RD-Probe: Scalable Monitoring With Sufficient Coverage In Complex Datacenter Networks

Rui Ding[1] Xunpeng Liu[1] Shibo Yang[1] Qun Huang[1*]
Baoshu Xie[2] Ronghua Sun[2] Zhi Zhang[2] Bolong Cui[2]
[1]School of Computer Science, Peking University [2]Huawei Technologies

ABSTRACT

Ensuring service availability in large-scale datacenters hinges on network monitoring. For monitoring quality, it is essential to attain sufficient coverage of all physical components. However, given the ever-evolving complexity of industrial environments, even measuring coverage metrics becomes challenging, let alone attaining *sufficient coverage*. In fact, insufficient coverage widely existed in our production datacenters and caused many missed failures.

To address this, we design RD-Probe, an industrial monitoring system with coverage and scalability guarantees. Specifically, it first constructs a network topology encoding the industrial complexity. Then, it combines Randomized and Deterministic methods to efficiently generate probe tasks that meet the coverage requirement.

We have deployed RD-Probe in three large production regions in Huawei Cloud. Within the first month, RD-Probe improved coverage from 80.9% to 99.5% and unearthed several previously unnoticed issues while tolerating numerous faults. Large-scale simulation of four industry solutions shows that RD-Probe is the only one achieving both sufficient coverage and scalability in complex datacenter networks. We plan to expand RD-Probe to other regions soon.

CCS CONCEPTS

• **Networks** → **Network measurement**; **Network monitoring**.

KEYWORDS

Network measurement; Active monitoring; Datacenter networks

ACM Reference Format:
Rui Ding, Xunpeng Liu, Shibo Yang, Qun Huang, Baoshu Xie, Ronghua Sun, Zhi Zhang, and Bolong Cui. 2024. RD-Probe: Scalable Monitoring With Sufficient Coverage In Complex Datacenter Networks. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3651890.3672256

1 INTRODUCTION

Network monitoring is vital for ensuring service availability in datacenter infrastructures [4, 5, 8, 16, 30, 42, 43, 76, 83]. To enhance monitoring effectiveness, we should attain *sufficient coverage* of all physical components in the underlay network [54, 57, 76]. Otherwise, with insufficient coverage, monitoring blind spots will emerge [57]. They cause diagnostic tools [19, 37, 72] to overlook potential failures, leading to degraded quality of service and eventual revenue loss. Indeed, analysis of our production network reveals that insufficient coverage on Layer 2 is the culprit for many missed failures and their prolonged MTTRs (mean-time-to-repair) (§2.2).

However, achieving sufficient coverage on Layer 2 is challenging within the production monitoring architecture. Three factors from industrial practices contribute to the challenges.
(1) *Virtualization techniques* are broadly utilized in the infrastructure to realize device flexibility and redundancy [1, 9, 56]. They abstract low-level physical components. However, they also occlude these components from monitoring systems that run on higher layers, making sufficient coverage on Layer 2 intractable.
(2) *Randomness techniques* are widely adopted to pursue load balance. They introduce nondeterministic forwarding and routing behaviors, causing unpredictable coverage variations. Derandomizing these techniques is challenging because the random seeds and secret keys used in their implementation are usually unknown from outside [36, 45, 76].
(3) *Path explosion* results from the rapidly increasing scales and device redundancy. For example, AWS provides an estimate of $O(10^{87})$ for intra-region link-path combinations and $O(10^{176})$ for inter-region link-path combinations [18]. At this formidable scale, any deployable solution must strive to balance the coverage level and resource overhead.

Unfortunately, existing solutions do not address sufficient Layer-2 coverage. We classify these solutions into *black-box* and *white-box* monitoring [34, 49]. Specifically, black-box monitoring examines externally visible switch information by injecting probe tasks. However, they only consider Layer-3 interfaces [6, 54, 69], and hardly extend to Layer 2 given the aforementioned complexity. White-box monitoring collects internal switch status (e.g., logs [24]) but depends on built-in monitoring capabilities, which do not cover all physical components like various switch cards. Therefore, we consider white-box monitoring a good complement and focus on black-box monitoring in this paper.

With such research and industry landscape in mind, we ask *"Given the current complexity of datacenter networks, can we design a monitoring system that guarantees sufficient coverage on Layer 2 without sacrificing scalability?"* We give an affirmative answer by presenting RD-Probe, a deployed black-box system with explicit coverage and scalability guarantees. It comprises two steps in sequence to tackle challenges.
(1) *Topology construction:* A *vanilla topology* whose nodes and edges have one-to-one mappings to Layer-3 switches and links cannot reflect infrastructure-level virtualization and randomness techniques.

*Qun Huang is the corresponding author.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672256

Our idea is to leverage switch configurations to transform it into a *de facto topology*. Specifically, a configuration is formatted text showing customized parameters of a technique. We parse configurations from bottom to top in network layers and modify the vanilla topology correspondingly. The eventual topology captures low-level components and their expected networking behaviors.
(2) *Probe generation:* Given the de facto topology, we generate probe tasks satisfying coverage and scalability requirements. There is a natural trade-off: A higher coverage demands more probing resources, challenging the system's scalability. To satisfy these requirements, we introduce a novel *two-phase* generation approach that combines *randomized* and *deterministic* methods. Specifically, the randomized phase selects probe tasks with random sources and destinations. It aims for an even coverage across switch components. Although it cannot guarantee sufficient coverage, it greatly reduces the planning complexity as most components will be covered in this phase. Next, the deterministic phase complements the first phase to fulfill the coverage requirement. It selects tasks that go through components demanding further coverage.

We have deployed RD-Probe in three regions for over six months, each containing $O(10^3)$ switches and $O(10^4)$ links and adopting various techniques. It fills in the coverage gap that widely existed in our datacenters. Furthermore, in the first month RD-Probe unearthed several previously unnoticed issues that dragged down network performance in the past. It has tolerated numerous faults that occurred during its launch and operation. We plan to replace our old black-box system with RD-Probe in other regions soon.

