Reasoning about Network Traffic Load Property at Production Scale

Ruihan Li$^{\mathbb{P}\mathbb{A}}$, *Fangdan Ye*$^{\mathbb{A}}$, *Yifei Yuan*$^{\mathbb{A}}$, *Ruizhen Yang*$^{\mathbb{A}}$, *Bingchuan Tian*$^{\mathbb{A}}$, *Tianchen Guo*$^{\mathbb{A}}$, *Hao Wu*$^{\mathbb{A}}$, *Xiaobo Zhu*$^{\mathbb{A}}$, *Zhongyu Guan*$^{\mathbb{A}}$, *Qing Ma*$^{\mathbb{A}}$, *Xianlong Zeng*$^{\mathbb{A}}$, *Chenren Xu*$^{\mathbb{P}}$, *Dennis Cai*$^{\mathbb{A}}$, *Ennan Zhai*$^{\mathbb{A}}$

$^{\mathbb{P}}$*Peking University* $^{\mathbb{A}}$*Alibaba Cloud*

Abstract

This paper presents JINGUBANG, the first reported system for checking network traffic load properties (*e.g.*, if any link's utilization would exceed 80% during a network change) in a production Wide Area Network (WAN). Motivated by our network operators, JINGUBANG should meet three important requirements: (**R1**) comprehensive support for complex traffic behavior under BGP, IS-IS, policy-based routes (PBR), and segment routes (SR), (**R2**) reasoning on traffic load of billions of flows across a period of time, (**R3**) real-time failure-tolerance analysis. These requirements pose challenges in modeling the complex traffic behavior and maintaining the checking efficiency. JINGUBANG has successfully addressed these challenges. First, we propose the traffic distribution graph (or TDG), capable of modeling equal-cost multipath (ECMP), packet rewriting, and tunneling, introduced by BGP/IS-IS, PBR, and SR, respectively. Second, we design an algorithm based on TDG to simulate traffic distribution for billions of flows across a time period both efficiently and accurately. Third, JINGUBANG proposes an incremental traffic simulation approach that first computes an incremental TDG and then simulates only the *differential* traffic distribution, avoiding the need to simulate the entire network traffic distribution from scratch. JINGUBANG has been used in the daily checking of our WAN for more than one year and prevented service downtime resulting from traffic load violations.

1 Introduction

Alibaba Cloud serves over one billion customers with its services including cloud computing, search, and video. To support these services, we operate a global Wide Area Network (WAN) infrastructure to interconnect tens of data centers. Our WAN is a traditional, distributed network, rather than an SDN network. This WAN maintains hundreds of routers and forwards a huge amount of service traffic by combining diverse protocols including BGP, IS-IS, policy-based routes (PBR), and segment routes (SR). According to our recent three-year records, more than 90% of outages caused by misconfigurations on our WAN were related to traffic load violations.[1]

Checking whether network traffic load meets our variety of specifications, therefore, is vital to the availability and reliability of our WAN. For example, in the scenario of planned network changes (*e.g.*, updating configurations and upgrading network routers), our network operators need to ensure that no link would be overloaded at any point during the entire planned change time window. As another example, our operators need to check whether any traffic load property (*e.g.*, no drastic traffic load increase on any link) may be violated if given links fail. Conventional techniques (*e.g.*, traffic engineering [4, 11, 17, 19, 25, 26]) do not help since they mainly focus on optimizing and controlling traffic load; however, what we need is a system capable of answering our queries about specified traffic load properties.

Requirements. We decided to build a system to model the traffic behavior and *proactively* check traffic load violations in our WAN. By surveying our operators, a practical checking system must meet the following requirements.

R1: Comprehensive protocol support. The checker should support comprehensive traffic behavior under BGP, IS-IS, PBR, SR, and static routes. Our WAN uses BGP and IS-IS for routing; our traffic scheduling and engineering heavily rely on PBR and SR, due to their precise traffic control and low operation cost. For example, SR can specify forwarding paths on a per-hop basis, while PBR can identify QoS and classify traffic into different classes.

R2: Reasoning on traffic load for billions of flows in a period of time. The checker should support the traffic load reasoning not only at a single time point, but also during a period of time. In a network change scenario, the change time window may last multiple hours; thus, our operators need to check whether the intended traffic load properties can hold during the entire change window. In addition, billions of flows can appear during the time period; thus, the checker must be scalable to reason on the traffic load of billions of flows.

R3: Real-time failure-tolerance analysis. Our operations frequently run failure-tolerance analysis to check if a set of failed routers and links would cause traffic load violations. The checker should offer an efficient what-if analysis to meet the real-time requirement of our operators.

State of the arts. No prior work, nevertheless, can simultaneously satisfy all the above requirements. Specifically, current network verification systems have been focused on checking reachability properties (*e.g.*, if packets/route advertisements sent from a router A can reach another router B) in terms of control plane [1, 2, 10, 12, 14, 15, 22, 31, 32, 34, 39–42] and data plane [3, 18, 20, 21, 23, 24, 28, 30, 35, 37]; thus, they are unable to reason about traffic load properties in the network.

A recent effort, QARC [36], closest to our goal, proposes a verification approach to checking whether links may be overloaded under any failure up to a given degree (*i.e.*, *k*-failure

[1] Since Hoyan [41] was used three years ago, Hoyan has prevented most misconfigurations causing routing reachability errors in our network. The remaining misconfiguration-related outages, in recent three years, happened tens of times a year and accounted for about 30% of all our network outages.

tolerance reasoning for network traffic load). QARC, however, cannot meet our requirements. First, QARC's encoding is based on the shortest path with flow quantities. It is nontrivial to extend the encoding to support PBR and SR, which introduces traffic behavior beyond shortest-path-forwarding, *e.g.*, packet rewriting and tunneling. Second, the verification algorithm of QARC mainly focuses on traffic load reasoning at a single time point. Extending the algorithm to *efficient* verification of traffic load in a period of time is not straightforward. Finally, QARC's main focus is k-failure tolerance typically with small k, it is reported that it cannot scale to a larger number of failed links/routers (*e.g.*, $k > 5$) [36].

Our approach: JINGUBANG.[2] In this paper, we present JINGUBANG, the first system for checking network traffic load properties for WANs. JINGUBANG simultaneously achieves the above-mentioned requirements proposed by our operators. In essence, JINGUBANG is a system that can simulate traffic distribution (*i.e.*, traffic load on each link) for traffic load property reasoning. Specifically, JINGUBANG takes as inputs (i) the network topology and routers' configurations, (ii) IP prefixes advertised into the network, and (iii) multiple traffic snapshots each recording the traffic information of a set of flows entering the network at a time point, then accurately simulates the traffic distribution, and finally checks whether it meets the given traffic load property.

Building JINGUBANG requires us to address three challenges (corresponding to the three requirements, respectively). First, the traffic behavior under BGP, IS-IS, PBR, and SR is complex. In particular, SR relies on *tunneling* and PBR offers fine-grained flow processing such as *packet rewriting*. In addition, routing protocols (*e.g.*, BGP, IS-IS) enforce protocol-based *equal-cost multipath* (ECMP), where traffic is load-balanced first between BGP next hops and then between IS-IS next hops. To comprehensively model the traffic behavior under those protocols, we introduce the *traffic distribution graph* (or TDG), translating the traffic behavior of different protocols into a uniform traffic forwarding representation. TDG can represent not only destination-based forwarding behavior (like BGP and IS-IS) with ECMP but also tunneling and packet rewriting used in SR and PBR. (§4.1)

The second challenge is efficiently reasoning on traffic load for billions of flows in a period of time. In our operation, the time period to be checked typically lasts several hours, involving a huge number of time points (1 minute for each in our settings). A strawman solution is to generate many TDGs, each corresponding to a time point, and then run traffic simulation on each of all these TDGs. Such a solution is inefficient. We observe that traffic flows across different time points highly overlap. Driven by this insight, we propose an approach that constructs only one TDG with the union of all the traffic flows, enabling us to simulate traffic distribution across a period of time within one run. To further improve the efficiency of handling billions of flows, we propose two optimizations based on sampling and equivalence classes which significantly reduce the number of flows to be considered while still maintaining accuracy. (§4.2-§4.4)

Third, daily operation requires a large number of what-if analysis on network link (or router) failures. Straightforwardly simulating traffic for each link (or router) failure request cannot meet the real-time requirement of our operators. To support real-time failure-tolerance analysis, JINGUBANG proposes an incremental traffic simulation approach that first computes an incremental TDG and then simulates only the *differential* traffic distribution, avoiding the need to simulate the entire network traffic distribution from scratch. (§5)

Scope. JINGUBANG operates under two key settings: (i) a time point is defined as a minute for the purpose of checking traffic load properties, (ii) traffic is assumed to be distributed evenly or based on pre-configured weights across ECMP paths. Thus, the examination of micro-burst congestion [45] and ECMP unfairness falls outside the scope of this paper, as these considerations are orthogonal to our operators' requirements. [3] In addition, JINGUBANG takes as input flows entering our WAN and external routes advertised into our WAN, and will suffer from inaccuracy if traffic measurement is imprecise [5, 38] or route monitoring is incomplete (§7).

Real-world deployment. JINGUBANG has been used to check network traffic load in our WAN and successfully prevented service downtime in operations such as network changes and failure-tolerance analysis. We share our experience with JINGUBANG in §6 and evaluate JINGUBANG in §7.

Ethics. This work does not raise any ethical issues.

2 Background and Motivation

This section starts by describing the background of our WAN (§2.1), and then presents our motivation (§2.2).

2.1 Background: Our Production WAN

Alibaba Cloud operates a private WAN infrastructure that interconnects all its data centers and peers with external ISPs, serving both traffic internal to Alibaba Cloud and traffic between Alibaba Cloud and external networks. Our WAN is a distributed setting without a centralized SDN controller. By Jan 2023, this WAN has tens of autonomous systems (ASes), nearly a thousand routers, and tens of thousands of links, where each router has a forwarding table with millions of entries. [4] Our operators conduct hundreds of network changes per week, with frequent changes such as link capacity expansions and IP prefix publications happening a dozen times per day. Our WAN uses BGP (including eBGP and iBGP), IS-IS, static routes, SR, and PBR to route our network traffic.

[2] In ancient Chinese mythology, Jingu Bang is a tool used by Yu the Great to measure water levels during his efforts to control floods. Therefore, we named our network traffic load checking system JINGUBANG.

[3] In >99% of our WAN links, the top flow consumes <1% bandwidth. With a large number of flows and no elephant flows, flow-level ECMP can be approximated by evenly distributing flow volumes across ECMP paths.

[4] We omit absolute numbers for the confidentiality reason.

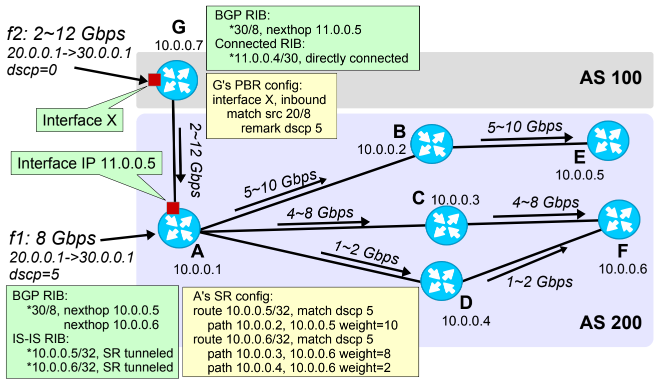


Figure 1: Motivating example. The green boxes represent RIBs for router A and G, respectively. Two yellow boxes are G's PBR configuration and A's SR configuration, respectively. In the given period of time (say one hour), the traffic rate of f_2 fluctuates between 2 Gbps and 12 Gbps, while f_1 keeps 8 Gbps. G belongs to AS 100, and A~F belong to AS 200. This example shows how BGP, IS-IS, PBR, and SR work together to distribute the traffic in our WAN.

