Towards Highly Available Clos-Based WAN Routers

Sucha Supittayapornpong
University of Southern California

Barath Raghavan
University of Southern California

Ramesh Govindan
University of Southern California

ABSTRACT

The performance and availability of cloud and content providers often depends on the wide area networks (WANs) they use to interconnect their datacenters. WAN routers, which connect to each other using trunks (bundles of links), are sometimes built using an internal Clos topology connecting merchant-silicon switches. As such, these routers are susceptible to internal link and switch failures, resulting in reduced capacity and low availability. Based on the observation that today's WAN routers use relatively simple trunk wiring and routing techniques, we explore the design of novel wiring and more sophisticated routing techniques to increase failure resilience. Specifically, we describe techniques to 1) optimize trunk wiring to increase effective internal router capacity so as to be resilient to internal failures, 2) compute the effective capacity under different failure patterns, and 3) use these to compute compact routing tables under different failure patterns, since switches have limited routing table sizes. Our evaluations show that our approach can mask failures of up to 75% of switches in some cases without exceeding routing table limits, whereas competing techniques can sometimes lose half of a WAN router's capacity with a single failure.

CCS CONCEPTS

• **Networks** → **Network design and planning algorithms**; **Network performance analysis**; **Topology analysis and generation**; *Routers*; • **Computer systems organization** → **Reliability**; **Fault-tolerant network topologies**; Availability;

KEYWORDS

Wide area network, Data center, Clos-based topology, WCMP, Capacity optimization, Robust router

ACM Reference Format:
Sucha Supittayapornpong, Barath Raghavan, and Ramesh Govindan. 2019. Towards Highly Available Clos-Based WAN Routers. In *SIGCOMM '19: 2019 Conference of the ACM Special Interest Group on Data Communication, August 19–23, 2019, Beijing, China.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3341302.3342086

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS 1413978 and CNS 1705086. Computation for the work described in this paper was supported by the University of Southern California's Center for High-Performance Computing (https://hpcc.usc.edu).

SIGCOMM '19, August 19–23, 2019, Beijing, China

ACM ISBN 978-1-4503-5956-6/19/08…$15.00
https://doi.org/10.1145/3341302.3342086

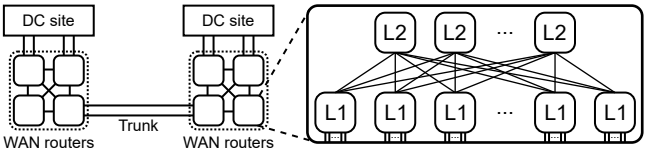

Figure 1: Globally-distributed WANs and WAN routers.

1 INTRODUCTION

Large cloud and content providers (like Google, Facebook, Netflix, and Microsoft) are expanding their own wide-area networks (WANs) to meet service-level latency and throughput objectives and to achieve high availability, all at low cost. These globally-distributed WANs consist of dozens of *sites* [19, 25]. At each site (Figure 1), one or more *WAN routers* forward (a) ingress and egress traffic to one or more datacenters at the site and (b) transit traffic between WAN sites. To achieve this, each WAN router connects to datacenters and to WAN routers at the same or other sites using *trunks*, which are logical collections of physical links that provide high aggregate capacity [34].

The design of the WAN topology and its routing is crucial to the performance and availability of the entire WAN. WANs must carry large traffic volumes, often in the terabits per second (Tbps), so they incorporate novel router designs that achieve high performance and high utilization at low cost. However, the effect of small failures within WAN routers, or within trunks, can disproportionately degrade the capacity of WAN routers, resulting in lower service availability or in degraded user-perceived performance. In this paper, we focus on the failure resilience of a common type of WAN router designed using a non-blocking Clos [12] topology.

Clos-Based WAN Routers. In the last decade, some content providers and router vendors have designed high-aggregate-capacity WAN routers by arranging merchant-silicon switching chips (*e.g.,* the Broadcom Trident series [24], Arista 7050X3 series [21]) in a topology shown in Figure 1. In this topology, traffic ingresses and egresses the WAN router at *external* ports attached to the lower half of the lower layer of switches (also called layer-1 or *L1* switches). Incoming traffic traverses internal links, bounces off layer-2 or *L2* switches, and then exits an external port towards a datacenter border router or another WAN router. The use of commodity merchant silicon ensures low cost, and the design of the topology ensures *non-blocking* performance; a non-blocking switch or router can satisfy *any traffic matrix*, which specifies the volume of traffic between each ingress-egress trunk pair.

The aggregate capacity of the Clos-Based WAN router (henceforth, simply WAN router) is a function of the number of switches used, which itself is a function of the switching chip *radix* (the number of switch ports) and the per-port capacity. With a 16-port switch, the WAN router will require 16 L1 switches and 8 L2 switches to achieve the non-blocking property and will have 128 full-duplex

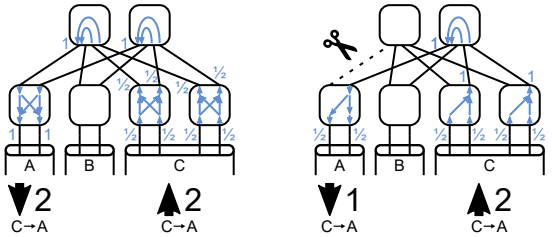

Figure 2: A Clos-based WAN router with three trunks. At left, we show how two units of traffic can be routed from trunk C to trunk A. When a single link fails, only 50% of the demand is satisfied.

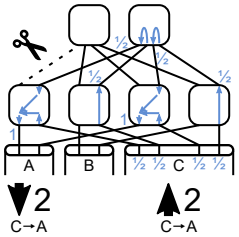

Figure 3: By carefully wiring trunks, and forwarding ingress traffic out on L1 switches whenever possible, a WAN router can *mask* failure of an internal link.

external ports [19]. If each port can support 40 Gbps of traffic, the entire WAN router has a capacity of 5.12 Tbps.

Internal Routing in Clos-Based WAN Routers. Today, WAN routers use a simple internal routing strategy. For example, consider Figure 2 which depicts a smaller WAN router constructed using 4-port switches. This router interconnects three external trunks, **A**, **B** and **C** with, respectively, 2, 2, and 4 links. Each link in this topology, whether internal or external, has one unit of capacity. Suppose that 2 units of traffic enter trunk **C** destined for **A**. The ingress L1 switch uses *ECMP-based forwarding*, in which ingress traffic is evenly load-balanced across the two internal links towards two L2 switches. These L2 switches each then forward the traffic to the switch connected to trunk **A**'s links.

The Impact of Internal Failures in a WAN Router. In a WAN router, one or more L1 or L2 switches and/or one or more internal links can fail (external links can also fail, and, while we do not consider such failures in this paper, our approach extends to this case (§7)). Such a failure can reduce the *effective capacity* of the switch. In Figure 2, the failure of a single internal link (out of a total of 8 internal links) reduces the router's effective capacity by 50%, and the router only satisfies one unit of the demand from **C** to **A**.

To understand how this example generalizes to more realistic settings, Table 1 shows the reduction in effective capacity in a 128-port switch with 4 trunks for different failure configurations. Specifically, the table shows the *maximum* reduction in effective capacity across *all possible traffic matrices*, using a methodology developed in this paper and described later. A 128-port switch has 8 L2 switches, 16 L1 switches, and 128 internal links. As Table 1 shows, *a single internal link failure can reduce effective capacity by 50%*, and four concurrent link failures (out of 128) can result in an effective capacity of a *quarter of the original capacity*. L1 switch failures can be equally catastrophic: removing 2 out of 16 L1 switch failures *can reduce the effective capacity to zero* for this trunk configuration. However, the WAN router *degrades gracefully* with L2 switch failures: each L2 switch failure reduces capacity by $1/8^{\text{th}}$, as it should.

Content and cloud providers strive to simultaneously achieve high utilization (especially in a WAN where the cost of wide-area links are high [25]) and high availability (to satisfy service-level objectives). To achieve this, WAN routers must mask as many failures as possible, and gracefully degrade when not. This motivates our search for techniques to improve the resilience of WAN routers.

Towards Better Failure Resilience in WAN Routers. Ideally, a WAN router *should be able to completely mask internal failures*. However, there are limits to failure masking. For example, when an L1 switch fails, capacity will necessarily degrade since its ports cannot ingress or egress traffic. Similarly, if enough L2 switches or internal links fail, it may not be possible to mask these failures. In these cases, we require that capacity *degrade gracefully*: the loss in capacity should be proportional to the fraction of failed hardware resources (links or switches).

To understand how to achieve these requirements, consider Figure 3 which explains how we can *mask* the single link failure in Figure 2. Figure 3 illustrates that, to minimize the impact of failures, we can: (a) *carefully arrange trunks* across the WAN router's external ports, and (b) *route traffic at L1 switches* when possible. These techniques can avoid the capacity degradation of Figure 2. For example, trunk **A** now connects to the first and third (from the left) L1 switch, instead of only the first L1 switch. This permits the first L1 switch to forward traffic from **C** to **A** and send less traffic up to the L2 switches. This *early forwarding* reduces the *upflow* (total traffic from L1 switches to L2 switches) and can completely mask the single link failure.

Contributions. Our paper leverages the above two insights to design topology and routing schemes to maximize failure masking and ensure graceful degradation in WAN routers. Indeed, our approach can *mask all L2 and link failures* in Table 1, and gracefully degrade L1 switch failures. To achieve this, our paper makes three contributions.

Our first contribution (§2) is the design of *minimal-upflow trunk wiring*. Re-arranging the ports assigned to each trunk (the trunk wiring) permits early forwarding of traffic between two trunks at the L1 switch, without even traversing internal links. However, early forwarding is not always possible: some traffic (the *upflow*) needs to traverse L2 switches. Intuitively, minimizing the upflow can improve the ability of the WAN router to sustain capacity in the face of L2 and internal link failures. Given a trunk configuration, we study how to *minimize the total upflow* from L1 to L2 switches. The challenge in doing this is that, in practice, while trunk configurations change on day, week, or month timescales, the inter-trunk

	Effective capacity		
No. failures	Link	L2 switch	L1 switch
1	50.0%	87.5%	50%
2	48.3%	75.0%	0%
3	25.0%	62.5%	0%
4	24.1%	50.0%	0%

Table 1: Effective capacity of a 128-port WAN router with 4 trunks consisting of (16, 32, 32, 48) links under different failures.

traffic matrix can change more frequently. Thus, we need to determine a trunk wiring that minimizes upflow across *all possible traffic matrices*. Simply enumerating all possible traffic matrices is infeasible at the scale of today's WAN routers. Instead, we observe that, given a trunk configuration, we can enumerate a smaller set of *extreme* traffic matrices that dominate the set of all traffic matrices. Using this observation, we develop a mixed-integer linear program (MILP) formulation for the minimal-upflow trunk wiring problem. We also prove that, for some trunk configurations, it is possible to derive minimal-upflow trunk wiring without solving an MILP. While our MILP formulation scales to reasonable problem sizes, it hits scaling limits for 512-port WAN routers built from 32-port switches. For these larger routers, we develop a fast heuristic for trunk wiring that achieves the minimal upflow almost every time.

Our second contribution (§3) is the design of a method to compute *effective capacity under failures*. Specifically, given a minimal-upflow trunk wiring, we need a way to determine, for, say, failures of links or of L1/L2 switches, the maximum capacity reduction in the WAN router across all possible traffic matrices. This is necessary because, in practice, a traffic engineering algorithm such as the one used in [19, 25] requires an estimate of the residual router capacity for a given failure pattern. Because traffic engineering needs to be fast, it may be infeasible to run an algorithm to determine the maximum capacity reduction across all traffic matrices when a set of failures occurs. Instead, we seek to *pre-compute* capacity reduction for each failure pattern, but the number of possible failure patterns can be prohibitively large. We show, however, that the symmetry in WAN routers permits the enumeration of a small number of *canonical failure patterns*, and any failure pattern is isomorphic to one of these canonical failure patterns. We develop an algorithm to determine a canonical failure pattern from any given failure pattern and use it to enumerate all canonical failure patterns. We then devise easily parallelizable optimization formulations to determine the effective capacity under failure.

Our third contribution (§4) is to develop *compact forwarding tables* for a minimal-upflow trunk wiring and a given failure pattern. While today's switches use ECMP, in which traffic is evenly load-balanced across links, our approach requires a *weighted* version of ECMP (called WCMP [40]). Unfortunately, today, the way WCMP is achieved in chips can *inflate* forwarding table sizes, and switches have limited forwarding tables. To meet table size constraints, one can quantize the weights for different flows, which can potentially result in lower effective capacity than computed in §3. We show that it is possible to optimize compact forwarding tables to achieve minimal-upflow trunk wiring *without sacrificing effective capacity under failures*, and provide scalable approximations for this problem.

Our evaluations (§5) show that our approach can mask up to 6 concurrent link or L2 switch failures in a WAN router, while a baseline wiring strategy that uses ECMP or WCMP *cannot even mask a single failure*. Our approach can tolerate failures of up to half of the L1 switches, but the baseline wiring can only tolerate 1-3 such failures. Random wiring is less effective than our approach, often having an upflow 2-3× higher, with correspondingly lower resilience. We also demonstrate that our approach's resilience does not require exceeding hardware table limits. Finally, we show that our optimizations for using extreme traffic matrices and canonicalizing failure patterns are effective: the latter can reduce computational complexity by 3-5 orders of magnitude.