We obtain insights into RD-Probe on even larger regions by conducting simulation studies involving $O(10^4)$ switches and $O(10^5)$ links. We further compare RD-Probe with four state-of-the-art solutions that also have large-scale deployment. We implement related techniques and inject equivalent behaviors into our simulator. Moreover, we customize some minor logic of the compared solutions since they were designed with a clean-slate view of datacenters in mind. Results show that RD-Probe is the only solution fulfilling the coverage and scalability requirements in complex datacenters.

2 BACKGROUND AND MOTIVATION

2.1 Monitoring Architecture in Production

Figure 1 shows the monitoring architecture in Huawei Cloud's datacenter networks, which host over $O(10^5)$ switches and $O(10^6)$ servers. This architecture has six modules.
(1) *Task generation* is in charge of generating probe tasks. It first fetches information about switches, NICs, and configurations from CMDB (configuration management database). It then selects healthy probe agents based on the health information from the health check module. A probe agent runs on a server. It executes and manages all the probe tasks assigned to that server. Each probe task injects synthetic traffic into the network by exercising one of ICMP, TCP, UDP, and IP-in-IP pings but is not limited to these. After exercising probe tasks, the module fetches coverage metrics from data storage.
(2) *Health check* discovers machines in a degraded state so that task generation will avoid the agents on them. It consumes two kinds of data: heartbeats sent from agents and historical statistics computed by the offline analysis module.

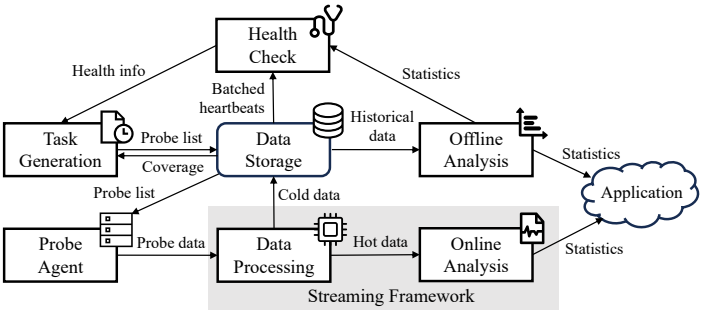


Figure 1: Monitoring architecture in Huawei Cloud's datacenter networks.

(3) *Data storage* persists probing-related data, which include system logs, switch configurations, and probe lists. It also batches heartbeats from agents and feeds them to the health check module.
(4) *Probe agents* periodically pull the latest probe list from the storage, parsing task types and source and destination agents. They set a unique identifier in the IP payload to distinguish probe packets from others. The default probe frequency and duration are one packet per second and three minutes, respectively, aligned with other industrial practices [54, 69].
(5) *Streaming framework* processes agents' data. It has two submodules. (i) The data processing submodule performs data cleansing, labeling, and simple calculations. It further distinguishes between hot and cold data. (ii) The online analysis submodule analyzes hot data to address requests with real-time requirements.
(6) *Offline analysis* extracts statistics from historical data, including logs and heartbeats. It handles requests that tolerate longer delays.

All the data shown in Figure 1 are tagged with a monitoring epoch. This is because probe tasks are generated every epoch, and agents only exercise current tasks till the end of an epoch. An epoch spans three minutes by default, in alignment with the default probe duration. Previous industry solutions adopted basically the same epoch length [54, 69].

2.2 Coverage Requirement

Meeting the coverage requirement is crucial within this monitoring architecture. We specify coverage requirements by deciding an appropriate granularity and the extent of sufficient coverage at this granularity. Next, we show how industrial complexity hinders achieving sufficient coverage. Finally, we present the consequences of insufficient coverage observed in our datacenters.

Port-level coverage. We choose granularity on the *port* level instead of the traditional *interface* level. Here, a *port* refers to a physical Layer-2 port, which Layer-3 interfaces can build upon as the destination or gateway of routing. The reason is two-fold. (i) Each port should be monitored as it has dedicated hardware resources like NIC and buffer. If we overlook some ports, we will be doomed to miss the runtime status of some switch components. (ii) Sufficient monitoring of switch ports erases blind spots. Specifically, by monitoring different ports on the same switch, we capture the runtime status of switch fabrics; By monitoring directly connected ports, we capture the runtime status of links. In contrast, we cannot erase blind spots if we only cover Layer-3 interfaces. This is because a Layer-3 interface can be a bundle of multiple Layer-2 ports. Some member ports can still be unmonitored.

Convention. In this paper, a *port* and an *interface* always mean a Layer-2 and 3 component, respectively. Coverage refers to the port level unless otherwise stated.

Sufficient coverage. We require that an overwhelming majority of ports are sufficiently covered. We define this requirement formally as an ordered pair $(\alpha, p) \in \mathbb{N} \times \mathbb{R}_{(0,1)}$, meaning that the portion of Layer-2 ports covered by $\geq \alpha$ different tasks is at least p (typically, $\alpha \in [1, 5]$). Next, sufficient coverage means $p \geq 99\%$ given the operators' desired α, which indicates good enough coverage from our experience (§6). Note that requiring $p = 1$ is not pragmatic due to two reasons. (i) Current industrial practices make the resource overhead for attaining $p = 1$ overly high. (ii) Operational benefits gained from strengthening $p \geq 99\%$ to $p = 1$ is only marginal (§6.3).

Challenges. However, sufficient coverage faces two challenges that stem from industrial practices.

• **(C1) Complexity.** Due to virtualization and randomness techniques, sufficient port-level coverage is much harder to achieve than the interface-level one. Specifically, virtualization techniques prevent the monitoring architecture from seeing switch ports directly. Randomness techniques induce unpredictable coverage variations across ports. We discuss four techniques adopted in our datacenters, which are also widely used in other long-running datacenters.

