Detecting Network Load Violations for Distributed Control Planes

Kausik Subramanian
University of Wisconsin-Madison
Madison, WI, USA
sskausik08@cs.wisc.edu

Anubhavnidhi Abhashkumar
University of Wisconsin-Madison
Madison, WI, USA
abhashkumar@wisc.edu

Loris D'Antoni
University of Wisconsin-Madison
Madison, WI, USA
loris@cs.wisc.edu

Aditya Akella
University of Wisconsin-Madison
Madison, WI, USA
akella@cs.wisc.edu

Abstract

One of the major challenges faced by network operators pertains to whether their network can meet input traffic demand, avoid overload, and satisfy service-level agreements. Automatically verifying if no network links are overloaded is complicated—requires modeling frequent network failures, complex routing and load-balancing technologies, and evolving traffic requirements. We present QARC, a distributed control plane abstraction that can automatically verify whether a control plane may cause link-load violations under failures. QARC is fully automatic and can help operators program networks that are more resilient to failures and upgrade the network to avoid violations. We apply QARC to real datacenter and ISP networks and find interesting cases of load violations. QARC can detect violations in under an hour.

CCS Concepts: • **Networks** → **Network reliability**; • **Software and its engineering** → *Formal software verification.*

Keywords: Quantitative Verification; Routing Protocols; Fault Tolerance; Abstract Representation

ACM Reference Format:
Kausik Subramanian, Anubhavnidhi Abhashkumar, Loris D'Antoni, and Aditya Akella. 2020. Detecting Network Load Violations for Distributed Control Planes. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '20), June 15–20, 2020, London, UK*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3385412.3385976

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PLDI '20, June 15–20, 2020, London, UK
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7613-6/20/06…$15.00
https://doi.org/10.1145/3385412.3385976

1 Introduction

Managing modern data center and ISP networks is an incredibly difficult task. Many of these networks serve a diverse set of customers who demand stringent guarantees from the network, such as high path availability and a variety of path-based properties (e.g., service chaining, and path isolation), and these requirements evolve over time. Thus, operators must face the challenge of programming a network that meets various requirements.

Various frameworks have been developed to ease the operators' task of programming the network to ensure requirements are met. Intent-based programming [31], where operators specify what they want the network to do instead of worrying about how the network must be configured, has made synthesizing network control and planes simple [7, 8, 13, 44, 45]. Using these tools, operators need to specify the network requirements at a high-level, and compliant low-level network implementations are synthesized automatically. Other frameworks validate if the current network satisfies important properties [4, 6, 15, 16, 20, 23], and automatically take corrective action otherwise [14, 19].

While these tools are invaluable, they focus on *qualitative* properties—informally, path properties that are all variants of reachability. In contrast, *quantitative* properties, which are also central to network management, have received little systematic treatment. One such property, which our paper focuses on, pertains to meeting demand, i.e., *can a given network (topology and control plane) accommodate input traffic demand without any link becoming overloaded?*

The focus on qualitative properties is well justified, because violation of such properties can lead to serious policy violations, e.g., broken isolation, circumventing firewalls, or network partitions. But violation of quantitative properties can have equally serious implications. For instance, when network links become overloaded, customers' applications may suffer from increased losses and latencies, violating customer guarantees and impacting operator revenues.

In this paper, we present an intent-based framework that helps programmers verify whether a network meets a complex quantitative property—absence of link overload when network links become unavailable due to failures. We believe that our *verification* framework forms the first step in the much harder challenge of developing an intent-based framework for generating compliant network control planes for qualitative *and* quantitative properties.

Verification. Verifying link overload is non-trivial. First, link failures are common in networks [24] due to various factors, e.g., human error and device overheating. Thus, even if link loads are low now, significant overload may occur when one or more links fail. In datacenter networks, packet losses due to high load (i.e., congestion) are common [51]. Ideally, we must *proactively verify for potential overload*: does there exist a failure where some link is overloaded? This is difficult because of the exponentially many failure scenarios.

Second, checking for overload needs an estimated model of traffic volumes, but actual traffic volumes may vary substantially around such estimated values [41]. The network's control plane may not be programmed to accommodate such variations, leading to overload when traffic spikes unexpectedly along certain paths. Thus, verification must account for *variations* in measured traffic volumes. Unlike the set of failures which can be enumerated, the set of traffic demands to consider could be infinite (as traffic quantities are rational).

Third, the control planes in most networks are *distributed* in nature and use a mix of different routing protocols [10, 22], such as Open Shortest Path First (OSPF) and Border Gateway Protocol (BGP), configured in low-level languages. The complexity of the control plane designs [9, 10, 22] and the interactions among the constituent routing protocols make it difficult to reason about the paths induced in a network under failures, and hence, about potential link overload.

Our first contribution is a verification framework that, given a network, its distributed router configurations, and an input traffic matrix with variable traffic volumes, identifies failure scenarios that can cause overload on some network link. Our framework also allows operators to focus the verification on failure events that occur with a certain minimum likelihood or involve k or fewer links.

To conduct verification, we need a model of how the distributed control plane reacts to failures and recomputes paths, and how traffic load is spread across recomputed paths. Furthermore, analyzing the model to determine potential load violations should be fast to enable quick corrective action. To this end, we develop QARC, a control plane model for quantitative analysis, with a focus on analyzing link overload. QARC builds on a weighted digraph-based control plane abstraction, and couples it with flow quantities and a novel mixed integer linear program encoding. QARC computes if links can become overloaded under some failure without enumerating failures or traffic matrices that adhere to a given bounded variation; without materializing paths under each failure; and without having to analyze all links and considering the load contributed by all source-destination pairs. These optimizations to verification are our second contribution, which help achieve substantial speedup of $5-800\times$ over SMT-based state-of-art verifiers [6].

Upgrade. Once verification has identified a potential load violation, it is crucial to understand how to avoid overload to mitigate impact on production traffic as much as possible. There are many aspects of a network's design and operation which can cause load violations, e.g., network topology, configuration, input traffic matrices, etc. It is cumbersome for operators to manually identify the root cause of a violation and reconfigure the network to ensure load violations do not happen in practice. Instead, we propose an intent-based approach where operators specify the input traffic characteristics and set of failure scenarios, and QARC will upgrade the link capacities such that no load violations will occur under these scenarios. Moreover, QARC will compute the *minimal* capacity upgrade to prevent unnecessary overprovisioning.

Analysis of real networks. Today, very little is known about how robust real-world network designs and control plane configurations are in avoiding overload under observed and expected traffic patterns. We conduct a detailed study of potential link overload in production networks.

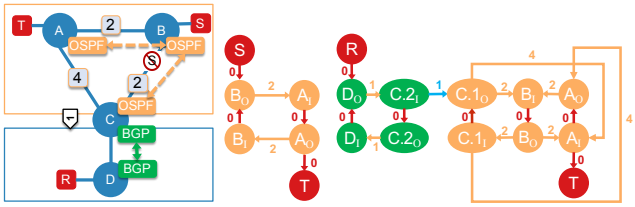
We apply QARC to 112 data center networks of a large service provider, and 86 ISP networks from the Topology zoo. We use a mix of real and synthetic control plane configurations. We apply a variety of realistic traffic matrices and traffic volume variations. We find that *70%* of the networks experience link overload under 1 or 2 link failures with different traffic characteristics and variation. We also find that the ISP networks in our study are susceptible to link overload under failures if overall traffic increases by 2-12%, while datacenter networks require 5-35% to experience link overload. We also show QARC is practical: QARC can verify real-world networks we study in under an hour (§9).

2 Motivation

We provide an overview of distributed network control planes, control plane abstractions and their shortcomings for reasoning about network load violations.

2.1 Network Control Planes

A distributed network has two components: the control and the data plane. The control plane exchanges information to routers and decides the best path for every packet, these best paths are installed in the data plane. In a distributed control plane, each router runs one or more routing processes, each process implementing a path selection algorithm, defined by a standard routing protocol—e.g., OSPF [36], BGP [39], and RIP [33]. A routing process will receive a set of routing advertisements from different routing processes: could be from



(a) (b)

Figure 1. (a) Example control plane with OSPF and BGP. The boxes denote the routing processes, the numbers on links are OSPF weights, and the redistribution cost of OSPF to BGP is 1. There is an ACL installed at router B for traffic class $S - T$. (b) ARC ETGs for (a).

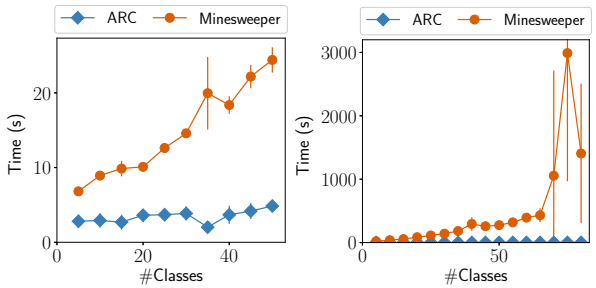
the same router or neighbouring routers, and could be of the same or different routing protocol. Each routing process's path computations are based on different protocol-specific configuration parameters and advertisements. Each routing process decides the best path(s) and could advertise the best path(s) to other routing processes. The steady-state outcome of the control plane's computation is a set of forwarding rules computed at each router, which encode the next-hop for different traffic classes, i.e., source-destination subnets. Operators typically decide which protocols and configuration to use based on network design and operational policies they wish to enforce [34] like reachability, waypointing etc.