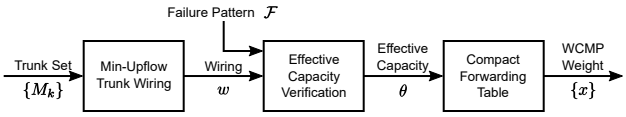

Figure 4: Overview of our approach.

Putting It All Together. Our contributions collectively result in the processing pipeline shown in Figure 4. The input to the first stage of the pipeline is a trunk description used to determine a minimal-upflow wiring. The next stage in the pipeline takes this wiring, together with a failure pattern, and produces the effective capacity of the WAN router under that failure pattern. This effective capacity is then used to optimize compact routing tables.

Implications. As cloud and content providers simultaneously strive to achieve high utilization, high availability, and low cost, they need tools that enable them to make these tradeoffs in a principled way. In the WAN setting, our work shows that it is possible to mask significant failures (*i.e.,* tolerate failures without losing capacity). Using our approach, they may also be able to reduce cost by reducing the number of L2 switches and internal links (in today's WAN routers, internal links use expensive optics).

Ethics. This work does not raise any ethical issues.

2 MINIMAL-UPFLOW TRUNK WIRING

In this section, we explore the first challenge: how to find the minimal-upflow wiring for a set of trunks in a WAN router.

2.1 Background

By virtue of being at the top of the routing hierarchy, WAN routers carry large volumes of aggregate traffic, often up to terabits per second. However, low-cost commodity switching chips offer per-port speeds of 40 Gbps to 100 Gbps. To meet capacity requirements using these chips, WAN routers must have a large number of ports. In Google's B4, routers have 128 or 512 ports. Moreover, the physical topology of WANs is *sparse*. WAN routers interconnect data centers in large metropolitan areas; since long-distance cables are expensive, each WAN router usually connects to a small number of other WAN routers. The sample topology in [25] shows a WAN router connected to 4-5 other WAN routers.

Thus, *multiple* ports on a WAN router connect to corresponding ports on an adjacent WAN router; this collection of physical links is *trunk*. All the links in a trunk are of the same capacity (*e.g.,* 40 Gbps) because all L1 and L2 switches use the same type of switching chip (*e.g.,* a 16x40 Gbps chip). A router receives traffic on one trunk and may forward this traffic to one or more other trunks. When doing so, it evenly splits outbound traffic across all links in the trunk between the two routers. This uniform splitting enables better utilization of trunk links, and allows a traffic engineering algorithm to abstract the WAN router as a single node with a fixed capacity [25]. All our techniques in this and subsequent sections model this crucial constraint. More important, this even split maximizes early forwarding opportunities.

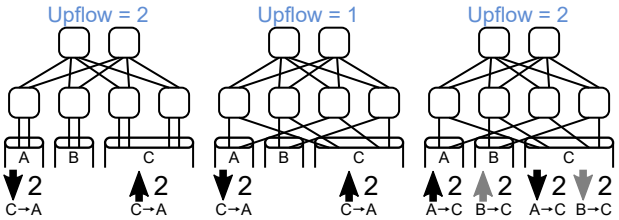

Figure 5: Upflow depends both on trunk wiring and traffic matrix.

Operators configure trunks when commissioning a router, and occasionally reconfigure them afterwards. However, trunk reconfiguration happens at timescales of weeks or months, since it requires manual labor to rewire the links on a trunk.

2.2 Goal and Challenges

Figure 3 shows how carefully wiring the trunks across the external ports of a WAN router can improve resilience by permitting early forwarding, which reduces upflow. Upflow (defined more formally below) is the aggregate traffic sent from L1 switches to L2 switches. In this section, we ask: *How can we design scalable methods to compute the trunk wiring pattern that minimizes upflow?*

This question is challenging because upflow depends not just on the trunk wiring but also on the trunk-to-trunk *traffic matrix* (the (i, j)-th entry in a traffic matrix represents the total traffic from trunk i to trunk j). Figure 5 shows an example that illustrates this (we omit the detailed upflow calculations for brevity). The topology on the left and in the middle have different wiring but the same traffic matrices, and have different upflow. The middle and the right topologies have the same trunk wiring but different traffic matrices, and also have different upflow. Unfortunately, traffic matrices at a WAN router can change frequently based on changes in application demand: numbers from [25] suggest that traffic engineering (TE) computations run once every 2.4 mins on average. Each such computation can potentially change the traffic matrix at the router. At these timescales, it is infeasible to re-wire trunks in response to each such change because trunk wiring is a manual operation.

The rest of this section describes an optimization formulation, and associated scaling methods to compute a minimal-upflow trunk wiring that addresses this challenge by computing a wiring that *minimizes the maximum upflow across all possible traffic matrices.*

2.3 Formalizing Upflow

Input and Output. The input to our algorithm is a set $\{M_1, \ldots, M_K\}$ of K trunks. We call this set a *trunk set*, where the k^{th} trunk has M_k links, for $k \in \mathcal{K}$ and $\mathcal{K} = \{1, \ldots, K\}$ is an index set. The output of the algorithm is a matching (association) between external ports of the WAN router and links in each trunk that minimizes upflow (the *minimal-upflow trunk wiring*). Recall that WAN router external ports are all connected to L1 switches.

Traffic Matrix. A traffic matrix $T = [t_{ij}]_{K\times K}$ is a K-by-K matrix containing traffic rate t_{ij} going from a trunk i to a trunk j for every $i, j \in \mathcal{K}$. We normalize the traffic rate t_{ij} by the link capacity without loss of generality, since every link in a trunk has the same capacity. We assume $t_{ii} = 0$ for all $i \in \mathcal{K}$, *i.e.,* that no traffic received on a trunk exits on the same trunk. Finally, let $\mathcal{T}$ be the set of all possible traffic matrices for a given trunk set $\{M_k\}$.

Defining Upflow. We call the normalized aggregate rate of traffic sent from L1 switches to L2 switches the *upflow*. To formalize upflow, let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the sets of L1 switches and L2 switches respectively. In a Clos-based WAN router, the number of L1 switches is twice as many as L2 switches. Let P denote the number of external ports in each L1 switch. As described above, the upflow at each L1 switch depends on the traffic matrix and how trunks connect to the WAN router. Specifically, let w_{sk} be the number of links from trunk k wired to switch s for all $s \in \mathcal{L}_1$ and all $k \in \mathcal{K}$, and let w be a vector of these w_{sk}'s. We call w a *trunk configuration.* Given a traffic matrix $T = [t_{ij}] \in \mathcal{T}$, the upflow rate (or, simply, upflow) for traffic from trunk i to j at switch s is

$$u_s^{ij}(w, T) = \left[\frac{w_{si}t_{ij}}{M_i} - \frac{w_{sj}t_{ij}}{M_j}\right]_+ \quad \text{for all } s \in \mathcal{L}_1, (i, j) \in \mathcal{K}^2, \quad (1)$$

where $[a]_+ = \max(a, 0)$ is a positive projection. This formulation relies on the observation that the total traffic on a trunk is evenly split across its constituent links. Then, the first term on the right hand side measures the fraction of incoming traffic on trunk i destined to trunk j that arrives at switch s. The second term measures, at switch s, the fraction of trunk j's outgoing capacity for traffic from trunk i. The two terms together determine how much of the incoming traffic on trunk i *cannot* be "early forwarded" (*i.e.,* how much must traverse L2 switches). It follows then that the total upflow to L2 switches for a given trunk configuration w and a traffic matrix T is:

$$U(w, T) = \sum_{s\in\mathcal{L}_1} \sum_{i\in\mathcal{K}} \sum_{j\in\mathcal{K}} u_s^{ij}(w, T). \quad (2)$$

Minimizing Upflow. Our approach tries to minimize total upflow because, in doing so, it reduces the internal capacity required in the WAN router, thereby enabling the router to mask many failures of internal links or L2 switches. Because upflow depends on the traffic matrix, we attempt to find that trunk wiring configuration w that minimizes the maximum upflow across *all* possible traffic matrices:

$$\min_{w\in\mathcal{W}} \max_{T\in\mathcal{T}} U(w, T), \quad (3)$$

$\mathcal{W}$ is a feasible set of wiring constraints, defined as:

$$\mathcal{W} = \left\{ w_{sk} \in \mathbb{Z}_+^{|\mathcal{L}_1|\times K} : \begin{array}{ll} \sum_{k\in\mathcal{K}} w_{sk} \le P & \text{for all } s \in \mathcal{L}_1 \\ \sum_{s\in\mathcal{L}_1} w_{sk} = M_k & \text{for all } k \in \mathcal{K} \end{array} \right\},$$

where $\mathbb{Z}_+$ a set of non-negative integers. The first constraint ensures that the wiring at a switch does not exceed the number of external ports P, and the second ensures that each link in every trunk connects to a port.

2.4 Scaling

Extreme Traffic Matrices. Unfortunately, this formulation is intractable because *there can be infinitely many traffic matrices* in $\mathcal{T}$. We observe that there is a simpler solution: it suffices to examine a smaller set of *extreme* traffic matrices, denoted by $\mathcal{E}$, rather than the full set of all possible traffic matrices $\mathcal{T}$. To understand why, consider that when a WAN router is non-blocking, traffic rates, going in and out of the router, are only limited by trunk capacity (the aggregate rate of trunk's links). Since each link has the same

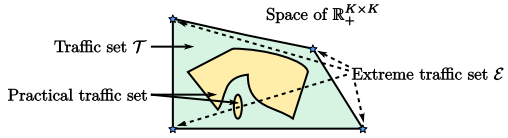

Figure 6: Traffic set and extreme traffic set.

capacity, we can represent each link as having unit capacity, so the number of links in each trunk constrains the space of all possible traffic matrices $\mathcal{T}$:

$$\mathcal{T} = \left\{ T \in \mathbb{R}_+^{K\times K} : \begin{array}{lll} \sum_{j\in\mathcal{K}} t_{ij} & \le M_i & \text{for all } i \in \mathcal{K} \\ \sum_{i\in\mathcal{K}} t_{ij} & \le M_j & \text{for all } j \in \mathcal{K} \\ t_{ii} & = 0 & \text{for all } i \in \mathcal{K} \end{array} \right\}, \quad (4)$$

where $\mathbb{R}_+$ is a set of non-negative real numbers. The set $\mathcal{T}$ bounds all feasible traffic matrices, as shown in Figure 6.

This set is a *convex polytope*, because constraints in Equation 4 are affine functions [6, 9]. For example, for a WAN router with a trunk set (2, 2, 4), 6 affine constraints define $\mathcal{T}$. One of them is $t_{12} + t_{32} \le 2$ and implies the total (normalized) rate to the second trunk is at most 2, even though the third trunk can send at most 4.

Using Extreme Traffic Matrices. The vertices of this convex polytope represent the extreme traffic matrix set $\mathcal{E}$ (Figure 6). In §A.1, we prove that it suffices to use $\mathcal{E}$ instead of $\mathcal{T}$ in our optimization formulation of Equation 3, as follows:

$$\min_{w\in\mathcal{W}} \max_{T\in\mathcal{E}} U(w, T). \quad (5)$$

We can transform this formulation into an MILP problem, and use an off-the-shelf MILP solver for reasonable problem sizes, *e.g.,* 128-port router with 4 trunks (§5). Larger WAN routers, or those with more trunks, need other scaling techniques, described next.

Symmetric Trunk Sets. For some trunk sets, we can obtain the minimal-upflow trunk wiring without using an MILP solver. Consider a WAN router with 128 ports and 16 L1 switches, and four trunks with (16, 32, 32, 48) links respectively. Now, suppose we wire each L1 switch with one link from the first trunk, 2 each from the second and third trunk, and 3 from the fourth trunk. It turns out that this trunk configuration achieves *zero* upflow across all traffic matrices. More generally, we say that a trunk set is *symmetric* if the number of links in each trunk is a multiple of the number of L1 switches. Specifically, a symmetric trunk set has $M_k = a_k\,|\mathcal{L}_1|$ when a_k is some positive integer for every $k \in \mathcal{K}$. In §A.2, we prove that the upflow for any symmetric trunk set is zero across any traffic matrix, so in these cases, computing upflow does not need Equation 5.

Approximating Minimal-Upflow. As the size of a WAN router increases, the number of L1 switches $|\mathcal{L}_1|$ increases. Also, a 128-port WAN router could serve more than 5 trunks. These two factors represent scaling challenges since the number of auxiliary constraints in Equation 5 increases as $O\left(|\mathcal{L}_1|\,2^{K^2}\right)$ (see §A.3 for a proof), which can cause solvers to exceed memory limits.

We have developed an approximation with better scaling behavior based on two ideas. The first is to approximate all the extreme traffic matrices by $\hat{T}$, a matrix whose entries are element-wise maxima across all the extreme traffic matrices. The second applies our observation that the minimal-upflow wiring usually tries to evenly distribute links of a trunk across all L1 switches: we only explore trunk configurations $\hat{\mathcal{W}}$ such that the number of links from a trunk assigned to two different L1 switches differ by no more than 1.