PBR and SR. Unlike routing protocols (*e.g.*, IS-IS and BGP), PBR and SR are non-destination-based forwarding techniques. A PBR policy can match packets by their destination addresses, source addresses, and DSCP values; for matched packets, it can set next hops or modify packet fields (*e.g.*, DSCP). SR forwards the packets along one or more explicit paths built on Multiprotocol Label Switching (MPLS) or IPv6 stacks (*i.e.*, SRv6); different paths may have different routing weights. The mainstream SR used in our WAN is SRv6.

Traffic monitoring. Our WAN employs a large-scale traffic monitoring system that collects detailed traffic information from the network via Netflow [8] and sFlow [29] at a sampling rate of 8192. The monitoring system records the information of traffic flows entering each interface of routers, including the value of each field (*e.g.*, the IP-port 5-tuple), the timestamp of report time, and the total traffic volume of each flow sent since the last report time. The system generates a *traffic snapshot* per minute, recording the traffic flows (entering our WAN) and their rates in that minute. JINGUBANG receives 60 one-minute traffic snapshots every hour.

2.2 Motivations and Goals

Motivating example. Figure 1 shows a small but illustrative example of a traffic distribution situation in our operation. Suppose two traffic flows f_1 and f_2 enter the network by router A and G, respectively. In the given period of time, the traffic rate of f_1 keeps 8 Gbps, while f_2's rate fluctuates between 2~12 Gbps. When receiving f_2, G sets f_2's DSCP to 5 according to G's PBR configuration; then, G routes f_2 to A based on its eBGP route. By receiving f_1 and f_2 (both with DSCP=5), A performs the longest prefix matching for the flow based on A's routing information base (RIB), identifying two iBGP next hops E and F. The traffic rate from A to E

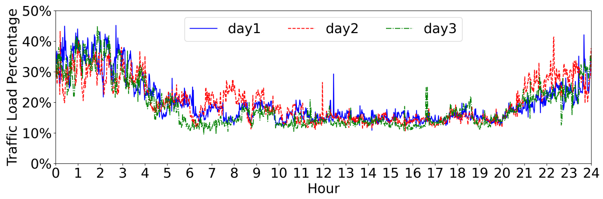


Figure 2: Traffic load fluctuation on one of the links in our WAN during randomly sampled three days.

and F is equal, *i.e.*, 10~20 Gbps/2=5~10 Gbps, because of the ECMP of iBGP. Router A looks up the direct next hop to E and F by checking A's SR configuration. According to A's SR configuration in Figure 1, A distributes the traffic into three paths ($A \to B$, $A \to C$, and $A \to D$) in traffic rates 5~10 Gbps, 4~8 Gbps, and 1~2 Gbps, respectively. Note that $A \to C$ and $A \to D$ forward traffic rates in a ratio of 4:1 due to weights 8 and 2 specified in A's SR configuration.

Requirements from our operators. It is hard to ensure whether the traffic meets the intended properties (say "whether all traffic on $C \to F$ keeps the rate lower than 8 Gbps even if $B \to E$ fails") in a real WAN which is much larger than the above-mentioned motivating example. By surveying our operators, Table 1 lists representative scenarios where our operators need to check traffic load properties. There are mainly two types of analysis: what-if analysis and auditing. The former mainly focuses on network changes and specified failure scenarios, while the latter checks a network without any change. As shown in Table 1, our operators want to check traffic not only at a single time point but also over a period of time, because the network traffic fluctuates significantly during a time period. Figure 2 shows the traffic fluctuation occurred on one link in our WAN in randomly sampled three days. We can observe that the network traffic load fluctuates between 10% and 50% of this link bandwidth during these three days. Furthermore, as shown in Table 1, we should support protocols including BGP, IS-IS, SR, and PBR, since our WAN employs these types of protocols to forward traffic.

Traffic load properties of interest. The network traffic load property our operators mainly focus on is whether the traffic load/rate (in absolute number or percentage) or the change of it on a given set of links is above or below the specified threshold, such as "the traffic rate on the given set of links should be lower than 3 Gbps" and "given a network change, the traffic load on a specified link should not increase by 40% during the change". The *prop* in Figure 4 shows the formal definition of properties of interest (detailed in §3.3).

Representative scenarios for what-if analysis. We now describe two representative scenarios (network change analysis and failure tolerance analysis) to illustrate why we need what-if analysis in our WAN.

Network change analysis. Network changes, *e.g.*, changing configuration and updating routers, are one of the most important operations. In a typical network change, the operator makes a step-by-step change plan including each atomic step

Table 1: The main specifications our operators express in their operations.

Type	Name	Specification	Protocols
What-if Analysis	Change analysis	Whether the traffic load property holds in the network change time window	BGP/IS-IS/SR/PBR
	Change safety	Whether the network change would cause a traffic violation for a specified period of time	BGP/IS-IS/SR/PBR
	Failure tolerance	Whether the traffic load property holds if a specified set of routers or links becomes unavailable	BGP/IS-IS/SR/PBR
Auditing	Daily auditing	Whether the traffic load property holds for a given period of time	BGP/IS-IS/SR/PBR

to be executed (*e.g.*, changing routing policy, turning off interfaces, and cutting off the connection with neighbor routers), and reserves a network change time window (*e.g.*, 1am-7am on the upcoming Thursday). Because network changes are prone to failures [27,41], our operators want to check whether the network change would cause traffic load violations at any point in the change time window (note that network traffic fluctuates significantly during a period of time as shown in Figure 2) based on the traffic monitored in similar time periods (*e.g.*, 1am-7am on last Thursday).

Failure tolerance analysis. Another key type of operations is to check whether the load property holds if a specified set of links (or routers) becomes unavailable, due to the following scenarios: (i) router upgrading/replacement, and (ii) link maintenance. In the above cases, the router or link should be specified as unavailable, and the operators want to know whether the "failure" may cause any property violation.

Non-goal. JINGUBANG can check failure tolerance under a given set of failed routers or links, but cannot check arbitrary k-failure tolerance (*i.e.*, whether the property holds if arbitrary k links fail). While QARC [36] presented a good first step toward checking arbitrary k-failure tolerance, our operation experience found that designing a production-scale k-failure-tolerance checking for quantitative properties is extremely hard. This is because such a checker needs to support more detailed protocol properties, *e.g.*, BGP add-path and SR tunnels, which significantly expands the encoding space. We leave scalable k-failure traffic load checking to future work.

2.3 Related Work

For the traffic load property reasoning, why the state-of-the-art efforts do not help? Existing verification systems [2, 3, 10, 12, 14, 15, 18, 20, 21, 23, 24, 28, 30–32, 34, 35, 37, 39–43] mainly focus on checking qualitative properties such as packet reachability (*e.g.*, "whether a given route sent from router A can reach another router B"). To the best of our knowledge, except QARC [36], none of the prior work is able to check quantitative properties related to the traffic load.

Traffic engineering. The major goal of traffic engineering (TE) is to optimize network traffic and resource usage [4, 11, 17, 19, 25, 26]. Our goal, on the contrary, is to check whether the network implementation (*e.g.*, network topology, router configurations, and injected routes) satisfies the high-level traffic load properties. In particular, Alibaba Cloud uses a hybrid approach to realizing the TE optimization objectives via reconfiguring routers (*e.g.*, setting up SR tunnels) and injecting routes (*e.g.*, advertising BGP routes). However, they are unable to answer what-if or auditing queries about traffic

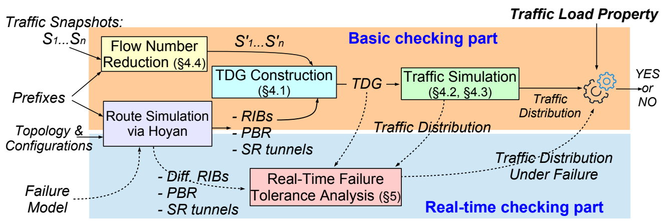


Figure 3: JINGUBANG's workflow overview. Solid arrows denote the basic traffic property checking process; dashed arrows mean the failure-tolerance checking process.

properties as shown in Table 1. In addition, our work can also be used in non-TE scenarios. For example, when a router needs to be updated from vendor A to B, JINGUBANG can check that the new configuration under vendor B is correct in that no traffic load properties are violated.

Network emulation. Network emulators, *e.g.*, CrystalNet [27], run the real control plane software in emulated environments and generate the corresponding forwarding behaviors for validation purposes. We do not choose emulation due to two issues. First, the emulator needs vendors to provide their router firmware within virtual machines or containers. However, we found it practically hard to get such support from all vendors for all router models in our WAN. Second, running real router software requires a large number of computing resources (*e.g.*, $100 per hour for emulating just one data center [27]). As routers in WAN typically have much more sophisticated firmware than data center switches, emulating a WAN can be even more expensive. Therefore, emulation is useful but not a good choice for our purpose.

Quantitative property analysis. Our focus on traffic properties is also different from the work on probabilistic analysis of the network control plane [16, 34, 44] and data plane [13, 33]. We check traffic at the network implementation level, which is different from the work [6, 46] that validates network designs.

3 JINGUBANG Overview

We built JINGUBANG, the first system for checking traffic load properties in production. JINGUBANG meets the requirements specified by our operators in §1.

3.1 JINGUBANG Workflow

Figure 3 shows the workflow of JINGUBANG. At a high level, JINGUBANG takes as inputs (i) the network topology and routers' configurations, (ii) IP prefixes advertised into the network, and (iii) multiple traffic snapshots each recording a set of traffic flow information at a time point (say one minute), then accurately simulates the *traffic distribution* (defined later), and finally checks whether the distribution meets

Table 2: Important terminologies.

Terms	Meaning
flow	A traffic flow, defined as $\langle$*dst*, *src*, *dscp*$\rangle$
located flow	A traffic flow at a particular interface defined as $\langle$*dst*, *src*, *dscp*, *interface*, *traffic volume*$\rangle$
traffic snapshot	A set recording all located flows entering the network at the given time point
traffic distribution	A vector recording the traffic load of each link in the network at the given time point

the specified traffic load property. For failure-tolerance analysis, JINGUBANG takes an additional failure model as input, which specifies a collection of unavailable routers or links.

As shown in Figure 3, we have two parts in JINGUBANG: the basic checking part (*i.e.*, the solid arrows) and the real-time checking part (*i.e.*, the dashed arrows). The real-time part is used for failure-tolerance analysis, *i.e.*, checking "whether the property holds if a specified set of routers or links becomes unavailable", while the basic part is responsible for the regular checking, such as network change analysis and daily auditing.

Basic checking part. In the basic part, we use Hoyan [41] to generate the RIBs, PBR, and SR tunnels for each virtual routing and forwarding (VRF) of all routers in the network.

Then, JINGUBANG reads multiple *traffic snapshots* $\mathcal{S}_1, \cdots, \mathcal{S}_n$. Each traffic snapshot records a set of *located flows* entering the network at the corresponding time point (*e.g.*, all traffic flow information from 12:30am to 12:31am on Aug 20, 2022), where a located flow is represented as $\langle$*dst*, *src*, *dscp*, *interface*, *traffic volume*$\rangle$. For example, in Figure 1, f_1 can be represented as $\langle$*30.0.0.1*, *20.0.0.1*, *dscp=5*, *interfaceA*, *8 Gbps*$\rangle$. By reading n traffic snapshots, $\mathcal{S}_1, \cdots, \mathcal{S}_n$, JINGUBANG's flow number reduction module (§4.4) generates n snapshots, $\mathcal{S}'_1, \cdots, \mathcal{S}'_n$, but each with a much smaller size due to our proposed sampling and equivalence class approaches.