Figure 1(a) shows a toy distributed control plane spanning four routers. Two routers run OSPF, one runs BGP, and one runs both and is configured to redistribute routes from OSPF to BGP. The OSPF edge weights are configured to support load balancing for $R - T$ traffic, while an Access Control List (ACL) is configured at router B to block $S - T$ traffic for a security policy. Distributed control planes are programmed at a low-level by configuring routers using vendor-specific languages (e.g., Cisco IOS [3]), either manually or using automated tools [7, 13, 44, 46].

Another mechanism commonly used in distributed control planes is load-balancing traffic between endpoints. ECMP (Equal Cost Multi-Path routing) [27], or its weighted variant (WCMP [50]), are used to distribute traffic across multiple paths between endpoints in the network. In an ECMP router, traffic is split *equally* among the "best paths" based on routing metrics.

2.2 Control Plane Abstractions

Analyzing whether a network satisfies certain properties over paths, qualitative or quantitative, requires knowing the network forwarding state. Understandably, inferring this based on low-level router configurations is difficult as the forwarding state is the result of complex distributed computations. Thus, many frameworks [4, 6, 15, 16, 20, 49] have been developed to model the distributed network control logic, so that different inferences can be made about network



(a) N1 **(b)** N2

Figure 2. Network load verification times for ARC and Minesweeper for networks N1 and N2 and varying number of traffic classes.

forwarding—under failures—without actually deploying the network configurations. These models are used widely in verifying qualitative properties, e.g., middlebox traversals.

However, verification of network overload, a quantitative property, is challenging in aspects that the state-of-art models cannot handle. Firstly, verifying this property requires a model that can reason about how much traffic a traffic class sends on a path (or multiple paths for ECMP/WCMP). Moreover, the model needs to reason about the collective load imposed by *all* traffic classes in the network. Contrarily, verifying qualitative properties only requires reasoning about small subsets of traffic classes.

Secondly, operators want to understand if the network can handle load under different failure scenarios. While in the worst-case scenario where all devices fail together the network will be overloaded, the probability of such scenario occurring is very small. Thus, the operator may constrain the set of requisite failure scenarios—e.g., only 1 and 2-link failures. The number of failure scenarios grows exponentially with number of failed links. Thus, the approach adopted by some control plane models, e.g., Batfish [16], of enumerating all failure scenarios can be *prohibitively* expensive. Therefore, we require our network model to symbolically encode network routing state even under failures.

2.3 ARC vs. Minesweeper

Two networks models are amenable for verifying network overload under failures: ARC [20] and Minesweeper [6]. Our work builds on ARC and in this section we motivate this choice.

The ARC network model uses the fact that most routing protocols use a *cost-based path selection algorithm*. E.g., OSPF uses Dijkstra's algorithm to compute minimum cost paths from a source to all destinations, where each link has a cost; and BGP selects paths based on numeric metrics, most importantly, the AS path length. ARC abstracts the different routing protocols, static routes, and ACLs based on the above principle, and models the network's collective forwarding

behavior under failures using a set of weighted directed graphs (called ETGs) satisfying the *path-equivalence* property: under *any arbitrary failure scenario*, the path taken by the traffic will correspond to the *shortest weighted path* in the graph. Figure 1(b) illustrates the ETGs for the two traffic classes (R-T and S-T) for the network shown in Figure 1. ARC cannot model every set of router configurations as certain routing constructs like BGP local preferences do not adhere to the cost-based path selection principle.

Minesweeper [6] encodes the converged routing state of the control plan using SMT (Satisfiability Modulo Theories) formulas [12] and supports iBGP and BGP local preferences, which ARC cannot handle. Minesweeper is primarily used to verify qualitative properties dealing with a single or small subset of traffic classes under 1-link or no failure.

Let us consider ARC v/s Minesweeper for detecting link overload under no failures. Using ARC, we can compute the exact network paths taken by the traffic classes using the polynomial-time Dijkstra's algorithm and thus compute individual link utilizations. On the other hand, for Minesweeper, due to the symbolic nature of the network abstraction, we need to use the SMT solver to find the network paths and compute utilizations. Handling network failures further slows down verification. We compare verification performance by using the Minesweeper and ARC abstractions for $k = 1$ link failure scenarios in Figure 2. We can observe that ARC is significantly faster than Minesweeper, achieving 5× speedup for N1 and $20 - 800\times$ for the larger N2 network.

Moreover, the ARC abstraction offers certain key benefits for network transformations [38] which can help reduce the search space efficiently and enables even faster quantitative verification than Minesweeper (§6). For example, in ARC, we know a certain path will not be traversed if a shorter path exists in the network (by simply comparing the path cost). This property can be used to reduce the search space—both by limiting which links to consider as candidates for overload, and by ignoring the contributions of some source-destination pairs' traffic volumes—in scenarios when a small number of links can fail together. In Minesweeper, the paths are symbolic (represented by SMT constraints), and thus, we cannot a priori tell which path would be preferred.

Since ARC is a more natural fit for quantitative analysis, ARC's natural attributes support effective speedup of verification, and ARC has near-universal control plane coverage, we adopt it as the basis for our framework. In § 3, we extend ARC to model (1) paths under failures without enumeration, (2) distribution of load on equal cost paths (due to ECMP or WCMP), and (3) variation of traffic volumes.

3 QARC (ARC with Quantities)

ARC. ARC abstracts the set of router configurations using a set of weighted directed graphs called *extended topology graphs* (ETGs). For each traffic class, ARC constructs one ETG to model the behavior of the network routing protocols and interactions among the routers for the traffic class (shown in Figure 1(b)). In the ETG, each vertex corresponds to a routing process; there is one incoming (I) and one outgoing (O) vertex for each process. Directed edges represent possible paths enabled by exchange of routing advertisements between connected processes. The weights of the edges of the ETG satisfy the *path-equivalence property*: under any failure scenario, the actual network path(s) are the shortest weighted path(s) in the ETG. Thus, the ARC abstraction can be used to compute the forwarding state of the network under any failure scenario, abstracting away the need to model route advertisements and path computations at each router. Note that the path-equivalence property of ARC can model ECMP-style load-balancing.

ARC was designed to use graph algorithms on the extended topology graphs (ETGs) to verify qualitative properties under *any* arbitrary failure scenario—e.g., a traffic class is connected under any k-link failure if the min-cut for the traffic class's ETG is $\geq k$.

QARC. Our model, QARC, extends ARC and it uses a mixed-integer linear program (MIP) encoding to add symbolic traffic quantities to the ARC ETGs to verify load properties under different failure scenarios.

Verification. Provisioning bandwidth is expensive, especially in wide area networks where adding new links is not an easy endeavor. Operators therefore want to keep link utilization high and, at the same time, tolerate link failures. QARC helps operators predict if the network will encounter load violations under failures. Operators have frameworks to periodically update routing configurations [7, 44, 46] and measure input traffic characteristics [25, 26, 29]. We envision QARC to be used for verification whenever traffic characteristics change significantly, or the control plane is reconfigured. Taking as input the new control plane and traffic matrix (between endpoints) with bounds on traffic variation, our verification detects if there exists a failure scenario leading to utilization exceeding capacity on any link.

Consider the network control plane in Figure 1(a) and the corresponding ARC ETGs in Figure 1(b). In this scenario, the operator sees a change in the input traffic: the current traffic for classes $R \rightarrow T$ and $S \rightarrow T$ are 80 Gbps and 30 Gbps, respectively. All links in the network have a capacity of 100 Gbps. When no links have failed, the traffic from $S \rightarrow T$ flows through the path $B \rightarrow A$, while the traffic from $R \rightarrow T$ is load-balanced by ECMP at C; 40Gbps traffic is sent through the two paths: $D \rightarrow C \rightarrow A$ and $D \rightarrow C \rightarrow B \rightarrow A$ (these are the shortest network paths in the ETGs). As we can observe, all links' utilizations are below capacity ($D \rightarrow C : 80$, $C \rightarrow B : 40$, $C \rightarrow A : 40$, $B \rightarrow A : 40 + 30 = 70$).

Suppose, the operator wants to inspect if the network, for the given traffic matrix, can experience some link becoming overloaded under a single link failure. Using our tool QARC,

the operator can discover that when $C \rightarrow A$ fails, the traffic on $B \rightarrow A$ will be 110 Gbps, exceeding the link's capacity.

Going further, the operator can discover other single link failures by asking our verification tool to find 1-link failures *other than* $C \rightarrow A$. Likewise, the operator can discover sets of k-link failures that cause links to overload.

Upgrade. When verification detects a possible load violation, the operator can invoke our upgrade framework. Taking as input the network configurations and the traffic matrix, QARC computes a minimal set of links whose capacities need to be upgraded to avoid overload under failures. For the network in Figure 1, QARC suggests to increase the $B \rightarrow A$ capacity by 10Gbps to ensure no load violations occur under any 1-link failures.