§A.4 lists this formulation in which the number of auxiliary constraints scales as $O(|\mathcal{L}_1|\,K^2)$. §5 shows that this formulation yields a wiring that match the optimal wiring obtained from the formulation in Equation 5 for most of the trunk sets we have been able to evaluate.

3 EFFECTIVE CAPACITY UNDER FAILURE

In this section, we discuss how to define, and compute, the effective capacity of a WAN router under failures.

3.1 Background

Minimizing upflow can make a WAN router resilient to failure, but it is important to quantify this resilience. To do so, we compute the *effective capacity* of the WAN router under (concurrent) failure of components. This effective capacity is an input to traffic engineering (TE) algorithms, such as those used by Google's TE Server [25], which use the effective capacity to route traffic demands based on application needs (*e.g.,* latency, traffic demand). It is also an input to our next step, computing compact forwarding tables (§4).

At run time, when a set of failures occurs in a WAN router, it may be possible to compute an estimate of the effective capacity, required for TE calculations. However, it is desirable in our setting to *pre-compute* this effective capacity for as many failure patterns as possible, because while computing the effective capacity of a given failure pattern is inexpensive (as we show below), the *next* step of our approach, computing compact forwarding tables (§4) requires an MILP formulation that can delay convergence of TE algorithms if run online. To ensure fast TE convergence [19], we pre-compute effective capacity *and* routing tables (in §4).

3.2 Goal and Challenges

Our goal in this section is to determine, for any given trunk configuration, the effective capacity when *multiple* internal links, or L1 or L2 switches fail *concurrently*. This problem is challenging because, if we want to pre-compute this effective capacity, we have to explore all possible concurrent failure scenarios, which increase exponentially. The second challenge is to define effective capacity precisely. A WAN router, by design, is non-blocking: it can support any possible traffic matrix. When failures occur, it may not be possible to support some traffic matrices.

We describe, for ease of exposition, separate algorithms for determining effective capacity (a) under L2 and internal link failures and (b) under L1 failures. This is because the characteristics of these failures are different: an L1 failure disables some ingress and egress trunk links, but a link failure or an L2 failure does not. We defer the discussion of L1 failures to §A.7. In §A.8, we describe a combined algorithm that estimates effective capacity under arbitrary combinations of internal link and L1/L2 switch failures.

At a high-level, to pre-compute the effective capacity under L2 and internal link failures, our approach enumerates all possible failure scenarios, and for each computes the effective capacity (and routing tables (§4)).

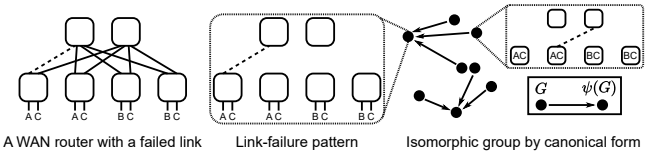

Figure 7: Link-failure model and canonicalization.

3.3 Modeling Link and L1 Switch Failures

When concurrent internal links fail, we model this as a graph G in which all the L1 and L2 switches are *nodes* and *only* the failed links are edges in the graph. Figure 7 shows an example with a single link failure. This graph represents a *failure pattern*. This approach can also model L2 switch failures; when an L2 switch fails, we mark all incident internal links in that switch to have failed. The total number of possible patterns is $2^{|\mathcal{L}_1|\times|\mathcal{L}_2|}$.

Graph Canonicalization and Isomorphism. To address this exponential complexity, we *leverage the symmetry* in WAN router topologies, and use *graph canonicalization* [29] which reduces a graph to an isomorphic *canonical* form. Two graphs are *isomorphic* when they are permutations of one another. For example, the middle and right failure patterns in Figure 7 are isomorphic because one can swap the rightmost L1 nodes (and L2 nodes) of either pattern to arrive at the other pattern. A *canonical form* of a graph G is a graph $\psi(G)$ that is isomorphic to G, such that every graph that is isomorphic to G has the same canonical form as G, and any two non-isomorphic graphs have different canonical forms. Canonicalizing failure patterns can result in fewer patterns to search and is crucial to scaling the pre-computation of effective capacity.

A Polynomial Time Algorithm. While it is not known whether polynomial time algorithms exist for general graph canonization, we have developed a polynomial time algorithm for Clos topologies. Our algorithm leverages the fact that Clos topologies are bipartite. Given a failure graph, the algorithm reorders links and nodes so that all links are on the left (to the extent possible). This transformation results in a canonical failure pattern. For example, in Figure 7, if there is a single failed link between the second L1 switch and the second L2 switch, by reordering the first and second L1 (respectively L2) switches, we can arrive at the canonical failure pattern where the failed link is between the first L1 switch and the first L2 switch (the pattern shown in Figure 7). From this example, it is tempting to assume that *all* single link failures are isomorphic, by symmetry. But this is *not* the case, because isomorphism *must also take into account the trunks to which external links belong*. For example, the failure pattern in Figure 7 is *not* isomorphic to one between the 3rd L1 switch and the 2nd L2 switch. Our canonicalization algorithm (§A.5), takes these dependencies into account.

With this algorithm, suppose we wish to determine the effective capacity for a failure pattern G: we first compute $\psi(G)$, then use the pre-computed effective capacity for $\psi(G)$. We now discuss how to compute effective capacity.

3.4 Effective Capacity

Given a trunk set, $\mathcal{T}$ is the set of all possible traffic matrices for that trunk set. By design, a WAN router (in the absence of failure) is non-blocking for all $T \in \mathcal{T}$. One way to define effective capacity is to enumerate the set of traffic matrices that the WAN router *can* support for each failure pattern. This is computationally challenging, and also complicates traffic engineering algorithms that must constrain their path computations to match these traffic matrices.

For this reason, we use a simpler definition of effective capacity. Consider the set of all traffic matrices $\theta\mathcal{T}$: every traffic matrix $T \in \mathcal{T}$ is scaled element-wise by a *scaling factor* $\theta \in [0, 1]$. We say that the effective capacity of the WAN router is the largest θ such that the router *is non-blocking for every matrix* in $\theta\mathcal{T}$. Defining effective capacity this way allows us to *keep the TE algorithm unchanged*; we can simply scale the capacity of each trunk incident on the WAN router by θ and run the TE algorithm to generate the paths.

Computing Effective Capacity. We obtain θ by solving an overloaded multicommodity flow problem [3, 8]. However, in formulating this, we need to be consistent with current practice in WANs, which splits traffic evenly across all links in a trunk (§2). Thus, the formulation must constrain the problem to ensure uniform splitting of traffic both on incoming traffic and on outgoing traffic. Equally important, we must find the effective capacity across all possible traffic matrices and all possible failure patterns. The input to the formulation includes a trunk set $\{M_k\}$, a trunk configuration $\{w_{sk}\}$, a traffic set $\mathcal{T}$, and a set of link-failure patterns $\mathcal{F}$.

The output is the effective capacity θ, defined using a min-max optimization objective:

$$\min_{F\in\mathcal{F}} \min_{T\in\mathcal{T}} \max_{\theta\in\Theta(F,T)} \theta \qquad \text{where} \qquad \Theta(F,T) =$$

$$\left\{\theta:\ \begin{array}{ll} \sum_{a\in\mathcal{L}_1} r_{ab}^{ij} = \sum_{a\in\mathcal{L}_1} r_{ba}^{ij} & ,\forall b\in\mathcal{L}_2,(i,j)\in\mathcal{K}^2 \\ \sum_{b\in\mathcal{L}_2} r_{ba}^{ij} + \theta t_{ij} w_{ai}/M_i = \sum_{b\in\mathcal{L}_2} r_{ab}^{ij} + \theta t_{ij} w_{aj}/M_j & \\ & ,\forall a\in\mathcal{L}_1,(i,j)\in\mathcal{K}^2 \\ \sum_{(i,j)\in\mathcal{K}^2} r_{ab}^{ij} \le \mathbb{I}[(a,b)\notin F] & \\ & ,\forall (a,b)\in\mathcal{L}_1\times\mathcal{L}_2\cup\mathcal{L}_2\times\mathcal{L}_1 \\ r_{ab}^{ij}, r_{ba}^{ij} \in \mathbb{R}_+ & ,\forall a\in\mathcal{L}_1, b\in\mathcal{L}_2,(i,j)\in\mathcal{K}^2 \\ \theta\in[0,1] & \end{array}\right\} \tag{6}$$

This formulation finds the smallest θ across every pair of traffic matrix and failure pattern. For each pair, it computes the largest θ satisfying the constraints in $\Theta(F,T)$. The first two constraints ensure flow conservation at L2 switches and L1 switches. The second one also imposes uniform splitting of ingress and egress traffic on each trunk. The third describes the link capacity under a given failure scenario, where $\mathbb{I}[\cdot]$ is an indicator function. The last two constrain the range of decision variables.

As in §2, this formulation is also intractable because the traffic matrix set $\mathcal{T}$ is infinite. Here too, we can leverage the fact $\mathcal{T}$ forms a convex polytope, and use the extreme traffic matrices in this polytope to compute effective capacity. We prove this in §A.6.

It follows then that we can find the effective capacity by considering every pair of (a) canonical failure pattern $F \in \mathcal{F}$ and (b) extreme traffic matrix $T \in \mathcal{E}$. For each such pair, we solve the *linear program* $\max_{\theta\in\Theta(F,T)} \theta$. Using the canonical failure patterns and the extreme traffic matrices reduces the complexity of the optimization significantly. Furthermore, we can parallelize the computation of effective capacity for each canonical pattern and each extreme traffic matrix, so this computation scales well. We defer the discussion of L1 switch failures and arbitrary combinations of failures to §A.7 and §A.8 respectively.

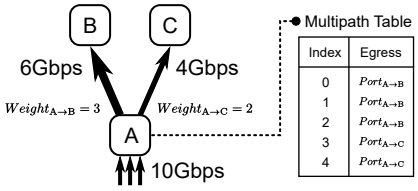

Figure 8: WCMP routing uses a multipath table. In today's switches, these tables have limited sizes.

4 COMPACT FORWARDING TABLES

In this section, we describe how we derive compact forwarding tables to minimize upflow in the presence of failures.

4.1 Background

As described in §1, WAN routers today use ECMP [20] to forward traffic: each L1 switch splits incoming traffic evenly across all links to L2 switches. However, our approach may require an uneven traffic split because, at an L1 switch, some fraction of incoming traffic may be subject to early forwarding, while the rest of the traffic needs to traverse L2 switches.

Prior work has described a weighted version of ECMP, called WCMP [40], which assigns weights in proportion to the desired traffic split ratio. Today's switches implement WCMP using a *multipath table* (Figure 8): they assign each split entries in this table in proportion to its weight. For example, if tunnel **A** should split traffic across tunnels **B** and **C** as 3 : 2, the multipath table will have 5 entries as shown. The switch hardware will evenly split the traffic across these 5 entries, achieving the desired 3 : 2 traffic split.

Unfortunately, modern switches have limited multipath table entries, and arbitrary weight ratios can exceed table capacity. For example, a weight ratio of two relatively prime numbers 233 : 767 requires 1000 entries.

4.2 Goal and Challenges

Motivated by this, we design *compact forwarding tables* by minimizing the number of entries needed for multipath tables for a given trunk configuration, a failure pattern, and effective capacity.

Our design must address two challenges. First, it must preserve early forwarding opportunities in order to minimize upflow. (One way to compact the forwarding table is to adjust weights at the cost of increased upflow, but this would negate the benefits of computing the minimal-upflow wiring in §2). Second, the resulting forwarding table must ensure that the WAN router *remains non-blocking across all traffic matrices*; computing and modifying WCMP weights in response to traffic matrix changes is infeasible both computationally and operationally since traffic matrices change quickly over time.

4.3 Compacting Forwarding Table

Input and Output. The input to our algorithm is a trunk wiring configuration $\{w\}$ (from §2), a failure pattern F and effective capacity (from §3). The output is a set of integer WCMP [40] weights that ensures non-blocking behavior under any traffic matrix for that failure pattern and uses the fewest multipath entries.

Decoupling Traffic Matrices from WCMP Weight Calculations. Conceptually, it seems difficult to compute WCMP weights that would ensure non-blocking behavior across all traffic matrices. However, given a trunk configuration $\{w\}$, the *proportion* of traffic sent from an L1 switch to an L2 switch, or vice versa, for a given pair of trunks, is independent of the traffic matrix. To understand why, consider the following two quantities defined for traffic from trunk i to trunk j:

$$u_s^{ij}(w,T) = t_{ij}\left[w_{si}/M_i - w_{sj}/M_j\right]_+ = t_{ij}\hat{u}_s^{ij}$$
$$d_s^{ij}(w,T) = t_{ij}\left[w_{sj}/M_j - w_{si}/M_i\right]_+ = t_{ij}\hat{d}_s^{ij}.$$

The first quantity is the *upflow volume* from switch s, for a given traffic matrix T and a given wiring $\{w\}$: *i.e.,* it measures the total volume of traffic at switch s sent up to L2 switches. The second quantity is the *downflow volume* at s: the total volume at s *received* from L2 switches forwarded on a trunk j at switch s.