By putting the above information together, JINGUBANG constructs a traffic distribution graph (or TDG) and simulates the traffic distribution (§4.2, §4.3). Finally, it checks whether the traffic distribution meets the specified property.

Real-time checking part. In the real-time part, JINGUBANG takes the failure model—containing a collection of specified failed routers or links—as input. We first use Hoyan to generate differential RIBs that record the difference of RIBs due to the failure. Then, JINGUBANG checks the TDG computed in the basic process and only updates the affected part. Finally, JINGUBANG computes traffic distribution under the given failure via the incremental traffic simulation module (§5).

3.2 Important Terminologies

Table 2 summarizes important terminologies, including flow, located flow, traffic snapshot, and traffic distribution. For the sake of simplicity, we may use the term "flow" for the located flow when the context is clear. Besides, the terms "traffic rate" and "traffic volume" of a flow may be used interchangeably at a given time point, since they are equal in unit time.

task	::=	*conf*; *flow*; [*fail*;] *prop*;	Checking task
		...	...
prop	::=	($\mathit{link}\ l : \mathit{expr}$)*	Traffic load property
expr	::=	(**load**$\mid$**diff**) ($\geq \mid \leq$) n (**Gbps**$\mid$%)	Per-link property

Figure 4: Specification of JINGUBANG checking tasks.

3.3 JINGUBANG Checking Task's Specification

The specification of an JINGUBANG's checking task is similar to previous verification systems [2,3,9,37] with the additional traffic load properties. As shown in Figure 4, our operators will use a domain-specific language (DSL) to specify four parts: *conf*, *flow*, *fail* (optional), and *prop*: (i) *conf* specifies the topologies and configurations of the network to check; (ii) *flow* specifies the traffic snapshots; (iii) *fail* is only used for real-time checking, and allows the operators to specify a failure model, *i.e.*, a collection of failed routers and links in this checking task; (iv) *prop* specifies the traffic load property, defined as a set of $\langle$*link*, *per-link traffic load property*$\rangle$ pairs, where a per-link traffic load property (defined in *expr*) specifies the condition that the traffic load or its difference (in the failure scenario) on a link should hold.

Given a task defined above, JINGUBANG simulates the traffic distribution and checks if the specified traffic load property holds. The checking part follows directly from the simulated traffic distribution. Below we discuss how to simulate the traffic distribution accurately and efficiently in detail.

4 Traffic Simulation Using TDG

Given the complex features used in our WAN, flows exhibit various behavior as described in §2.2. In order to accurately simulate the traffic load on each link, we need to correctly model the traffic behavior of each flow in the traffic snapshots, such as ECMP, packet rewriting, and tunneling.

In this section, we first introduce the traffic distribution graph (TDG), capable of modeling the above-mentioned traffic behavior of flows. We then describe how to simulate traffic distribution based on TDG. Finally, we propose additional optimizations to further improve the simulation efficiency.

4.1 Traffic Distribution Graph

There are two existing ways that may be used to model the network behavior: 1) the shortest-path-based approach [14, 36] and 2) the SMT-based approach [2]. Neither of them can model the traffic behavior in our WAN. First, the shortest-path-based approach uses a weighted graph to model the behavior of each traffic class, where the shortest paths correspond to the actual forwarding paths. This approach is fundamentally limited in expressiveness: it cannot model iBGP and local preference which are inevitable in WANs. Second, the SMT-based approach encodes the entire control plane into an SMT formula and employs a solver to solve the formula. While there are encodings [2] for BGP, it is nontrivial to extend them to model the complex features in our WAN. In addition, SMT solving often induces prohibitively high overhead in

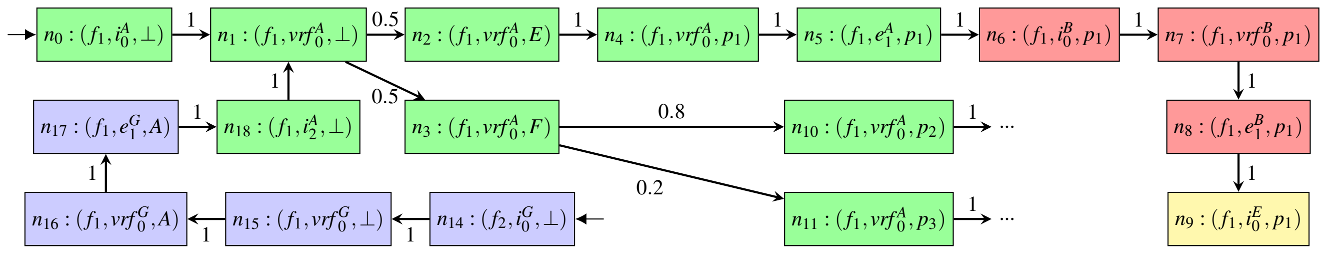

$n_0 : (f_1, i_0^A, \perp)$
$n_1 : (f_1, vrf_0^A, \perp)$
$n_2 : (f_1, vrf_0^A, E)$
$n_4 : (f_1, vrf_0^A, p_1)$
$n_5 : (f_1, e_1^A, p_1)$
$n_6 : (f_1, i_0^B, p_1)$
$n_7 : (f_1, vrf_0^B, p_1)$
$n_8 : (f_1, e_1^B, p_1)$
$n_9 : (f_1, i_0^E, p_1)$
$n_3 : (f_1, vrf_0^A, F)$
$n_{10} : (f_1, vrf_0^A, p_2)$
$n_{11} : (f_1, vrf_0^A, p_3)$
$n_{17} : (f_1, e_1^G, A)$
$n_{18} : (f_1, i_2^A, \perp)$
$n_{16} : (f_1, vrf_0^G, A)$
$n_{15} : (f_1, vrf_0^G, \perp)$
$n_{14} : (f_2, i_0^G, \perp)$
1
0.5
0.5
0.8
0.2
···

Figure 5: A (partial) TDG representing the example scenario in Figure 1. The green, red, yellow, and purple nodes correspond to router A, B, E, and G, respectively. Other nodes are omitted due to space limits.

runtime [36], which makes it impractical for our WAN with a large number of routers and prefixes.

To model the complex behavior under BGP, IS-IS, PBR, and SR, while still supporting efficient reasoning for traffic load properties, we introduce the *traffic distribution graph* (or TDG). Below, we first define the TDG and illustrate the TDG for the motivating example in Figure 1. The formal definition and construction of TDG can be found in Appendix B.

Definition. A TDG, corresponding to a traffic snapshot, models the entire traversal of *all* flows in this traffic snapshot in a fine-grained manner. Specifically, a TDG is a directed acyclic graph (JINGUBANG performs loop checking beforehand and will report errors if the TDG contains loops), where a *node* contains three elements: (i) the corresponding flow, *i.e.*, $\langle dst$, src, $dscp\rangle$, (ii) its location (*e.g.*, receiving at an interface or being processed at a VRF), and (iii) its next hop (*e.g.*, an IP address or an SR tunnel) based on the RIB lookup and policy matching results on the router. An *edge* between two nodes denotes a single processing step for this flow, such as forwarding the flow from an interface on router A to an interface on router B, looking up the RIB on a router's VRF, and so on. Each edge is associated with a non-negative weight representing the fraction of traffic volume distributed on that edge; thus, it is required that the sum of weights on all outgoing edges of a node equals 1. Finally, the sources in the TDG denote the initial state of flows entering the network.

Example. Figure 5 shows a (partial) TDG for the motivating example shown in Figure 1. For readability, nodes are colored based on the routers they correspond to; specially, green, red, yellow, and purple nodes correspond to routers A, B, E, and G, respectively. Each node is labeled with a name n_k. We use vrf_k^X to denote the k-th VRF on router X. We also use i_k^X (e_k^X, resp.) to denote the k-th interface labeled with the incoming (outgoing, resp.) flow direction on router X.

The node n_0 denotes the initial state of flow f_1 entering the network on router A's interface i_0^A, where $\perp$ means that the next hop is currently unknown. The edge (n_0, n_1) corresponds to the step that f_1 will be matched against the RIB on vrf_0^A. The two edges from n_1 correspond to the BGP lookup step, which finds the next hop E and F. Note that the weights on the two edges are both 0.5, reflecting the BGP ECMP mechanism implemented by router vendors. The edge (n_2, n_4) corresponds to the step that A resolves the next hop for E and finds the SR tunnel p_1; the two edges from n_3 denote that two tunnels are resolved for F and the weights reflect the ones set in the SR configurations. Then (n_4, n_5) and (n_5, n_6) denote that the flow is sent to the outgoing interface and forwarded to B's interface i_0^B; the meaning of other edges follows directly.

Similarly, the node n_{14} denotes the initial state of flow f_2 entering router G's interface i_0^G with no next hops. The edge (n_{14}, n_{15}) corresponds to the PBR step which modifies f_2 to f_1. The edges from n_{15} to n_{18} and finally to n_1 are similar to the ones described above. Note that after f_2's DSCP is modified, it undergoes processing identical to f_1. Therefore, the TDG does not have repetitive nodes for f_2 after n_{18}, showcasing its efficiency in WAN scenarios where it is common for flows to enter through different ingress points but later be processed identically. In contrast, the shortest-path-based modeling [14, 36] has to build complete graphs for both f_1 and f_2, even though the two graphs may share a large common subgraph.

4.2 Simulating A Single Traffic Snapshot

Given a TDG modeling the traffic behavior of all flows in a traffic snapshot, we can compute the traffic distribution of each flow on the TDG following its structure. By aggregating the traffic load of all flows traversing a link, we can obtain the traffic load of that link for the given traffic snapshot.

Algorithm 1 shows the traffic simulation algorithm for a single traffic snapshot $\mathcal{S}$, implementing the above idea. First, the function TRAFFICSIM constructs a TDG for all located flows in the traffic snapshot $\mathcal{S}$. After that, the function TRAFFICSIMONTDG takes the generated TDG and the traffic snapshot $\mathcal{S}$ as inputs to compute TL, *i.e.*, the traffic distribution at the given time point. Recall that the traffic distribution is a vector that records the traffic load of each link in the given snapshot (§3). Specifically, TRAFFICSIMONTDG calculates the traffic volume entering each node and distributes it on each edge of the TDG in topological order. For each node n in the TDG, we compute the traffic volume that reaches n (*i.e.*, $V[n]$) by summing up all volumes $V[e]$ on each incoming edge e of n (line 7). Then, the volume at n is distributed into each outgoing edge of n determined by their weights, *e.g.*, ECMP and SR weight (line 8). Finally, a link l's volume (or rate) $TL[l]$ is computed by aggregating $V[e]$ for each edge e where e denotes the forwarding step along the link l (line 10). Thus, we obtain a traffic distribution for a given time point.

Algorithm 1: Traffic simulation at a time point

```
1  Function TRAFFICSIM($\mathcal{S}$):
2      Construct the TDG $(N,E,w)$ for all $(f_k, i_k, v_k)$ in $\mathcal{S}$;
3      return TRAFFICSIMONTDG($\mathcal{S}$, $TDG(N,E,w)$);
4  Function TRAFFICSIMONTDG($\mathcal{S}$, $TDG(N,E,w)$):
5      Add an incoming edge $e'_k$ to source node $n_k$ for each located
         flow in $\mathcal{S}$ and let $V[e'_k] \leftarrow v_k$;
6      forall n in N sorted in topological order do
7          $V[n] \leftarrow \text{sum}\{V[e]\}$ for all $n$'s incoming edge $e$;
8          $V[e] \leftarrow V[n] \times w(e)$ for all $n$'s outgoing edge $e$;
9      forall link l in the network do
10         $TL[l] \leftarrow \text{sum}\{V[e]\}$ for all $e$ in $E$ that share the same pair
             of interfaces with $l$;
11     return $TL$;
```

4.3 Simulating Multiple Traffic Snapshots

We now consider the case where the system accepts multiple traffic snapshots as input and needs to validate traffic load properties for a period of time. A naive solution may simply apply the TRAFFICSIM function, which is designed for a single traffic snapshot, to each input traffic snapshot. However, it is inefficient for scenarios requiring a large number of traffic snapshots. For example, a planned change with a 2-hour time window requires 120 per-minute traffic snapshots.