3.1 Problem Definition

Let TC be a set of traffic classes, $N = (Routers, Links)$ be the network topology and $FL \subseteq Links$ be the failure scenario—i.e., the set of links that failed. For each traffic class $tc \in TC$, the routing processes compute paths from source to destination; the paths form a directed acyclic graph (DAG), which we call the *flow graph*. We represent tc's flow graph as $FG(tc, FL) = (V_{tc}, L_{tc})$. Traffic will not flow on failed links, thus $L_{tc} \cap FL = \emptyset$. ARC constructs a weighted directed graph $ETG(tc)$ for each traffic class tc with the following property:

Theorem 3.1 (Path-equivalence [21])**.** *For every traffic class $tc \in TC$, if the destination is reachable from the source router in the actual network, then, after removing edges corresponding to failed links from $ETG(tc)$, the shortest path in the ETG from the source router to destination router is equivalent to the path computed by the actual network.*

Thus, the flow graph $FG(tc, FL)$ will be a sub-graph of $ETG(tc)$ and every path from source to destination in $FG(tc, FL)$ will be the shortest weighted path based on $ETG(tc)$ weights.

Since there are multiple paths in the flow graph, the traffic is split across different paths based on the load-balancing scheme deployed in the network. We define the flow function $F(l, tc, FL)$ as the amount of flow of tc on a link l in $FG(tc, FL)$. In ECMP, each router divides the total incoming flow equally among the outgoing links leading to shortest paths. For a node r, we define the set of nodes which have an incoming edge into r as $prev(r) = \{r' | (r', r) \in L_{tc}\}$, and the set of nodes connected by an outgoing edge from r as $next(r) = \{r' | (r, r') \in L_{tc}\}$. Thus, we can define ECMP behavior in terms of the flow function as:

$$\forall r. \forall r' \in next(r).\ F((r, r'), tc, FL) = \frac{\sum_{r'' \in prev(r)} F((r'', r), tc, FL)}{|next(r)|}$$

For each link $l \in Links$ and failure scenario FL, we define the utilization l under the failure scenario FL as $Util(l, FL) = \sum_{tc \in TC} F(l, tc, FL)$. We represent the link capacity of link l as $Cap(l)$. A network load violation occurs when a link's utilization exceeds its capacity.

Table 1. QARC variables

Name	Description	Range
$Flow(e, tc)$	Fraction of traffic for class tc flowing on edge e	[0,1]
$\Delta(tc)$	Fraction of traffic variation for class tc	[0,1]
$\Delta(e, tc)$	Fraction of traffic variation for class tc flowing on edge e	[0,1]
$Dist(n, tc)$	Shortest path distance of node n to destination in ETG	$\mathbb{Q}$
$Fail(e)$	Link e ($\forall tc$) failure status (1 ≡ failed)	$\{0, 1\}$
$Load(e)$	Link e load status (1 ≡ overloaded)	$\{0, 1\}$

Table 2. QARC constants

Name	Description
TC	Set of all traffic classes
$\overline{FL}$	Set of all failure scenarios
$T(tc)$	Traffic sent by class tc ($T(tc) \in \mathbb{Q}$)
$W(e, tc)$	Weight of edge e in tc ETG
$Src(tc)$	Source node of traffic class tc
$Dst(tc)$	Destination node of traffic class tc
$In(n)$	Set of in-edges of node n in ETG
$Out(n)$	Set of out-edges of node n in ETG
$OEdges(tc)$	Set of all outgoing edges in tc ETG
$Links$	Set of all physical links in the network
$Cap(e)$	Capacity of link e ($Cap(e) \in \mathbb{Q}$)
Π	Threshold of total traffic variation

Definition 3.2 (Verification)**.** Given a set of failure scenarios $\overline{FL}$, the *verification problem* is to find a failure scenario $FL \in \overline{FL}$ that leads to a network load violation—i.e., $\exists FL \in \overline{FL}.\ \exists l \in Links.\ Util(l, FL) \geq Cap(l)$.

4 QARC Encoding

We now present the mixed-integer linear program (MIP) encoding the semantics of QARC. Our encoding makes new technical contributions. First, it extends ARC shortest-path-forwarding ETGs with flow quantities while accounting for failures and permitting bounded variation of traffic characteristics. This is crucial to determining link overload. Second, it *models flow being split among multiple shortest paths* even under failures, i.e., QARC models ECMP, which is a key construct used in most networks. Third, it does not require disjunctions and only uses integer variables to represent failures and link overload, therefore enabling fast verification.

Table 1 and Table 2 describe QARC's variables and constants, respectively. For each section, the sentences within boxes state what property the presented constraints encode.

4.1 Flow Constraints

Traffic flows from source to destination along *active* links and flow is *conserved* at every node in the ETG.

The $Flow$ and Δ variables represent respectively the fraction of normal and variation of traffic flowing along the network links. For class tc and edge e, the actual traffic on

the edge is $(Flow(e,tc) + \Delta(e,tc)) \times T(tc)$. The distinction of *Flow* and Δ variables ensures the verification constraints (13) are linear while providing variation in traffic characteristics. For every ETG and corresponding traffic class tc, the outgoing flow at tc source and the incoming flow at tc destination should equal $1 + \Delta(tc)$ which are constrained separately:

$$\sum_{e \in Out(Src(tc))} Flow(e,tc) = 1 \qquad \sum_{e \in In(Dst(tc))} Flow(e,tc) = 1$$
$$\sum_{e \in Out(Src(tc))} \Delta(e,tc) = \Delta(tc) \qquad \sum_{e \in In(Dst(tc))} \Delta(e,tc) = \Delta(tc) \tag{1}$$

For all other nodes $n \notin \{Src(tc), Dst(tc)\}$ in the ETG, flow is conserved, i.e., incoming flow is equal to outgoing flow.

$$\sum_{e_{in} \in In(n)} Flow(e_{in},tc) = \sum_{e_{out} \in Out(n)} Flow(e_{out},tc)$$
$$\sum_{e_{in} \in In(n)} \Delta(e_{in},tc) = \sum_{e_{out} \in Out(n)} \Delta(e_{out},tc) \tag{2}$$

Next, we need to accommodate failures. When a link fails, no traffic must flow on it:

$$Flow(e,tc) + Fail(e) \leq 1 \bigwedge \Delta(e,tc) + Fail(e) \leq 1 \tag{3}$$

Thus, if $Fail(e) = 1$, then $Flow(e,tc) = 0$ and $\Delta(e,tc) = 0$.

Operators can bound the individual variation for each traffic class and/or bound the total extra variation of traffic in the network by a threshold Π:

$$\sum_{tc \in TC} \Delta(tc) * T(tc) \leq \Pi \tag{4}$$

Given a solution to the above constraints, we construct the flow graph of tc by picking all edges e where $Flow(e,tc) > 0$, which indicate all the links traffic flows on.

4.2 Distance Constraints

A computed path in the real network matches the shortest path in the ETG between the corresponding source and destination (Theorem 3.1). Thus:

Traffic must only flow on shortest-distance ETG paths.

We formulate shortest path distances to the destination node of the ETG in an inductive fashion - the shortest path from a node must traverse through one of its neighbors, and thus, distance of node n can be defined inductively as the shortest among distances from the neighbors. Again, failures introduce complications, because distances can change under different link failures. To this end, we need to two sets of constraints. First, we use the following constraints to ensure that for every node n and traffic class tc, the value of the variable $Dist(n,tc)$ is *at most* the distance of the shortest path that traverses only active links. For all $e = n \longrightarrow n' \in Out(n)$:

$$Dist(n,tc) \leq W(e,tc) + Dist(n',tc) + \infty \times Fail(e) \tag{5}$$

The intuition is that when edge $n \longrightarrow n'$ has failed and $Fail(n \longrightarrow n') = 1$, the shortest distance from n will not depend on the path through n' as the equation will be satisfied trivially due to the large constant in front of $Fail(n \longrightarrow n')$.

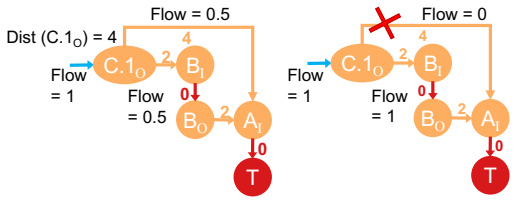


Figure 3. ECMP behavior of node $C.1_O$ under two different scenarios for the partial $R-T$ ETG from Figure 1(b). Note that Sum of Flow in the network weighted with edge weights is the distance of the path taken by the flow for both scenarios.

The above constraints provide an upper-bound on *Dist* variables, but do not ensure that the variable values are exactly equal to actual shortest distances in the ETG. E.g., setting all *Dist* variables to 0 trivially satisfies Constraints (5).