Notice that both of these quantities have two components: a traffic matrix component t_{ij} and a (respectively) *upflow fraction* $\hat{u}_s^{ij}$ or a *downflow fraction* $\hat{d}_s^{ij}$. Our key insight is that WCMP weight calculations *can be designed independent of traffic matrix by basing the weight calculations on upflow and downflow fractions.* (As an aside, a TE algorithm does not compute a traffic matrix but routes tunnels (or tunnel groups [25]) on trunks. Thus, trunk i might carry traffic for multiple tunnels. For some tunnels, traffic will exit the WAN router using trunk j. We have abstracted this detail by describing the total volume of such traffic using the term t_{ij}).

Minimizing Multipath Table Entries. To minimize multipath table entries we observe that, at a given switch s, for traffic between trunks i and j, *if the upflow fraction is non-zero, there cannot be any downflow.* This is by design: if there is upflow, it means that all the links of trunk j at s are used for early forwarding, so there is no capacity left for traffic from other switches to exit on trunk j at s.

Flow Counts. Thus, at each switch we can define a quantity called the *flow count* v_s^{ij} as follows:

$$v_s^{ij} = \begin{cases} \hat{u}_s^{ij}/\alpha_{ij} & ,\hat{u}_s^{ij} > 0 \\ \hat{d}_s^{ij}/\alpha_{ij} & ,\hat{d}_s^{ij} > 0 \\ 0 & ,\text{otherwise} \end{cases} \qquad \forall s \in \mathcal{L}_1, (i,j) \in \mathcal{K}^2.$$

which is either the upflow fraction or the downflow fraction depending on the trunk wiring. For a reason described below, we scale these fractions by the fractional greatest common divisor (FGCD) α_{ij} of each traffic pair (i,j) across L1 switches, so that all v_s^{ij} values are integers, hence the name flow count.

An Example. Figure 9 illustrates this idea for a trunk pair (C,A) and two different values of traffic between these trunks t_{CA}. In the example on the left, the incoming traffic on the rightmost L1 switch is 1 unit, and the switch forwards 3/4ths to the L2 layers (early forwarding the rest), which is evenly distributed across the other three L1 switches. In the example on the right, the incoming traffic is half that. Despite this, the upflow/downflow fractions and the flow counts are the same in all cases.

Now, suppose at switch s the flow count for trunk pair (i,j) corresponds to an upflow. Consider two other switches s_1 and s_2 have (downward) flow counts. To compute the WCMP weights across the network, we solve an optimization that determines how to route these (upflow) flow counts from L1 switches (*e.g., s*) to L2 switches, and subsequently from L2 switches to the corresponding L1 switches (*e.g.,* s_1 and s_2) which have "available" downward flow counts.

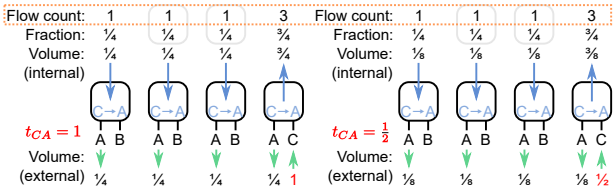

Figure 9: Fraction of different volumes is the same and is converted to flow count by FGCD.

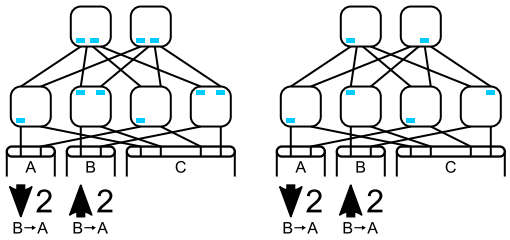

Figure 10: Compact routing does not assign unnecessary weights to multiple ports.

The key intuition for why our approach compacts routing tables rests on the observation that, when "matching" the upflow flow counts to the downflow flow counts, we can avoid splitting the traffic unless not doing so would reduce the effective capacity. Figure 10 illustrates this and shows which switch ports have associated WCMP weights for forwarding traffic from **B** to **A**. Without our approach (left), the second L1 switch splits its traffic (*i.e.,* divides its upflow) on both internal ports. This assignment uses 2 entries in the multipath table. Instead, the compact routing (right) realizes that this multiple-interface assignment is unnecessary and only uses the port connected to the first L2 switch, resulting in fewer WCMP entries.

The Optimization Objective. Thus, at a high-level, our optimization objective is to match the upflow and downflow counts, subject to the effective capacity constraint. To formalize this, let x_{ba}^{ij} represent the total flow count from *L2 switch b* to *L1 switch a* for trunk pair (i, j). Let's define X_b^{ij} as the sum of these values across all L1 switches a. We make two observations. First, that X_b^{ij} is *proportional to the number of multipath table entries* at L2 switch b for trunk pair (i, j). To see why, consider a trunk pair (i, j) whose X_b^{ij} value is 10. Now, assuming that switch b forwards these 10 flow counts to L1 switches, any split with integer weights of these X_b^{ij} flow counts can at most be 10. Second, we observe that L2 switches will need more WCMP entries in our approach than L1 switches for internal links (overall, L1 switches need more entries because they need to handle egress links as well, §5.4). An L1 switch may not see traffic for all trunk pairs (i, j), but an L2 switch may see traffic for all trunk pairs (i, j) with non-zero upflow (because, in general, an L1 switch must distribute the upflow across L2 switches to ensure the effective capacity constraint).

These observations motivate our optimization objective: to find flow count assignments x_{ba}^{ij} and x_{ab}^{ij} which minimize maximum total flow count at an L2 switch, across all possible traffic matrices, for a given failure pattern. We present the optimization formulation in §A.9. In §A.9, we also discuss an important detail: computing WCMP weights for egress links in L1 switches (our formulation focuses on weight assignments for internal links).

Scaling. To this formulation, we apply two scaling techniques discussed in previous sections. First, we replace every occurrence of the infinite traffic matrix set $\mathcal{T}$ with the finite extreme traffic matrix set $\mathcal{E}$, resulting in an MILP problem solvable by an off-the-shelf solver. Second, we run the optimization only for canonical failure patterns.

Improved Scaling with an Approximate Solution. Even so, our formulation does not scale well to 512-port routers. To address this, we use the same approximation technique as in §2: instead of iterating over all extreme traffic matrices, we evaluate for one matrix whose elements are the element-wise maximum across all extreme matrices in $\mathcal{E}$.

5 EVALUATION

In this section, we compare the resilience of our approach to other approaches both for 128 and 512-port switches, explore the efficacy of our routing table compaction, and quantify the benefits of our techniques to scale the computations[1].

5.1 Methodology

Goal. We compare our approach against a *baseline* wiring scheme that sequentially assigns external ports to each trunk one by one. For example, in a 128-port WAN router with 16 L1 switches each with 8 egress ports, a trunk set $(8, 32, 32, 56)$ would be assigned as follows: the first trunk connects to all the egress ports on the leftmost L1 switch, the next trunk connects to egress ports in the next 4 L1 switches, and so on. For this baseline wiring, we evaluate both ECMP (which splits traffic from L1 to L2 switches evenly) and WCMP (which weights the traffic split using the algorithm in §4). We also compare against a *random wiring* that randomly assigns external ports to trunks. For this strategy as well, we evaluate ECMP and WCMP based routing. Overall, we explore a space defined by two dimensions: a routing strategy dimension consisting of two alternatives (ECMP and WCMP), and a wiring dimension with three alternatives (our approach, baseline wiring, and random wiring).

Metrics and Methodology. To understand the efficacy of our approach, we use three metrics: upflow (§2), effective capacity (§3), and table size (§4). We also quantify the benefits of our optimizations: finding extreme traffic matrices, the upflow approximation, failure pattern canonicalization, and routing approximation. Our evaluations use two sizes of WAN routers. In a 128-port WAN router, each switch has 16 ports, and there are 16 L1 and 8 L2 switches. For this router, we evaluate all possible trunk sets with four trunks where the number of links in each trunk is divisible by 8. There are 34 such trunk sets, shown on the x-axis in Figure 11. In a 512-port WAN router, each switch has 32 ports, and there are 32 L1 switches and 16 L2 switches. For this router, we evaluate all possible trunk sets with 5 trunks (there are 480 such sets), where the number of links in each trunk is divisible by 16.

Implementation. Our experiments use the `cdd` [16] library to generate the extreme traffic set. We use Gurobi [28] to solve all LP and MILP problems, and Open MPI [33] to parallelize our computations.

[1] Our code is available at https://github.com/USC-NSL/Highly-Available-WAN-Router.

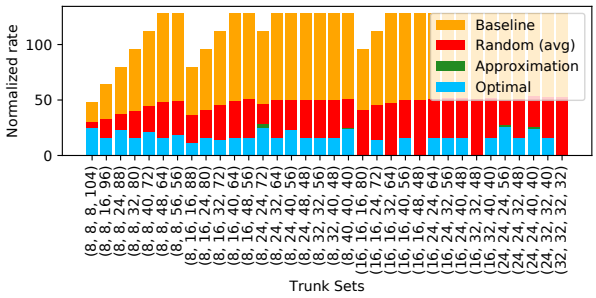

Figure 11: Upflow in a 128-port router with 4 trunks.

5.2 Resilience: 128-port WAN Router

Figure 12 (left) quantifies resilience by showing the effective capacity for our approach over different trunk sets, under link failure and L2-switch failure (§3) and under L1-switch failure (§A.7).

Upflow. To understand the results Figure 12, it is important to first understand the efficacy of minimal-upflow trunk wiring. Figure 11 shows the upflow across all the trunk sets in a 128-port router. We use Equation 5 to compute upflow for our approach, random wiring, and baseline wiring. Baseline wiring does not employ early forwarding, but random wiring does. Our optimal wiring approach leads to the lowest upflow rate in all scenarios because it spreads links from the same trunk across L1 switches and maximizes early routing opportunities to minimize upflow. Trunk sets where each trunk has a multiple-of-16 links have no upflow, a consequence of Theorem A.2. By comparison, baseline wiring has an upflow that is sometimes 10× higher. With baseline wiring, the upflow can vary with trunk set. Some trunk sets have more constrained traffic matrices than others: for example, in the trunk set $(8, 8, 8, 104)$ the largest trunk can send at most 24 (normalized) units of traffic even though the trunk capacity is 104. Finally, random wiring yields upflow (averaged over 100 random wirings for each trunk set) that is 2-3× worse than optimal.

Link Failures. The upper left plot of Figure 12 plots the resilience of trunks across link failures. The resilience varies by trunk set, but instead of plotting resilience across all trunk sets, we group them into 4 classes by their upflow: these classes demonstrate qualitatively different behaviors.

Our approach is able to *completely mask* up to *six link concurrent link failures across all trunk sets*, and the effective capacity only drops below one after the 7^{th} failure. This is because the maximum upflow across all trunk sets, in Figure 11, is well below 32. In a 128-port WAN router, there are 16 L1 switches and 8 internal links between a pair of L1 and L2 switches. Each switch has at most 2 units of upflow (since the total upflow is less than 32), which requires 2 links to carry the upflow (per L1 switch), so the router can tolerate up to 6 failures.

There are 4 classes with respect to link failures. Trunk sets with zero upflow always have an effective capacity 1 under any link failure. Any trunk set with upflow in $(0, 16]$ starts to degrade after 7 link failures. Similarly, any trunk set with upflow in $(16, 32]$ starts to degrade after 6 link failures. These trunk sets differ slightly in the drop in effective capacity resulting from the 7^{th} failure because of the way the trunk sets are configured. Finally, every trunk set with non-zero upflow has effective capacity 0 under 8 failures: no

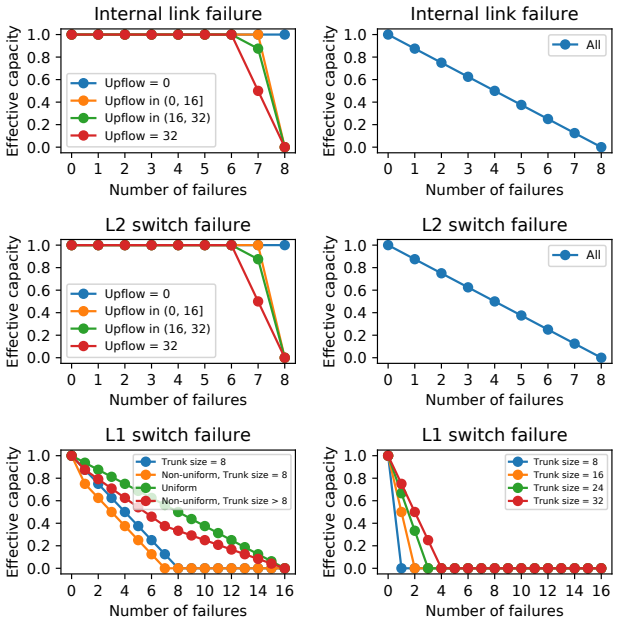

Figure 12: Minimal-upflow wiring (left) vs. Baseline wiring with WCMP (right).

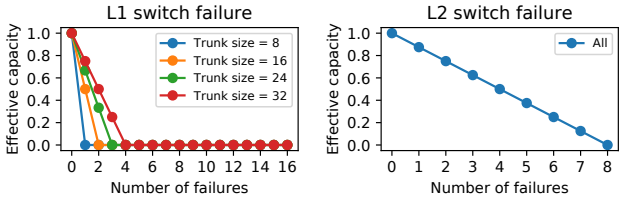

Figure 13: Baseline wiring with ECMP routing.