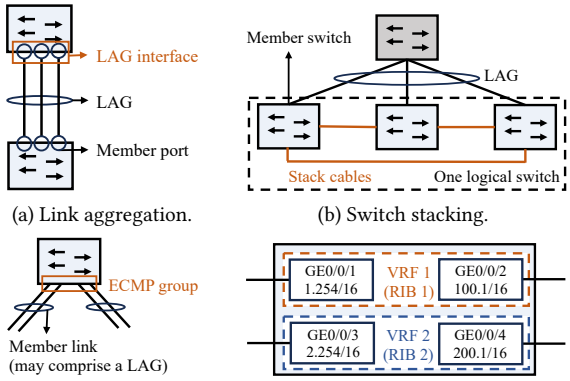
(1) *Link aggregation* introduces Layer-2 virtualization and randomness (Figure 2(a)). For virtualization, it bundles multiple ports into a logical interface called a link aggregation group (LAG). For randomness, it implements a Layer-2 hash to select a member port in the LAG to forward a packet. The concrete hash and forwarding port are invisible from the upper layers, so monitoring systems can only determine the traversed LAG of a packet, but not the specific port. Therefore, link aggregation precludes the precise calculation of port-level coverage. Moreover, switch vendors generally do not disclose the detailed hash and per-packet port selection within a LAG due to proprietary technology and security concerns.

(2) *Switch stacking* introduces Layer-2 virtualization (Figure 2(b)). It interconnects multiple physical switches with stack cables to form a logical switch. The stacked switch is multi-homed to other devices (e.g., the gray switch), with all the connection links forming a LAG. Regarding port-level coverage, we must consider peer ports connected via stack cables. These internal ports are invisible to Layer 3 or above.

(3) *Equal-cost multipath routing (ECMP)* introduces Layer-3 randomness (Figure 2(c)). Specifically, interfaces on a switch are assigned to different ECMP groups. When an IP prefix matches an ECMP group, a Layer-3 hash chooses one next hop (or member link) within the group. The member link may comprise a LAG, so the final forwarding port is collectively determined on Layers 2 and 3. Switch vendors typically provide white-box ECMP metrics up to the interface level.

(4) *Virtual routing and forwarding (VRF)* introduces Layer-3 virtualization (Figure 2(d)). It creates multiple logically isolated switches on a physical switch by assigning each interface to a VRF plane. Each plane has a separate routing information base (RIB), so it cannot access other planes. VRF renders many seemingly connected paths (in the physical topology) invalid. It restricts the space of probe tasks and hardens the achievement of sufficient coverage.

• **(C2) Scalability.** Maintaining scalability while fulfilling coverage requirements is difficult due to path explosion. As there are more



(a) Link aggregation. (b) Switch stacking. (c) ECMP routing. (d) Virtual routing and forwarding.

Figure 2: Virtualization and randomness techniques.

Month	MTTR	Root Cause	Details
Nov./21	10.5 h	Port flapping	One of the eight aggregated ports flapped
Aug./22	15.7 h	Chip error	Slot 4, Chip 1 malfunctioned on a core switch
Dec./22	41.5 h	Chip error	VXLAN inner address overrode in LPU 1
Feb./23	3.5 h	Chip error	SerDes lane connected to Chip 1 of SFU 6 was down

Table 1: Example failures overlooked due to insufficient coverage (LPU: line processing unit, SFU: switch fabric unit).

and more switches and redundancies, two critical resources dominate the overhead: probe volume and probe planning time. Thus, we use them to characterize scalability. Other resource consumption is minor in black-box monitoring (e.g., switch CPU utilization).

(1) *Probe volume* refers to the amount of probe data collected during monitoring. It is proportional to the task number. A higher volume consumes more probing resources on agents and aggravates the data processing burden.

(2) *Probe planning time* refers to the time spent on generating tasks. It determines the delay from observing a change in the network (e.g., physical rewiring or configuration update) until running new tasks. The ideal time should be within $O(1)$ minutes for timeliness.

Consequence of insufficient coverage. Insufficient coverage entails unnoticed failures on the network side. We analyzed failure events in our production network over the past two years. We found a typical troubleshooting pattern among the worst failures: Users complained about performance anomalies while our monitoring systems raised no alarms. Then, engineers had to test devices on the degraded path one by one and eventually narrowed down the failure to a buggy port with insufficient coverage.

Table 1 shows four example events overlooked due to insufficient coverage. They had notably long MTTRs ranging from several hours to nearly two days. The datacenter only achieved a coverage of $(1, 80.9\%)$. All these failures occurred in the remaining 19.1% of ports. Take the port flapping failure for example. Only one port in an eight-port LAG alternated between up and down state every 30 seconds. Although 12 probe tasks went through this LAG, none was hashed to the flapping port. Meanwhile, no white-box metrics (e.g., from SNMP, Syslog, and Netflow) showed anomalies. However, improving this coverage is nontrivial. The datacenter ran a Pingmesh-like [30] system with three tasks between each pair of servers. If we increase it to four, some agents will overload and fail to work properly.

2.3 Limitations of Prior Systems

Our datacenters have adopted numerous black-box, white-box, and verification tools. However, they did not address the challenges of sufficient coverage. The black-box tool we previously adopted was similar to Pingmesh [30]. The adopted white-box tools include SNMP [12], Syslog [24], NetFlow [13, 55], BGP monitoring protocol (BMP) [59], ERSPAN [21], and so on. The adopted verification tool is Batfish [10].

Black-box monitoring. It treats switches as black boxes and only examines externally visible behaviors. The common practice is to send probe packets from end hosts or vantage points [4, 5, 15, 16, 42, 47, 76]. Since it offers the same perspective as users, we use black-box metrics as our primary information to detect anomalies.

Prior black-box methods cannot address the two challenges (§2.2). (i) Their intended coverage is up to the interface level. Due to various low-level techniques, there is no direct way to extend their coverage to the port level. Consequently, they cannot achieve sufficient port-level coverage. (ii) They face scalability issues at today's scales. We give two examples. First, tomography solutions [6, 11, 14, 23, 25, 54, 64] search for a minimal set of paths that can infer the status of individual links. These paths often satisfy multiple objectives like path independence and link identifiability. Even with heuristics, this process suffers from massive computation and prolonged planning due to path explosion. Second, Pingmesh [30] and its descendant [69] send probes in a full-mesh manner, incurring prohibitive volume. Indeed, a recent report [57] by Microsoft itself indicates the scaling concern of Pingmesh.

