common, the fault ratio (i.e., the fraction of elements that fail) is low. Thus, it suffices to guard against a small number of faults (k). Solving for high values of k would be computationally intensive and impose a high throughput overhead.

4. Basic FFC Formulation

We now describe how to formulate and solve FFC for a basic TE setting. Our formulation is highly flexible, and in §5 we extend it to a range of other TE settings.

4.1 Basic TE (without FFC)

The basic TE problem can be formulated as follows, with key notations summarized in Table 1. The input is a graph G=(V, E), where V and E are sets of switches and directed links between switches. Each link $e \in E$ has a capacity c_e. The traffic demand is a set of flows, where each flow f is (aggregated) traffic from an ingress to an egress switch. The bandwidth demand of f in a TE interval is d_f and its traffic can be carried on a set of pre-established tunnels T_f.

iting their sending rate at source (to which they can adapt). For instance, detecting and recovering from a dropped TCP SYN takes time, which magnifies flow completion time for short transfers.

TE Input	G	Network graph with switches V and links E.
	F = $\{f\}$	Flows aggregated by ingress-engress switches.
	d_f	The bandwidth demand of f in a TE interval.
	c_e	The bandwidth capacity of link e.
	T_f	The set of tunnels that are set up for flow f.
	$L[t, e]$	1 if tunnel t uses link e and 0 otherwise.
	$S[t, v]$	1 if tunnel t's source is switch v and 0 otherwise.
TE Output	b_f	The granted bandwidth to flow f.
	$a_{f,t}$	The bandwidth allocated for flow f on tunnel t.
TE Others	$\beta_{f,t}$	The upper-bound of flow f's traffic.
	$w_{f,t}$	The traffic splitting weight of flow f on tunnel t.
FFC	p_f & p	The max number of f's tunnels traverse a link.
	q_f & q	The max number of f's tunnels traverse a switch.
	k_c, k_e, k_v	The number of configuration, link and switch failures that FFC protects the network against.
	τ_f	The min number of f's residual tunnels with up to k_e link and k_v switch failures.

Table 1: Key notations in FFC formulation.

The TE output is bandwidth allocation $\{b_f|\forall f\}$ of each flow and how much of the flow can traverse each tunnel $\{a_{f,t}|\forall f, t \in T_f\}$. In networks where the flow rate cannot be controlled, the TE output is only the latter (and $b_f = d_f$).

The TE problem can be solved based on path-constrained multi-commodity flow problem [14], as follows:

$$\max \quad \sum_{f \in F} b_f \tag{1}$$

$$\text{s.t.} \forall e \in E: \quad \sum_{f \in F, t \in T_f} a_{f,t} L[t,e] \leq c_e \tag{2}$$

$$\forall f \in F: \quad \sum_{t \in T_f} a_{f,t} \geq b_f \tag{3}$$

$$\forall f \in F, t \in T_f: \quad 0 \leq b_f \leq d_f;\ 0 \leq a_{f,t} \tag{4}$$

where the binary indicator $L[t, e]$ denotes if tunnel t traverses link e. The TE objective in this formulation is to maximize network throughput (Eqn. 1). We consider other objectives in §5.3. Eqn. 2 states that no link should be overloaded, and Eqn. 3 states that the sum of the allocation of a flow across all its tunnels should not be less than its allocated rate.[3] Eqn. 4 states that the rate allocated to a flow should not exceed demand, and all variables are non-negative.

The formulation above captures TE for both wide area networks and data center networks (DCN). One difference is that in DCNs, given the larger scale, TE focuses only on large flows (elephants) and link capacity refers to what is not used by small flows (mice).

To implement the computed solution, the TE controller updates the flow's rate limiters to $\{b_f\}$ and ingress switches to use traffic splitting weights of $w_{f,t} = a_{f,t} / \sum_{t \in T_f} a_{f,t}$.

4.2 Modeling control plane faults

For control plane faults, the goal of FFC is to compute the new configuration ($\{b_f\}$, $\{a_{f,t}\}$) such that no congestion will occur as long as k_c or fewer switches fail to update their old configuration ($\{b'_f\}$, $\{a'_{f,t}\}$). Another type of control plane fault is a failure to configure a rate limiter, which we will consider in §5.5. Let λ_v=1 denote a configuration failure for at least one of the flows with v as the ingress switch; λ_v=0 denotes that configuration for *all* flows starting at v succeeds. An individual case of control plane faults in the network can be represented by a vector $\boldsymbol{\lambda}$=$[\lambda_v|v \in V]$ that indicates the status of each switch. Thus, FFC that is robust to k_c faults requires that the network have no overloaded link under the set of cases $\Lambda_{k_c} = \{\boldsymbol{\lambda}| \sum_{v \in V} \lambda_v \leq k_c\}$.

This requirement can be captured as:

[3] Using '$\geq$' instead of '=' in Eqn. 3 simplifies the exposition of FFC for data plane faults. For TE without FFC, given the goal of maximizing b_f, using '$\geq$' is equivalent to using '='.

$$\forall e \in E, \boldsymbol{\lambda} \in \Lambda_{k_c} : \sum_{v \in V} \{(1-\lambda_v)\hat{a}_{v,e} + \lambda_v \hat{\beta}_{v,e}\} \leq c_e \quad (5)$$

where $\hat{a}_{v,e}$ is the total traffic that can arrive at link e from flows starting at v if there is no configuration fault. That is:

$$\forall v \in V, e \in E : \quad \hat{a}_{v,e} = \textstyle\sum_{f \in F, t \in T_f} a_{f,t} L[t,e] S[t,v] \quad (6)$$

where indicator $S[t,v]$ denotes if tunnel t's source switch is v.

In Eqn. 5, $\hat{\beta}_{v,e}$ is the upper bound on link e 's traffic from flows starting at v when a fault occurs ($\lambda_v = 1$). That is:

$$\forall v \in V, e \in E : \quad \hat{\beta}_{v,e} = \textstyle\sum_{f \in F, t \in T_f} \beta_{f,t} L[t,e] S[t,v] \quad (7)$$

where $\beta_{f,t}$ is the upper bound on flow f's traffic on tunnel t when a faults occurs for f. Since we assume the updates in rate limiters are successful, $\beta_{f,t}$ can be modeled as:

$$\forall f \in F, t \in T_f : \quad \beta_{f,t} = \max\{w'_{f,t} b_f, a_{f,t}\} \quad (8)$$

where $w'_{f,t}$ is flow f's splitting weight for tunnel t in the old configuration (which is known).

Adding Eqns. 5–8 to the basic TE formulation can, in theory, find TE configurations that are robust to k_c control plane faults. However, Eqn. 5 contains $|E| \sum_{j=1}^{k_c} \binom{|V|}{j}$ constraints because Λ_{k_c} has $\sum_{j=1}^{k_c} \binom{|V|}{j}$ failures cases. Directly solving for so many constraints is computationally intractable. We outline in §4.4 how we reduce this problem to a smaller number of equivalent constraints.

4.3 Modeling data plane faults

For data plane faults, the goal of FFC is to compute flow allocations such that no congestion occurs even after up to k_e links fail *and* up to k_v switches fail. The guarantee is for link failures that are not incident on the failed switches. Since switch failures imply link failures, one could protect against them by considering only link failures [15]. But we explicitly consider switch failures because switches can have a large number of incident links; protecting against switch failures implicitly (using link failures) would require a high value of k_e. This approach would significantly hurt throughput because it will protect against arbitrary combinations of up to k_e links, a much stronger condition than protecting against k_e incident links on the same switch.