128-port WAN router can ensure non-blocking behavior under the worst-case failure pattern with 8 concurrent link failures (which occurs when all uplinks on an L1 switch fail).

By comparison, *baseline wiring with WCMP cannot mask a single failure* (top right figure in Figure 12). The effective capacity in this case is independent of the trunk set. The WAN router capacity degrades gracefully under failure: every link failure drops capacity by $1/8^{\text{th}}$. Baseline wiring with ECMP performs worse than baseline wiring with WCMP. Table 1 shows the resilience of baseline wiring with ECMP for up to four failures: effective capacity drops by 50% with a single failure, and by nearly $3/4^{\text{th}}$ with 4 failures. While it might be tempting to conclude that routers should implement WCMP to increase failure resilience, we note that an alternative strategy which treats each link failure as the failure of the corresponding L2 switch has the same resilience as baseline wiring with WCMP, so there is really no incentive to deploy WCMP for link failure resilience.

L2 Switch Failures. Resilience to L2-switch failures (center left of Figure 12) is *identical* to that for link failures. An L2 failure removes 1 link from each L1 switch, so our upflow-based categorization still applies. Here too, there are four classes categorized by the value of upflow, and baseline wiring with WCMP (middle right of Figure 12)

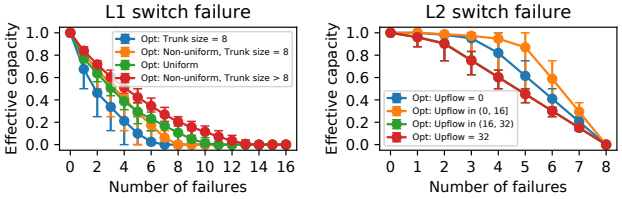

Figure 14: Effective capacity of random wiring. Trunk sets are selected from the categories of minimal-upflow wiring.

and ECMP (Figure 13) provide no masking and have identical behavior: with each L2 failure, capacity drops by $1/8^{\text{th}}$. Random wiring with WCMP (Figure 14), is more resilient than baseline wiring, but has lower effective capacity than minimal-upflow wiring across all failure configurations. For example, minimum upflow wiring ensures full effective capacity with 6 concurrent failures when total upflow is 32. However, random wiring, for the same setting only has an effective capacity of 0.3. The effective capacity of random wiring with link failure is identical to L2 switch failure, so we have omitted a description of the former.

L1 Switch Failures. No approach can mask L1 failures, since these reduce trunk capacity in addition to internal capacity. However, *our approach degrades much more gracefully than competing approaches*, but the behavior depends on the trunk set configuration. Our 34 trunk sets fall into four classes (lower left of Figure 12) with qualitatively different behavior. These sets depend on two factors: the size of the smallest trunk in the trunk set (either 8, or larger), and whether the minimal-upflow wiring (§2) wires the trunks uniformly across the L1 switches or not. Non-uniform wiring introduces a little asymmetry with a slightly different resilience.

When the minimum trunk size is 8, because our approach spreads the links of these trunks across 8 L1 switches, they can tolerate up to 8 L1 switch failures, with each failure degrading capacity by $1/8^{\text{th}}$, as shown by the line "Trunk size = 8". With non-uniform wiring with a minimum trunk size of 8, our approach can tolerate up to 7 failures resulting from non-uniform spreading of the wires (only 7 L1 switches connect to the 8-wire trunk). For a similar reason, trunk sets with a minimum of 16 links in each trunk can tolerate up to 15 L1 switch failures, with capacity drops of around $1/16^{\text{th}}$ at each step (there are slight variations resulting from non-uniformity described earlier).

By contrast, baseline wiring with ECMP (Figure 13) or WCMP (bottom right of Figure 12) *can only tolerate up to 4 L1 switch failures*, and, for some trunk configurations, *may have zero capacity even with a single L1 switch failure*. Finally, random wiring with WCMP (Figure 14), performs generally worse than minimal-upflow wiring: the latter generally degrades gracefully, but the capacity degradation in the former is more dramatic (*e.g.,* for the uniform wiring case).

Simultaneous L1 and L2 Switch Failures. Our approach can handle simultaneous failures of links, L1 switches, and L2 switches (§A.8). To demonstrate this, Figure 15 shows effective capacity resulting from simultaneous failure of L1 and L2 switches for a specific trunk set in the L2-category "Upflow in (16, 32)" and L1-category "Non-uniform, Trunk size > 8". The effective capacity is a combination of the results from those categories: for instance, the

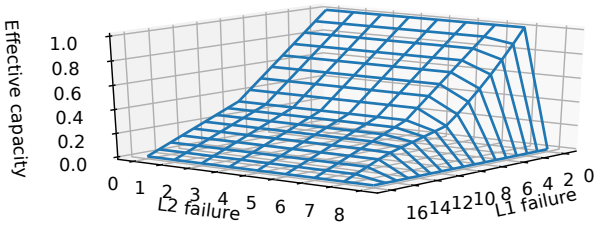

Figure 15: Effective capacity of trunk set (16, 24, 24, 64) under simultaneous L1 and L2 switch failures.

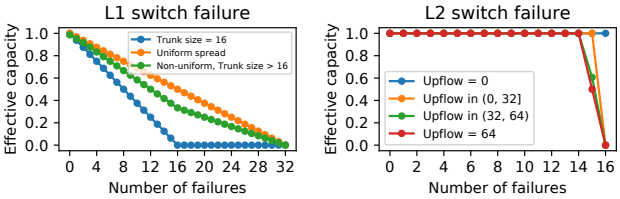

Figure 16: Effective capacity under L1 and L2 failures for a 512-port router.

dip in effective capacity for the 7th L2 switch failure in Figure 12 is also visible in this plot.

5.3 Resilience: A 512-port WAN Router

Figure 16 shows the effective capacity of 512-port router trunk sets under L1 and L2 switch failures. For this router, there are 480 trunk sets. In computing the effective capacity, we use the approximation formulation (§2) to compute the minimal-upflow wiring. That we are able to obtain these results demonstrates the scalability of our approximation: in §5.5, we quantify the optimality gap introduced by the approximation. We have also computed effective capacity for random wiring for a 512-port router. These results are qualitatively similar to those for the 128-port router, so we omit them for brevity.

L1 Switch Failures. Our trunk sets exhibit three qualitatively different classes of behavior. These classes depend on two factors: the minimum trunk size in a trunk set, and whether the optimal wiring spreads a trunk's links uniformly across L1 switches or not. When trunks are uniform, degradation is graceful, but, for obvious reasons, when the minimum trunk set size is 16, the trunk set can only tolerate up to 16 failures. With non-uniform wiring, the degradation is steeper for the first few failures as a result of asymmetry.

L2 Switch Failures. As with the 128-port case, we observe four qualitatively different types of behavior with L2 switch failures. Every trunk set with zero upflow can mask all L2 switch failures. Every trunk set having upflow in $(0, 32]$ requires one internal link per L1 switch to carry traffic, so can sustain 15 L2 switch failures. Every trunk set with 64 units of upflow requires 2 L2 switches to carry traffic, so the effective capacity becomes 0.5 when 15 L2 switches fail. For trunk sets with upflow in $(32, 64)$, the effective capacity is slightly better than the 64-unit upflow case due to slightly lower upflow.

Link Failures. As with the 128-port router, we find that the resilience under link failure is similar to that under L2 switch failures, so we omit this graph for brevity.

Router	Baseline ECMP No failure	Random ECMP No failure	Optimal ECMP No failure	Optimal WCMP (No, L1, L2)
128 ports	48	186	192	(64, 388, 64)
512 ports	128	640	640	(346, 3310, 346)

Table 2: Maximum number of multipath entries at any switch, across all trunk sets.

5.4 Compact Routing Tables

We now show that our routing table compaction technique (§4) results in tables that *do not exceed hardware routing table limits*. To do this, we compute, for the two sizes of routers, the largest table size at any switch, across all trunk sets, for any combination of L1 failures, and (separately) for any combination of L2 failures that are completely masked. For the 512-port router, we compute the tables using the scaling approximation (§4).

Table 2 shows the table sizes. For calibration, the hardware limit on the multipath table in modern switches is 65K [32]. Our approach (last column) uses at most 388 and 64 entries for a 128-port router and at most 3310 and 346 entries for a 512-port router under L1 and L2 failures. We also see that the routing table sizes are relatively insensitive to L2 failures because the optimization assigns WCMP weights sparsely to minimize table sizes: every L1 switch sends traffic over a few links to L2 switches, and other links that do not carry traffic are not assigned WCMP weights. So, when a link with non-zero weight fails, our algorithm moves, to another active link, the weights assigned to that link without increasing table sizes.

ECMP with baseline wiring in the absence of failures (first column) uses fewer entries, because links of the same trunk connect to the same L1 switch, which permits grouping of entries (an L2 switch only needs 1 entry per cross-trunk pair and per L1 switch). The grouping is less effective for arbitrary wiring (two middle columns). Random wiring requires 186 and 640 entries, while the minimal-upflow wiring requires 192 and 640 entries for 128-port and 512-port routers. Our routing optimization compacts the entries (in the last two columns) from 192 to 64 for a 128-port router and from 640 to 346 for a 512-port router in the absence of failures by assigning weights sparingly.

5.5 Impact of Optimizations

The Importance of Routing Optimizations. Minimal-upflow wiring, together with early forwarding, alone does not provide high resilience; our WCMP routing is also necessary. To demonstrate this, we conduct an experiment on a 128-port router with no failures. For minimal-upflow wiring, we configure routing tables to use early forwarding when possible, but use ECMP routing to split traffic equally to L2 switches for upflow traffic and egress ports for early forwarding traffic. Figure 17 shows the resulting effective capacity across all trunk sets. Our minimal-upflow wiring together with compact routing tables achieves an effective capacity of 1 *across all trunk sets* (not shown in the figure). However, the simpler routing technique that uses ECMP is able to achieve full capacity for only 5 of the trunk sets, whose upflow is zero, with minimal upflow wiring, and much lower effective capacity with random wiring.

Canonicalization. Canonicalizing failure patterns often reduces the number of link failures by orders of magnitude. Figure 18 shows the average numbers of canonical forms across our 34 trunk sets

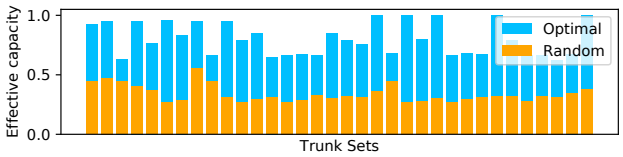

Figure 17: Effective capacity constrained by ECMP routing under no failure.

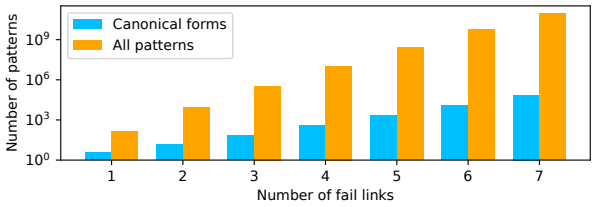

Figure 18: The benefits of canonicalization.

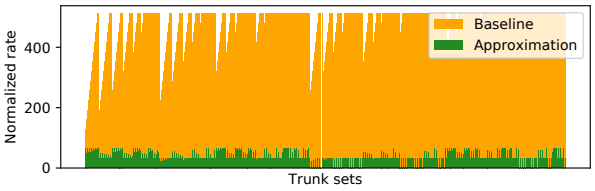

Figure 19: Upflow in a 512-port router with 5 trunks.

Trunk Set	Optimal	Approximation
(8, 8, 56, 56)	9 min.	0.21 sec.
(24, 24, 32, 48)	38 min.	0.16 sec.
(96, 96, 96, 112, 112)	–	0.88 sec.
(96, 96, 96, 96, 128)	–	0.63 sec.

Table 3: Micro benchmark of wiring approaches.

under different numbers of link failures (y-axis is in log scale). With 7 failures, the reduction is a 5 order of magnitude, from 10^{11} to 10^6.

Upflow Approximation. For the 128-port router, our upflow approximation (§2) matches the optimal upflow computed using the formulation of Equation 5 for all but 4 of the trunk sets (Figure 11). For those 4 cases, the differences are extremely small. To demonstrate that our approximation helps compute upflow with larger WAN routers and trunk sets, Figure 19 shows the upflow across *all 5-trunk trunk sets* (480 such combinations) for a 512-port WAN router. The resulting upflow is 20-30× lower than the baseline wiring.

Finally, our approximation noticeably speeds up upflow computation (Table 3). For The 4-trunk cases, the formulation of Equation 5 can take up to 38 min to find the minimal upflow using a multi-core desktop (20 cores 2 Intel Xeon CPU E5-2650 @ 2.30GHz), while our approximation can compute this in a fraction of second. It can also compute upflow for some 5 trunk configurations when the formulation of Equation 5 does not even complete.

Large providers may have resources to compute minimal-upflow wiring using Equation 5. When using a cluster of say 500 multicore machines, the computation for 4-trunk trunk sets can complete in 10s of minutes because the number of extreme traffic matrices range from 200 to about a 1000 (Figure 20).