White-box monitoring. It exploits built-in features to observe switch internals. There are three directions: sampling, in-network telemetry (INT), and measurement. (i) Sampling-based methods passively sample traffic and analyze traffic correlations across switches [13, 17, 55, 58, 60, 61, 65, 73]. (ii) INT-based methods utilize switch programmability to tag packets with switch-local metrics, such as switch/link ID [32, 66–68], timestamps [8, 40], counters [53, 71], and custom data [38, 84]. They collect metrics at edge routers or end hosts. (iii) Measurement-based methods utilize switch programmability to measure traffic statistics in the data plane [31, 50, 74]. They use memory-efficient data structures like sketches [43, 44, 74, 81], hash tables [35, 48, 63], and ring buffers [39, 83].

Backed by other industrial guides [62], we only analyze white-box metrics to troubleshoot root causes when black-box metrics indicate network issues. Three reasons explain this choice. (i) White-box methods build on switches' built-in capabilities. It overlooks anomalies outside such capabilities, like NIC-related silent drops and blackholes. It is also impractical to ask the switch manufacturer to expose the metrics of every component due to proprietary and security concerns. (ii) White-box methods are more delicate and intrusive than black-box methods. Proper implementation and verification are labor-intensive (e.g., special ASICs required by INT methods). Moreover, richer switch functionalities often correlate with decreased reliability [2, 26]. (iii) White-box metrics come in large volumes. Processing them and resolving inconsistency (which arises from noise in switch counters and various delays) incur a high overhead [62]. While exporting only coarse-grained or aggregated metrics reduces this overhead, it does so at the cost of losing low-level coverage and missing abnormal events.

Network verification. It verifies whether the data and control planes produce the expected behaviors or have the expected properties [7, 75, 77–79, 82]. It typically aims at static properties of the network (e.g., compliance with policies and absence of misconfigurations). In contrast, our work aims at dynamic and continuous monitoring of the underlay network via black-box probing.

2.4 Design Specification of a New System

Due to prior limitations, we have an urge to design a new black-box system to satisfy the coverage requirement. The new system should (i) fit into the existing monitoring architecture and (ii) address the two challenges. We state its design specifications as follows.

Input. It consists of the switch configurations (from CMDB) and coverage requirement (α, p) specified by operators. As configuration updates are incremental, CMDB only notifies the system of the changes since the last epoch.

Output. It is the complete probe list in the current epoch. Note that the task generation module does not have to generate all tasks at once. It can extend existing tasks by appending new tasks to the probe list. The only cost is that the appended tasks will run for less time (since they expire at the end of the epoch).

Assumptions. We make two mild assumptions about the target datacenters, which are already satisfied in ours. (i) It has a multi-tier architecture. Specifically, each switch connects to devices in the higher/lower/same layer via up/down/peer links without cross-layer links. Numerous datacenter designs [27–29, 51] satisfy this assumption. We also recognize the emergence of spineless datacenters [33, 41], and we leave the extension to such architecture as our future work. (ii) We assume that the combined effect of ECMP and link aggregation control protocol (LACP) is that each flow will be hashed to one port in an ECMP group uniformly and independently [70, 76, 80]. This encourages load balance. Operators have a natural incentive to configure switches to approach this effect.

3 RD-PROBE OVERVIEW

We propose RD-Probe, a complexity-aware and scalable monitoring system that guarantees sufficient coverage. As shown in Figure 3, RD-Probe seamlessly integrates with the existing monitoring architecture. It only modifies the task generation and data processing modules. Specifically, the task generation module constructs a de facto topology, which reflects low-level complexity and serves as the base of probe generation. Then, probe generation fulfills coverage and scalability requirements in two phases. The data processing module computes coverage counters, thereby providing feedback to the generation module.

Topology construction (§4.1). To address (C1), RD-Probe encodes complexities into a de facto topology, which can be vastly different from the physical topology. It is an attributed graph whose nodes correspond to different routing planes on physical switches. Its links correspond to peer ports with direct physical connection. Its attributes indicate possible choices of routing given certain network prefixes. To obtain such a topology, we start with a vanilla one whose nodes and links match the Layer-3 topology. We incrementally modify it by translating related switch configurations into equivalent graph operations from bottom to top.

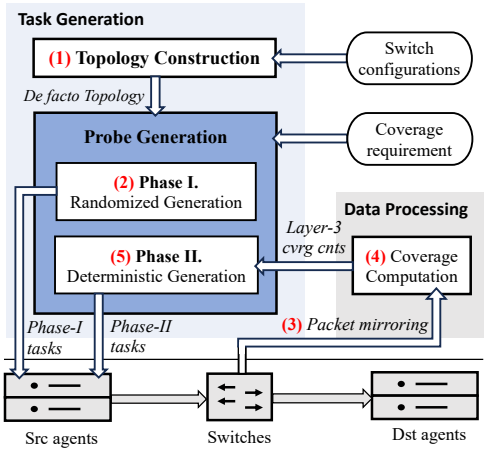


Figure 3: RD-Probe's design.

Previous solutions only use the vanilla topology as the base of probe generation. This is fine for simple datacenters that avoid low-level virtualization and randomness techniques. In such datacenters, the meaning of port-level and interface-level coverage coincides, and RD-Probe can guarantee 100% coverage in this scenario (§4.3). However, this is untrue for many long-running datacenters with accumulated complexity, especially ours.

Probe generation (§4.2-4.3). To address (C2), we propose a novel two-phase generation scheme. Specifically, Phase I uses randomized generation, which dispatches tasks to randomly selected agents. Its goal is that most ports will reach α-coverage quickly, so the coverage problem narrows down to a small subset of ports in Phase II. This goal is achievable owing to two observations. (i) Given randomness techniques, Phase-I tasks can cover different ports evenly. (ii) Given the coverage requirement, we can quantify the required task number in Phase I using the theory of coupon collector [46]. This number is orders of magnitude lower than path numbers. We rigorously prove that Phase I realizes its goal (§4.2).