We model data plane FFC as follows. Let μ_e=1 denote that link e has failed, and η_v=1 denote that switch v has failed; the indicator values are 0 otherwise. Then, a case of data plane fault can be represented by $(\boldsymbol{\mu}, \boldsymbol{\eta})$ in which vector $\boldsymbol{\mu} = [\mu_e | e \in E]$ and $\boldsymbol{\eta} = [\eta_v | v \in V]$. Thus, TE that is robust to k_e link failures and k_v switch failures requires that there is no overloaded link under the set of hardware failure cases $U_{k_e,k_v} = \{(\boldsymbol{\mu}, \boldsymbol{\eta}) | \sum_e \mu_e \leq k_e, \sum_v \eta_v \leq k_v\}$.

Recall that data plane faults can cause congestion because they alter traffic distribution over the network when ingress switches rescale traffic, that is, move it from the impacted to the residual tunnels for the flow. Given a fault case $(\boldsymbol{\mu}, \boldsymbol{\eta})$, we know the residual tunnels $T_f^{\boldsymbol{\mu},\boldsymbol{\eta}}$ of each flow f—those that do not traverse any failed link or switch. FFC requires that f's residual tunnels be able to hold its allocated rate.

$$\forall f \in F, (\boldsymbol{\mu}, \boldsymbol{\eta}) \in U_{k_e,k_v} : \quad \textstyle\sum_{t \in T_f^{\boldsymbol{\mu},\boldsymbol{\eta}}} a_{f,t} \geq b_f \quad (9)$$

Further, if a flow f has no residual tunnels ($T_f^{\boldsymbol{\mu},\boldsymbol{\eta}} = \emptyset$) under a failure case $(\boldsymbol{\mu}, \boldsymbol{\eta})$, its flow size b_f should be fixed to 0.

Eqn. 9 also guarantees that no link will be overloaded:

LEMMA 1. *A TE configuration ($\{a_{f,t}\}$, $\{b_f\}$) that satisfies constraints Eqns. 2–4, 9 under fault case $(\boldsymbol{\mu}, \boldsymbol{\eta})$ causes no link overload after all ingress switches rescale.*

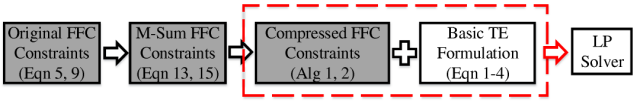


Figure 7: Overall process for FFC-TE computation.

PROOF. When a data plane failure case $(\boldsymbol{\mu}, \boldsymbol{\eta})$ happens, the traffic load of a flow f on a residual tunnel $t \in T_f^{\boldsymbol{\mu},\boldsymbol{\eta}}$ is:

$$b_{f,t}^{\boldsymbol{\mu},\boldsymbol{\eta}} = \frac{a_{t,f}}{\sum_{t \in T_f^{\boldsymbol{\mu},\boldsymbol{\eta}}} a_{f,t}} * b_f \leq \frac{a_{t,f}}{b_f} * b_f = a_{f,t} \quad (10)$$

which is directly derived from Eqn. 9. Therefore, we know the total traffic load on a link e is:

$$\forall e \in E : \quad \textstyle\sum_{f, t \in T_f^{\boldsymbol{\mu},\boldsymbol{\eta}}} b_{f,t}^{\boldsymbol{\mu},\boldsymbol{\eta}} L[t,e] \leq \sum_{f, t \in T_f} a_{f,t} L[t,e] \leq c_e$$

which finishes the proof. □

As for control plane faults, adding these constraints to the basic TE will in theory yield a solution that is robust to data plane faults, but directly solving for these constraints is intractable given the large number of possible failure cases in U_{k_e,k_v}. Before describing how to solve these constraints, we briefly discuss how careful tunnel layout can improve robustness to data plane faults as well as reduce the overhead of data plane FFC.

Robust tunnel layout One observation from Eqn. 9 is that higher the number of residual tunnels, greater the network throughput. Thus, we can improve throughput by creating tunnels such that flows lose as few tunnels as possible when faults occur. Ideally, each flow would lose at most one tunnel upon a fault, but this guarantee requires switch-disjoint (and thus also link-disjoint) tunnels, which limits flows to a small number of tunnels in networks with low path diversity. That would in turn restrict network throughput, as more tunnels are better able to utilize network capacity.

To balance these needs, we recommend (p, q) link-switch disjoint tunnels. For an individual flow, at most p tunnels should traverse a link and at most q tunnels should traverse a switch. The parameters p and q can be flow specific. Algorithms to find link and switch disjoint paths can be extended to find (p, q) link-switch disjoint tunnels; we omit details for space constraints.

4.4 Efficiently solving FFC constraints

To tractably solve the large number of FFC constraints, we transform them to a "bounded M-sum" problem and then encode the transformed problem using a sorting network. Figure 7 shows the overall process of the FFC-TE computation.

4.4.1 Transformation to bounded M-sum

We define the *bounded M-sum* problem as: Given a set of N variables, the sum of any M of those should be less (or more) than a bound B. Formally, if N_M is the set of all possible variable subsets with cardinality $\leq M$, we have:

$$\forall S \in N_M : \sum_{n_i \in S} n_i \leq B \quad (11)$$

The interesting aspects of this problem are: 1) FFC constraints can be transformed to it; and 2) while the original formulation has a large number of constraints, all of them are satisfied as long as one constraint involving the largest M variables is satisfied. If n^j is an expression for the j-th largest variable in N, all constraints above hold if:

$$\sum_{j=1}^{M} n^j \leq B \quad (12)$$

Thus, if we can find efficient (linear) expressions for the largest M variables in N, we can replace all original constraints with one constraint. Before showing how to find such expressions, we first show how to transform FFC constraints into bounded M-sum problem.

Algorithm 1: LargestValues(X, M)

```
1  [Input] X: an array of variables
2  [Input] M: the number of largest values to extract
3  [Output] Y: an array of new variables in which Y[i] (0 ≤ i ≤ M) is
     the ith largest element in X
4  [Output] C: a set of constraints between X and Y
5  Y ← ∅; C ← ∅;
6  // Pop M largest variables from X.
7  while |Y| < M do
8      y*, X, C' ← BubbleMax(X) ;
9      Y ← Y + {y*}; C ← C + C' ;
10 return Y, C ;
```

Algorithm 2: BubbleMax(X)

```
1  [Input] X: an array of variables
2  [Output] x*: a variable that represents max{X}
3  [Output] Y: an array that represent X \ {x*}
4  [Output] C: a set of constraints among X, Y and x*
5  x* ← X.pop(); Y ← ∅; C ← ∅ ;
6  while X ≠ ∅ do
7      x ← X.pop() ;
8      x_max, x_min ← new variables ;
9      // Make two new constraints.
10     c1 ← 2 * x_max = x + x* + |x − x*| ;
11     c2 ← 2 * x_min = x + x* − |x − x*| ;
12     x* ← x_max; Y ← Y + {x_min}; C ← C + {c1, c2};
13 return x*, Y, C ;
```

Control plane faults Eqn. 5 of control plane FFC can be equivalently re-written as:

$$\forall e \in E, \boldsymbol{\lambda} \in \Lambda_{k_c} : \sum_{v \in V} \lambda_v(\hat{\beta}_{v,e} - \hat{a}_{v,e}) \leq c_e - \sum_v \hat{a}_{v,e} \qquad (13)$$

Let $D = \{\hat{\beta}_{v,e} - \hat{a}_{v,e} | v \in V\}$ and d^j be the jth largest element in D. Since $\hat{\beta}_{v,e} - \hat{a}_{v,e} \geq 0$, Eqn. 13 is equivalent to

$$\forall e \in E : \sum_{j=1}^{k_c} d^j \leq c_e - \sum_v \hat{a}_{v,e} \qquad (14)$$

Thus, we have transformed the original $|E| \times |\Lambda_{k_c}|$ constraints into $|E|$ constraints, one for each link.