Computing Effective Capacity. Computing effective capacity of a link failure pattern takes 30 seconds on a single core of a multi-core machine. This means that the total time largely depends on the

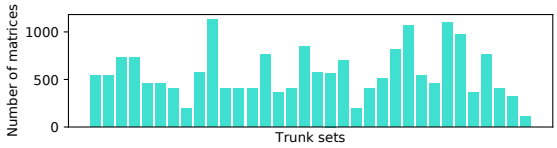

Figure 20: Extreme traffic matrices of a 128-port router.

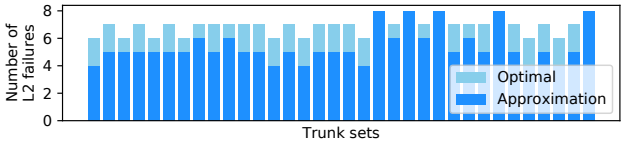

Figure 21: Performance gap of the approximate routing.

number of failure patterns. We can estimate this time from Figure 18. For example, finding effective capacities of all combinations of 7 link failures would take about 5.8 hours on a single 24 core machine in the presence of canonicalization. Without the optimization, it would take about 66 *years*.

Computing Routing Tables. Calculating a routing table is also well within the compute power available to cloud and content providers. It takes at most 2 minutes for a given trunk wiring and a 2-link failure pattern for a 128-port WAN router, across all possible trunk sets and 2-link failure combinations.

Routing Approximation. For the 512-port router, we were unable to find optimal routing tables using our compute cluster. However, our approximation formulation (§4) completed in a few minutes for this size of router. Figure 21 shows the optimality gap for our approximation for a 128-port router. It reports, for each trunk set, the maximum number of L2 switch failures which preserve full capacity. We observe that the approximation underestimates this quantity by at most 2, relative to the optimal.

6 RELATED WORK

Prior work has considered fault tolerance in multi-stage switching networks [2] (and references therein). This line of work considers interconnection networks where, unlike our setting, (a) packets traverse the network in one direction from the first stage and exit at the last stage so early forwarding opportunities do not exist, and (b) do not incorporate trunks. Since early forwarding is not possible, designers over-provision the networks [1, 11, 14, 15, 26, 30, 35, 36], by replicating stages, links, or the entire network. Our work achieves fault tolerance without over-provisioning.

Our work might apply to FatTrees [4] and F10 [27]. The latter focuses on limiting the blast radius of failures in datacenters by carefully striping a Clos; it is complementary to our work that seeks to improve failure resilience by adapting trunk wiring and routing to provide non-blocking behavior in the presence of failures. We do not know of WAN routers that incorporate other topology designs[2] proposed for datacenters, such as FatClique [39], random graphs: Jellyfish [37], Xpander [38], and server-centric designs: BCube [17], DCell [18], so our work focuses on Clos-based WAN routers.

Our work draws inspiration from Google's original B4 network [25] and more recent incarnation [19]. The B4 network uses various complementary techniques to improve availability including side-links between WAN routers at a site to increase resilience. Our work can be directly applied to these WAN routers to further improve overall resilience.

Prior work [40] formulated a non-linear integer optimization to minimize over-subscription in asymmetric topologies while fitting WCMP entries within table limit constraints. Our work considers a different problem: finding WCMP weights for non-blocking behavior of a WAN router.

Finally, our formulations and proof techniques draw inspiration from ideas from robust validation [10], robust optimization [5, 7], linear programming [8], and convex analysis [6, 9]. Robust validation [10] approximates solutions of max-min problems in robust optimization [5, 7]. Instead, our work finds the exact solutions by leveraging the convex polytope property of traffic matrices.

7 DISCUSSION

External Link Failures. External links can fail in practice. Our work extends easily to cope with such failure in two different situations. A total trunk failure, in which every link of a trunk fails, neither decreases effective capacity nor changes internal routing. Therefore, our approach applies directly. However, a partial trunk failure, where some links in a trunk fail, requires recalculation of effective capacity and routing (§A.10), which can be pre-computed.

Non-Uniform Internal Path Length. Because some incoming traffic on a trunk can be early forwarded, flows within a trunk may experience slightly different latencies. However, packets within a flow do not experience re-ordering because WCMP hashes all packets in a flow to the same path.

Cell-Based Routing. Some multi-chip routers, such as Stardust [41] designed from Broadcom Jericho2 [22] and Ramon [23] chips, use cell-based routing. In this approach, the router's ingress ports divide packets into fixed size cells and spray them uniformly across the fabric, re-assembling the packet at the egress ports. For such routers, our optimal wiring can increase effective capacity (*e.g.,* over random wiring in Figure 17) but, because it is yet unclear how to do weighted forwarding in these fabrics, it remains an open question how to compute WCMP-like forwarding tables for them.

8 CONCLUSION

This paper discusses an approach to optimizing trunk wiring and forwarding weights to increase the resilience of WAN routers in large content- and cloud-provider networks. Based on the observation that early forwarding in L2 switches can create excess internal capacity in the WAN router, enabling it to be more resilient to internal failures, we formulate an efficient optimization to derive the minimal-upflow trunk wiring. Then, given this wiring and an arbitrary failure pattern, we devise an efficient optimization to compute the effective capacity under failure, and finally describe a technique to compute compact forwarding tables that can ensure non-blocking behavior subject to this effective capacity. Our evaluations show that our approach can greatly increase the resilience of WAN routers without sacrificing a precious resource in today's switches, routing tables.

Acknowledgements. We thank Nathan Bronson, the SIGCOMM reviewers, and Subhasree Mandal for their comments and feedback that improved the paper greatly.

[2] In general, the topology of a WAN router can be arbitrary, but it must have non-blocking behavior under some routing scheme.

REFERENCES

[1] Adams and Siegel. 1982. The Extra Stage Cube: A Fault-Tolerant Interconnection Network for Supersystems. *IEEE Trans. Comput.* C-31, 5 (May 1982), 443–454. https://doi.org/10.1109/TC.1982.1676021

[2] George B. Adams, III, Dharma P. Agrawal, and Howard Jay Siegel. 1994. Interconnection Networks for High-performance Parallel Computers. IEEE Computer Society Press, Los Alamitos, CA, USA, Chapter A Survey and Comparison of Fault-tolerant Multistage Interconnection Networks, 654–667. http://dl.acm.org/citation.cfm?id=201173.201276

[3] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.* 38, 4 (Aug. 2008), 63–74. https://doi.org/10.1145/1402946.1402967

[5] A. Ben-Tal, L. El Ghaoui, and A.S. Nemirovski. 2009. *Robust Optimization.* Princeton University Press.

[6] D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar. 2003. *Convex Analysis and Optimization.* Athena Scientific.

[7] D. Bertsimas, D. Brown, and C. Caramanis. 2011. Theory and Applications of Robust Optimization. *SIAM Rev.* 53, 3 (2011), 464–501. https://doi.org/10.1137/080734510 arXiv:https://doi.org/10.1137/080734510

[8] Dimitris Bertsimas and John Tsitsiklis. 1997. *Introduction to Linear Optimization* (1st ed.). Athena Scientific.

[9] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization.* Cambridge University Press, New York, NY, USA.

[10] Yiyang Chang, Sanjay Rao, and Mohit Tawarmalani. 2017. Robust Validation of Network Designs Under Uncertain Demands and Failures. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17).* USENIX Association, Berkeley, CA, USA, 347–362. http://dl.acm.org/citation.cfm?id=3154630.3154658

[11] L. Ciminiera and A. Serra. 1986. A Connecting Network with Fault Tolerance Capabilities. *IEEE Trans. Comput.* C-35, 6 (Jun. 1986), 578–580. https://doi.org/10.1109/TC.1986.5009436

[12] C. Clos. 1953. A study of non-blocking switching networks. *The Bell System Technical Journal* 32, 2 (Mar. 1953).

[13] Shagnik Das. [n. d.]. A brief note on estimates of binomial coefficients. http://page.mi.fu-berlin.de/shagnik/notes/binomials.pdf

[14] Daniel M. Dias and J. Robert Jump. 1982. Augmented and pruned n log n multistaged networks: topology and performance. In *International Conference on Parallel Processing, ICPP'82, August 24-27, 1982, Bellaire, Michigan, USA.* 10–12.

[15] C. C. Fan and J. Bruck. 2000. Tolerating multiple faults in multistage interconnection networks with minimal extra stages. *IEEE Trans. Comput.* 49, 9 (Sep. 2000), 998–1004. https://doi.org/10.1109/12.869334

[16] Komei Fukuda. [n. d.]. cdd and cddplus Homepage. https://www.inf.ethz.ch/personal/fukudak/cdd_home/

[17] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. *SIGCOMM Comput. Commun. Rev.* 39, 4 (Aug. 2009), 63–74. https://doi.org/10.1145/1594977.1592577

[18] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. 2008. Dcell: A Scalable and Fault-tolerant Network Structure for Data Centers. *SIGCOMM Comput. Commun. Rev.* 38, 4 (Aug. 2008), 75–86. https://doi.org/10.1145/1402946.1402968

[19] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. 2018. B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and e in Google's Software-defined WAN. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'18).* ACM, New York, NY, USA, 74–87. https://doi.org/10.1145/3230543.3230545

[20] C. Hopps. 2000. Analysis of an Equal-Cost Multi-Path Algorithm.

[21] Arista Networks Inc. [n. d.]. Arista 7050X3 Series Switch Architecture. https://www.arista.com/assets/data/pdf/Whitepapers/7050X3_Architecture_WP.pdf. Accessed: 2019-1-30.

[22] Broadcom Inc. [n. d.]. BCM88690: 10 Tb/s StrataDNX Jericho2 Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/stratadnx/bcm88690. Accessed: 2019-6-13.

[23] Broadcom Inc. [n. d.]. BCM88790 Scalable Fabric Element 9.6 Tbps Self-Routing Switching Element. https://www.broadcom.com/products/ethernet-connectivity/switching/stratadnx/bcm88790. Accessed: 2019-6-13.

[24] Broadcom Inc. [n. d.]. High-Capacity StrataXGS Trident 3 Ethernet Switch Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56870-series/. Accessed: 2019-1-30.

[25] 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. 2013. B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 3–14. https://doi.org/10.1145/2534169.2486019

[26] Menkae Jeng and Howard Jay Siegel. 1986. A Fault-Tolerant Multistage Interconnection Network for Multiprocessor Systems Using Dynamic Redundancy.. In *ICDCS.* 70–77.

[27] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-tolerant Engineered Network. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI'13).* USENIX Association, Berkeley, CA, USA, 399–412. http://dl.acm.org/citation.cfm?id=2482626.2482665

[28] Gurobi Optimization LLC. [n. d.]. The Fastest Mathematical Programming Solver. http://www.gurobi.com/

[29] Brendan D. McKay. 1981. Practical Graph Isomorphism.

[30] Robert J. McMillen and Howard Jay Siegel. 1982. Performance and Fault Tolerance Improvements in the Inverse Augmented Data Manipulator Network. *SIGARCH Comput. Archit. News* 10, 3 (Apr. 1982), 63–72. http://dl.acm.org/citation.cfm?id=1067649.801714

[31] P. McMullen. 1970. The maximum numbers of faces of a convex polytope. *Mathematika* 17, 2 (1970), 179–184. https://doi.org/10.1112/S0025579300002850

[32] Cumulus Networks. [n. d.]. Equal Cost Multipath Load Sharing - Hardware ECMP. https://docs.cumulusnetworks.com/display/DOCS/Equal+Cost+Multipath+Load+Sharing+-+Hardware+ECMP

[33] The Open MPI Project. [n. d.]. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/

[34] Barath Raghavan, Subhasree Mandal, Mohammad Alfares, John McCullough, Fei Ye, Min Zhu, and Aravind Ravisankar. 2016. High performance and resilience in wide area networking. https://patents.google.com/patent/US9369408B1/en US Patent 9369408B1.

[35] S.M. Reddy and V.P. Kumar. 1984. On Fault-Tolerant Multistage Interconnection Networks. In *International Conference of Parallel Processing.* 155–164.

[36] C S. Raghavendra and A Varma. 1984. INDRA: A Class of Interconnection Networks with Redundant Paths. *Proceedings of Real-Time Systems Symposium*, 153–164.

[37] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: Networking Data Centers Randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12).* USENIX Association, Berkeley, CA, USA, 17–17. http://dl.acm.org/citation.cfm?id=2228298.2228322

[38] Asaf Valadarsky, Michael Dinitz, and Michael Schapira. 2015. Xpander: Unveiling the Secrets of High-Performance Datacenters. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets-XIV).* ACM, New York, NY, USA, Article 16, 7 pages. https://doi.org/10.1145/2834050.2834059

[39] Mingyang Zhang, Radhika Niranjan Mysore, Sucha Supittayapornpong, and Ramesh Govindan. 2019. Understanding Lifecycle Management Complexity of Datacenter Topologies. In *Proceedings of the 19th USENIX Conference on Networked Systems Design and Implementation (NSDI'19).* USENIX Association, Berkeley, CA, USA.