Second, we use another set of constraints to ensure that the ETG traffic flow only uses the shortest paths in the network. (We will consider load-balancing in §4.3 and ignore it here). We illustrate the idea of our encoding using the example ETG in Figure 3. For the *Flow* quantities, the cost of the path taken by the flow can be computed by the sum of *Flow* on all ETG edges multiplied by the edge weights – i.e., $\sum_e W(e,tc) \times Flow(e,tc)$ (regardless of whether the flow is sent on the shortest path or not). The following constraint ensures that the flow is sent on the shortest distance path from source to destination of the ETG of tc:

$$Dist(Src(tc)) = \sum_{e \in OEdges(tc)} Flow(e,tc) * W(e,tc) \tag{6}$$

Since Constraint (5) guarantees upper bounds on the shortest distance, the above constraint will ensure the traffic will not be sent on a longer path. Also, by virtue of Constraints (5) and (6), the *Dist* variables for all nodes in the ETG will be exactly equal to the shortest distances to the destination. Notice, that *the constraint will guarantee this property even if traffic flows across multiple shortest paths.*

To ensure that the traffic variation only flows on shortest paths, we add the following constraint:

$$\forall e \in OEdges(tc).\ \Delta(e,tc) \leq Flow(e,tc) \tag{7}$$

By virtue of Constraint (6), $Flow(e,tc)$ is 0 on non shortest path links, and consequently, $\Delta(e,tc)$ will be 0.

4.3 Load Balancing Constraints

The outgoing *Flow* and Δ of each node are *split* equally among the shortest neighbors connected by active links.

Figure 3 demonstrates how ECMP operates under no failures, and the differences that arise when a link failure occurs.

At any router, by virtue of Constraints 1-7, traffic will never be sent along longer paths. The constraints presented in this section ensure traffic is split equally among the active links at a router which are on the currently available

shortest paths to the destination. First, we show a simple disjunctive constraint that encodes the desired behavior and we then show how the same behavior can be captured without disjunction.

For node n and all outgoing edge pairs $e_1 = n \rightarrow n_1$ and $e_2 = n \rightarrow n_2$ in $Out(n)$, ECMP for $Flow$ (similarly Δ) is:

$$[W(e_1, tc) + Dist(n_1, tc) = W(e_2, tc) + Dist(n_2, tc)]$$
$$\wedge \neg Fail(e_1) \wedge \neg Fail(e_2) \implies Flow(e_1, tc) = Flow(e_2, tc)$$

The above constraints ensure that if the distance to the destination along two active outgoing edges is equal, then the flow on them will also be equal (if these edges do not lie on the shortest path, then flow will be 0 on both edges). However, these constraints use implications (i.e., disjunctions) and cannot be provided in this form to an ILP solver.

By introducing new *rational* variables we can write constraints that express the ECMP load balancing constraints using only linear inequalities. This forms a *key innovation of* QARC *encoding*. Specifically, we define sets of variables $minFlow(n, tc) \in [0, 1]$ and $maxFlow(n, tc) \in [0, 1]$ to capture the minimum and maximum non-zero $Flow$, respectively, out of node n for traffic class tc. The non-zero flow restriction holds for flows along the shortest paths (flow along non-shortest paths will be 0), and thus we can impose constraints on $minFlow$ and $maxFlow$ to model ECMP routing. We also add similar constraints for the traffic variation Δ variables so that the total traffic adheres to ECMP.

Adding constraints for $maxFlow(n, tc)$ to be the maximum non-zero flow value is trivial, as the zero flow values will not affect the $maxFlow$ variable:

$$\forall e = n \rightarrow n' \in Out(n).\ Flow(e, tc) \leq maxFlow(n, tc)$$
$$\forall e = n \rightarrow n' \in Out(n).\ \Delta(e, tc) \leq max\Delta(n, tc) \quad (8)$$

Adding constraints for $minFlow(n, tc)$ is trickier: to ensure that $minFlow(n, tc)$ is the minimum non-zero flow value, we need to know the flow values *for all possible failure scenarios*. To bypass this problem, we use the distance variables $Dist$ to identify the next-hops that lie on the shortest paths, and thus, have non-zero flows. Based on this insight, we impose the following constraints for all $e = n \rightarrow n' \in Out(n)$ to ensure that $minFlow$ variables have the correct values:

$$\forall e = n \rightarrow n' \in Out(n).\ minFlow(n, tc) - Flow(e, tc) \leq$$
$$\infty \times (W(e, tc) + Dist(n', tc) - Dist(n, tc) + Fail(e))$$
$$\forall e = n \rightarrow n' \in Out(n).\ min\Delta(n, tc) - \Delta(e, tc) \leq$$
$$\infty \times (W(e, tc) + Dist(n', tc) - Dist(n, tc) + Fail(e)) \quad (9)$$

First, notice that the quantities $W(e, tc) + Dist(n', tc) - Dist(n, tc)$ and $W(e, tc) + Dist(n', tc) - Dist(n, tc) + Fail(e)$ are always greater or equal than 0. Hence, there are three scenarios for each edge which are all encoded by the above constraint: (1) the edge e has failed, in which case $Fail(e) = 1$, the RHS of the constraint is infinity, and the constraint is trivially satisfied, (2) the edge e is not on the shortest path, thus $W(e, tc) + Dist(n', tc)$ is greater than $Dist(n, tc)$ and thus, the RHS is a large positive constant, and (3) the edge e is active and on the shortest path, so the RHS of the constraint is 0—i.e., $minFlow(n, tc) \leq Flow(e, tc)$. Therefore, $minFlow(n, tc)$ is smaller than all the non-zero edge flows (which can only flow on the shortest paths).

Thus, for a node n, we have two variables for the lower bound and upper bound of all the non-zero edge flows out of the node. For ECMP, we require all non-zero edge flows on the shortest paths out of a node to be equal, which can be enforced by ensuring the lower bound $minFlow(n, tc)$ and upper bound $maxFlow(n, tc)$ are equal (similarly for Δ):

$$minFlow(n, tc) = maxFlow(n, tc)$$
$$min\Delta(n, tc) = max\Delta(n, tc) \quad (10)$$

Constraints 1-10 ensure the total flow to neighbors on the shortest paths are equal to one another, adhering to the ECMP load-balancing model. The above constraints can be modified by multiplying constant weight factors to $Flow$ variables to model Weighted Cost Multipathing (WCMP) [50].

4.4 Failure Constraints

Failure scenarios can be restricted by the operator, e.g., by number of links or likelihood of failures.

Operators may want to restrict the failure scenarios of interest ($\overline{FL}$) to make useful observations about how the network control plane reacts under failures. The following constraint restricts the number of link failures to k:

$$\sum_{e \in Links} Fail(e) \leq k \quad (11)$$

Theorem 4.1. *For every traffic class $tc \in TC$ and failure scenario $FL \in \overline{FL}$, Constraints 1-11 ensure that the flow Graph $FG(tc, FL)$ is a directed acyclic graph such that each path from $Src(tc)$ to $Dst(tc)$ is the shortest path in $ETG(tc)$, and for every router n in flow graph $FG(tc, FL)$, the flows on outgoing edges which lie on shortest paths from n to $Dst(tc)$ are equal., i.e., $\forall n_1, n_2 \in next(n).\ F((n, n_1), tc, FL) = F((n, n_2), tc, FL)$.*

Theorem 4.1 and Theorem 3.1 together prove that the flow graphs constructed by the QARC constraints faithfully represent the routing paths and flow distributions in the actual network.

Operators can also assign failure probabilities to individual links and restrict the search to scenarios that have probability above a threshold. Suppose, the operator has probabilities assigned to individual link failures ($P_{fail}(l)$) and is only interested in link failure scenarios that have joint probability above a threshold ω. The failure probabilities can be derived from telemetry data in real-world networks [35]. We assume that link failures are independent, thus, the probability of a certain link failure scenario is the product of individual link failure probabilities. Then, we would want to enforce

the following constraint to search for failure scenarios with probability greater than ω:

$$\prod_{l \in Links.Fail(l)=1} P_{fail}(l) \geq \omega$$

We use the logarithm of probabilities to generate the equivalent linear constraint.

$$\sum_{l \in Links} log(P_{fail}(l)) \times Fail(l) \geq log(\omega) \tag{12}$$

We can have a theorem similar to Theorem 4.1 for link failure probabilities.

5 Verification using QARC

Finally, verifying for network load violations using QARC is done by adding load-related constraints.

$Load(e) = 1$ iff the utilization of link e exceeds capacity.

The following constraint correctly sets the value of the variable $Load(e) \in \{0, 1\}$:

$$\frac{\sum_{tc \in TC}[Flow(e, tc) + \Delta(e, tc)] \times T(tc)}{Cap(e)} - Load(e) \geq 0 \tag{13}$$

The numerator of the fraction captures the total traffic flow on link e, while the denominator, $Cap(e)$ (a constant), captures the capacity of e. $Load(e)$ can only be 1 when the traffic on link e exceeds its capacity. To find if there exists a failure scenario where at least one of the links is overloaded, we can constrain the sum of load variables to be at least 1:

$$\sum_{e \in Links} Load(e) \geq 1 \tag{14}$$

When feeding the constraints to the ILP solver, there can be two outcomes. First, the solver returns a satisfiable model from which we can extract the link failure scenario under which one or more links are overloaded. Second, the solver returns unsatisfiable, which means there is no k-link failure scenario that can cause link overload. There are two explanations for this outcome: (a) the network has sufficient capacity to handle rerouted input traffic under failures, or (b) a subset of traffic classes that do not have high path redundancy are disconnected due to the failures, and the remaining active traffic classes do not have sufficient traffic to cause overloads. Note that, even when the solver finds a failure scenario where link overload occurs, certain traffic classes could be disconnected in the satisfying solution, with the remaining active traffic classes still able to overload the link(s). To summarize, QARC, as presented, will try to find potential for link overload even in the face of disconnections.