Data plane faults Assume that the tunnels of flow f are (p_f, q_f) link-switch disjoint. The values (p_f, q_f) are computable for any given tunnel layout; the layout does not have to use the robust strategy above. Then, for any data plane fault case $(\boldsymbol{\mu}, \boldsymbol{\eta}) \in U_{k_e,k_v}$, the number of residual tunnels is no less than $\tau_f = |T_f| - k_e p_f - k_v q_f$. Suppose $a^j_{f,t}$ is the jth smallest (not largest) element in $A_f = \{a_{f,t} | t \in T_f\}$, then the following guarantees that all constraints in Eqn. 9 are satisfied:

$$\forall f : \quad \sum_{j=1}^{\tau_f} a^j_{f,t} \geq b_f \qquad (15)$$

The guarantee holds because the left-hand side of Eqn. 15 is the worst-case rate allocation that flow f can have from its residual tunnels under any case in U_{k_e,k_v}.

Unlike control plane faults, the transformation from Eqn. 9 to Eqn. 15 does not preserve equivalence. Satisfying Eqn. 15 satisfies Eqn. 9, but not vice versa. In the special cases link failures with link-disjoint tunnels and switch failures with switch-disjoint tunnels, the two are equivalent.

Interestingly, however, the imprecision of Eqn. 15 allows it to protect against fault cases beyond U_{k_e,k_v}. It essentially protects the network against any fault case where the number of tunnel failures is no more than $k_t = k_e p_f + k_v q_f$. Suppose (p_f, q_f)=(1, 3) and our desired protection level is up to 3 links failures and no switch failure (k_e=3, k_v=0), with Eqn. 15 we also simultaneously protect the network against one arbitrary switch failure and no link failure (k_e=0, k_v=1). We leverage this effect in our experiments.

4.4.2 *Encoding for largest (or smallest) M variables*

We now explain how we express the largest M variables as linear constraints. When added to other TE constraints, they help efficiently compute FFC traffic distribution.

Our method is based on sorting networks [10], which are networks of compare-swap operators that can sort any array of N values. An example network to sort 4 values is shown in Figure 8(a). This network is based on the merge sort algorithm. Each compare-swap operator takes two inputs from left, and it moves the higher input upwards and the lower downwards.

The characteristic of sorting networks that we exploit is that the sequence of compare-swap operations is independent of the input, unlike many sorting algorithms (*e.g.* quick sort) where the next comparison depends on the outcome of previous ones. This characteristic allows us to encode the relationship between original variables and the jth largest element in them with linear expressions.

We also exploit that we are interested in only the largest M values, rather than sorting them all, which allows us to use a partial network. Practical sorting networks require $O(Nlog(N)^2)$ compare-swap operators, while we use a partial network with $O(NM)$ operators. This difference is substantial since N is large and M is small in our setting—equivalent to the number of faults we guard against.

We base our network on bubble sort; unlike other algorithms, its premature termination after M stages yields the largest M values. Figure 8(b) illustrates our strategy for the case of finding the largest 2 of 4 values. The first stage finds the largest element, and the second finds the second largest.

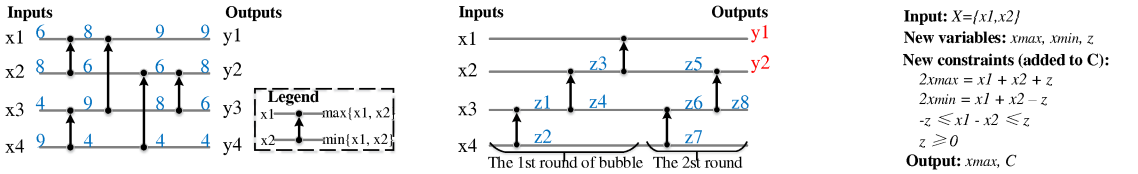
Algorithm 1 shows the pseudo-code for generating expressions for the largest M values. It operates in M steps, and in each step, it builds an expression for the largest of the remaining values. Algorithm 2 shows the pseudo-code for building the expression for the largest value. Figure 8(c) shows a concrete example of Algorithm 2 in which $|x - x^*|$ in Lines 10 and 11 is encoded with linear formulations.

A similar approach can find the expressions for the smallest M values. The only difference is that compare-swap operators that push the lower (not higher) of the two values upwards are used.

4.4.3 *Throughput and computational overhead*

Our methods have the following properties with respect to throughput overhead: 1) Our control plane FFC scheme is optimal; and 2) Our data plane FFC scheme is optimal for the special cases of link failures with link disjoint tunnels and switch failures with switch disjoint tunnels. Optimal means that no other scheme will have lower overhead for the same degree of protection. We omit proofs from this paper due to space constraints.

The computational overhead of our methods can be characterized using the number of additional variables and constraints they introduce in the LP. The basic (non-FFC) TE problem has $2|F|+|E|$ constraints and $\sum_f |T_f| + |F|$ variables. Control plane FFC introduces $|E|$ constraints (Eqn. 14) plus up to $4k_c|V||E|$ constraints and up to $3k_c|V||E|$ variables to encode the partial sorting network. (Even though we show only 2 new variables and 2 constraints in Algorithm 2, multiplicative factors of 4 and 3 stem from converting the absolute values in Lines 10∼11 into standard linear constraints.) Data plane FFC introduces up to $|F| + 4\sum_f |T_f| \min\{|T_f| - \tau_f, \tau_f\}$ constraints and up to $3\sum_f |T_f| \min\{T_f - \tau_f, \tau_f\}$ variables. Recall that $\tau_f = |T_f| - k_e p_f - k_v q_f$.



(a) An example of sorting networks (b) A network to find the largest-2 elements (c) A concrete example of Alg-2

Figure 8: Sorting networks and finding largest elements among variables.

4.5 Combined FFC for both faults types

To simultaneously protect against control and data plane faults, we simply include both types of constraints in TE computations. It will take three parameters (k_c, k_e, k_v) and the guarantee is that no congestion will occur as long as switch configuration failures, link failures, and switch failures are up to k_c, k_e , and k_v , respectively.