To improve efficiency, we make the key observation that the input traffic snapshots are highly correlated. In the above example, all the traffic snapshots are extracted from the continuous 2-hour period. Thus, a located flow may appear across multiple snapshots which allows us to avoid repetitive computation for each appearance.

We then propose a global-construction separate-evaluation approach. As shown in Algorithm 2, JINGUBANG constructs a global TDG representing the union of all flows in all traffic snapshots (line 2) and calls TRAFFICSIMONTDG with the global TDG and each given traffic snapshot (line 3). Thus, the output of the algorithm *TLM* is a matrix recording the traffic distributions across multiple time points.

4.4 Flow Number Reduction Optimization

To further improve the efficiency of traffic simulation, we next propose two optimization techniques that can significantly reduce the number of flows needed to be simulated.

Traffic sampling optimization. First, we propose a sampling approach with *provable* guarantees on the accuracy loss to reduce the number of flows in the traffic snapshot. Our key insight is that a flow with a higher traffic volume should have a higher probability of being sampled, and the total traffic volume should be preserved. For example, if we sample only one flow from the traffic snapshot $\mathcal{S}$, the sampled flow should carry the total volume of all flows in $\mathcal{S}$.

Based on this insight, we propose the traffic sampling algorithm. This algorithm attempts to draw K located flows from the traffic snapshot $\mathcal{S}$. In each round, the algorithm draws a flow from $\mathcal{S}$ such that a flow with a volume v gets drawn with probability v/V, where V is the total traffic volume of all flows. Then, we assign V/K as the volume to the sampled flow and add it to the new traffic snapshot $\mathcal{S}'$. Repeating this process K times, we sample K located flows with a preserved total volume. We leave the algorithm's mathematical representation and pseudocode in Appendix A.1.

It is important to note that the number of *distinct* flows in the sampled traffic snapshot is likely to be much lower than the parameter K. This is because high-volume flows are likely to be sampled multiple times, and their volumes can be aggregated in the sampling process. We define the *flow sample ratio* as the ratio of the number of distinct flows in the sampled snapshot $\mathcal{S}'$ to the number of distinct flows in $\mathcal{S}$, which is related not only to the sample parameter K, but also to the distribution of volumes in $\mathcal{S}$.

Traffic sampling can introduce errors in estimating the traffic load on a link. Intuitively, for links with low/high traffic load, it is reasonable to require that the estimated traffic load does not deviate by an unacceptable absolute/relative value, respectively. We rigorously establish a lower bound of K such that traffic volumes of *all* links are guaranteed not to deviate by given absolute and relative values at a given level of confidence. The significance of K's proven bound lies in its tightness, making it practical for the use in our WAN, which is detailed in Appendix A.2. In the derivation of K's bound, we start from the law of large numbers and make no assumptions about the distribution of flows in $\mathcal{S}$. We also evaluate traffic sampling by experiments in §7.

Traffic equivalence class optimization. Besides, we observe that many flows exhibit the same behavior in the network despite that they differ in field values and incoming locations. For example, suppose in the motivating example there is another flow f_3 entering the network at some interface of A and matches the same BGP route and SR policy as f_1 does. Instead of performing the computation for both f_1 and f_3, we may only perform the computation for one of them, assuming it carries the volume of both flows.

Leveraging this insight, we propose the *traffic equivalence class* (TEC). Two flows are equivalent if they have the same traffic distribution over all links of the network; a TEC is a set of located flows that are equivalent to one another. See Appendix C for the formal definition and generation of TECs.

Among multiple flows in a TEC, we only need to consider one for traffic simulation. Therefore, given a traffic snapshot $\mathcal{S}$, we can aggregate flows in the same TEC by replacing them with a single flow that has the combined volume of all flows in the TEC. As a result, we generate a new traffic snapshot $\mathcal{S}'$ that has a much smaller number of flows.

5 Real-Time Failure-Tolerance Analysis

Checking whether the network satisfies the intended traffic load properties under a variety of failure scenarios is a key task in our daily operation (see Table 1). In such a task, our op-

Algorithm 2: Traffic simulation for a period of time

```
1 Function TrafficSimMul($\{\mathcal{S}_k\}_{k=1}^{m}$):
2     Construct the TDG $(N,E,w)$ for all $(f_k,i_k,v_k)$ in $\bigcup_{k=1}^{m}\mathcal{S}_k$;
3     $TLM[\mathcal{S}_k] \leftarrow$ TrafficSimOnTdg($\mathcal{S}_k$, $(N,E,w)$) for all
        $k = 1,\cdots,m$;
4     return $TLM$;
```

erators submit a large number of checking requests for failure-tolerance analysis, each specifying a set of failed routers or links, and expect to obtain results within *seconds*.

To meet the high-efficiency challenge, we observe that a typical checking request contains a small number (*i.e.*, $O(10)$) of link and router failures. Thus, the new TDG for the failure scenario may only differ slightly from the one without failure. For example in Figure 1, if $B \rightarrow E$ fails, the new TDG has all the nodes and edges except the edges $n_1 \rightarrow n_2 \rightarrow \cdots \rightarrow n_9$ in Figure 5. This observation allows us to adopt a two-phase design, where we first compute the TDG and store the necessary information for the network (without failure) in the basic checking part (see Figure 3), and then perform an *incremental* computation in the real-time checking part (see Figure 3) for each received failure-tolerance checking request.

There are two key steps to be conducted incrementally: TDG construction (§5.1) and traffic simulation (§5.2).

5.1 Incremental TDG Construction

When a failure happens, RIBs on routers converge to a new state (including the new SR tunnels); thus, the routes/tunnels may change, and the nodes in the old TDG corresponding to changed routes/tunnels may need to be updated (*e.g.*, connecting to different nodes or changing the weights of outgoing edges). In addition, the nodes corresponding to the failed links or routers should also be updated by removing their associated edges. All other nodes irrelevant to the failure (*e.g.*, receiving a flow at an active interface) remain unchanged. Therefore, to construct the new TDG, we keep most nodes and edges in the old TDG, and only re-run the construction for changed nodes.

Based on this observation, we construct the TDG for a given failure model incrementally. The algorithm takes the old TDG, the set of changed routes (*e.g.*, next hops changed) and tunnels (*e.g.*, tunnel removed) ΔRib, and the new RIBs (denote Rib') for the failure as input, and generates the new TDG incrementally. The algorithm also outputs the set of changed nodes *chg* for the post incremental traffic simulation (§5.2). Note that simulating Rib' and ΔRib is easy; many existing systems [12, 41] can extract the information.

The pseudocode and detailed description of the algorithm can be found in Appendix D.1. We illustrate how the algorithm works using the following example.

Example. Suppose $B \rightarrow E$ in Figure 1 fails. Due to the failure, the SR tunnel $A \rightarrow B \rightarrow E$ becomes invalid and thus the set *chg* contains the nodes for matching tunnel p_1 (*i.e.*, n_2 and n_4-n_9 in Figure 5). Furthermore, the BGP next hop E is also

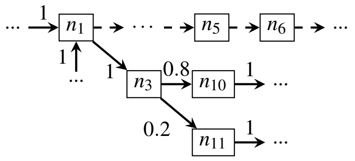


Figure 6: The (partial) TDG representing the example in Figure 1 when link $B \rightarrow E$ fails. Solid lines represent edges in both the old (*i.e.*, before failure) and new (*i.e.*, after failure) TDGs; dashed lines represent edges only in the old TDG.

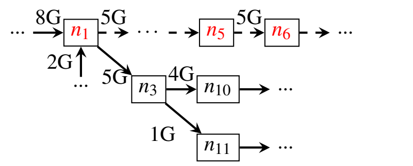
inactive, so *chg* also contains n_1 (in Figure 5) representing the match of the BGP route that is changed by the failure. Then the algorithm removes all outgoing edges for the nodes in *chg*. Using BFS exploration, the algorithm only re-constructs the edge from n_1 to n_3 with weight 1. The constructed TDG is shown in Figure 6.

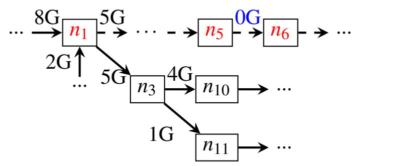
5.2 Incremental Traffic Simulation

With the new TDG, one may re-run the traffic simulation algorithm on it to compute the volume on each node and edge as in Algorithm 1. However, since the new TDG largely overlaps with the old TDG, this straightforward approach leads to numerous redundant computations. In particular, if a node's volume remains unchanged and its outgoing edges/weights are also unchanged, the traffic distribution on the outgoing edges of that node should remain the same as that in the old TDG. We therefore only need to run the simulation for the *affected* nodes where either the volume or the outgoing edges/weights change. Clearly, all nodes in *chg* are affected nodes since their outgoing edges are changed; all nodes with incoming edges (in the old TDG) from any node in *chg* may also be affected, since their volumes may be changed. In addition, when updating the volume on the new TDG, other nodes may also be affected due to the change in their volume; thus, we also need to update the set of affected nodes along with the execution of traffic simulation.

Algorithm 3 shows the incremental traffic simulation for a single traffic snapshot; it can be naturally generalized to support multiple snapshots similar to the approach described in §4.3. The detailed description is in Appendix D.2.

Example. We illustrate the incremental computation of the example in §5.1. Suppose without failure, the volume of n_1's incoming edges is 8 and 2 Gbps respectively as shown in Figure 7. As shown in Figure 7a, the affected nodes α initially contain n_1, n_2, and all nodes from n_4 to n_9, since all of them are in *chg* as described in §5.1. Thus, the first node that needs to be updated is n_2 or anyone from n_4 to n_9, as none of them have any incoming edges in the new TDG G'. Suppose n_5 gets updated first. Since n_5 has no incoming edges in G', its volume is updated to 0; since the outgoing edge $n_5 \rightarrow n_6$ only appears in the old TDG G, the algorithm also decreases the traffic distribution on the corresponding link from $A \rightarrow B$, as shown in Figure 7b. The update of n_2 and others from n_4 to n_9 is similar and we omit the detailed description due


(a) Initially, affected nodes α contains n_1, n_2, and all nodes from n_4 to n_9 as described in §5.1.


(b) After updating n_5, the volume on $n_5 \to n_6$ is decreased to 0 Gbps. Since $n_5 \to n_6$ corresponds to link $l = A \to B$ in Figure 1, $TL[l]$ is updated to 0.

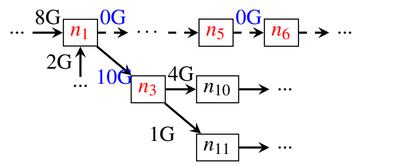

(c) After updating n_1, the volume on $n_1 \to n_3$ is increased to 10 Gbps and n_3 is added to α.

Figure 7: Representative steps of Algorithm 3 running on the TDG in Figure 6. The black (blue, resp.) number on an edge e denotes the value of v_e (v'_e, resp.), and the nodes in red are in α, after the corresponding step.