[40] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14).* ACM, New York, NY, USA, Article 5, 14 pages. https://doi.org/10.1145/2592798.2592803

[41] Noa Zilberman, Gabi Bracha, and Golan Schzukin. 2019. Stardust: Divide and Conquer in the Data Center Network. In *Proceedings of the 19th USENIX Conference on Networked Systems Design and Implementation (NSDI'19).* USENIX Association, Berkeley, CA, USA.

A APPENDICES

Appendices are supporting material that has not been peer reviewed.

A.1 Using Extreme Matrices for Minimal-Upflow Wiring

Theorem A.1. *The following equation holds:*

$$\min_{w\in\mathcal{W}} \max_{T\in\mathcal{T}} U(w,T) = \min_{w\in\mathcal{W}} \max_{T\in\mathcal{E}} U(w,T).$$

Proof. Given a fixed wiring w, we first show that $\max_{T\in\mathcal{T}} U(w,T) = \max_{T\in\mathcal{E}} U(w,T)$. From the definitions of upflow in Equation 1 and Equation 2 with a constant w, the optimization $\max_{T\in\mathcal{T}} U(w,T)$ is a linear program with a compact (closed and bounded) feasible set, and optimal solutions

exist at the boundary, including extreme points, of the traffic set $\mathcal{T}$. Therefore, some of these optimal solutions are extreme points in the extreme traffic set $\mathcal{E}$, a well-known result in Linear programming [6, 8]. This is equivalent to solving $\max_{T\in\mathcal{E}} U(w,T)$, which directly gives an optimal extreme point. Finally, since $\max_{T\in\mathcal{T}} U(w,T) = \max_{T\in\mathcal{E}} U(w,T)$ for any given w, it follows that $\min_{w\in\mathcal{W}} \max_{T\in\mathcal{T}} U(w,T) = \min_{w\in\mathcal{W}} \max_{T\in\mathcal{E}} U(w,T)$, which proves the theorem. □

A.2 Symmetric Trunk Sets

Theorem A.2. *For a symmetric trunk set with $\{a_k\}_{k\in\mathcal{K}}$ and a_k is some positive integer, the trunk wiring that uniformly distributes links of each trunk over L1 switches, such that $w^*_{sk} = a_k$ for every $s \in \mathcal{L}_1$ and $k \in \mathcal{K}$, optimally solves problem in Equation 3. Further, the total upflow is always* 0.

Proof. Given a symmetric scenario with $\{a_k\}_{k\in\mathcal{K}}$, the upflow rate in Equation 1 under any traffic matrix T is

$$u^{ij}_s(w^*,T) = \left[\frac{a_i t_{ij}}{a_i\,|\mathcal{L}_1|} - \frac{a_j t_{ij}}{a_j\,|\mathcal{L}_1|}\right]_+ = 0.$$

Therefore, the total upflow is 0, the maximum total upflow is also 0, *i.e.,* $\max_{T\in\mathcal{T}} U(w^*,T) = 0$, and the wiring is optimal, since every total upflow is at least 0. □

A.3 Number of Auxiliary Constraints

Lemma A.3. *The number of auxiliary constraints for formulation in Equation 5 increases like $O\left(|\mathcal{L}_1|\,2^{K^2}\right)$.*

Proof. Every upflow rate in Equation 1 requires an auxiliary constraint for the positive projection. From the total upflow in Equation 2, every extreme traffic matrix requires $O(K^2\,|\mathcal{L}_1|)$ auxiliary constraints.

The number of extreme traffic matrices can increase exponentially in the square of trunks as $|\mathcal{E}| = O\left(2^{K^2}\right)$. This bound is a consequence of [31] that the number of extreme points is upper bounded by $O\left(\binom{p-\lfloor d/2\rfloor-1}{\lfloor d/2\rfloor}\right)$, where $d = K^2 - K$ is the dimensions of the traffic matrix, and $p = d + 2K$ is the number of constraints from positivity and Equation 4. It follows that

$$|\mathcal{E}| \le O\left(\binom{K^2/2 + 3K/2}{K^2/2 - K/2}\right) \le O\left(2^{K^2}\right).$$

Note that the last inequality uses an approximation of binomial coefficients in [13]. Therefore, the number of auxiliary constraints increases like $O\left(K^2\,|\mathcal{L}_1|\,2^{K^2}\right) = O\left(|\mathcal{L}_1|\,2^{K^2}\right)$. □

A.4 Approximating Minimal-Upflow Wiring

$$\min_{w\in\hat{\mathcal{W}}} U(w,\hat{T}) \qquad \text{where}$$

$$\hat{T} = [\hat{t}_{ij}], \quad \hat{t}_{ij} = \max_{T\in\mathcal{E}} t_{ij} \quad ,\forall (i,j) \in \mathcal{K}^2$$

$$\hat{\mathcal{W}} = \mathcal{W} \cap \left\{ w_{sk} \in \left\{ \left\lfloor \frac{M_k}{|\mathcal{L}_1|} \right\rfloor, \left\lceil \frac{M_k}{|\mathcal{L}_1|} \right\rceil \right\}, \forall s \in \mathcal{L}_1, k \in \mathcal{K} \right\}$$

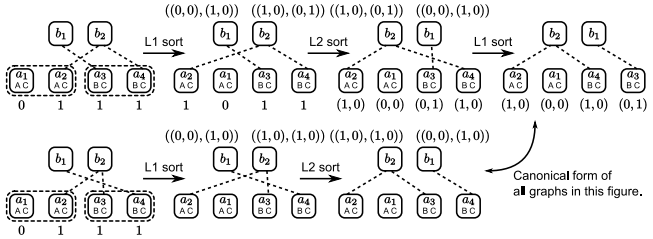

Figure 22: Steps in the canonical-form resolution.

A.5 Polynomial Time Canonicalization

Algorithm 1 outputs a canonical form of a given failure pattern. Figure 22 shows an example of the steps performed by the algorithm: recall that a link in this graph corresponds to a failed link in a WAN router. Intuitively, the algorithm (Algorithm 1) leverages the fact that the topology is bi-partite and re-orders nodes and links in a deterministic fashion to arrive at a canonical form.

In the first step (Lines 1-3), the algorithm re-orders L1 switches. In Line 1, it groups L1 switches by similarity of trunk link distribution, *e.g.,* switches a_1 and a_2 belong to a group because they have links to the same two trunks (**A** and **C**) and switches a_3 and a_4 belong to the (**B**, **C**) group in Figure 22. In creating groups, only the number of links to each trunk matters. So, if one 4-port L1 switch has links in this order (**A**, **A**, **B**, **C**) and another in this order (**A**, **B**, **C**, **A**), they belong to the same group, but another switch with links (**A**, **B**, **B**, **C**) does not belong to that group. In Lines 2 and 3, the algorithm sorts nodes within a group in descending order according to the number of failed links (or *cardinality*) associated with the node. The sort moves nodes with more failed links to the left within each group.

In the next step (Lines 4 and 5), the algorithm attempts to re-order L2 switches in a canonical order. To do this, it defines a *label* for each L2 switch. This label captures the link wiring from that switch, while preserving group structure. The label is an ordered list of tuples, where each tuple represents a group and enumerates the cardinality of each L1 switch in the group in descending order. For example, consider the L2 switch b_3 in Figure 23. Its label is $((0,0,0,0),(2,1,0,0))$ because it has no links to the first group, but has links to a_5 and a_6. The first two elements in the second tuple are 2 and 1, which are the cardinality of a_5 and a_6 respectively. Line 4 assigns these labels, and Line 5 re-orders L2 switches lexicographically in descending order of labels. This is shown in the 3rd step in Figure 22, which moves b_2 to the left.

The third step of the algorithm (Lines 6-10) attempts to re-arrange L1 switches within the same group and with the same cardinality by the rank of the L2 switch they are connected to. To achieve this, we "back propagate" the tuples from the L2 switches to the corresponding groups (Line 9), then, among all switches with the same cardinality (Line 7), we re-order them in descending lexicographic order (Line 10). This results in the fourth graph in the first row of Figure 22. This is the canonical form of the original failure graph.

The second row of Figure 22 shows another failure pattern that reduces to the *same* canonical form (this pattern does not require the third step).

Algorithm 1: Canonical-form resolution

Input : Graph F
Output: Canonical form $\psi(F)$

Group L1 nodes by their trunk wiring.
for *each group of L1 nodes* **do**
  Sort nodes in descending order by their cardinality
Label each L2 node by a tuple of L1 tuples, where each L1 tuple is an L1 group of sorted (in descending order) L1-node cardinalities associated with the L2 node.
Sort L2 nodes in descending order by their label.
for *each group of L1 nodes* **do**
  Group L1 nodes by their cardinality to subgroups.
  for *each subgroup of L1 nodes* **do**
    Label each node by a tuple of L2 node indices.
    Sort nodes in descending order by their label.

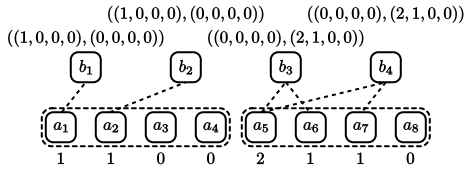

Figure 23: L2 label and L1 node's cardinality. Each L1 tuple in an L2 tuple is the sorted cardinalities of L1 nodes that the L2 node connects to. Unconnected L1 node corresponds to 0 in an L1 tuple.

A.6 Using Extreme Traffic Matrices for Effective Capacity

Theorem A.4. *The following equality holds:*

$$\min_{F\in\mathcal{F}}\min_{T\in\mathcal{T}}\max_{\theta\in\Theta(F,T)}\theta=\min_{F\in\mathcal{F}}\min_{T\in\mathcal{E}}\max_{\theta\in\Theta(F,T)}\theta.$$

Proof. Given a fixed F, we first prove that

$$\min_{T\in\mathcal{T}}\max_{\theta\in\Theta(F,T)}\theta=\min_{T\in\mathcal{E}}\max_{\theta\in\Theta(F,T)}\theta. \tag{7}$$

Let $\theta^*=\max_{\theta\in\Theta(F,T^*)}\theta$ be an optimal solution of the left hand side attained at traffic matrix $T^*\in\mathcal{T}$. We will show by contradiction that at least one extreme traffic $T\in\mathcal{E}$ leads to this θ^*. Specifically, $\theta^*=\min_{T\in\mathcal{E}}\max_{\theta\in\Theta(F,T)}\theta$. Suppose there is no such extreme point. Let $\hat{\theta}>\theta^*$ and $\hat{\theta}=\min_{T\in\mathcal{E}}\max_{\theta\in\Theta(F,T)}\theta$ be the minimum achieved by the extreme traffic set $\mathcal{E}$. Caratheodory's theorem [6] implies there exists $|\mathcal{K}|^2+1$ extreme points $\{T_x\}$ in the extreme traffic set $\mathcal{E}$ such that

$$T^*=\sum_{x=1}^{|\mathcal{K}^2|+1}\lambda_x T_x,\qquad \sum_{x=1}^{|\mathcal{K}^2|+1}\lambda_x=1,\quad \lambda_x\in[0,1]\quad\forall x.$$

Then, it is possible to construct $\hat{r}$ from a convex combination of $\{r_x\}$ derived from $\hat{\theta}$, $\{T_x\}$ and $\{\lambda_x\}$ such that all constraints in Equation 6 are satisfied by $\hat{\theta}$, $\hat{r}$ and T^*. This means the feasible set $\Theta(F,T^*)$ contains $\hat{\theta}$, and we have $\max_{\theta\in\Theta(F,T^*)}\theta=\theta^*\geq\hat{\theta}$, which is a contradiction. Thus, there exists an extreme traffic $T_x\in\mathcal{E}$ that $\theta^*=\max_{\theta\in\Theta(F,T_x)}\theta$, and the equality in Equation 7 holds. Since the equality holds for any F, it also holds at the minimum. □

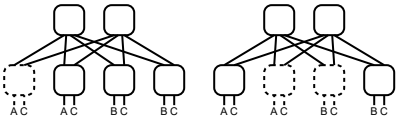

Figure 24: An L1-switch failure can reduce capacity on some trunks.

A.7 Failure of L1 switches

Unlike internal link or L2-switch failures, an L1-switch failure not only disables internal links, but also reduces capacity of trunks whose links connect to the switch, as shown in Figure 24. As a result, computing the effective capacity uses a slightly different formulation from the L2-switch case, but our definition of effective capacity is the same: we define the effective capacity as the fraction γ by which we scale the capacity of each trunk incident on a WAN router, such that the router is non-blocking under any set of traffic matrices with the reduced-capacity trunks.

Specifically, let $\mathcal{H}$ be the set of L1-switch failures, and $M_k(H)$ be the number of trunk k's active links under a failure H for $H\in\mathcal{H}$. The effective capacity γ is obtained by solving:

$$\min_{H\in\mathcal{H}}\min_{T\in\mathcal{T}}\max_{\gamma\in\Gamma(H,T)}\gamma \qquad \text{where}$$

$$\Gamma(H,T)=\left\{\gamma\in[0,1]:\ \begin{array}{ll}\gamma\sum_i t_{ij}\leq M_j(H) & ,\forall j\in\mathcal{K}\\ \gamma\sum_j t_{ij}\leq M_i(H) & ,\forall i\in\mathcal{K}\end{array}\right\}.$$

As in the previous section, we leverage the fact that the set of all traffic matrices is a convex polytope, and only consider the (finite) set of extreme traffic matrices $\mathcal{E}$ (Theorem A.5). Algorithm 2 depicts our algorithm, which iterates over every combination of failure pattern and extreme traffic matrix. Note that, in Algorithm 2, the number of operational trunk links $\{M_k(\cdot)\}$ can be derived from $\{M_k\}$ and $\{w_{sk}\}$.