A subtle issue can arise in combined protection settings right after data plane faults bigger than the protection level (*e.g.*, number of failed links $> k_e$). Due to such faults, some links may get congested and there may be no way to move traffic away from them while being also robust to control plane faults. For instance, assume that due to a big data plane fault, after rescaling, a link e with capacity 10 gets 7 units of traffic each from three flows that start at different ingress switches. Moving the extra 11 units of traffic away from the link requires updating at least two switches. But if we are protecting against two control plane faults (k_c=2) planning for this movement is impossible; there would be no feasible solution to the FFC constraints. To handle this issue, we allow unprotected moves for overloaded links by setting k_c=0 for such links in Eqn. 5. In theory, overloaded links can arise after big control plane faults as well, but in our experience, such faults are unlikely to create congestion that is so severe that traffic cannot be moved in a manner that is robust to (further) control plane faults.

5. Extending Basic FFC

Our FFC formulation is not only efficient but also flexible. We now show how it extends to a wide range of TE settings.

5.1 Traffic with different priorities

Earlier, we assumed that all traffic has the same priority; some networks may use multiple priorities to differentiate between applications with different performance requirements [1, 2]. FFC can be extended to this setting to offer different levels of protections to different priorities. The TE solution for higher priority traffic is computed first with a custom protection level (k_c^h, k_e^h and k_v^h), and the TE solution of lower priority traffic is computed next with its own protection level (k_c^l, k_e^l and k_v^l). This cascading computation is already done to support multiple priorities [1, 2]; computation for lower priority traffic uses residual link capacity (not actually used higher priority traffic).

A requirement for the extension above is that the protection level for high priority should not be smaller ($k_x^h \geq k_x^l, x \in \{c, e, v\}$), which is a desirable property anyway. Otherwise, the FFC-TE for lower priorities may not have a feasible solution because the configuration for high priority traffic may violate FFC constraints.

5.2 Congestion-free updates

Some networks use multi-step updates to preclude transient congestion caused by different switches updating their configuration at different times [2, 16]. The basic idea is to find a chain of intermediate TE configurations A^i=$\{a^i_{f,t}\}(0 < i < m)$ that update the network from current configuration A^0 to the desired configuration A^m. The transition between each pair of adjacent TE configurations is guaranteed to be congestion free irrespective of the order in which the switches apply updates. Such intermediate TEs are found using the following key constraint:

$$\forall e \in E, i : \sum_v \max\{\hat{a}^{i-1}_{v,e}, \hat{a}^i_{v,e}\} \leq c_e \quad (16)$$

This constraint captures the condition that each link e should be able to accommodate the maximum traffic, across adjacent configurations, it gets from each ingress switch v. After computing the intermediate configurations, the TE controller updates the network step-by-step: $A^0 \rightarrow A^1 \ldots A^m$.

With congestion-free updates, control plane faults will not cause congestion, but will block the update process; the preceding step must complete before the next step can be taken. In this setting, FFC can ensure that the update can proceed from A^{i-1} to A^i as long as the cumulative number of faults (across all steps thus far) is k_c or fewer. We can find such intermediate configurations by replacing Eqn. 16 with:

$$\forall e \in E, \lambda \in \Lambda_{k_c}, i :$$
$$\textstyle\sum_v \lambda_v \max\{\hat{\beta}^0_{v,e}, \ldots \hat{\beta}^i_{v,e}\} + (1 - \lambda_v)\max\{\hat{a}^{i-1}_{v,e}, \hat{a}^i_{v,e}\} \leq c_e$$

This is a large number of constraints, but we can solve them efficiently by transforming them to the bounded M-sum problem (§4.4). We omit details for space constraints.

5.3 Optimizing for fairness

Earlier, we assumed that TE objective was to maximize network throughput; another common objective is fairness among flows [1, 2]. Fairness typically introduces more constraints in the TE problem, and simply including those constraints will yield FFC-TE with fairness. As a concrete example, consider the iterative approximate-max-min fair method of SWAN [2]. It solves the basic TE (Eqns. 1–4) multiple times, each time with an upper-bound on flow allocations (b_f). The bound is iteratively increased by multiplying it by a factor α. The allocation of flows that are unable to reach the bound in a given iteration are frozen for future iterations, and the iterations continue until the bound goes above the maximum flow demand (d_f). This process ensures that flows with large demands cannot get a higher allocation until the allocation of other flows cannot be increased to at least that level. It yields flow allocations that are provably at most α away from true max-min fair allocation (which is computationally hard to calculate).

The same process can be used largely unmodified to compute TE solutions that are both fair and provide FFC. The only difference is that each iteration includes FFC-related constraints as well.

5.4 TE without flow rate control

In some networks, such as ISP backbones, controlling the rates of incoming flows is not possible. Instead of allocating flow rates, the goal of TE tends to be to configure the network such that maximum link utilization (MLU) is minimized while carrying the offered demand ($\{d_f\}$). The basic (non-FFC) network configuration ($\{a_{f,t}\}$) can be computed for this setting by replacing Eqns. 1 and 2 with the following objective function and constraint:

Figure 9: The network topology emulated in our testbed.

$$\begin{aligned} &\text{min.} && \Theta(u) \\ &\text{s.t.}\forall e: && u \geq \textstyle\sum_v \hat{a}_{v,e}/c_e \end{aligned}$$

where u denotes MLU and Θ is a function of MLU that needs to be minimized. Eqn. 2, which limited the amount of traffic a link had to carry, is replaced because the offered demand may be higher than what is routable while honoring link capacities. In the equations above, u is allowed to be higher than 1 (i.e., over-subscribed links).

To compute FFC configuration, constraints for data plane FFC stay the same, but control plane FFC requires changing the objective function and additional constraints, as follows:

$$\begin{aligned} &\text{min.} && \Theta(u) + \sigma\Theta(u_\text{f}) \\ &\forall e \in E, \lambda \in \Lambda_{k_c}: && u_\text{f} \geq \textstyle\sum_v \{\lambda_v \hat{\beta_{v,e}} + (1-\lambda_v)\hat{a}_{v,e}\}/c_e \end{aligned}$$

where $\sigma > 0$ is a coefficient that balances the importance of MLU in normal cases u and MLU when faults occur u_f. These constraints can be solved using the method of §4.4.

5.5 Control plane faults for rate limiters

Earlier, we assumed that configuration updates for rate limiters always succeed; some networks may experience update failures for rate limiters as well. If ingress switches and rate limiters are updated independently, the traffic load of a flow on a tunnel can be a mix of old or new traffic splitting weights and old or new flow sizes. We can account for this by modifying Eqn. 8 to:

$$\beta_{f,t} = \max\{a'_{f,t}, b'_f w_{f,t}, b_f w'_{f,t}, a_{f,t}\} \tag{17}$$

In some networks, the updates on switches and rate limiters are ordered to ensure that there is no congestion due to transient inconsistencies in flow sizes and tunnel weights [2]. The order is: if f's size is increasing ($b'_f < b_f$), the traffic splitting weights at ingress switches are updated first and the rate limiter is updated after (and only if) that succeeds; otherwise, the rate limiter is updated first and the splitting weights are updated after that succeeds. Thus, if $b'_f < b_f$, the combination of new flow size and old weights ($b_f w'_{f,t}$) will not occur and $b'_f w_{f,t} < a_{f,t}$; similarly if $b'_f > b_f$, the combination of old flow size and new weights ($b'_f w_{f,t}$) will not occur and $b_f w'_{f,t} < a'_{f,t}$. We can then simplify Eqn. 17 to:

$$\beta_{f,t} = \max\{a'_{f,t}, a_{f,t}\} \tag{18}$$

Besides simplifying the FFC formulation, ordering of updates on switches and rate limiters also helps lower the overhead of FFC. It reduces the number of possible traffic configurations for which we must be prepared.