Theorem 5.1 (Correctness). *A failure scenario FL satisfies Constraints 1-14 if and only if there exists a link $l \in Links$ such that $Util(l, FL) \geq Cap(l)$.*

6 Optimizations

We now describe two techniques that take advantage of the properties of QARC abstraction and constraints to speed up verification performance. First, we use ARC's shortest-distance path property to efficiently minimize the traffic class ETGs (§6.1). Second, we show how to use minimized ETGs to partition the set of network links and perform verification in a parallel fashion (§6.2).

6.1 ETG Minimization

Due to Theorem 3.1, in an ETG, the traffic will not traverse a path with a higher cost if one with a lower cost exists. Therefore, for each traffic class, we can prune the ETG and remove edges and nodes that will not be traversed under the targeted number of failures. This pruning strategy reduces the size of the generated constraints and can result in faster verification.

To illustrate this property, consider the network in Figure 4. For any 1-link failure scenario, the traffic from S to T will never traverse router C on links $B \rightarrow C$ and $C \rightarrow A$ (because 2 shorter paths exist in the network). Thus, in the ETG, we can prune all nodes and edges corresponding to router C, forming a minimized ETG.

Identifying the redundant nodes and edges for general k link failures can be challenging—the naive approach of enumerating all k-link failures will be expensive. Instead, we modify QARC's encoding to perform this task. The core insight of our approach is to find the maximum shortest distance between the source and the destination under any k-link failure scenario—all nodes and edges that are farther than such a distance will never be traversed and can be pruned from the ETG. The same minimization cannot be performed efficiently when using Minesweeper's symbolic SMT-based abstraction: one would need to check each node and edge to detect if the node/edge is reachable under k failures. In ARC, since the edge weights are not symbolic, we can eliminate a path if "shorter paths" exist.

Minimization Constraints. Unlike verification which deals with all traffic classes, minimization only involves constraints for a single traffic class. The minimization of multiple ETGs can be done in parallel since one ETG's routing does not depend on other ETGs. For an ETG of traffic class tc, we use Constraints (5) to specify upper bounds on distances and Constraints (11) to bound the number of failures. The maximum shortest distance between the source and destination of the ETG can be found using the objective:

$$\texttt{maximize}\ Dist(Src(tc), tc) \tag{15}$$

Let us denote the objective value provided by the ILP solver as $MaxDist$. Given an edge $e : n_1 \rightarrow n_2$, if the shortest path from source to destination using the edge $n_1 \rightarrow n_2$ is greater than $MaxDist$, the edge will never be traversed under any k link failures and can be pruned from the ETG. We check for

such edges using the condition:

$$SP(Src(tc), n_1) + W(e, tc) + SP(n_2, Dst(tc)) > MaxDist \quad (16)$$

where $SP(n, n')$ denotes the length of the shortest path between n and n' in the graph (computed using Dijkstra's shortest path algorithm). After pruning the edges that are never traversed, we prune the nodes that do not have incoming or outgoing edges to obtain a minimized ETG $ETG_{min}(tc)$. ETG minimization is sound and complete, i.e., it will remove edges where traffic will never flow under k failures and will not remove edges where traffic could flow under some k failure scenario.

6.2 Parallel Verification

A major bottleneck of QARC verification is solving a Mixed Integer Linear Program (MIP) that has size linear in the number of network links and traffic classes. Consider the network-wide constraint (14) used to find load violations where at least one link's utilization exceeds its capacity:

$$\sum_{e \in Links} Load(e) \geq 1$$

If we restrict the above constraint to only consider a subset of links, QARC will find scenarios where one of these links' utilization exceeds its capacity. Thus, if we partition the set of links into N partitions, we can run N instances of verification on separate machines. By splitting the verification N-way, each instance will effectively have a smaller search space of links to find load violations, thus, speeding up verification.

There are numerous ways to partition the set of links. Instead of simply splitting the set of links into N equal-sized partitions, we leverage ETG minimization to generate a partitioning scheme that also minimizes the number of the traffic classes that contribute to load on links in each partition. The insight of our partitioning scheme is as follows: a minimized ETG prunes all links where traffic for the particular class does not flow under any k-link failure scenario. To detect if link e can be overloaded, we can prune the constraints for traffic classes that do not contain e in their minimized ETGs. Thus, for each partition, we only need to consider a subset of traffic classes, reducing verification time further.

We formulate our partitioning scheme as an Integer Linear Program (ILP). Our partitioning scheme is tied inherently to the ETG minimization, which we can perform efficiently due to the ARC abstraction. Without ETG minimization, all traffic classes would contain all links in their ETG (barring ACLs and filters), thus, we would not be able to further reduce the constraints in each partition.

7 Upgrading Link Capacities in QARC

Once a load violation is found, the operator might want to modify the network to ensure that all link utilizations do not exceed capacity under all considered failure scenarios. One way to achieve this property is to modify the control plane.

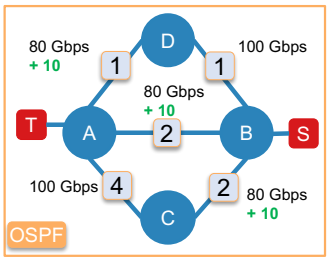


Figure 4. Example OSPF network where a link gets overloaded under 1 and 2 link failure scenarios. The capacities in green are the minimum link capacity additions computed by QARC to ensure no violations occur under ≤ 2 failures.

However, modifying the control plane may cause violations of *qualitative* properties that the current configuration satisfies (access control, waypointing for middleboxes, etc.). Finding policy-compliant control-plane modifications is already a computationally hard problem [19], and accounting for network load on all links for all traffic classes complicates the problem further.

Instead, we propose to use the QARC abstraction to *find new link capacities guaranteeing all link utilizations do not exceed capacity under all considered failure scenarios.* There are multiple mechanisms for realizing the new capacities—physically adding more cables to increase capacity, or dynamically changing capacity in the case of optical links [42]. Increasing bandwidth is expensive, thus, we need to ensure we do not increase capacities beyond what is necessary.

Upgrade Formulation. Concretely, the upgrade problem is as follows: *Given a set of network configurations and input characteristics, find the minimal set of links and the minimal capacity additions to these links such that the network is not overloaded under the given failure scenarios.*

Figure 4 illustrates the upgraded link capacities computed by QARC. The traffic sent by class $S \rightarrow T$ is 90 Gbps. As we can see, under 2-link failure scenarios (e.g., $D - A$ and $B - A$), some links' utilization exceeds capacity (e.g., $B - C$). QARC computes the minimum capacity additions to the links (shown in green—e.g., +10); by increasing the capacities, no 1- or 2-link failure scenario can cause violations.

We use QARC to compute new link capacities such that link utilization never exceeds the "new" capacity under the provided failure scenarios. We use the QARC constraints (§4.1-§4.4) and compute the minimum link capacity that link l should have using the following objective:

$$\texttt{maximize} \sum_{tc \in TC} [Flow(l, tc) + \Delta(l, tc)] \times T(tc) \quad (17)$$

QARC computes the maximum utilization for the link under the different failure scenarios given the input traffic characteristics with bounded variation, which is the *minimum new capacity* required on the link to ensure that the particular

link is not overloaded. Note that, this capacity is not dependent on the capacity of other links, as the distributed control plane chooses the "best" active path(s) under failures based on configuration parameters such as static link weights and path lengths. Thus, similar to verification, we can parallelize the phase of computing the upgraded link capacities.

The set of links whose capacities need an upgrade is *minimal*—i.e., if any of these links have capacities less than the computed ones, there exists a failure scenario under which a link will be overloaded.

8 Limitations

QARC relies on the underlying ARC's graph abstraction and the path-equivalence property to model the flow of traffic in the network to detect load violations. Thus, QARC can only be used for network configurations that can be faithfully represented by ARC. ARC cannot model local path selection criteria, e.g., BGP local preference, because it is not possible to statically assign edge weights to ETGs such that local decisions (e.g., local preferences) override the global costs (e.g., path lengths) without compromising the path equivalence property. Similarly, iBGP's path selection does not use global costs like path lengths, and hence, cannot be modeled by ARC. ARC also does not support BGP communities, which are tags on route advertisements that influence path selection at routers. While state-of-art frameworks like Minesweeper [6] can support the above configuration constructs, the generality comes at the cost of performance (Figure 2). Tiramisu [4] also supports a wide range of configuration constructs like Minesweeper, but Tiramisu's abstraction is tied to qualitative verification algorithms, and cannot be extended to support quantitative verification.