Algorithm 3: Incremental traffic simulation

```
Function IncTrafficSim(G, V, TL, G', chg):
    Let $G = (N, E, w)$, $G' = (N', E', w')$;
    $V'[n] \leftarrow V[n]$ for all $n \in N'$;
    $V'[e] \leftarrow V[e]$ for all $n \in E'$;
    $\alpha \leftarrow chg \cup \{n | \exists n' \in chg \text{ s.t. } (n', n) \in E\}$;
    forall n in N' sorted in topological order of G' do
        if n is not in $\alpha$ then
            continue;
        $V'[n] \leftarrow \text{sum}\{V'[e]\}$ for all n's incoming edge $e \in E'$;
        forall $e = (n, n') \in E' \cup E$ do
            $V'[e] \leftarrow V'[n] \times w'(e)$ if $e \in E'$;
            Let $v_e = V[e]$ if $e \in E$ else 0;
            Let $v'_e = V'[e]$ if $e \in E'$ else 0;
            if $v'_e \neq v_e$ then
                Add $n'$ to $\alpha$ ;
                $TL[l] \leftarrow TL[l] + v'_e - v_e$ if e corresponds to link l;
    return TL;
```

to space limits. After those nodes are updated, the traffic distribution on link $B \to E$ is correctly decreased to 0 as expected. The next node of interest to be updated is n_1. As shown in Figure 7c, since the volume on the only outgoing edge $n_1 \to n_3$ increases, the algorithm adds n_3 to α. In the following iterations, the algorithm updates n_3 and all other nodes afterward, which results in the correct increase of traffic distribution for all links on the paths from A to F.

6 Deployment and Use Cases

JINGUBANG has been used in our WAN for more than one year and covers operation scenarios in Table 1. During this time, no outage resulting from traffic load violations occurred, since JINGUBANG successfully detected many violations ahead of time; before JINGUBANG, tens of outages caused by load violations happened every year. As a result, JINGUBANG prevents millions of dollars in losses. We next present real cases for two representative operation scenarios, *i.e.*, validating network changes and validating failure tolerance.

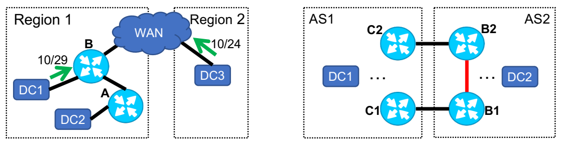
6.1 Validating Network Changes

JINGUBANG has been regularly used to check whether network change plans are correctly designed. JINGUBANG successfully detected several severe network change risks. Table 3 shows the statistics of root causes of network changes that were detected by JINGUBANG in the past one year. We classify the root causes into three types: change plan errors, unexpected routes, and existing misconfiguration. Specifically, change plan errors refer to errors in the change plan itself; unexpected routes mean that the change triggers issues because certain routes were not considered during the change; existing misconfiguration refers to hidden misconfigurations in the network that were not triggered because there was no traffic, but this change triggered the error configuration. We now detail several real, tricky cases as examples for demonstrating the effectiveness of JINGUBANG.

Table 3: Network change risks detected by JINGUBANG: root causes and their frequency of occurrence.

Root causes	Change plan errors	Unexpected routes	Existing misconfiguration
Percentage	44%	33%	23%

Change plan errors. Given the complexity and large size of our WAN, it is nontrivial to change the network correctly even for simple configuration-changing tasks. Our operators use JINGUBANG to check a change plan against intended link load properties during the time window of network changes. In one of our real network changes, our operators planned an important topology architecture upgrade for one business region in our WAN. The original topology in that region has many edge routers and border routers. Each edge router A_i is directly connected with all the border routers B_j with 100 Gbps links, while all the edge routers are connected in full mesh with 10 Gbps links. All routers run IS-IS as underlay and SR as overlay; the IS-IS cost for the links between the edge and border routers is 10000, and the cost between the edge routers is 5. For the new topology, our operators planned to add a layer of core routers C_k between the edge layer and border layer connected using 100 Gbps links. The IS-IS cost of links between the core routers and the border routers should be 10, while the cost of the links between core routers and edge routers is 10000. After the upgrade, traffic from border routers to edge routers should pass the core routers. In this network change, our operators first added all the core routers and set up the links without disabling old links between the border routers and the edge routers. Then, our operators disabled



(a) Unexpected routes. (b) Existing misconfiguration.

Figure 8: Severe risks detected by JINGUBANG.

IS-IS on the links between edge and border layers one by one via changing the interfaces on the edge routers to passive.

Given the change plan, JINGUBANG checked link overload for all links in the WAN using traffic snapshots in a similar time window. JINGUBANG detected that just after disabling the first edge router's (A_1) interfaces to the border routers, the utilization of links between A_1 and other edge routers suddenly increases to up to 150% during the time period, indicating severe link overload that would result in service downtime. The reason for the link overload is that: after disabling A_1's link to the border routers, the IS-IS shortest path from the border router B_j to A_1 becomes B_j-A_i-A_1 instead of B_j-C_k-A_1 as in the desired topology; meanwhile, the links between A_1 and other edge routers as backup links have a capacity of only 10 Gbps, significantly smaller than the 100 Gbps of other links. As a result, the traffic from border routers to A_1 caused links between A_1 and other edge routers to overload.

This situation is very hard to be detected without JINGUBANG. First, without JINGUBANG, our operators have to manually check link overload for each step in the change plan, which is error-prone. Second, what we describe is just a subset topology of our WAN. Our operators have to analyze IS-IS shortest paths for the entire scope, which should cover at least nearby regions. Third, the backup links between edge routers are designed to carry a small amount of traffic; thus, in addition to identifying that IS-IS shortest paths pass through those links, our operators have to manually analyze the traffic rate passing those links. As a result, our operators cannot detect such complex outages without JINGUBANG. Before JINGUBANG, similar problems occurred multiple times.

Unexpected routes. Our WAN carries millions of routes; some of them are well understood while some of them are unexpected. It is impossible for the operators to examine the effect of a change on all of the routes, especially for the unexpected routes. JINGUBANG offers an intuitive approach to checking the correctness of a change plan under the existence of unexpected routes. Figure 8(a) shows such a case. In this scenario, router A was configured with a default route (*i.e.*, 0/0) to forward DC2's traffic to router B while router B did not advertise any routes to router A. Due to business reasons, our operators needed to block the traffic from DC2 going to Region 2. After carefully checking that no traffic flowing from DC2 to Region 2, our operators planned to update the policy on router B such that it can advertise routes from Region 2 (those routes have pre-defined communities) to router A, and also add policies on router A to explicitly drop traffic matching those routes. Our operators used JINGUBANG to check that the traffic load should remain the same during the change. However, JINGUBANG identified load decreases on links in Region 1, violating the intended traffic load property. The root cause is that, DC3 in Region 2 was advertising a 10/24 route, while a service in DC1 in Region 1 unexpectedly used and advertised 10/29. Before the change, traffic from DC2 can reach the service in DC1 by matching the default route on router A and the 10/29 route on router B. After the change on routers A and B, the traffic would match the 10/24 route on router A and thus get dropped, which may cause a service outage. Without JINGUBANG, such risk is very hard to detect due to the unawareness of those routes; but fortunately, JINGUBANG successfully prevented it by identifying the violation to the specified traffic load property.

Existing misconfiguration. A typical network change may only involve a small set of routers. However, given the existing misconfiguration on other routers in the network, such changes can introduce severe risks. Figure 8(b) shows a real case in our WAN. In this scenario, traffic from DC1 to DC2 used to exit AS1 via router C2. Our operator planned to shift the exit point from C2 to C1. To achieve this, our operators first changed the policies on the routers that act as BGP route reflectors [7] (not shown in the figure) to propagate routes that would be advertised by C1 to DC2; then changed the policies on C1 to advertise these routes while assigning them a higher local preference to make them effective. Our operators use JINGUBANG to check that no links in the network would be overloaded during the change. JINGUBANG successfully detected overloaded links between B1 and B2 (highlighted in red). The root cause of this risk was that, while the change plan could successfully steer the traffic from C2 to C1, however, due to the incorrect IS-IS cost setup in AS2, traffic transited from B1 to B2 instead of flowing directed to DC2, causing the low-capacity links between B1 and B2 to overload. This risk is extremely hard for our operators to identify manually because (i) the change plan correctly altered the exit point on AS1 as intended and (ii) the misconfigured IS-IS costs in AS2 were not a problem before the change because there was no large volume of traffic going through those links. Fortunately, JINGUBANG successfully prevented the severe risk by detecting the overloaded links.

6.2 Validating Failure Tolerance

Our WAN is designed to tolerate failures up to a certain level by configuring redundant links and backup routers, which we call the redundancy property. However, due to the high-churn network changes and carelessness of our operators, redundancy property is violated in many regions in our WAN. Our operators cannot know the violations without a checker. Once the bottlenecked links or routers fail, a cascading failure would span across the entire WAN.

Our operators systematically inject router and link failures and use JINGUBANG to check two properties: (i) no link overload and (ii) no drastic drop in link load (which is typically caused by reachability issues). JINGUBANG has successfully detected tens of failure risks. One of them happened on a remote site of our WAN, where we deployed two border routers to provide service to our customers abroad. After our operators injected a failure model that made one of the border routers unavailable, JINGUBANG detected that the traffic load on several links close to that site dropped hugely. Further analyzing each flow using JINGUBANG's capability of flow-level traffic simulation, JINGUBANG successfully identified a prefix where flows with destinations in that prefix got dropped on the other router. We examined this situation with our operators, and confirmed that the prefix belongs to a new service and one of the routers is configured incorrectly such that the prefix is only advertised out of another router.

7 Evaluation

We evaluate the accuracy and efficiency of JINGUBANG using real-world data obtained from our production WAN.

Experiment setup. Besides the entire WAN, we select two sub-networks, N1 and N2, to represent diverse scales of network traffic. N1 and N2 are chosen as they respectively account for 7% and 85% of the total traffic in our WAN. In each hour, there are billions of flows entering N2 and hundreds of millions of flows entering N1. There are hundreds of routers in both N1 and N2. All networks are configured with BGP, IS-IS, SR, and PBR. Unless otherwise specified, we use JINGUBANG to check traffic load across one hour (*i.e.*, 60 time points). All the experiments are conducted on a server with 768 GB RAM and a 2.50 GHz 104-core processor.

Performance. We first use JINGUBANG to check traffic load for each hour across one randomly selected day. As shown in Figure 9, the time cost for checking these hours is within an order of magnitude, ranging from 19.6 s to 25.6 s for N1, 168.2 s to 272.4 s for N2, and 225.5 s to 308.4 s for the WAN.

Accuracy. We inject failures that actually occurred in the selected hours. We define the accuracy for each link as $\frac{\min(r_{\text{sim}},r_{\text{real}})}{\max(r_{\text{sim}},r_{\text{real}})} \times 100\%$ where r_{sim} represents the traffic rate computed by JINGUBANG and r_{real} represents the rate measured by the router. Based on this, we further determine the overall accuracy of JINGUBANG by averaging accuracy of all links with weights proportional to the link's traffic load. Figure 9 depicts such accuracy of each hour. The overall accuracy ranges from 88.2% to 94.6% for N1, 87.4% to 92.5% for N2, and 87.8% to 92.1% for the WAN, which is stable and meets our operators' requirements. Nevertheless, the accuracy is not 100%, mostly because our route simulation takes external routes advertised into our WAN as input and they are provided by our internal route monitoring system, which occasionally misses certain routes due to coverage issues.

We further demonstrate the variation of r_{sim} and r_{real} during one hour using an example link. The link connects two service areas of our WAN and its link load varies between 40% and 95% within a single hour. As shown in Figure 10, JINGUBANG can accurately track these fluctuations. Over the 60 time points, the median, 90th-percentile, and maximum error rate is 4.0%, 8.7%, and 19.0%, respectively, indicating the high accuracy of JINGUBANG.

Multiple traffic snapshots. [5] We evaluate the checking time of JINGUBANG for different numbers of time points. As shown in Figure 11, JINGUBANG's checking speed at 60 time points is only 4.2× (from 4.5s to 18.9s) and 5.2× (from 67.5s to 353.9s) slower than at one time point for networks N1 and N2, respectively. The reason behind this is that the sets of flows at consecutive time points have a significant amount of overlap with each other. By constructing a single TDG for all 60 time points, JINGUBANG avoids duplicate processing and speeds up the checking by 14.1× (11.4×) for N1 (N2), compared to construct separate TDGs for each time point.