5.6 Uncertainty in current TE

Earlier, we assumed that while computing the next TE configuration, the controller exactly knows the current configuration ($\{a'_{f,t}\}$, $\{b'_f\}$) of each flow. Sometimes, however, there may be uncertainty in the configuration of some flows, e.g., if update commands were sent in the last round to change configuration from ($\{a''_{f,t}\}, \{b''_f\}$) to ($\{a'_{f,t}\}, \{b'_f\}$) but the success of some updates could not be confirmed. In such an event, the configuration of the flow can be either ($\{a''_{f,t}\}, \{b''_f\}$) or ($\{a'_{f,t}\}, \{b'_f\}$).

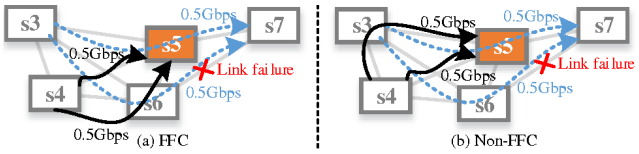


Figure 10: Traffic distribution before link s6-s7 fails.

We can explicitly account for this uncertainty in FFC computations. Suppose $\mathcal{F}$ is a set of flows with uncertain configurations. Instead of computing yet another configuration for them, we try to bring their configuration up-to-date and, in terms of link capacity allocation, we plan for them to be in either of the last two configurations. The following constraints capture this strategy:

$$\begin{aligned} &\forall f \in \mathcal{F}, t \in T_f: && b_f = b'_f; a_{f,t} = a'_{f,t} \\ &\forall f \in \mathcal{F}, t \in T_f: && \beta_{f,t} = \max\{a''_{f,t}, a'_{f,t}\} \end{aligned}$$

6. Implementation

Our FFC controller is implemented as a drop-in replacement for existing TE controllers. It takes as input traffic demand (per-priority ingress-egress flow), network topology, and current traffic, and produces the allocation of each flow and configuration for each switch. The additional configuration for our controller includes the protection level (k_c, k_e, k_v) for each priority. It implements all the extensions in §5. We use Solver Foundation v3.0.2 with CPLEX plugin v12.5.0 as our LP solver.

Our implementation includes a few optimizations that lower computational burden without practically impacting FFC properties. For control plane FFC, observe that not all ingress switches have traffic on each link. Thus, in Eqn. 14, if a switch v has no load on a link e in the old TE, we ignore v when considering the safety of e. Similarly, we also ignore switches that have little traffic load—less than 0.001% of capacity—on e in the old TE because the impact of such switches not updating is negligible. For data plane FFC, we pick mice flows—those that collectively carry less than 1% of the traffic—and fix their tunnel bandwidth allocations with the constraint $a_{f,t} = \frac{b_f}{\tau_f}$, which suffices to satisfy Eqn. 15, rather than using sorting networks.

7. Testbed Evaluation

We begin our evaluation of FFC by first performing experiments on a testbed. These experiments show the value of FFC with real switches and actual delays in detecting and reacting to failures. The next section shows the value of FFC at scale using simulations with data from real networks.

Our testbed emulates a WAN with 8 sites spread across 4 continents, as shown in Figure 9. Each site has 5 servers that generate traffic and 1 WAN-facing switch (Arista 7050T). The capacity of every cross-site link is 1 Gbps. The TE controller is in New York ($s5$), and we emulate delays for control messages based on geographic distances. We update switch rules via a custom software agent running on the switches (which we have found to be more performant and reliable than the built-in software). We use *iperf* on the servers to generate UDP traffic flows with specified rates and measure packet loss according to both iPerf's server reports and the packet counters in the switches. All switches run link liveness detection protocol, and they report any failures to ingress switches. Upon hearing about a failure, ingress switches rescale traffic away from the impacted tunnels.

We conducted several experiments to study the behavior of FFC and non-FFC TE, but we present results only from a simple, representative experiment that illustrates their differences for a data

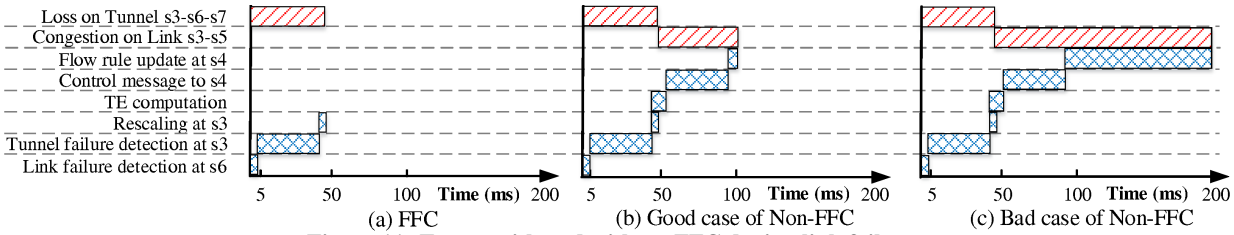


Figure 11: Events with and without FFC during link failures.

plane fault. This experiment has two flows, $s3 \rightarrow s7$ and $s4 \rightarrow s5$, each with demand 1 Gbps. Figure 10 shows how this traffic was spread in the case of FFC and non-FFC. The main difference is that FFC uses tunnel s4-s6-s5, instead of s4-s3-s5, to transmit 0.5 Gbps traffic, which provides protection against any single-link failure. We conducted many trials with this setup; in each we failed link s6-s7 and observed the ensuing events.

FFC behavior was consistent across trials. Figure 11(a) shows an example. The x-axis is a time line relative to when the link failure was injected and y-axis lists the events that could happen in a failure reaction. Shadowed blocks denote the start and end of the event. We see $s6$ detects link failure within 5 ms, and $s3$ hears about it within 45 ms. $s3$ rescales and moves all traffic to the residual tunnel within 2 ms. Packet loss on tunnel s3-s6-s7 stops immediately after that. We thus see that losses in FFC are a purely a function of the time it takes for fault detection and rescaling delay. We see that modulo propagation delay, which is fundamental, today's switches can detect faults and rescale quickly. FFC ensures that these activities suffice at eliminating congestion, and the TE controller does not need to react.

The situation is more complex in the non-FFC case. After s3 rescales, link s4-s5 will get congested as it starts getting 1.5 Gbps of traffic; see Figure 9(b). To remedy this, the TE controller computes a new solution in which s4 moves 0.5 Gbps of traffic to tunnel s4-s6-s5 and updates the switch s4. Figure 11(b) shows the best case for non-FFC in our trials, in which the update itself was quick (within 5 ms). We see rescaling-related loss on s3-s5 stops within 45 ms, given the propagation delay from the controller to the switch s4. Overall, the network was congested for twice the time compared to FFC. Figure 11(c) shows a bad case for non-FFC, in which the switch s4 took a long time to apply the update. Consequently, the congestion lasted for much longer.

8. Data-driven Evaluation

We now evaluate FFC with topology, traffic, and failure data from real networks. We first micro-benchmark its throughput and computation cost, then study its end-to-end impact in single- and multi-priority networks, and finally study its impact on update time for multi-step updates. We start by describing our data sources and experimental methodology.