Next, Phase II uses deterministic generation, whose tasks complement Phase I and finally fulfill the coverage requirement. It has two steps. (i) It locates the under-covered interfaces using coverage counters collected during Phase I. (ii) It constructs new probe tasks that go through these interfaces until all their member ports meet coverage requirements. We rigorously prove that Phase II realizes its goal (§4.3).

Two-phase generation fulfills coverage and scalability requirements. As for coverage, it leverages the coupon collector theory to deduct the required task number. It relies on two phases to attain the coverage requirement collaboratively. As for scalability, its probe volume is far less than that of full-mesh probing [30, 69]. Moreover, probe planning completes in a few seconds, thanks to the low algorithmic complexity of both phases. In contrast, some solutions cannot finish planning in even several hours due to the NP-hard nature of the planning problem [25, 54] (§6.3).

Data processing. We extend the data processing module with a coverage computation logic. It counts the number of Phase-I tasks through each interface, which we call Layer-3 *coverage counters*. They will guide Phase II in locating under-covered interfaces.

To compute coverage counters, the module listens for probe packets mirrored from switches. Specifically, switches match probe packets based on the identifier in the IP payload (§2.1). They mirror the matched packets to the data processing module. The mirrored packets contain information about the interfaces they go through. Thus, the computation logic simply increments the counter of an interface once it sees a new task through it. This match-and-mirror function is widely available in commodity datacenter switches [85], e.g., ERSPAN [21].

Workflow. RD-Probe has five steps in sequence during one epoch (Figure 3). Previous black-box systems also work in epochs [30, 69].

(1) *Topology construction (§4.1):* It transforms the vanilla topology directly seen by the monitoring system into the de facto topology using switch configurations.

(2) *Randomized generation (§4.2):* Based on the input, it selects a proper number of random Phase-I tasks and dispatches them to agents. Tasks are active until the epoch's end.

(3) *Packet mirroring:* Switches match and mirror probe packets to the data processing module, where the coverage computation logic maintains the Layer-3 coverage counters.

(4) *Coverage computation:* It listens for the mirrored packets and obtains the interfaces they go through. It then increments the associated counters on seeing packets from new tasks and persists the updated counters to storage.

(5) *Deterministic generation (§4.3):* Phase II reads the current coverage counters after a fixed amount of time since the epoch begins (e.g., 20s). It then decides which interfaces are still under-covered and finds additional tasks to boost their coverage. These tasks are also active till the end.

Discussion. RD-Probe is designed to be a black-box monitoring system as it does not solicit runtime white-box metrics on switches like counters and logs. Although we leverage switch configurations stored in CMDB, they are mostly static and only dictate the enabled protocols and modes. RD-Probe can complement other white-box techniques to enhance monitoring quality.

4 RD-PROBE DESIGN

4.1 Topology Construction

Topology construction transforms the vanilla Layer-3 topology into a de facto one using switch configurations. Its significance is to expose low-level components and configurations that affect coverage but are not directly visible in the physical topology.

Vanilla topology. We describe it by an attributed graph $G' = \langle V', E', L' \rangle$, where

- V' comprises the Layer-3 switches, each identified by its management IP address. The monitoring system communicates with a switch via this address.
- $E' \subset V' \times V'$ comprises Layer-3 links among these switches. Two ends of a link represent two peer interfaces in one network domain.
- L' is the attribute function on interfaces. Given a $v' \in V'$ and an interface i' on v', $L'(v'.i')$ shows the address of i'.

De facto topology. We also describe it as an attributed graph $G = \langle V, E, L \rangle$, where

- V is the vertex set. Each vertex represents a routing plane on a physical switch. The property of a routing plane is that ports on the same plane are interconnected while different planes are isolated from each other.

Category	Semantics of a config	Graph ops	Explanation of the ops	Config of example techniques			
				LACP	Stacking	VRF	ECMP
Virtualization	Combine multiple ports as one	`SplitEdge`	Split an edge into many, each corresponding to a port	✔			
	Combine multiple switches as one	`SplitVertex`	Split a vertex into many, each corresponding to a switch		✔		
	Enable a port for inter-switch comm. only	`AddEdge`	Add a new edge, corresponding to the internal port		✔		
	Create traffic isolation within a switch	`SplitVertex`	Split a vertex into many, each corresponding to an isolated plane			✔	
Randomness	Affect next-hop selection	`AssignAttr`	Assign a new attribute, indicating the range of selection	✔			✔

Table 2: Semantics regarding virtualization and randomness in our datacenters and the corresponding operations.

Algorithm 1 Randomized Generation

Input: Coverage requirement (α, p), de facto topology G
Output: Phase-I tasks
1: **procedure** RandGen
2: $T \leftarrow$PropNum(α, p, G) ▷ Generate T tasks in Phase I
3: $A \leftarrow$ # of healthy agents
4: **for** each healthy agent **do**
5: Generate T/A new tasks with this agent as the source
6: **for** each of the T/A tasks **do**
7: Randomly choose a distinct destination agent

- $E \subset V \times V$ is the edge set. Two ends of an edge represent two peer ports with direct physical connection. Moreover, all the edges incident to a vertex have a one-to-one correspondence with all the ports on it.
- L is the attribute function on ports. Given a $v \in V$ and a port i on v, $L(v.i)$ is the group of ports on v (including $v.i$) among which the switch can load-balance traffic. It expresses routing choices for certain network prefixes.

Transformation procedure. We parse configurations of related techniques in a bottom-up order. This order is because techniques are layered and rely on the lower abstraction. We translate each configuration's semantics regarding virtualization and randomness into equivalent graph operations. Table 2 summarizes all the five related semantics in our datacenters and their corresponding operations. For example, if a configuration enables a port for inter-switch communication only, we will call `AddEdge` on the corresponding switch vertex. This operation adds a new edge, representing a communication channel missed in the vanilla topology. Note that de facto topology is changeable even when the physical topology is fixed (e.g., due to configuration updates).

Examples. We demonstrate four virtualization and randomness techniques (from bottom to top): link aggregation, switch stacking, VRF, and ECMP (§2.2). The right half of Table 2 illustrates their corresponding transformation.

(1) A configuration of link aggregation shows member ports in a LAG and enables load balance among them. It matches the semantics of port combining and next-hop selection, so we translate it into a `SplitEdge` and a `AssignAttr` operation. The assigned attribute is the LAG to which a port belongs.