Traffic equivalence classes. In Figure 12, we show the impact of applying TEC on the performance of JINGUBANG for different flow sample ratios in network N1. When traffic sampling is disabled, the use of TEC significantly accelerates the checking, from 601.8s to 18.4s, resulting in a 32.7× speedup. This improvement is substantial due to the large number of flows that can be grouped into a single class. However, when the flow sample ratio is 11%, the speedup is only 3.8× (from 33.4s to 8.8s) as many low-rate flows are not sampled, but their equivalence class still exists because of either the large flow number in the class or the presence of an elephant flow. We do not evaluate the use of TEC in N2 as it would result in memory exhaustion without TEC.

Traffic sampling. Figure 13 illustrates the relationship between the flow sample ratio and the time cost of JINGUBANG in network N2. By setting the sample ratio to 11%, the checking time can be significantly reduced from 358.3s to 113.7s, a 3.2× speedup. To assess its impact on accuracy, we define the maximum accuracy loss as the maximum reduction in link load accuracy for all links that use $\geq$5% of their bandwidth. As also shown in Figure 13, the accuracy loss does not exceed 1.1% across 10 runs. Besides, for links with bandwidth utilization <5%, the absolute changes in their traffic rates due to traffic sampling never exceed 60 Mbps. Therefore, for most links, the accuracy loss is not substantial at all. As an example, in Figure 10, we additionally plot r_{sim} after sampling at the flow ratio of 11%, and the difference is barely noticeable.

Incremental simulation. The daily operation of our WAN involves handling up to $O(100)$ checking requests per hour, each of which asks us to validate a few traffic load properties under up to $O(10)$ failed links. To compare the checking time between incremental and full simulation, we randomly selected $O(100)$ checking requests and depict the CDF of time cost using two methods in Figure 14. We apply the traffic sampling with a flow ratio of 11% in N2. As demonstrated in

[5]Since our operation suggests that JINGUBANG can check N2 and the WAN at a similar time cost (as also shown in Figure 9), and for simplicity, we do not evaluate JINGUBANG on the WAN in the following experiments.

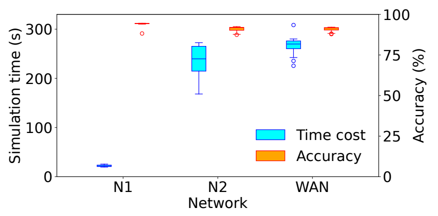


Figure 9: Performance and accuracy.

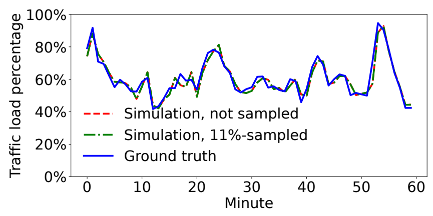


Figure 10: Per-minute accuracy.

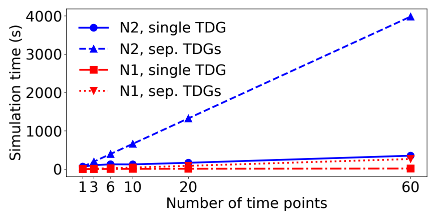


Figure 11: Effects of the single TDG.

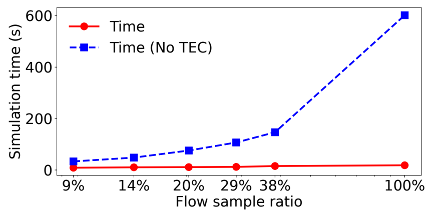


Figure 12: Effects of TEC.

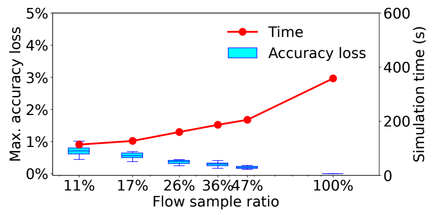


Figure 13: Effects of traffic sampling.

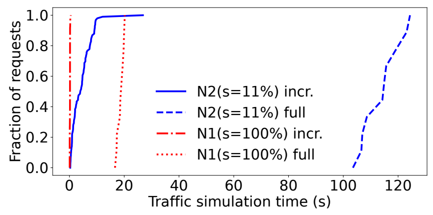


Figure 14: Failure-tolerance check time.

the figure, the average time cost for incremental simulation is 4.3s, which is a significant improvement over the average time cost of 113.7s for full simulation. This 26.2× speedup meets our deployment requirements and shows the effectiveness of using incremental simulation in our WAN.

JINGUBANG is also capable of handling router failures in an incremental manner. However, such requests are relatively infrequent, occurring dozens of times per month. Nevertheless, with ∼100 checking requests that involve supported kinds of router failures, we find that incremental simulation in N1 (N2) costs 2.3s (14.5s) on average, which is still 8.0× (7.8×) faster than full simulation.

8 Discussions and Lessons

We now present our lessons and discuss limitations.

Verifying arbitrary *k*-failure tolerance. The *k*-failure problem lies beyond the scope of this paper, as discussed in §2.2. Meanwhile, we argue that JINGUBANG provides a solid foundation by efficiently modeling comprehensive traffic behaviors at a production scale. We plan to extend the model to support arbitrary *k*-failure verification in future research.

Addressing vendor-specific behaviors. Improving JINGUBANG's precision is often hindered by vendor-specific traffic behaviors and unexpected corner cases. For instance, SR tunnels are configured with specific next-hop IPs and DSCP values. If traffic flows match the next-hop IP but exhibit different DSCP values, routers from one vendor attempt alternative SR tunnels configured for other DSCP values, while other routers default to the IS-IS shortest path. [6] We refine JINGUBANG's accuracy over years of operation and defer the proactive identification of such behaviors to future work.

Enhancing user experience. JINGUBANG can indicate if traffic properties remain intact under various conditions but falls short in diagnosing reasons for property violations. Even with additional traffic flow visualization utilities, pinpointing the root issues remains time-intensive for both system developers and network operators. We envision an automated system that can find victim routes, identify configuration errors, and even offer potential repair solutions as the next step.

[6]We use real examples, but omit the vendor names.

Handling altered traffic snapshots. JINGUBANG cannot reason about shifts in traffic snapshots. Failures, for example, can prompt routers to announce altered routes to ISPs, leading to changes in the traffic entering our WAN. Since ISP routers fall outside our simulation domain and are not covered by our monitoring system, we currently rely on heuristic rules defined by our operators to adapt to such changes in traffic snapshots. A rigorous methodology to understand variations in traffic snapshots is left for subsequent research.

9 Conclusion

This paper presents JINGUBANG, the first traffic load property checking system for production WAN. JINGUBANG makes three contributions: (i) a new model, named the traffic distribution graph, capable of encoding complex traffic behavior under BGP, IS-IS, SR, and PBR; (ii) an efficient traffic simulation approach, which can handle billions of flows in a period of time; (iii) an incremental simulation approach, enabling real-time failure-tolerance checking. JINGUBANG has been used to check our WAN for more than one year and prevented many potential failures resulting from traffic load violations.

Acknowledgments

We thank our shepherd, Stefano Vissicchio, and anonymous reviewers for their insightful comments. This work is supported in part by National Key Research and Development Plan, China (Grant No. 2023YFB2903902), National Natural Science Foundation of China (Grant No. 62022005, 62272010, and 62061146001), and Alibaba Cloud through Alibaba Research Intern Program and Alibaba Innovative Research Program. Ennan Zhai and Chenren Xu are co-corresponding authors. Ruihan Li and Chenren Xu are affiliated with School of Computer Science at Peking University, Zhongguancun Laboratory, and Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education.

References

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. Tiramisu: Fast multilayer network verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 201–219, Santa Clara, CA, February 2020. USENIX Association.

[2] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. A general approach to network configuration verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 155–168, New York, NY, USA, 2017. Association for Computing Machinery.

[3] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 328–341, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. Teavar: striking the right utilization-availability balance in wan traffic engineering. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 29–43, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Tobias Bühler, Romain Jacob, Ingmar Poese, and Laurent Vanbever. Enhancing global network monitoring with magnifier. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1521–1539, Boston, MA, April 2023. USENIX Association.

[6] Yiyang Chang, Sanjay Rao, and Mohit Tawarmalani. Robust validation of network designs under uncertain demands and failures. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 347–362, Boston, MA, March 2017. USENIX Association.

[7] Enke Chen, Tony J. Bates, and Ravi Chandra. BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). RFC 4456, April 2006.

[8] Benoît Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[9] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. NetComplete: Practical Network-Wide configuration synthesis with autocompletion. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 579–594, Renton, WA, April 2018. USENIX Association.

[10] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. Efficient network reachability analysis using a succinct control plane representation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 217–232, Savannah, GA, November 2016. USENIX Association.

[11] Mikel Jimenez Fernandez and Henry Kwok. Building express backbone: Facebook's new long-haul network, 2017.

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

[13] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. Probabilistic netkat. In Peter Thiemann, editor, *Programming Languages and Systems*, pages 282–309, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[14] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. Fast control plane analysis using an abstract representation. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 300–313, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Nick Giannarakis, Ryan Beckett, Ratul Mahajan, and David Walker. Efficient verification of network fault tolerance via counterexample-guided refinement. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 305–323, Cham, 2019. Springer International Publishing.

[16] Nick Giannarakis, Alexandra Silva, and David Walker. Probnv: probabilistic verification of network control planes. *Proc. ACM Program. Lang.*, 5(ICFP), aug 2021.

[17] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 15–26, New York, NY, USA, 2013. Association for Computing Machinery.

[18] Alex Horn, Ali Kheradmand, and Mukul Prasad. Delta-net: Real-time network verification using atoms. In

14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pages 735–749, Boston, MA, March 2017. USENIX Association.

[19] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 3–14, New York, NY, USA, 2013. Association for Computing Machinery.

[20] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. Validating datacenters at scale. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 200–213, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. Automated analysis and debugging of network connectivity policies. Technical Report MSR-TR-2014-102, Microsoft, July 2014.

[22] Siva Kesava Reddy Kakarla, Ryan Beckett, Behnaz Arzani, Todd Millstein, and George Varghese. Groot: Proactive verification of dns configurations. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 310–328, New York, NY, USA, 2020. Association for Computing Machinery.

[23] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 113–126, San Jose, CA, April 2012. USENIX Association.

[24] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide invariants in real time. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, Lombard, IL, April 2013. USENIX Association.

[25] 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. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 1–14, New York, NY, USA, 2015. Association for Computing Machinery.

[26] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. Traffic engineering with forward fault correction. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, page 527–538, New York, NY, USA, 2014. Association for Computing Machinery.

[27] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. Crystalnet: Faithfully emulating large production networks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 599–613, New York, NY, USA, 2017. Association for Computing Machinery.

[28] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. Checking beliefs in dynamic networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 499–512, Oakland, CA, May 2015. USENIX Association.

[29] Sonia Panchen, Neil McKee, and Peter Phaal. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, September 2001.

[30] Aurojit Panda, Katerina Argyraki, Mooly Sagiv, Michael Schapira, and Scott Shenker. New Directions for Network Verification. In Thomas Ball, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morriset, editors, *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–220, Dagstuhl, Germany, 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[31] Aurojit Panda, Ori Lahav, Katerina Argyraki, Mooly Sagiv, and Scott Shenker. Verifying reachability in networks with mutable datapaths. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 699–718, Boston, MA, March 2017. USENIX Association.

[32] B. Quoitin and S. Uhlig. Modeling the routing of an autonomous system with c-bgp. *IEEE Network*, 19(6):12–19, 2005.