8.1 Experimental methodology

Networks We experiment with two wide area networks. The first, which we call `L-Net`, has $O(50)$ sites globally with $O(100)$ switches and $O(1000)$ links. We have data on link capacities, traffic, and data plane faults for `L-Net`. To ensure that our results are robust to network topology, we also use B4's site-level topology with 12 sites [1], which we call `S-Net`. Since we do not know the internal details of `S-Net`, we assume that there are two switches per site, and each site-level link is composed of four 10 Gbps switch-level links between each of the four inter-site switch pairs.

Traffic demand We collect network traffic logs and aggregate it into ingress-egress flows. We partition time into 5-minute bins (the TE interval) and the demand of the flow for an interval is the average bandwidth it consumed. For `L-Net`, we use its traffic logs directly. For `S-Net`, we use traffic logs from another inter-datacenter network (not `L-Net`) and synthesize demand by mapping sites from the other network onto `S-Net`, as in earlier work [2].

For multi-priority experiments, we partition traffic into three priorities based on their source services. Following SWAN [2], high priority is for interactive services, which are highly sensitive to loss and delay; medium priority is for services that are less sensitive but are still impacted by packet loss (*e.g.* deadline-driven transfers); and low priority is for background services (*e.g.* data replications).

To understand the impact of network provisioning level, we study three cases. The first is a well-utilized network where capacity matches demand. To mimic this, we scale the demands of all flows (uniformly) such that 99% of demands per interval are satisfied. The results below refer to this case as traffic scale of 1. The other two cases are a well-provisioned network and an under-provisioned network, and we use traffic scales of 0.5 and 2 to mimic them.

In terms of relationship to practice, ISP networks are typically well-provisioned and use single priority traffic. Inter-datacenter networks are well-utilized and use multiple priorities to protect high priority traffic from short-term demand increases of lower priority traffic [1, 2].

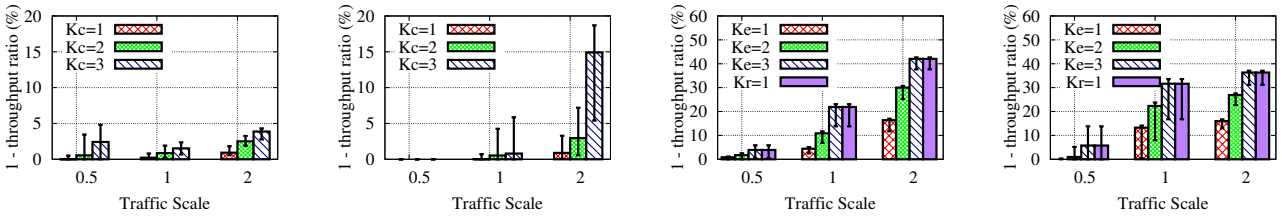
Failures and switch update models For `L-Net`, we inject data plane faults as per logs from the network. For `S-Net`, we inject faults based on the per-link and per-switch failure probabilities derived from `L-Net` logs. We assume that it takes 5 ms for a switch to detect a link failure, and the time it takes for an ingress switch to hear about the failure and rescale depends on the propagation delay.

We consider two models of switch update behaviors. In the `Realistic` model, we use the update delay distribution reported for B4 (§2.3) and a configuration failure rate of 1%. In the `Optimistic` model, we use the update delay distribution we measured in a controlled environment (§2.3) and no configuration failures.

TE approaches We compare TE with and without FFC. Without FFC, when a link or switch fails, the TE controller immediately computes a new traffic distribution and updates the network. With FFC, the controller does not react to data plane faults unless it is on the edge of protection level. If the link protection level k_e=2, the controller reacts only after 2 links have failed. reaction logic is per-priority; when a given priority traffic is at the edge of protection level, only that traffic is re-adjusted. Both approaches use the same set of tunnels. We use (1, 3)-link-switch disjoint strategy to establish six tunnels for each flow. Except in the micro-benchmark experiments, if a flow's demand is not satisfied in an interval, the remaining bytes add to its demand in the next interval.

We evaluated both max-throughput and max-min fairness as TE objectives but present results only for the former. Results for the latter are qualitatively similar.

Metrics We use two metrics to capture the behavior of FFC:



(a) Control Plane FFC (L-Net) (b) Control Plane FFC (S-Net) (c) Data Plane FFC (L-Net) (d) Data Plane FFC (S-Net)

Figure 12: Throughput overhead of FFC. Bars show the 90th %-ile value and error bars show the 50th and 99th %-iles.

i) *Throughput ratio* Network throughput with FFC versus without FFC. One minus this ratio is the overhead of FFC.

ii) *Data loss ratio* Bytes lost with FFC versus without FFC. We count bytes lost due to both blackholes and congestion. Blackhole losses occur during the time between when a link fails and when ingress switches rescale; all packets that traverse the failed link are considered lost in this period. Congestion losses are computed based on link capacity and the duration and degree to which the link is oversubscribed. This simple measure overestimates loss if traffic is congestion controlled, as such traffic will reduce sending rate in response to early losses. But it serves as a good proxy for capturing the intensity and duration of congestion, without requiring us to model the intricate relationship between loss and performance for a diverse set of applications.

8.2 Micro-benchmarks

Throughput overhead To micro-benchmark the overhead of FFC, we compute traffic distributions with and without it for successive TE intervals. In each interval, we exclude any unfinished data from previous intervals, so that both approaches operate on the same demand in each interval, independent of preceding allocations. Failure and switch models do not impact these experiments as we do not inject faults but study overhead when no faults happen.

Figures 12(a) and 12(b) show the overhead (1 - throughput ratio) of control plane FFC in isolation (no data plane FFC) for three protection levels. For each traffic scale, they plot three percentile values. We see that the overhead of control plane FFC is small—under 5% even at 90th percentile level for all but one setting—and, expectedly, increases with the protection level. We also see that the overhead generally increases with the traffic scale. As the network gets busier, it becomes harder to accommodate all traffic in a way that modifies traffic spread robustly. The results are similar for L-Net and S-Net.

Figures 12(c) and 12(d) show the overhead of data plane FFC for 1-3 links failures and 1 switch failure. Because we use (1, 3)-link-switch disjoint strategy to construct tunnels, the cases with k_e = 3 and k_v = 1 are identical. We see that the overhead is low at traffic scale 0.5 (a well-provisioned network), but it grows quickly as traffic scales and protection level increases.

Based on these results and that multiple link failures in a short amount of time and switch failures are uncommon (but do occur), we recommend using a protection level of (k_c, k_e, k_v) =$(2, 1, 0)$ in single-priority networks. In multi-priority networks, we recommend a higher protection level for high-priority flows. As the relative volume of this traffic is typically under 50%, FFC's overhead will still be small.

Computation time We benchmark computation time on an ordinary PC with Intel i5 M540 2.53 Ghz CPU (2 cores) and 4GB RAM. Table 2 lists the average computation time for FFC with different protection levels and without FFC. Because of the imprecision of Eqn. 15 (§4.4.1), FFC configuration of $(3, 3, 0)$ simultaneously provides protection for $(3, 0, 1)$. We use $(3, 0, 1) \cup (3, 3, 0)$ as a shorthand for this combined protection level. We see that even at a high protection level, FFC computation takes only 1.2 seconds for large network like L-Net.