(2) A configuration of switch stacking shows the neighboring member switches and local ports connected to them. It matches the semantics of switch combining and inter-switch communication, so we translate it into a `SplitVertex` and an `AddEdge` operation.

(3) A configuration of VRF shows which interfaces belong to a VRF plane. It matches the semantics of traffic isolation, so we translate it into a `SplitVertex` operation.

(4) A configuration of ECMP shows the group of interfaces that can load-balance traffic. It matches the semantics of next-hop selection, so we translate it into an `AssignAttr` operation. The assigned attribute is the ECMP group to which a port belongs.

Discussion. Table 2 can vary in other companies' datacenters if they adopt techniques with different virtualization and randomness semantics. The corresponding graph operation should then express those semantics. Since techniques provide precise semantics, we believe topology construction generalizes to other datacenters.

4.2 Phase I: Randomized Generation

Given the de facto topology and coverage requirement (α, p), Phase I attains a coverage of (α, p') where p' is quite close to p. Specifically, it generates tasks with random sources and destinations in the hope that the randomness techniques will distribute them evenly across different ports. However, the crux is to determine the proper number of tasks. A small number will cause $p' \ll p$, overburdening Phase II. A large number will waste probing bandwidth. We leverage the coupon collector theory to derive this number, which gives a desired trade-off between coverage and probing overhead.

Algorithm. As shown in Algorithm 1, Phase I first computes the proper number of tasks by analyzing the coverage requirement and the structure of the de facto topology (Line 2). It then assigns an equal number of tasks to each healthy agent for load balancing (Line 5). Next, it randomly chooses distinct destination agents for tasks on the same agent (Line 7).

Proper number of tasks. We compute this number in two steps. First, we find the number of tasks required per switch. The idea is to model port selection as a coupon collector process [46]. In this process, there are n types of coupons, and each type has an unlimited supply. We try to collect α copies of each coupon in several independent attempts. Each attempt draws one coupon randomly chosen from the n types. The analogy between coupon drawing and port coverage is clear when viewing ports in an ECMP group as coupons and a Phase-I task as a draw of coupons. This is because each Phase-I task goes through one port uniformly at random in the matching ECMP group (§2.4). Then, we derive the required number per switch using the mathematical properties of this process. Second, we determine the task number for the whole topology after knowing the per-switch task number. The idea is that a task will traverse different tiers in sequence in multi-tier datacenters (§2.4). We analyze tier by tier to find the number.

Algorithm 2 shows the details. It has two procedures. Let $p_\alpha(n, m)$ be the probability that we collect α copies of each of the n types after m draws of coupons.

(1) The procedure ProbNumSw finds the task number required by a switch (Lines 7-13). Suppose an ECMP group contains n ports

Algorithm 2 Proper Number of Tasks

```
1:  procedure PropNum(α, p, G)
2:    for Tier l in G (except the top tier) do
3:      T_l ← 0                                  ▷ Need T_l tasks to cover Tier l
4:      for each device d in Tier l do
5:        T_l += PropNumSw(α, p, d)
6:    return max_l T_l
7:  procedure PropNumSw(α, p, d)
8:    for each ECMP group E on d do
9:      n ← # of ports in E
10:     t_n ← (1/n) min{m : p_α(n, m) ≥ p}
11:   t ← max t_n among all ECMP groups on d
            ▷ Any port requires ≤ t tasks to reach coverage requirement (α, p)
12:   N ← # of ports on the up-/peer-links of d
13:   return Nt
```

(Line 9). Then each port in this group requires $t_n := \frac{m}{n}$ tasks, where m is the minimal integer satisfying $p_\alpha(n, m) \geq p$ (Line 10). Let t be the largest t_n among all ECMP groups (Line 11). The task number the switch requires is the multiple of t and the number of ports on its up-links and peer-links (Lines 12-13).

(2) The procedure PropNum finds the task number required by the whole topology (Lines 1-6). For any tier, the required task number should equal the sum of each switch's required number in this tier (Line 5). As each task traverses multiple tiers, the required number should be the maximum value among all tiers (Line 6).

Analysis. We analyze Phase I from three aspects: the computation of $p_\alpha(n, m)$, its scalability, and coverage guarantee.

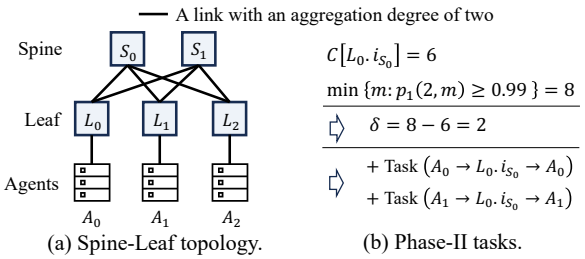
(1) $p_\alpha(n, m)$ can be precisely and efficiently computed for any triplet (α, n, m). We give an explicit formula of $p_\alpha(n, m)$ in Appendix A.1 using the coupon collector theory. We also show how to evaluate this formula using dynamic programming in $O(\alpha^2 m^2 n^3)$ time. In practice, we precompute the values for all possible triplets and load them into memory in advance. This consumes < 20 MB of memory given $\alpha \in [1, 5]$, $n \in [1, 100]$ and $m \in [1, 5000]$.

(2) We analyze the scalability in terms of probe volume and planning time (§2.2). As for volume, we show that an N-port switch only requires $O(N \ln N)$ Phase-I tasks in Appendix A.2 using the coupon collector theory. Thus, Phase I incurs at most a logarithmic multiplicative overhead since $\Theta(N)$ tasks are necessary to achieve α-coverage in the optimal planning (although finding it is NP-hard [25, 54]). As for planning time, Phase I only iterates over all the tiers and ECMP groups once. Thus, given S switches and a maximum of N ports per switch, Phase I finishes in $O(S \cdot N \ln N)$ time.