Moreover, a majority of real-world network configurations do not deploy these complex BGP features. Gember-Jacobson et. al [20] analyzed the network configurations of 314 datacenter networks operated by a large Online Service Provider and showed that ARC produces path-equivalent abstractions for more than 95% of the networks. Those networks did not use BGP local preferences or other protocol features not supported by ARC. Thus, QARC has sufficiently high feature coverage to be useful in real-world settings.

9 Evaluation

QARC is implemented in Java using the open-source ARC [2] and Batfish [1] frameworks. QARC uses the Gurobi ILP solver [37] for solving the verification and repair constraints. Our evaluation answers the following questions:

Q1: Do real datacenter and ISP networks suffer from link overload under failures? (§9.1)

Q2: How quickly can QARC detect load violations or report the absence of violations? (§9.2)

Q3: Do our optimizations speed up verification? (§9.3)

Q4: How quickly can QARC compute links to upgrade? (§9.4)

Experiments were conducted on a 5-node 40-core Intel-Xeon 2.40GHz CPU machine with 128GB of RAM.

Networks. We study 112 datacenter networks operated by Microsoft, and 86 ISP networks obtained from the Topology Zoo [28] dataset. The network configurations use OSPF, BGP, static routing constructs, and ECMP load-balancing which are all modeled by ARC. We refer to the datacenter networks as DC*i*, and the ISP topologies by their names from Topology Zoo. We also study two synthetic fat-tree [5] datacenter topologies: Fat6 (45 routers) and Fat8 (80 routers).

Link Capacities. We vary the link capacities to be either 40Gbps or 100Gbps picked randomly or dependent on topology structure (e.g., for fat-tree: core-aggregate links are 100Gbps while edge-aggregate links are 40Gbps).

Traffic Matrices (TM). To verify the datacenter networks on realistic input traffic characteristics, we sample the datacenter packet traces from the Fbflow dataset [41]. We use the gravity [40] model to generate traffic matrices for our datacenter and ISP networks. The traffic matrices describe the traffic between all edge-edge router pairs for the datacenter networks or all endpoint pairs for the ISP networks. We denote the maximum link utilization of our network with 0 failures as MLU_0. We run our experiments using the traffic matrices obtained from Fbflow or gravity which have the property that $MLU_0 \in [0.65, 0.8]$ [1]; Under no failures, the max link utilization in the network is less than 80%. Our traffic matrices model scenarios when the network is not overloaded when all links are active, but certain 1 or 2 link failures could potentially lead to some links getting overloaded. Unless otherwise mentioned, we bound the variation of an individual traffic class so that it cannot increase more than 50%, and we bound the total traffic variation summed over all traffic classes to 10% of the total network traffic.

9.1 Verifying Real Networks

We use QARC to detect if the datacenter (DC) and ISP networks experience link overload under $k = \{1, 2\}$ link failures. We generate 20 random traffic matrices (Fbflow for datacenter and gravity for ISP) and run QARC's verification. We report our findings in Table 3.

QARC finds link overload events for 1-link failures for 139 of the 200 networks. Moreover, 87 networks show link overload violations for more than 10 out of 20 TMs, indicating that these networks are quite susceptible to link overload events even under a single-link failure scenario. QARC likewise finds overload for over 90% of the ISP networks, compared to 50% for the datacenter networks, indicating that datacenter networks are less susceptible to link overload due to higher path diversity than ISP networks. Notice that, the number of networks experiencing violations decreases for 2-link compared to 1-link failures. This is because 2-link

[1] We extract matrices using gravity/fbflow and multiply each field of the matrix with a constant factor to obtain the desired MLU_0

Table 3. Networks which experience link overload for one or more TMs (>0%) and more than 10 of 20 TMs (>50%) at $k = 1$ and $k = 2$.

Class	Total	k=1 (>0%)	k=1 (>50%)	k=2 (>0%)	k=2 (>50%)
DC	114	56	20	52	23
ISP	86	83	67	82	63

failures disconnect more traffic classes; the number of active traffic classes reduces, and thus, links' utilizations do not exceed capacities.

Min Traffic Variation. QARC can be used to find the minimum total variation of traffic (each traffic class cannot increase by more than 50%) which can cause overload under 1-link and 2-link failure scenarios. Minimum variation can be used to judge how the network handles unexpected spikes in load under failures. For this experiment, we use traffic matrices such that $MLU_0 = 0.7$ (high utilization) and find the average minimum traffic variation. For our ISP and datacenter networks in Figure 5, we report the variation as a percentage of total network volume. We can observe that the ISP networks require lower variations (2-12%) to cause link overload, i.e., if the total traffic increases by 12%, the network is likely to be overloaded under failures. Meanwhile, datacenter networks require more traffic variation for links to get overloaded (2-40%), highlighting intrinsic robustness pertaining to meeting traffic demands and higher path diversity. Also, $k = 2$ requires lower variation than $k = 1$.

Effect of Traffic Matrices. We also study the effect of different types of traffic matrices causing link overloads. We consider 5 datacenter networks and use QARC to find link overload for $k = 1$ failure scenarios. We generate 100 TMs using both Fbflow and gravity models such that $MLU_0 = [0.65, 0.8]$ and bounded variation is 10%; in Table 4 we report the percentage of violations found in these networks for the different matrices. We observe that Fbflow matrices lead to fewer load violations than gravity matrices for some of the datacenter networks. This shows the impact of using QARC to verify if significant changes to traffic patterns can affect a network's load properties under failures.

Q1: QARC **finds network load violations under failures** in 70% of real datacenter and ISP networks, and ISP networks are more susceptible to link overload than datacenter networks. To the best of our knowledge, this is the first study of finding potential violations for control planes.

9.2 Verification Performance

We now evaluate the performance of QARC's algorithm for detecting link overload for the datacenter and ISP networks using 20 different traffic matrices with $MLU_0 \in [0.65, 0.8]$ for 1 and 2-link failure scenarios. The datacenter networks have on the order of 10 to order of 100 links (exact numbers hidden for confidentiality) and order of 100 traffic classes,

Table 4. Percentage overload violations for 1-link failures for datacenter networks with Fbflow and Gravity traffic matrices

Matrix	DC1 (<100)	DC2 (<100)	DC3 (<100)	Fat6 (216)	DC4 (<1000)
Fbflow	74%	72%	28%	100%	100%
Gravity	83%	96%	38%	100%	100%

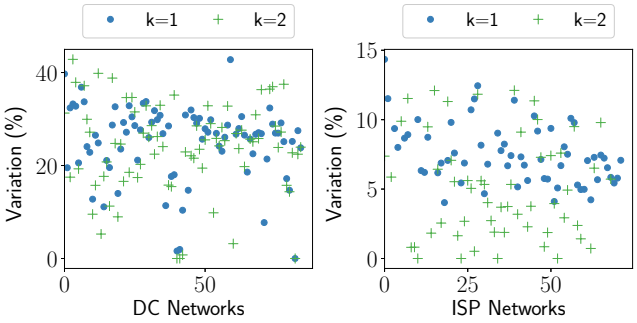


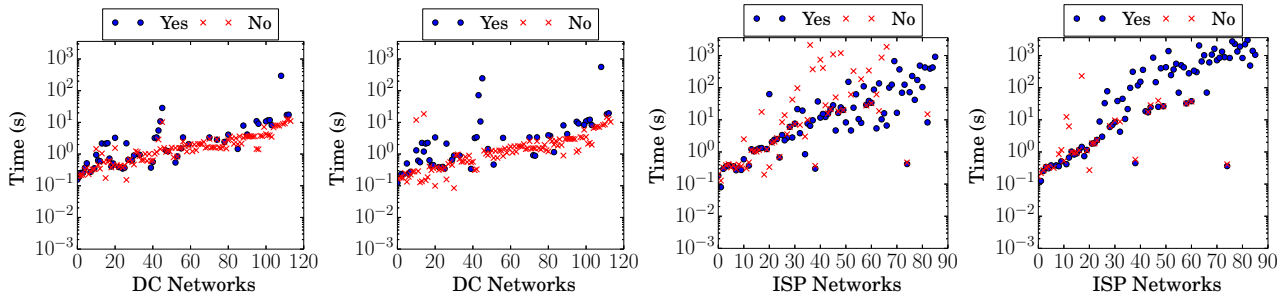
Figure 5. Variation % for different ISP and datacenter networks for 1 and 2 link failure scenarios.

and the ISP networks have 8-125 links and 10-2000 traffic classes. We use 5 nodes to run parallel verification and report the average time to successfully find violations (Yes) or verify no violations (No) for the networks.

Figure 6 shows the results. We observe that verification time for $k = 2$ is greater than $k = 1$ due to the larger search space. We sort the networks by number of network links and observe that network time in general increases with size of the network. Also, the ISP topologies have higher verification times compared to the datacenter networks (due to larger number of traffic classes). For most networks, QARC is able to verify if the network will experience link overload (Yes) or not (No) in under an hour (6.7% of the runs timed out due to the large number of links or traffic classes). With our choice of traffic matrices with $MLU_0 \in [0.65, 0.8]$, the times taken to find a violation or proving the absence of violations follow similar trends, indicating similar difficulty.