[33] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. Cantor meets scott: se-

mantic foundations for probabilistic networks. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 557–571, New York, NY, USA, 2017. Association for Computing Machinery.

[34] Samuel Steffen, Timon Gehr, Petar Tsankov, Laurent Vanbever, and Martin Vechev. Probabilistic verification of network configurations. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 750–764, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Symnet: Scalable symbolic execution for modern networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 314–327, New York, NY, USA, 2016. Association for Computing Machinery.

[36] Kausik Subramanian, Anubhavnidhi Abhashkumar, Loris D'Antoni, and Aditya Akella. Detecting network load violations for distributed control planes. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 974–988, New York, NY, USA, 2020. Association for Computing Machinery.

[37] Bingchuan Tian, Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng, and Ben Y. Zhao. Safely and automatically updating in-network acl configurations with intent language. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 214–226, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative network monitoring on a budget. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 467–482, Renton, WA, April 2018. USENIX Association.

[39] Yaron Velner, Kalev Alpernas, Aurojit Panda, Alexander Rabinovich, Mooly Sagiv, Scott Shenker, and Sharon Shoham. Some complexity results for stateful network verification. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 811–830, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[40] Anduo Wang, Limin Jia, Wenchao Zhou, Yiqing Ren, Boon Thau Loo, Jennifer Rexford, Vivek Nigam, Andre Scedrov, and Carolyn Talcott. Fsr: Formal analysis and implementation toolkit for safe interdomain routing. *IEEE/ACM Transactions on Networking*, 20(6):1814–1827, 2012.

[41] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, Duncheng She, Qing Ma, Biao Cheng, Hui Xu, Ming Zhang, Zhiliang Wang, and Rodrigo Fonseca. Accuracy, scalability, coverage: A practical configuration verifier on a global wan. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 599–614, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. Check before you change: Preventing correlated failures in service updates. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 575–589, Santa Clara, CA, February 2020. USENIX Association.

[43] Peng Zhang, Aaron Gember-Jacobson, Yueshang Zuo, Yuhao Huang, Xu Liu, and Hao Li. Differential network analysis. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 601–615, Renton, WA, April 2022. USENIX Association.

[44] Peng Zhang, Dan Wang, and Aaron Gember-Jacobson. Symbolic router execution. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 336–349, New York, NY, USA, 2022. Association for Computing Machinery.

[45] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, IMC '17, page 78–85, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Yunmo Zhang, Hong Xu, Chun Jason Xue, and Tei-Wei Kuo. Probabilistic analysis of network availability. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, pages 1–11, 2022.

APPENDIX

A Traffic Sampling

In this appendix, we will formally define the traffic sampling process (first presented in §4.4), then prove a practical bound for the error rate, and demonstrate its use in our WAN.

A.1 Algorithm

Initially, we consider a single network and a single traffic snapshot at a specific time point. We will extend our analysis to encompass multiple networks (*e.g.*, networks under different failure scenarios) and multiple time points later.

Suppose that there are N located flows $f_1, \cdots, f_N$ in the traffic snapshot $\mathcal{S}$, with volumes $v_1, \cdots, v_N$, respectively. We denote the set of flows as $\mathcal{F}$, *i.e.*, $\mathcal{F} := \{f_1, \cdots, f_N\}$ and the total volume as V, *i.e.*, $V := \sum_{i=1}^{N} v_i$. The network contains M links, where the volume of the i-th link is l_i, which is a function of the traffic snapshot.

Lemma 1. *Assuming the absence of traffic loops, the volume on each link in the network is a linear function of the volumes of the flows in the traffic snapshot. More specifically,*

$$l_i = \sum_{j=1}^{N} w_i(f_j) v_j,$$

where

$$w_i : \mathcal{F} \to [0, 1]$$

is the flow weight function for the i-th link, for all $i = 1, \cdots, M$.

Proof. This lemma follows from the construction of TDGs during the traffic simulation process. The flow weight functions can be easily calculated according to these TDGs. □

Then we give a formal definition of traffic sampling, corresponding to Algorithm 4.

Definition 1. *In traffic sampling, a new traffic snapshot $\mathcal{S}'$ is generated by selecting K flows, $F_1, \cdots, F_K$, from the original traffic snapshot $\mathcal{S}$, each with a volume of $\frac{V}{K}$. These K flows are considered as K independent and identically distributed (i.i.d.) random variables, and the probability of selecting the i-th flow in the original traffic snapshot is defined as*

$$\mathbb{P}(F_1 = f_i) = \frac{v_i}{V}$$

for all $i = 1, \cdots, N$.

As noted previously in Appendix A.1, the number of distinct flows in the sampled traffic snapshot is typically much less than K, since a flow with a large volume can easily be sampled multiple times.

Algorithm 4: Traffic sampling algorithm

```
1 Function SAMPLE($\mathcal{S}$, $K$):
2     Let $V$ be the total volume of all flows in $\mathcal{S}$;
3     Let $P$ be the probability distribution s.t.
        $\forall (f_k, i_k, v_k) \in \mathcal{S}, P(f_k, i_k) = v_k/V$;
4     $\mathcal{S}' \leftarrow \emptyset$;
5     repeat $K$ times
6         Draw a $(f_k, i_k)$ from the probability distribution $P$;
7         Add $(f_k, i_k, V/K)$ to $\mathcal{S}'$;
8     return $\mathcal{S}'$;
```

A.2 Error Bound

We next introduce the error bound for traffic sampling.

Theorem A.1. *Suppose that M links have volumes $l_1, \cdots, l_M$ before sampling. After sampling, these links have volumes represented by M random variables $L_1, \cdots, L_M$. Given a maximum relative error μ, a maximum absolute error Δ, and a confidence parameter δ, the following holds with probability at least $1 - \delta$:*

$$|L_i - l_i| \le \max\{\Delta, \mu l_i\} \quad \forall i \in \{1, \cdots, M\}$$

if the number of samples K satisfies

$$K \ge \frac{2V}{\Delta}\left(\frac{1}{\mu} + \frac{1}{3}\right) \ln\left(\frac{2M}{\delta}\right).$$

Proof. For the i-th link, we define K random variables

$$X_j^{(i)} = w_i(F_j)\frac{V}{K} \quad \forall j \in \{1, \cdots, K\}.$$

Since $F_1, \cdots, F_K$ are i.i.d., $X_1^{(i)}, \cdots, X_K^{(i)}$ are i.i.d., too. Besides, we have

$$\begin{aligned}\mathbb{E}\left[X_1^{(i)}\right] &= \sum_{r=1}^{N} w_i(f_r)\frac{V}{K}\mathbb{P}(F_1 = f_r) \\ &= \sum_{r=1}^{N} w_i(f_r)\frac{V}{K}\frac{v_r}{V} \\ &= \frac{1}{K}\sum_{r=1}^{N} w_i(f_r) v_r = \frac{l_i}{K}\end{aligned}$$

and

$$\begin{aligned}\mathrm{Var}\left[X_1^{(i)}\right] &= \mathbb{E}\left[\left(X_1^{(i)}\right)^2\right] - \left(\mathbb{E}\left[X_1^{(i)}\right]\right)^2 \\ &\le \sum_{r=1}^{N}\left(w_i(f_r)\frac{V}{K}\right)^2 \mathbb{P}(F_1 = f_r) \\ &\le \sum_{r=1}^{N} w_i(f_r)\frac{V^2}{K^2}\frac{v_r}{V} = \frac{V}{K^2} l_i.\end{aligned}$$

According to Lemma 1, we know

$$L_i = \sum_{i=1}^{K} w_i(F_k)\frac{V}{K} = \sum_{i=1}^{K} X_i^{(i)},$$

which implies $\mathbb{E}[L_i] = K\mathbb{E}\left[X_1^{(i)}\right] = l_i$ as well as $\text{Var}[L_i] = K\,\text{Var}\left[X_1^{(i)}\right] \leq \frac{V}{K} l_i$.

As $0 \leq X_1^{(i)} \leq \frac{V}{K}$ holds almost surely, the use of Bernstein's equalities on $\left\{X_j^{(i)} - \frac{l_i}{K}\right\}_{j=1}^{K}$ yields

$$\begin{aligned}\mathbb{P}\left[\left|L_i - l_i\right| \geq t\right] &\leq 2\exp\left(-\frac{\frac{1}{2}t^2}{\text{Var}[L_i] + \frac{1}{3}\frac{V}{K}t}\right)\\ &\leq 2\exp\left(-\frac{K}{V}\frac{3t^2}{6l_i + 2t}\right)\end{aligned}$$

for any $t > 0$.

Taking t as $\max\{\Delta, \mu l_i\}$, we have

$$\begin{aligned}\mathbb{P}\left[\left|L_i - l_i\right| \geq \max\{\Delta, \mu l_i\}\right] &\leq 2\exp\left(-\frac{K}{V}\left(\frac{2l_i}{t^2} + \frac{2}{3t}\right)^{-1}\right)\\ &\leq 2\exp\left(-\frac{K}{V}\left(\frac{1}{\mu}\frac{2}{\Delta} + \frac{1}{3}\frac{2}{\Delta}\right)^{-1}\right).\end{aligned}$$

So

$$\mathbb{P}\left[\left|L_i - l_i\right| \geq \max\{\Delta, \mu l_i\}\right] \leq \frac{\delta}{M}$$

as given

$$K \geq \frac{2V}{\Delta}\left(\frac{1}{\mu} + \frac{1}{3}\right)\ln\left(\frac{2M}{\delta}\right).$$

Finally, by the union bound,

$$\begin{aligned}&\mathbb{P}\left[\forall i \in \{1, \cdots, M\}\ \left|L_i - l_i\right| < \max\{\Delta, \mu l_i\}\right]\\ &= 1 - \mathbb{P}\left[\exists i \in \{1, \cdots, M\}\ \left|L_i - l_i\right| \geq \max\{\Delta, \mu l_i\}\right]\\ &\geq 1 - \sum_{i=1}^{M}\mathbb{P}\left[\left|L_i - l_i\right| \geq \max\{\Delta, \mu l_i\}\right]\\ &\geq 1 - M \times \frac{\delta}{M} = 1 - \delta,\end{aligned}$$

which proves the theorem. □

This theorem bounds either the relative error or absolute error. As a corollary, we can bound only the relative error for links whose volumes are greater than a threshold value.

Corollary A.1. *Given a maximum relative error μ, a threshold volume V_T, and a confidence parameter δ, the inequality $\left|L_i - l_i\right| \leq \mu l_i$ holds for all links with volumes above the threshold (*i.e.*, $l_i \geq V_T$) with probability at least $1 - \delta$, if the number of samples K satisfies*

$$K \geq \frac{2V}{V_T}\left(\frac{1}{\mu^2} + \frac{1}{3\mu}\right)\ln\left(\frac{2M}{\delta}\right).$$

Proof. Setting $\Delta = \mu V_T$, we have

$$\left|L_i - l_i\right| \leq \max\{\Delta, \mu l_i\} = \mu l_i$$

for all links satisfying $l_i \geq V_T$. Therefore, applying Theorem A.1 directly proves this corollary. □

Thus far, we have demonstrated the theorem for a single failure model and a single traffic snapshot. However, it is straightforward to extend these results to accommodate multiple models and snapshots. This can be achieved by treating links associated with different failure models or traffic snapshots as distinct entities. As a result, instead of examining M interfaces, we consider $M \times A \times B$ interface-failure-snapshot combinations, where A is the number of failure models and B is the number of snapshots. The proof of the theorem remains valid. It is worth noting that the parameter M appears within the logarithmic function in the inequality. Thus, its increase has a minimal impact on the overall result.