	FFC (3, 3, 0) ∪ (3, 0, 1)	FFC (2, 1, 0)	Non-FFC
L-Net	1.2 sec	0.3 sec	0.05 sec
S-Net	0.03 sec	0.02 sec	0.015 sec

Table 2: TE computation time with and without FFC.

In contrast, we find that directly computing TE with the original FFC constraints (Eqns. 5 and 9) takes longer than 12 hours in each case. Such long computation time renders TE ineffective given that the traffic demands and network topology likely change before a solution is obtained.

8.3 Single-priority traffic

We now conduct end-to-end evaluation of FFC with realistic failure and switch models. This section focuses on the single-priority case, and the next on multi-priority case. Per above, we configure FFC as (k_c, k_e, k_v) =$(2, 1, 0)$.

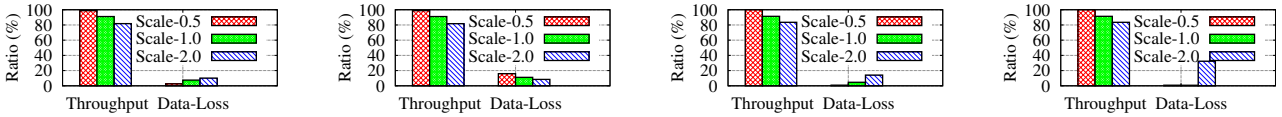
Figure 13 shows the results for the two networks, the two switch models, and the three traffic scales. Focusing first on the case of well-provisioned networks (traffic scale 0.5), which is the common case for single-priority traffic, we see that throughput difference with and without FFC is negligible. At the same time, FFC offers 10–20 times reduction in data loss. We do not report absolute amount of lost data to maintain confidentiality for link capacities in L-Net. But we note that the loss is substantial—well above $O(100GB)$ per day—and in almost all cases of link oversubscription, the amount of data lost is well above typical buffer capacities.

For the case of well-utilized networks (traffic scale 1), we see that FFC carries over 90% of the traffic carried without FFC and cuts data loss by a factor of 7–130 (0.72–11.5%).

Closer inspection of lost data reveals that both with and without FFC, blackhole losses due to delay in rescaling after a link failure are negligible. With FFC, the primary factor behind losses is cases where the number of data plane faults is greater than the protection level. Without FFC, any control and data plane fault leads to congestive losses, and both types of faults contribute roughly equally.

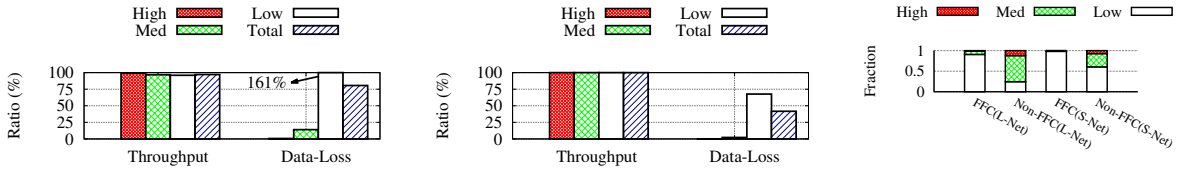
Though the absolute amount of data loss is lower for the Optimistic switch model, we observe in Figure 13 that the relative gain of FFC is similar for both switch models. Thus, FFC helps even if switch updates times could be improved to those in controlled environments today and all configuration failures could be eliminated.

Trade-off between throughput and data loss Using link failures as an example, Figure 15 shows the tradeoff between data loss and throughput overhead as we change the protection level. This experiment uses the Realistic switch model and no protection from control plane faults or switch failures. For each traffic scale, the point at (100, 100) corresponds to no protection $(k_e = 0)$ and successive points to the left correspond to increasing values of link protection. Expectedly, the less data we want to lose, the higher the throughput overhead, though throughput overhead grows slower than loss reduction (linear versus exponential).



(a) `Realistic` on `L-Net` (b) `Optimistic` on `L-Net` (c) `Realistic` on `S-Net` (d) `Optimistic` on `S-Net`

Figure 13: Throughput and data loss ratio for FFC with single-priority traffic.



(a) `L-Net` (b) `S-Net` (c) Fractions of data loss

Figure 14: Throughput and data loss ratio for FFC with multi-priority traffic with `Realistic` switch model.

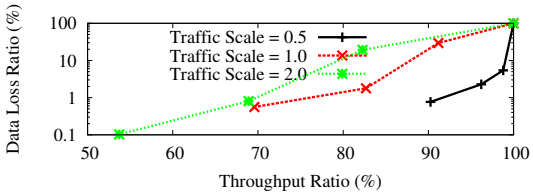
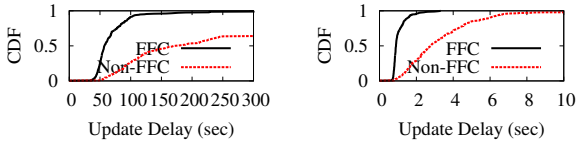


Figure 15: The tradeoff of data loss and throughput.



(a) `Realistic` (b) `Optimistic`

Figure 16: Update time for congestion-free updates.

8.4 Multi-priority traffic

We now consider multi-priority traffic. FFC offers the opportunity to provide greater protection for high-priority traffic with minimal loss in total throughput because low-priority traffic can be safely carried over the capacity that is set aside to protect high-priority traffic. When congestion occurs, priority queueing, which preferentially drops lower-priority packets, protects high-priority traffic as long as its rate does not exceed link capacity.

We use different protection levels for different priorities: $(k_c, k_e, k_v) = (3, 0, 1) \cup (3, 3, 0)$ for high-priority traffic to provide it strong protection, $(2, 1, 0)$ for medium priority, and $(0, 0, 0)$ for low priority. Thus, low priority traffic is not protected at all, which lets it use all network capacity.

Figure 14 shows the results for both networks. This experiment uses traffic scale of 1 (well-utilized network). In Figures 14(a) and 14(b), we see that the throughput ratio is close to 100%, for total traffic as well as for individual priorities. Recall that in a single-priority network, throughput ratio is 90% and loss ratio is 0.72–11.5% for this traffic scale. Given the basic FFC trade-off, the increase in total throughput for this multi-priority network must accompany a decrease in protection from congestion. The loss ratio for total traffic bears this out; it is 40–80%.

What is interesting, however, is the data loss ratios of different priorities. The high-priority traffic suffers almost no loss (< 0.01%), and the loss has been concentrated towards low-priority traffic. The effect is extreme for `L-Net`, where low-priority traffic loses more bytes with FFC than without FFC.

Figure 14(c) shows the relative fraction of lost bytes for each priority. With FFC, there is negligible loss (< 0.01%) for high-priority traffic and a small amount (2–7%) of loss for medium-priority traffic. In contrast, without FFC, 5–15% of the bytes lost are high-priority and 30–70% are medium-priority. Thus, priority queueing by itself is not sufficient to prevent congestion for high priority traffic. FFC provides strong protection without throughput loss.

8.5 Congestion-free network updates

We now consider the case of congestion-free, multi-step updates. Recall that in this setting, control plane faults do not lead to congestion losses, but network updates can be slow and can even stall. We evaluate the speed of network updates with and without FFC.