We also look at QARC's performance with increasing number of link failures. We consider 3 datacenter networks (DC2, DC4, Fat6) and one ISP network (Abilene) and use 50 random gravity matrices with $MLU_0 \in [0.65, 0.8]$ and 10% traffic variation, and verify for $k = [1, 4]$ failure scenarios. We present the median verification times in Table 5. As k increases, verification times generally increase (Fat-6 and DC4 experience a drastic increase at $k = 3$ and $k = 4$, respectively). For $k \leq 2$, QARC terminates within an hour for most instances (<1% instances timed out). QARC experiences more timeouts for $k = 3$ (7%) and $k = 4$ (25%) due to the exponential complexity of verification.

We now consider how QARC performs compared to naive enumeration. We fix the input traffic matrix, find the time



(a) k=1 **(b)** k=2 **(c)** k=1 **(d)** k=2

Figure 6. Verification time (log scale) sorted by network links for different WAN and datacenter networks for 1 and 2 link failure scenarios.

Table 5. Verification times (s) for $k = [1, 4]$ link failures.

Network(Links)	k=1	k=2	k=3	k=4
Abilene(28)	0.7	7.9	9.1	10.7
DC2 (<100)	1.4	17.5	17.1	20.1
Fat6 (216)	3.3	2.8	1,550.6	1,357.7
DC4 (<1000)	9.6	22.1	81.2	1,496.0

taken to verify one failure and extrapolate by the number of failures. We consider a 5-node parallelism, so enumeration is spread across 5 machines equally. For a network with >100 links, the naive enumeration time versus median QARC time for different failure scenarios are: (a) $k = 1$: 12.8s vs 9.6s, (b) $k = 2$: 1,017s vs 22s, (c) $k = 3$: 14.8hrs vs 81s, and (d) $k = 4$: 584hrs vs 0.4hrs . While for 1-link failures, the naive enumeration is comparable, QARC has significant speedup for $k \geq 2$. Moreover, verification with non-zero traffic matrix variation cannot be performed by enumeration, as there are infinitely many rational-value traffic matrices to consider.

Q2: QARC **can verify network overload for medium-sized datacenter and ISP networks** for different failure scenarios in under an hour.

9.3 QARC Optimizations

We now evaluate how the optimizations presented in §6 improve QARC's performance. We consider 3 datacenter (DC2, DC4, Fat6) and one ISP network (Abilene) and use 50 random gravity matrices with $MLU_0 \in [0.65, 0.8]$ and no traffic variation, and verify for $k = 1$ failures.

ETG Minimization. Table 6 reports the speedup obtained by ETG minimization (speedup = time without minimization/ time with minimization) and edge reductions for the networks. For the bigger networks, the speedup in verification is significantly larger, aiding in making QARC's verification more tractable. When verifying the Fat6 and DC4 network with $k = 1$, we are able to reduce 86% and 95% of the ETG edges respectively, achieving significant speedups of 69× and 18× for Fat6 and DC4. The ETG minimization phase is quick, taking under 5 seconds for all the networks.

Table 6. Speedups due to optimizations for $k = 1$ failures.

Optimizations	Abilene(28)	DC2 (<100)	Fat6 (216)	DC4 (<1000)
Edges Removed	54%	67%	86%	95%
Minimize Speedup	2.3	4.5	69	18
Parallel Speedup	1.3	1.3	5.0	9.5
Partition Classes	68%	50%	42%	36%
Partition Speedup	1.00	1.19	1.38	1.52

Parallelization. We now consider the speedup achieved by parallelizing the verification (with optimal partitions) over verification run on a single node. We run verification on 5 nodes and compare the performance with 1-node verification. In the parallel scenario, we terminate verification if any one node finds a link load violation, otherwise we wait till all nodes report no violations (with a timeout of 1 hour). Table 6 shows the verification speedup achieved due to parallelization. For the bigger networks (Fat6 and DC4), we achieve average speedup of over 5× and 10× over the single node case; thus, QARC parallelization further improves the tractability of verification.

Partitioning. Finally, we evaluate the speedup due to the optimal partitioning scheme which minimizes the number of traffic classes in each of the 5 partitions. We compare the performance of 5-node optimal partition verification to to a 5-node random partitioning scheme which randomly divides the links in equal sized partitions. For this experiment, we pre-compute the optimal partitioning (not included in verification time) and compare verification using the optimal and random naive partitions on 5 nodes. Table 6 shows the verification speedup achieved by the optimal versus naive partitions. The speedup achieved for DC2, DC4 and Fat6 is about 20-50%. We report the percentage of traffic classes in the optimal partitions (Partition Classes) for each network in Table 6; we are able to reduce more than half of the traffic classes in each partition for DC2, DC4 and Fat6.

Q3: QARC **optimizations speed up verification** significantly for different networks and failure scenarios, with the

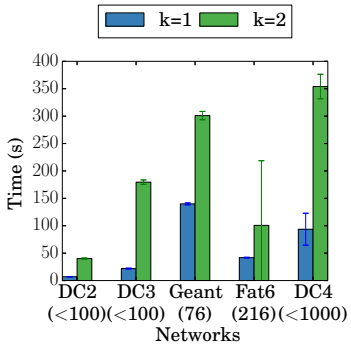


Figure 7. Capacity upgrade computation time

most benefits coming from ETG minimization and parallel verification for the bigger Fat6 and DC4 networks.

9.4 Upgrade Performance

We use QARC to generate the minimal link capacity additions required to prevent link overload under 1-link and 2-link failures. For this experiment, we consider traffic matrices with $MLU_0 \in [0.8, 0.95]$ and zero variation. Similar to verification, we parallelize the phase of computing the new link capacities to 5 nodes. We report the median upgrade time for 5 different networks in Figure 7. QARC is able to compute new link capacities for the networks in under 5 minutes using the 5 nodes. In our runs, QARC requires changing capacities of order of 10 links (Fat6 repair has the highest number of links whose capacity has to be increased). Finding the repair for all 2-link failure scenarios takes more time than 1-link scenarios due to the larger search space.

Q3: QARC **can upgrade link capacities to prevent network overload under failures** for different networks in under 400 seconds.

10 Other Related Work

Config. verification/repair. Our work complements configuration analysis tools [4, 6, 7, 13, 15, 16, 19, 20, 23, 44, 47, 49] that focus on qualitative properties; to the best of our knowledge, we are the first to show how to reason about quantitative properties in networks with distributed control planes. Our ETG minimization approach is inspired by the idea of surgery proposed by Plotkin et. al [38] to slice the network and headers to speed up reachability verification. The key difference is that QARC identifies network components which are not traversed under different failure scenarios, while Plotkin et. al do not verify reachability under failures.
Quantitative properties. ProbNetKAT [18, 43] supports probabilistic routing behavior and can answer congestion and latency queries about the network using a link failure probability distribution. ProbNetKAT cannot express complex routing strategies—e.g., distributed routing protocols such as OSPF—that recompute new paths under failures based on the global network state. Chang et. al. [11] propose a framework to validate network design under failures and uncertain demands. Specifically, the input is a global routing strategy that can adapt to the current network topology to optimize the max link utilization (MLU); examples are multi-commodity flow (which finds optimal network paths) and MPLS tunneling (best path chosen from a set of prespecified tunnels). The system finds the worst case (max) value that MLU can achieve under a set of failure scenarios/-traffic demands when the adaptive routing strategy is in use by relaxing a non-linear formulation to a linear program.

The routing in distributed control planes does not seek to minimize the global objective of maximum link utilization (MLU), and, thus, cannot be reasoned about directly using this framework. QARC leverages ARC to reason about real router configurations deployed in different real-world networks, while Chang et. al's framework deals with the abstract routing strategies like multi-commodity flow. Moreover, our formulation is sound, i.e., it will find a failure scenario where link overload happens if one exists, however Chang et. al's framework solves a relaxed formulation, so cannot provide soundness guarantees for all routing strategies.
Traffic Engineering. Our work is orthogonal to traffic engineering (TE) works [17, 30, 32, 48] that develop/configure routing strategies to react to uncertain traffic demands or failure scenarios. In a sense, TE systems solve a narrow "synthesis" problem, whereas we focus on analyzing the joint impact of the routing in use in a network, input traffic characteristics, and arbitrary failures on the network's current link bandwidths, i.e., *verification*. Also, the scope of TE systems is narrow in the sense that they only consider generating routes that satisfy quantitative properties, but ignore effects of such routes on qualitative properties.

11 Conclusion and Future Work

We presented QARC, a control plane abstraction to support verification of network link overload under different failure scenarios. QARC can efficiently verify complex network configurations thanks to its new Mixed-Integer-Programming encoding. QARC can also be used for determining which link capacities need to be upgraded to prevent load violations. We use QARC to show network load violations can occur in existing datacenter and ISP networks. For future work, QARC can be extended to support verification of other quantitative properties like latencies and rate-limits. Finally, to tackle general repair for quantitative properties, joint changes to the network control plane and link capacities are required.
Acknowledgments. We thank the anonymous reviewers, and our shepherd Arjun Guha for their insightful feedback and suggestions. We also thank the network engineers at Microsoft who provided us with real network configurations. This work is supported by the National Science Foundation grants CNS-1637516, CNS-1763512, CNS-1763871 and CCF-1750965.