Evaluation. The proven bound is practical enough for use in our WAN. In network N2 (as defined in §7), we employ a flow sample ratio of 11% to expedite the checking process by a factor of 3.2. The application of the theorem ensures that the error is limited to either 150 Mbps in absolute terms or 2.4% in relative terms for all links, with probability at least 99%. In our experiments (as detailed in §7), across 10 trials, either the relative error does not exceed 1.1% or the absolute error does not exceed 60 Mbps for all links in network N2.

B Traffic Distribution Graph

In this appendix, we will formally define the traffic distribution graph (*i.e.*, TDG, first presented in §4.1), and then detail our TDG construction algorithm.

B.1 Formal Definition

Formally, let $\mathcal{F}$ denote the space of all flows, $\mathcal{V}$ denote the set of all VRFs in the network, I denote the set of all interfaces, $\mathcal{IP}$ denote the set of all IP addresses, and $\mathcal{T}$ denote the set of tunnels (*e.g.*, established by SR and MPLS) in the network. To differentiate the direction of a flow at an interface, we use $I^* = I \times \{in, out\}$ to denote the set of interfaces labeled with flow directions. A traffic distribution graph is defined as a triple (N, E, w), where $N \subset \mathcal{F} \times (I^* \cup \mathcal{V}) \times (\mathcal{IP} \cup \mathcal{T} \cup \{\perp\})$ is the set of nodes in the graph, $E \subset N \times N$ is the set of edges, $w : E \to R^{[0,1]}$ is the weight function where $R^{[0,1]}$ denotes the set of real numbers ranging from 0 to 1 (inclusive).

B.2 TDG Construction

Algorithm 5 shows the overall construction algorithm. Function CONSTRUCTTDG takes a set of located flows as input and constructs the TDG in a standard BFS fashion as shown in function BFSEXPLORE. Function EXPLORENODE takes a newly generated node and explores new edges and nodes according to the network's behavior on that node. Specially, we currently support the following five types of flow processing used in our production network.

• (Line 12-14) When a flow is received at an interface, a PBR policy defined on that interface may be applied to the flow. In this case, the algorithm needs to generate a node corresponding to the PBR's modification to the flow and its

Algorithm 5: Construction of TDG

```
Function ConstructTdg($\{(f_k, i_k)\}_{k=1}^m$):
    $S \leftarrow \{(f_k, i_k, \perp) : k = 1,..,m\}$; $N \leftarrow \emptyset$; $E \leftarrow \emptyset$; $w$ is undefined;
    BfsExplore($S$, $N$, $E$, $w$);
    return $(N, E, w)$;
Function BfsExplore($S$, $N$, $E$, $w$, $Rib$):
    while $S$ is not empty do
        Pop a node $n$ from $S$ and add it to $N$;
        ExploreNode($n$, $N$, $E$, $w$, $Rib$);
Function ExploreNode($n$, $N$, $E$, $w$, $Rib$):
    Let $n = (f, x, h)$;
    switch $(f, x, h)$ do
        case $x$ is an interface and $f$ is incoming to $x$ do
            Generate a node $n' = (f', y, h)$ if the PBR policy defined on $x$
              modifies $f$ to $f'$ and change its location to $y$;
            Add $e = ((f, x, h), (f', y, h))$ to $E$ and set $w(e)$ to 1;
        case $x$ is an interface and $f$ is outgoing do
            Generate a node $n' = (f, y, h')$ where $y$ is the interface
              connected with $x$, $h'$ is $h$ if $h$ is a tunnel otherwise $\perp$;
            Add $e = ((f, x, h), n')$ to $E$ and set $w(e)$ to 1;
        case $x$ is a VRF and $h$ is an SR tunnel do
            If the router is the endpoint of $h$, then generate a node
              $n' = (f, x, \perp)$; otherwise generate a node $n' = (f, y, h)$
              where $y$ is the outgoing interface of $h$ on the corresponding
              router;
            Add $e = ((f, x, h), n')$ to $E$ and set $w(e)$ to 1;
        case $x$ is a VRF and performs BGP matching for $f$ do
            Generate nodes $n_k = (f, x, h_k)$ for all BGP next hops $h_k$ using
              $Rib$;
            Add $e_k = ((f, x, h), n_k)$ to $E$ and set $w(e_k)$ to $1/m$ for all $k$
              where $m$ is the total number of next hops;
        case $x$ is a VRF and $h$ is an indirect next hop do
            Generate nodes $n_k = (f, x, h_k)$ for all resolved next hops $h_k$
              (can be IP addresses or tunnels) using $Rib$;
            Add $e_k = ((f, x, h), n_k)$ to $E$ and set $w(e_k)$ according to the
              weight defined on corresponding the SR policy or IS-IS;
    Add new generated nodes into $S$ if they are not in $N$;
```

location (*e.g.*, redirect the flow to a VRF), and add an edge with weight 1 to the node.

• (Line 15-17) When a flow is ready to be forwarded out to another router, the algorithm generates a node for the receiving interface and adds an edge with weight 1.

• (Line 18-20) When a flow is forwarded in an SR tunnel, the algorithm needs to generate nodes based on whether the flow reaches the endpoint. If so, the algorithm generates a node with next hop $\perp$ indicating that the flow reaches the endpoint of the tunnel and needs further lookup for the next hops; otherwise, the algorithm generates a node with the same next hop and the corresponding outgoing interface. The weight of the edge is set to 1 in both cases.

• (Line 21-23) When a flow needs to look up the BGP RIB on a VRF, the algorithm needs to generate nodes for all next hops. To reflect the BGP ECMP mechanism implemented by router vendors, the weights of the edges to those nodes should be set equally.

• (Line 24-26) If an indirect next hop needs to be resolved by IS-IS or SR, the algorithm needs to generate nodes for each resolved next hop. The weights on the edges to the nodes should be correctly set based on the SR configuration or the IS-IS protocol (*e.g.*, normal ECMP).

C Traffic Equivalence Class

In this appendix, we will formally define the traffic equivalence class (*i.e.*, TEC, first presented in §4.4), and then detail our TEC construction algorithm.

C.1 Formal Definition

We define traffic equivalence class (TEC) using the notion of TDG. Given a TDG G, we use $G[f_1 \mapsto f_2, i_1 \mapsto i_2]$ to denote the TDG obtained by changing f_1 to f_2 and i_1 to i_2 for all nodes in G.

We first define the notion of *flow equivalence*.

Definition C.1. *Two located flows* $(f_k, i_k), k = 1, 2$ *are equivalent, denoted as* $(f_1, i_1) \equiv (f_2, i_2)$*, iff* $G_1 = G_2[f_2 \mapsto f_1, i_2 \mapsto i_1)]$ *and* $G_2 = G_1[f_1 \mapsto f_2, i_1 \mapsto i_2)]$*, where* $G_k =$ ConstructTdg$(\{f_k, i_k\})$ *is the TDG constructed for* (f_k, i_k).

Now we define the notion of traffic equivalence class below.

Definition C.2. *A set of located flows* $\{(f_k, i_k), k = 1, \cdots, m\}$ *forms a traffic equivalence class, if* $(f_j, i_j) \equiv (f_k, i_k)$ *for all* $j, k \in \{1, \cdots, m\}$.

C.2 TEC Generation

Ideally, to achieve optimal efficiency we may want each TEC to contain as many flows as possible. However, this task is computationally hard due to the large number of flows. To balance computational efficiency with the optimality of TECs, we adopt a *compositional* approach. We establish equivalence classes for the value space of each field based on a global view of the network and then compose these classes together.

Let D and S be the sets of all prefixes (which can be efficiently collected from RIBs and router configurations) used to match destination and source IPs, respectively. Two flows $((s_1, d_1, v), i_1)$ and $((s_2, d_2, v), i_2)$ are considered equivalent if (i) the longest prefix match in S (D, resp.) for s_1 (d_1, resp.) is the same as that for s_2 (d_2, resp.) and (ii) two interfaces i_1 and i_2 sit in the same VRF and are configured with the same PBR policy. The correctness relies on the fact that at any router, there must be a subset of prefixes $D' \subset D$ ($S' \subset S$) used to match destination (source) IPs; therefore, d_1 and d_2 (s_1 and s_2) still share the same longest prefix match, leading to the same forwarding behavior and traffic distribution. While establishing equivalence for DSCPs is feasible, we do not further consider such equivalence, since our WAN only uses a small number of them.

D Incremental TDG Construction and Traffic Simulation

In this appendix, we provide the detailed description of our incremental TDG construction and traffic simulation algorithms, in §D.1 and §D.2, respectively.

Algorithm 6: Incremental TDG construction

```
1 Function IncUpdateTdg(($N,E,w$), $\Delta Rib$, $Rib'$):
2     $N' \leftarrow N$; $E' \leftarrow E$; $w' \leftarrow w$;
3     let $chg$ be the set of nodes $n \in N$ s.t. $n$ representing the
        matching of some $r \in \Delta Rib$ or a failed router/link;
4     forall $n \in chg$ do
5         remove $e$ from $E'$ and $w'(e)$ ←undefined for all $n$'s
            outgoing edge $e$;
6     BfsExplore($chg$, $N'$, $E'$, $w'$, $Rib'$);
7     return $G' = (N', E', w')$, $chg$
```

D.1 Algorithm for Incremental TDG Construction

Algorithm 6 presents how to incrementally construct a TDG for a given failure model. The algorithm takes the old TDG, the set of changed routes (*e.g.*, next hops changed) and tunnels (*e.g.*, tunnel removed) ΔRib, and the new RIBs (denote Rib') for the failure as input, and generates the new TDG incrementally. Note that simulating Rib' and ΔRib is easy, since many existing systems [12,41] can extract the information.

In the first step, we find the nodes that may change in the given TDG. As described above, a node changes if (i) it represents route/tunnel matching for some r in ΔRib; or (ii) its location field is associated with a failed link or router (*e.g.*, the location field is an interface of some failed link). To efficiently identify those nodes, we maintain a mapping from routers, links, and routes/tunnels respectively to nodes in the TDG, when constructing a TDG (in the basic part).

Second, for those changed nodes *chg*, the algorithm removes all their outgoing edges and invalids all associated weights, since they may be inconsistent with the failure case. To reflect the new behavior under the failure, the algorithm performs the standard BFS algorithm from those changed nodes to construct the new edges and nodes using the new RIBs Rib'. Finally, the algorithm returns a new TDG with the set of changed nodes *chg* for the post processing (§5.2).

D.2 Detailed Description of Incremental Traffic Simulation Algorithm

Algorithm 3 shows the incremental traffic simulation for a single traffic snapshot. Using the approach described in §4.3, it can be naturally generalized to support multiple snapshots.

Algorithm 3 takes two parts of input: (i) the information computed in the basic part including the old TDG G, the volume V for nodes and edges in G, and the traffic distribution TL for all links, and (ii) the new TDG G' and the changed nodes *chg* computed by the incremental TDG construction (as detailed in §5.1).

The algorithm then computes the volume V' on nodes and edges incrementally based on the old volume V and also updates the traffic distribution TL incrementally based on the change of volume on edges. Similar to the traffic simulation algorithm in the basic part, the incremental algorithm computes the new volume of nodes and edges following the topological order of the new TDG G' but only for the affected nodes (line 6-16). Initially, the affected nodes α contains all nodes in *chg* and those having incoming edges from *chg* (line 5) as described above, and keeps adding potentially affected nodes to α. For an affected node n, the algorithm updates its volume (line 9) and computes the volume of all its outgoing edges in G' (line 11). Then the algorithm checks if any outgoing edges' volumes get changed for all the edges in both G and G'. If some edge's volume has changed, the node on the other end of the edge may become affected and so the algorithm adds that node to α (line 15). Furthermore, the algorithm also updates the traffic distribution by adding the changed volume to the corresponding link if needed. Finally, the updated TL contains the traffic distribution for all links under the failure and is returned.