(3) We summarize the coverage guarantee of Phase I in the following theorem. Its proof is in Appendix A.3. Since large-scale datacenters heavily adopt high-radix switches (e.g., chassis switch with $N \geq 64$), this guarantee eliminates over 80% of ports for Phase II. In practice, the fraction of α-covered ports is much higher than the theoretical prediction, as Theorem 1 only gives a lower bound.

Theorem 1. *After exercising Phase-I tasks, for a switch with N ports, at least $1 - \frac{1}{2N^{1/4}}$ fraction of ports has attained α-coverage on average when N is large.*

Example. We consider $(\alpha, p) = (1, 0.99)$ in a two-tier datacenter with three leaves and two spines (Figure 4). Suppose each link has an aggregation degree of two. Let us focus on a leaf switch to determine the proper number of tasks. There is one ECMP group



(a) Spine-Leaf topology. (b) Phase-II tasks.

Figure 4: An example of two-phase generation.

containing $N = 4$ ports on the uplinks. Since $p_1(4, 20) \doteq 0.987$ and $p_1(4, 21) \doteq 0.99$, we know $t = \frac{21}{4}$. Therefore, each leaf switch requires $Nt = 21$ tasks, and Phase I requires $21 \times 3 = 63$ tasks.

Discussion. NetSonar [76] also uses the coupon collector theory but with a different focus. While it aims to cover all links on some ECMP paths between pairs of agents, we determine the proper number of tasks to achieve sufficient port-level coverage.

4.3 Phase II: Deterministic Generation

Phase II complements Phase I to reach coverage requirements. It first locates under-covered interfaces in Phase I using coverage counters from the data processing module (§3). It then generates additional tasks through these interfaces. Thus, Phase II starts when coverage counters are ready. In principle, counters are ready shortly after each Phase-I task has some packets reaching the destination and their mirrored packets reaching the data processing module. Therefore, we expect Phase II to commence shortly. In our datacenters, we let Phase II start in the 20s after Phase I.

Algorithm. As shown in Algorithm 3, Phase II begins by finding under-covered interfaces (Line 2). Coverage requirement demands that m tasks go through this interface, where m is the minimal value satisfying $p_\alpha(n, m) \geq p$ and n is the link aggregation degree (Line 3). We denote by δ the difference between m and the coverage counter (Line 4). A positive δ indicates that we need δ new tasks through this interface (Line 5). Thus, we utilize breadth-first search (BFS) to find the δ-nearest agents (Line 6). For each of the δ agents, we construct an IP-in-IP ping task from and to this agent (Lines 8-9). Moreover, we add the under-covered interface as an intermediate hop (Line 10). IP-in-IP ping ensures that packets traverse this interface using IP tunneling [69]. If other types of ping were used, randomness techniques might select different paths that bypass this interface.

Analysis. We analyze Phase II from two aspects: its scalability and coverage guarantee.

(1) We analyze the scalability in terms of probe volume and planning time (§2.2). As for volume, Phase II's tasks are much fewer than Phase I's for two reasons. (i) The coverage guarantee of Phase I says that a switch only has $\leq N \cdot \frac{1}{2N^{1/4}} = \frac{N^{3/4}}{2}$ under-covered ports on average. (ii) The δs are mostly small since ports are close to α-coverage very likely before Phase II. As for planning time, Phase II is fast since it iterates over each interface just once. The BFS is efficient as the number of layers in datacenters is typically small (e.g., less than five).

(2) We state the final coverage guarantee in the next theorem. It says that our two-phase generation scheme ensures the ultimate achievement of any prescribed coverage requirement. We defer its proof to Appendix B.

Algorithm 3 Deterministic Generation

Input: Coverage requirement (α, p), de facto topology G,
Layer-3 coverage counters C
Output: Phase-II probes
1: **procedure** DetGen(α, p, G, C)
2: **for** each interface i on each vertex $v \in G$ **do**
3: $n \leftarrow$ # of aggregated ports in $v.i$
4: $\delta \leftarrow \min\{m : p_\alpha(n, m) \geq p\} - C[v.i]$
5: **if** $\delta > 0$ **then** ▷ $v.i$ needs further coverage
6: $v_1, \ldots, v_\delta \leftarrow$ the δ-nearest agents to $v.i$
7: **for** each l in $[1, \delta]$ **do**
8: Generate a new task with source and destination both as v_l
9: Set the task type as IP-in-IP ping
10: Add $v.i$ as the intermediate hop

Theorem 2. *After exercising Phase II tasks, the coverage requirement (α, p) is achieved.*

Example. Continuing with the previous example, suppose the coverage counter of the interface $L_0.i_{S_0}$ is six (Figure 4(b)). This interface has $n = 2$ ports. Since $p_1(2, 7) \doteq 0.984$ and $p_1(2, 8) \doteq 0.992$, we need eight tasks through this interface. Thus, we need $\delta = 8 - 6 = 2$ additional tasks. The 2-nearest agents are A_0 and A_1, so Phase II adds the two IP-in-IP tasks shown in Figure 4(b).

Discussion. Two reasons explain why we do not use Phase II solely from the very beginning. (i) Phase II's task type must be IP-in-IP ping to impose path control. However, it is crucial to have most probe tasks exercising other protocols to increase task diversity. (ii) Using only Phase II will result in excessive tasks. Essentially, Phase II greedily increases the coverage of an under-covered interface by one each time. This greedy policy is only effective when Phase I sufficiently covers most ports.

To extend RD-Probe to spineless architectures (§2.4), we only need to modify Algorithm 2 in Phase I. Specifically, the generation logic of random tasks in Phase I (i.e., Algorithm 1) and complementary tasks in Phase II (i.e., Algorithm 3) remains valid. As for Algorithm 2, we cannot analyze the task number tier by tier. Consequently, the coverage guarantee in Phase I can be different. We will explore the details in future work. On the other hand, the coverage guarantee for Phase II still applies as it is topology-independent (see Appendix B). Thus, spineless architectures establish a new trade-off point between coverage and probing overhead.

5 RD-PROBE IN PRODUCTION

We have deployed RD-Probe in three production regions for over six months, each with $O(10^3)$ switches and $O(10^4)$ links. Table 3 shows their structural information. They have diverse topologies and different adoption rates of certain techniques. We first introduce RD-Probe's implementation in production (§5.1) and then discuss insights from monitoring statistics (§5.2). Next, we discuss the resulting benefits (§5.3). We conclude with future efforts (§5.4).