References

[1] 2018. Batfish. https://github.com/batfish/batfish.

[2] 2019. ARC. http://bitbucket.org/uw-madison-networking-research/arc.

[3] 2019. Cisco IOS Configuration Fundamentals Command Reference. http://cisco.com/c/en/us/td/docs/ios/fundamentals/command/reference/cf$_b$ook.html.

[4] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: Fast Multilayer Network Verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 201–219. https://www.usenix.org/conference/nsdi20/presentation/abhashkumar

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 63–74.

[6] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. ACM, New York, NY, USA, 155–168. https://doi.org/10.1145/3098822.3098834

[7] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitu Padhye, and David Walker. 2016. Don't Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations. In *Proceedings of the ACM SIGCOMM 2016 Conference on SIGCOMM (SIGCOMM '16)*.

[8] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2017. Network configuration synthesis with abstract topologies. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 437–451.

[9] Theophilus Benson, Aditya Akella, and David Maltz. 2009. Unraveling the Complexity of Network Management. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation* (Boston, Massachusetts) *(NSDI'09)*. USENIX Association, Berkeley, CA, USA, 335–348. http://dl.acm.org/citation.cfm?id=1558977.1559000

[10] Theophilus Benson, Aditya Akella, and Aman Shaikh. 2011. Demystifying Configuration Challenges and Trade-offs in Network-based ISP Services. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. ACM, New York, NY, USA, 302–313. https://doi.org/10.1145/2018436.2018471

[11] Yiyang Chang, Sanjay Rao, and Mohit Tawarmalani. 2017. Robust Validation of Network Designs under Uncertain Demands and Failures. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 347–362. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/chang

[12] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 14th International Conference*. 337–340. https://doi.org/10.1007/978-3-540-78800-3$_2$4

[13] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2017. Network-wide Configuration Synthesis. In *29th International Conference on Computer Aided Verification, Heidelberg, Germany, 2017 (CAV'17)*.

[14] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2018. NetComplete: Practical Network-Wide Configuration Synthesis with Autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 579–594. https://www.usenix.org/conference/nsdi18/presentation/el-hassany

[15] Seyed K Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient Network Reachability Analysis using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 217–232.

[16] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A general approach to network configuration analysis. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 469–483.

[17] Bernard Fortz and Mikkel Thorup. 2000. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM 2000. Nineteenth annual joint conference of the IEEE computer and communications societies. Proceedings. IEEE*, Vol. 2. IEEE, 519–528.

[18] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic netkat. In *European Symposium on Programming Languages and Systems*. Springer, 282–309.

[19] Aaron Gember-Jacobson, Aditya Akella, Ratul Mahajan, and Hongqiang Harry Liu. 2017. Automatically repairing network control planes using an abstract representation. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 359–373.

[20] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. ACM, New York, NY, USA, 300–313. https://doi.org/10.1145/2934872.2934876

[21] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. *Fast Control Plane Analysis Using an Abstract Representation.* Technical Report. University of Wisconsin-Madison.

[22] Aaron Gember-Jacobson, Wenfei Wu, Xiujun Li, Aditya Akella, and Ratul Mahajan. 2015. Management Plane Analytics. In *Proceedings of the 2015 Internet Measurement Conference* (Tokyo, Japan) *(IMC '15)*. ACM, New York, NY, USA, 395–408. https://doi.org/10.1145/2815675.2815684

[23] Nick Giannarakis, Devon Loehr, Ryan Beckett, and David Walker. 2020. NV: An Intermediate Language for Verification of Network Control Planes. In *PLDI*. ACM.

[24] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) *(SIGCOMM '11)*. ACM, New York, NY, USA, 350–361. https://doi.org/10.1145/2018436.2018477

[25] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (Barcelona, Spain) *(SIGCOMM '09)*. ACM, New York, NY, USA, 51–62. https://doi.org/10.1145/1592568.1592576

[26] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM* (Hong Kong, China) *(SIGCOMM '13)*. ACM, New York, NY, USA, 15–26. https://doi.org/10.1145/2486001.2486012

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

[28] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. 2011. The Internet Topology Zoo. *Selected Areas in Communications, IEEE Journal on* 29, 9 (october 2011), 1765 –1775. https://doi.org/10.1109/JSAC.2011.111002

[29] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C. Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, Mathieu Robin, Aspi Siganporia, Stephen Stuart, and Amin Vahdat. 2015. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. ACM, New York, NY, USA, 1–14. https://doi.org/10.1145/2785956.2787478

[30] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-Oblivious

Traffic Engineering: The Road Not Taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 157–170. https://www.usenix.org/conference/nsdi18/presentation/kumar

[31] Dave Lenrow. 2015. Intent: What. Not How. http://opennetworking.org/?p=1633&option=com_wordpress&Itemid=471.

[32] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. 2014. Traffic Engineering with Forward Fault Correction. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. ACM, New York, NY, USA, 527–538. https://doi.org/10.1145/2619239.2626314

[33] Gary Scott Malkin. 1998. *RIP Version 2*. STD 56. RFC Editor. http://www.rfc-editor.org/rfc/rfc2453.txt http://www.rfc-editor.org/rfc/rfc2453.txt.

[34] David A. Maltz, Geoffrey Xie, Jibin Zhan, Hui Zhang, Gísli Hjálmtýsson, and Albert Greenberg. 2004. Routing Design in Operational Networks: A Look from the Inside. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Portland, Oregon, USA) *(SIGCOMM '04)*. ACM, New York, NY, USA, 27–40. https://doi.org/10.1145/1015467.1015472

[35] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. 2008. Characterization of failures in an operational IP backbone network. *IEEE/ACM Transactions on Networking (TON)* 16, 4 (2008), 749–762.

[36] John Moy. 1998. *OSPF Version 2*. STD 54. RFC Editor. http://www.rfc-editor.org/rfc/rfc2328.txt http://www.rfc-editor.org/rfc/rfc2328.txt.

[37] Gurobi Optimization. 2019. Gurobi. http://www.gurobi.com/.

[38] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling Network Verification Using Symmetry and Surgery. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg, FL, USA) *(POPL '16)*. ACM, New York, NY, USA, 69–83. https://doi.org/10.1145/2837614.2837657

[39] Yakov Rekhter, Tony Li, and Susan Hares. 2005. *A border gateway protocol 4 (BGP-4)*. Technical Report.

[40] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. 2002. Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment* (Marseille, France) *(IMW '02)*. ACM, New York, NY, USA, 91–92. https://doi.org/10.1145/637201.637213

[41] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (London, United Kingdom) *(SIGCOMM '15)*. ACM, New York, NY, USA, 123–137. https://doi.org/10.1145/2785956.2787472

[42] Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. 2018. RADWAN: Rate Adaptive Wide Area Network. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. ACM, New York, NY, USA, 547–560. https://doi.org/10.1145/3230543.3230570

[43] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. 2017. Cantor Meets Scott: Semantic Foundations for Probabilistic Networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages* (Paris, France) *(POPL 2017)*. ACM, New York, NY, USA, 557–571. https://doi.org/10.1145/3009837.3009843

[44] Kausik Subramanian, Loris D'Antoni, and Aditya Akella. 2018. Synthesis of Fault-Tolerant Distributed Router Configurations. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 22 (April 2018), 26 pages. https://doi.org/10.1145/3179425

[45] Kausik Subramanian, Loris DâĂŹAntoni, and Aditya Akella. 2017. Genesis: Synthesizing Forwarding Tables in Multi-Tenant Networks. *SIGPLAN Not.* 52, 1 (Jan. 2017), 572âĂŞ585. https://doi.org/10.1145/3093333.3009845

[46] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. 2016. Robotron: Top-down network management at facebook scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 426–439.

[47] Anduo Wang, Limin Jia, Wenchao Zhou, Yiqing Ren, Boon Thau Loo, Jennifer Rexford, Vivek Nigam, Andre Scedrov, and Carolyn Talcott. 2012. FSR: Formal Analysis and Implementation Toolkit for Safe Interdomain Routing. *IEEE/ACM Trans. Netw.* 20, 6 (Dec. 2012), 1814–1827. https://doi.org/10.1109/TNET.2012.2187924

[48] Hao Wang, Haiyong Xie, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert Greenberg. 2006. COPE: Traffic Engineering in Dynamic Networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Pisa, Italy) *(SIGCOMM '06)*. ACM, New York, NY, USA, 99–110. https://doi.org/10.1145/1159913.1159926

[49] Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. 2016. Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Amsterdam, Netherlands) *(OOPSLA 2016)*. ACM, New York, NY, USA, 765–780. https://doi.org/10.1145/2983990.2984012

[50] 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* (Amsterdam, The Netherlands) *(EuroSys '14)*. ACM, New York, NY, USA, Article 5, 14 pages. https://doi.org/10.1145/2592798.2592803

[51] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. 2017. Understanding and Mitigating Packet Corruption in Data Center Networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. ACM, New York, NY, USA, 362–375. https://doi.org/10.1145/3098822.3098849