Figure 16 shows the results for `L-Net` for both switch models. With the `Realistic` model, without FFC, 40% of the updates do not finish within 300 seconds. Since that is the TE interval, it is the maximum time we wait for the update to finish. This poor performance stems from the fact that even a single switch that takes a long time or fails to update altogether hurts the update process. FFC allows for faster updates by being robust to a small number of control plane faults (in this case k_c=2). FFC provides significant reduction in update time even with the `Optimistic` model, in which there are no configuration failures, only occasional delays. The median and 99th percentile speedup is a factor of three.

Faster updates lead to a more nimble network that can quickly react to demands bursts. They can also improve throughput by enabling a shorter TE interval, which enables the network to handle shorter-term demand variations.

9. Related Work

We build upon the rich line of work in TE algorithms and systems. Researchers have studied various aspects of this problem, including i) how to implement close-to-optimal traffic distributions in different settings such as link-state routing [17], MPLS environments [18], and SDNs [1, 2, 19, 20]; ii) how to stably adapt to changing traffic demands [21, 22]; iii) how to reroute traffic after failures such that minimal changes are needed [23]; and iv) how to find efficient backup paths [24, 25].

For brevity, we focus below on proactive techniques to make TE robust. Our key contributions in this space are i) proactively handling control plane faults; and ii) proactively handling data plane faults in a way that is both robust to any combination of up to k failures and works with today's commodity switches.

Data plane faults To prevent rescaling-induced congestion after a data plane fault, Suchara *et al.* [26] modify the ingress switch's rescaling behavior. Instead of simple proportional rescaling, tunnel splitting weights are based on the set of residual tunnels. These weights are pre-computed and configured at the switch. Unlike our data plane FFC, which protects against any combination of up to k

faults, this approach can handle only a modest number of potential failure cases as there are exponential number of residual tunnel sets.

For a distributed TE setting, R3 [15] proposes an approach for congestion-free fast re-route [27], in which adjacent routers route around failed links. The routing behavior is determined by a fast online computation, aided by offline computation that is done in advance. Like FFC, R3 protects against any combination of up to k link failures. There are two key differences, however. Our constraint reduction framework handles both control and data plane faults; R3's approach handles only data plane faults (and we could not extend it to control plane faults). Second, while R3 mixes proactive and reactive elements, FFC is purely proactive, which is a better approach when control plane faults can occur.

Further, both works above [15, 26] require changes to the switch hardware or software. We build solely on existing primitives.

Control plane faults Dionysus enables faster network updates in the face of switch configuration delays and failures, by dynamically deciding the order in which rule updates are applied [28]. While it reduces update time in general, it does not address the fundamental limitations of reactive approaches—they kick in after congestion occurs and the slowest switch can bottleneck the update process—which we seek to address. FFC and Dionysus are complementary. The former ensures that reactions are not needed for common case faults, and the latter ensures that, when needed, reactions are fast.

Demand uncertainty In networks where incoming traffic rate is not controlled, oblivious routing [29] and COPE [30] compute TE configurations that are robust to changes in traffic demand or errors in demand prediction. They do not consider control and data plane faults which are our focus. An interesting area of future investigation is if our approach for control faults in rate limiting, which produce uncertainty in traffic entering the network, can be extended to handle demand uncertainty. That would enable a common framework for handling both faults and demand uncertainty.

10. Conclusions

We developed FFC methods that proactively protect a network from congestion and packet loss due to data and control plane faults. These methods have low overhead in terms of network throughput—even optimal in some cases—and are computationally efficient. Using testbed experiments and data from real networks, we showed how FFC is useful in a variety of settings. For instance, in well-provisioned networks, it can reduce packet loss by a factor of 7–130; in well-utilized networks that carry traffic with multiple priorities, it can reduce loss for high-priority traffic to almost zero, with negligible reduction in total network throughput.

Acknowledgments Mohan Nanduri helped collect data for our experiments; Mohit Singh suggested the use of sorting networks for constraint reduction; Meg Walraed-Sullivan, Vyas Sekar (our shepherd), and the anonymous reviewers provided valuable feedback on drafts of this paper. We thank them all.

11. 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 *SIGCOMM'13*.

[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 *SIGCOMM'13*.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *NSDI'10*.

[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalag, P. Sharma, and S. Banerjee, "Devoflow: Scaling Flow Management for High-Performance Networks," in *SIGCOMM'11*.

[5] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting Data Center Networks with Multi-gigabit Wireless Links," in *SIGCOMM'11*.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *CoNext'11*.

[7] P. Gill, N. Jain, and N. Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," in *SIGCOMM'11*.

[8] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of Failures in an Operational IP Backbone Network," *IEEE/ACM Transactions Networking*, 2008.

[9] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California Fault Lines: Understanding the Causes and Impact of Network Failures," in *SIGCOMM'10*.

[10] K. E. Batcher, "Sorting Networks and Their Applications," in *AFIPS'68 (Spring)*.

[11] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, "Cloud Control with Distributed Rate Limiting," in *SIGCOMM'07*.

[12] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards Predictable Datacenter Networks," in *SIGCOMM'11*.

[13] "OpenFlow 1.1." `http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf`.

[14] S. Even, A. Itai, and A. Shamir, "On the Complexity of Timetable and Multicommodity Flow Problems," *SIAM Journal on Computing*, 1976.

[15] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang, "R3: Resilient Routing Reconfiguration," in *SIGCOMM'10*.

[16] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz, "zUpdate: Updating Data Center Networks with Zero Loss," in *SIGCOMM'13*.

[17] D. Xu, M. Chiang, and J. Rexford, "Link-state Routing with Hop-by-hop Forwarding Can Achieve Optimal Traffic Engineering," *IEEE/ACM Transactions on Networking*, 2011.

[18] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS Adaptive Traffic Engineering," in *INFOCOM'01*.

[19] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A Retrospective on Evolving SDN," in *HotSDN'12*.

[20] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNS with Openflow," in *SIGCOMM'11*.

[21] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the Tightrope: Responsive Yet Stable Traffic Engineering," in *SIGCOMM'05*.

[22] M. Kodialam, T. V. Lakshman, and S. Sengupta, "Efficient and Robust Routing of Highly Variable Traffic," in *HotNets'04*.

[23] D. Applegate, L. Breslau, and E. Cohen, "Coping with Network Failures: Routing Strategies for Optimal Demand Oblivious Restoration," in *SIGMETRICS'04*.

[24] K. Kar, M. Kodialam, and T. V. Lakshman, "Routing Restorable Bandwidth Guaranteed Connections Using Maximum 2-route Flows," *IEEE/ACM Transactions on Networking*, 2003.

[25] M. Kodialam, A. Member, T. V. Lakshman, and S. Member, "Dynamic Routing of Restorable Bandwidth-guaranteed Tunnels using Aggregated Network Resource Usage Information," *IEEE/ACM Transactions on Networking*, 2003.

[26] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network Architecture for Joint Failure Recovery and Traffic Engineering," in *SIGMETRICS'11*.

[27] E. A. Atlas and E. A. Zinin, "Basic Specification for IP Fast Reroute: Loop-Free Alternates."

[28] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic Scheduling of Network Updates," in *SIGCOMM'14*.

[29] D. Applegate and E. Cohen, "Making Intra-domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs," in *SIGCOMM'03*.

[30] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "COPE: Traffic Engineering in Dynamic Networks," in *SIGCOMM'06*.