Theorem A.5. *The following equality holds:*

$$\min_{H\in\mathcal{H}}\min_{T\in\mathcal{T}}\max_{\gamma\in\Gamma(H,T)}\gamma=\min_{H\in\mathcal{H}}\min_{T\in\mathcal{E}}\max_{\gamma\in\Gamma(H,T)}\gamma$$

Proof. The proof is similar to Theorem A.4 and is omitted. □

Algorithm 2: Finding effective capacity γ

Input : $\{M_k\},\{w_{sk}\},\mathcal{E},\mathcal{H}$
Output: Effective capacity γ

$\gamma\leftarrow 1$
for $(H,T)\in\mathcal{H}\times\mathcal{E}$ **do**
  for $i\in\mathcal{K}$ **do**
    if $\sum_{j\in\mathcal{K}}t_{ij}>M_i(H)$ **then**
      $\gamma\leftarrow\min\left[\gamma,\frac{M_i(H)}{\sum_{j\in\mathcal{K}}t_{ij}}\right]$
    if $\sum_{j\in\mathcal{K}}t_{ji}>M_i(H)$ **then**
      $\gamma\leftarrow\min\left[\gamma,\frac{M_i(H)}{\sum_{j\in\mathcal{K}}t_{ji}}\right]$

A.8 Effective Capacity Under Arbitrary Combinations of Failures

To compute the effective capacity under an arbitrary combination of internal link and L1/L2 switch failures, let $\mathcal{L}_1(H)$ be the remaining

L1 switches under L1-switch failure pattern H, i.e., $\mathcal{L}_1(H) = \mathcal{L}_1 \backslash H$. The effective capacity under given failure sets $\mathcal{F}$ and $\mathcal{H}$ is a solution of the following optimization:

$$\min_{H\in\mathcal{H}} \min_{F\in\mathcal{F}} \min_{T\in\mathcal{T}} \max_{\omega\in\Omega(F,H,T)} \omega \quad \text{where} \quad \Omega(F,H,T) =$$

$$\left\{ \omega : \begin{array}{ll} \sum_{a\in\mathcal{L}_1(H)} r_{ab}^{ij} = \sum_{a\in\mathcal{L}_1(H)} r_{ba}^{ij} & ,\forall b \in \mathcal{L}_2, (i,j) \in \mathcal{K}^2 \\ \sum_{b\in\mathcal{L}_2} r_{ba}^{ij} + \omega t_{ij} \frac{w_{ai}}{M_i(H)} = \sum_{b\in\mathcal{L}_2} r_{ab}^{ij} + \omega t_{ij} \frac{w_{aj}}{M_j(H)} & \\ & ,\forall a \in \mathcal{L}_1(H), (i,j) \in \mathcal{K}^2 \\ \sum_{(i,j)\in\mathcal{K}^2} r_{ab}^{ij} \le \mathbb{I}[(a,b) \notin F] & \\ & ,\forall (a,b) \in (\mathcal{L}_1(H) \times \mathcal{L}_2) \cup (\mathcal{L}_2 \times \mathcal{L}_1(H)) \\ \omega \sum_{i\in\mathcal{K}} t_{ij} \le M_j(H) & ,\forall j \in \mathcal{K} \\ \omega \sum_{j\in\mathcal{K}} t_{ij} \le M_i(H) & ,\forall i \in \mathcal{K} \\ r_{ab}^{ij}, r_{ba}^{ij} \in \mathbb{R}_+ & ,\forall a \in \mathcal{L}_1(H), b \in \mathcal{L}_2, (i,j) \in \mathcal{K}^2 \\ \omega \in [0,1] & \end{array} \right\}$$

The set $\Omega(F,H,T)$ is similar the set $\Theta(F,T)$ except that failed L1 switches in H are not considered in $\Omega(F,H,T)$. This reflects in the usage of $\mathcal{L}_1(H)$ and $M_k(H)$.

As before, the convex polytope property of the traffic set $\mathcal{T}$ can be used to simplify the optimization to

$$\min_{H\in\mathcal{H}} \min_{F\in\mathcal{F}} \min_{T\in\mathcal{E}} \max_{\omega\in\Omega(F,H,T)} \omega.$$

Theorem A.6. *The following equality holds:*

$$\min_{H\in\mathcal{H}} \min_{F\in\mathcal{F}} \min_{T\in\mathcal{T}} \max_{\omega\in\Omega(H,F,T)} = \min_{H\in\mathcal{H}} \min_{F\in\mathcal{F}} \min_{T\in\mathcal{E}} \max_{\omega\in\Omega(H,F,T)}$$

Proof. The proof is similar to Theorem A.4 and is omitted. □

A.9 Compact Forwarding Table Formulation

The Formulation. Our compact forwarding table optimization seeks to find flow count assignments x_{ba}^{ij} and x_{ab}^{ij}, such that the maximum total flow count assignment at an L2 switch is minimized:

$$\min_{(x,\mu)\in\Phi(F,\mathcal{T})} \max_{b\in\mathcal{L}_2} \left[\sum_{a\in\mathcal{L}_1} \sum_{(i,j)\in\mathcal{K}^2} x_{ba}^{ij} \right] \quad \text{where} \quad \Phi(F,\mathcal{T}) =$$

$$\left\{ (x,\mu) : \begin{array}{ll} \sum_{a\in\mathcal{L}_+^{ij}} x_{ab}^{ij} = \sum_{a\in\mathcal{L}_-^{ij}} x_{ba}^{ij} & ,\forall b \in \mathcal{L}_2, (i,j) \in \mathcal{K}^2 \\ \sum_{b\in\mathcal{L}_2} x_{ab}^{ij} = \mu v_a^{ij} \mathbb{I}[a \in \mathcal{L}_+^{ij}] & ,\forall a \in \mathcal{L}_1, (i,j) \in \mathcal{K}^2 \\ \sum_{b\in\mathcal{L}_2} x_{ba}^{ij} = \mu v_a^{ij} \mathbb{I}[a \in \mathcal{L}_-^{ij}] & ,\forall a \in \mathcal{L}_1, (i,j) \in \mathcal{K}^2 \\ \sum_{(i,j)\in\mathcal{K}^2} \theta(F) t_{ij} x_{ab}^{ij} \alpha_{ij} \le \mu \mathbb{I}[(a,b) \notin F] & \\ & ,\forall (a,b) \in \mathcal{L}_1 \times \mathcal{L}_2, T \in \mathcal{T} \\ \sum_{(i,j)\in\mathcal{K}^2} \theta(F) t_{ij} x_{ba}^{ij} \alpha_{ij} \le \mu \mathbb{I}[(a,b) \notin F] & \\ & ,\forall (a,b) \in \mathcal{L}_1 \times \mathcal{L}_2, T \in \mathcal{T} \\ \mu, x_{ab}^{ij}, x_{ba}^{ij} \in \mathbb{Z}_+ & ,\forall (a,b) \in \mathcal{L}_1 \times \mathcal{L}_2, (i,j) \in \mathcal{K}^2 \end{array} \right\}$$

In the formulation, the first constraint ensures flow count conservation at each L2 switch for each trunk pair. We define $\mathcal{L}_+^{ij}$ and $\mathcal{L}_-^{ij}$ respectively be the set of L1 switches with upflow flow count and the set of L1 switches with downflow flow counts. The second and third constraints ensure conservation between flow counts at a switch having upflow and downflow respectively. In some cases, these flow counts may cause a link to exceed its capacity, in which case we scale the flow counts by a factor μ. The fourth and fifth constraints limit the flow counts on each internal link by link capacity. In this step, we also scale the traffic matrix by the effective capacity $\theta(F)$ for the given failure pattern, as computed in §3. The last constraint defines the domain of the variables.

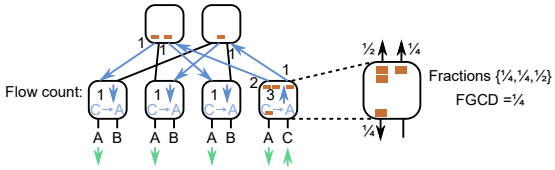

Figure 25: WCMP weights are derived at an L1 switch and an L2 switch.

Computing WCMP Weights for Egress Links in L1 Switches. As discussed above, the assigned flow counts $\{x_{ba}^{ij}\}$ from the compact routing optimization can be directly used as WCMP weights in L2 switches. However, at an L1 switch, $\{x_{ab}^{ij}\}$ (also an output of the optimization) only assigns flow counts to links to L2 switches. L1 switches also have egress links and *these must be considered when assigning weights at L1 switches.* The rate of egress traffic from trunk i to trunk j at an L1 switch s is $t_{ij}w_{sj}/M_j$, and each switch port carries t_{ij}/M_j. Then, the *egress fraction* is $1/M_j$ per switch port.

WCMP weights can be derived from these egress fractions and the upflow fractions. At an L1 switch s, the FGCD of the fractions per trunk pair (i,j) is FGCD $\left(1/M_j, \{\alpha_{ij}x_{sb}^{ij}\}_{b\in\mathcal{L}_2}\right)$. WCMP weights are the fractions divided by that FGCD. This is illustrated in Figure 25 which shows the WCMP weights assigned to the rightmost L1 switch as 2 and 1 for upflows and 1 for early forwarding.

A.10 Partial External Trunk Failure

We can extend our approach to handle partial external trunk failures, in which one or more links in a trunk can fail. Such a failure can increase upflow because traffic on the failed links is evenly distributed over the remaining links of the same trunk. The increased upflow requires re-calculating (a) traffic matrices, (b) effective capacity, and (c) routing tables.

Traffic Matrices. When some external links fail, the capacity of the trunk associated with the failed links decreases, from M_k to, say, M'_k. This change in capacity changes the set of traffic matrices Equation 4 so we need to re-compute the extreme traffic set for the new trunk set $\{M'_k\}$.

External Link Failure Pattern. An external link failure affects how traffic flows internally in the router, since there can be no ingress or egress traffic on the failed link. We assume a given wiring cannot be rewired, so an external link failure "removes" the failed links from the wiring and yields a *residual wiring* that contains only active links. For example, in Figure 25, if the link connecting trunk **A** and the leftmost L1 switch fails, the residual wiring at the switch, say 1, is $(w'_{1A}, w'_{1B}, w'_{1C}) = (0,1,0)$. Now, for a given minimum-upflow wiring, each external link failure pattern can induce a new residual wiring and for each such residual wiring, we would need to pre-compute effective capacity and routing tables.

To reduce the space of residual wirings, we can *canonicalize* each external failure pattern (in much the same way as we canonicalize internal failures §3.3), so we would only need to consider as many residual wirings as the number of canonical patterns. Specifically,

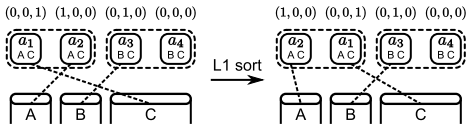

Figure 26: Steps in the canonical-form resolution for external failure.

given an external failure pattern Q, Algorithm 3 can find a canonical form $\psi'(Q)$ of the external failed links. Figure 26 shows an example of applying this algorithm to the external failure pattern on the left. The first step labels each L1 node by the number of failed external links it associated with. For example, the first L1 switch a_1 has a failed link from trunk **C**. So, the algorithm assigns it a label $(0, 0, 1)$. Steps 2-4 group L1 nodes that have the same wiring and sort the nodes according to their label in descending order. For example, a_2 becomes the leftmost L1 switch in its group $\{a_1, a_2\}$. Intuitively, the algorithm moves failed links to the left within each L1 group.

Algorithm 3: Canonical-form resolution for external failure

Input : Graph Q
Output: Canonical form $\psi'(Q)$

1 Label each L1 node by the numbers of trunk's failed links.
2 Group L1 nodes by their trunk wiring.
3 **for** *each group of L1 nodes* **do**
4 Sort nodes in descending order by their labels.

Once canonical forms are available, the residual wiring w' is just the original wiring w with failed links in the canonical form $\psi'(Q)$. Each residual wiring is then used to generate internal failure patterns (§3.3).

Effective Capacity and Routing. Under external link failure, the steps above generate a new trunk set $\{M'_k\}$, residue wiring w', a traffic set $\mathcal{T}'$, an extreme traffic set $\mathcal{E}'$, and a set of internal failure patterns $\mathcal{F}'$. They are the inputs to an effective capacity calculation, which is a modified version of the optimization in §A.8. Intuitively, the modified version replaces $(\{M_k\}, w, \mathcal{T}, \mathcal{E}, \mathcal{F})$ with $(\{M'_k\}, w', \mathcal{T}', \mathcal{E}', \mathcal{F}')$, and it assumes that all internal and external links of an L1 switch fail if the switch fails. After obtaining the effective capacity, a compact routing table could be optimized by computing flow counts and solving the formulation in §A.9 using the new effective capacity, $\mathcal{T}'$, $\mathcal{E}'$, and $\mathcal{F}'$.