5.1 Production Implementation

We have dedicated clusters to execute the generation module and streaming framework in production regions. The generation cluster has 16 nodes, each with an 8-core 2.80 GHz CPU and 64 GB of memory. The streaming cluster has 48 nodes, each with a 16-core 2.80 GHz CPU and 32 GB of memory.

To fully utilize resources in these clusters, we distribute jobs among cluster nodes during topology construction, probe generation, and packet mirroring.

(1) *Topology construction:* We logically divide the whole region into intra-pod and inter-pod topologies. Specifically, all nodes except one in the generation cluster manage different pods. They construct intra-pod topologies for their managed pods. Besides, a central node constructs the inter-pod topology. It aggregates intra-pod topologies sent from other nodes to form a complete graph. Fortunately, the central node is not a communication bottleneck because other nodes only send differences since the last epoch. As long as there are no significant configuration updates, the time spent on topology construction is negligible.

(2) *Probe generation:* Two-phase generation solely runs on the central node thanks to its light overhead. Meanwhile, the other 15 nodes construct ring-ping lists for their managed pods. A ring-ping list monitors the health status of servers in a rack. It shuffles all the agents under a ToR into a ring structure. Each agent only pings its left neighbor in the ring.

(3) *Packet mirroring:* We load-balance the job of maintaining coverage counters among streaming nodes via hardware Mux (HMux) [22, 85]. Specifically, we configure some switches as HMuxes based on the traffic volume. They share a common virtual IP address, which is also the destination of packet mirroring. Upon receiving a mirrored packet, they use the same hash on the packet's 5-tuple to select a streaming node. It then replaces the virtual IP with the node's direct IP and forwards the packet. Thus, each node maintains coverage counters for different sets of interfaces.

5.2 Monitoring Statistics

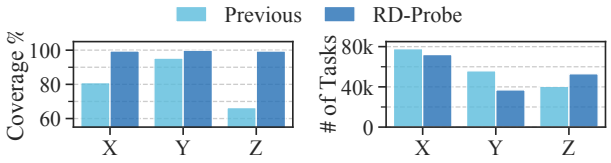
Coverage metrics. Figure 5(a) shows the estimated coverage before and after adopting RD-Probe. We set the coverage requirement as $(1, 99\%)$. We obtain the estimated coverage using Layer-3 coverage counters after an epoch ends. Previously, there were huge coverage gaps in Region X and Z, where only 80.9% and 66.4% of ports were covered, respectively. Gaps mainly came from two kinds of ports: (i) those in large LAGs and (ii) peer ports that connect to switches in the same tier. The reasons are as follows. (i) Highly aggregated ports require the number of tasks multiple times (typically more than four) its aggregation degree. Pingmesh did not generate enough tasks through such ports. (ii) Peer ports exist in stacked switches and middleboxes like firewalls. Pingmesh was unaware of these ports, as many of them are invisible to Layer 3. After deploying RD-Probe, we successfully improve the coverage of any region to above 99.5%. This is because RD-Probe reveals these ports, improving the quality of tasks.

Resource overhead. Besides sufficient coverage, RD-Probe further reduces probe volume and planning time. Moreover, overheads of other resources remain almost unchanged before and after adopting RD-Probe. For example, the CPU consumption remains 0.6-5% per core on agents; Mirroring consumes less than 0.1% of bandwidth on switches and basically does not consume switch CPU.

(1) Planning time is below 2.2 seconds for Phase I and 3.1 seconds for Phase II in the three regions. Moreover, Phase I and Phase II spend over 60% of their time performing network IOs with other

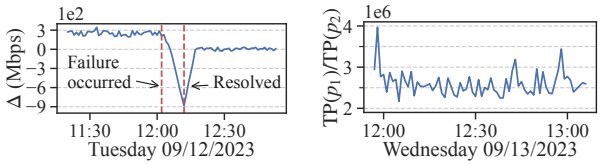
Name	Scale	Topology	Max LAG size	# of stacked switches	# of VRF planes	ECMP
X	1,403 switches, 30.2k links, 26 pods	Clos	16	627 (44.7% of total)	13	Per-flow load balancing
Y	747 switches, 10.1k links, 20 pods	Spine-Leaf	4	587 (78.6% of total)	14	Per-flow load balancing
Z	740 switches, 20.4k links, 19 pods	Spine-Leaf	16	716 (96.8% of total)	18	Per-flow load balancing

Table 3: Structural information of the three deployed regions.



(a) Coverage. (b) Probe volume.

Figure 5: Monitoring statistics in the deployed regions.



(a) LPU silent drop. (b) Unexpected L2 load imbalance.

Figure 6: Example failures found by RD-Probe.

nodes and the CMDB. These data imply that RD-Probe has low algorithmic complexity.

(2) Figure 5(b) shows the volume in terms of task number. In Region X and Y, we achieve sufficient coverage with 7.5% and 33.9% fewer tasks, respectively. In Region Z, we improve the coverage by 33.6% while incurring only 30.9% more tasks. This gain comes from two advantages of RD-Probe over Pingmesh. (i) It leverages asymmetry among pods. Asymmetry is common as it accumulates over the years. For example, the largest and smallest pods in Region X contain 156 and 12 switches, respectively. Larger pods need more inter-pod tasks, but Pingmesh generates 75 inter-pod tasks for any pod in Region X. In contrast, RD-Probe generates different numbers of inter-pod tasks based on the pod's scale. (ii) RD-Probe reduces redundancy in Pingmesh tasks. Redundancy means some ports are covered too many times. Indeed, many pods do not need full-mesh pinging to attain sufficient coverage.

Fault tolerance. During RD-Probe's launch and operation, we come across various faults caused by internal (e.g., sudden device outages) and external factors (e.g., occasional interference from other teams). Fortunately, RD-Probe remains operational as long as physical conditions allow. In the following, we present how RD-Probe tolerates three typical faults in production environments: lost mirrored packets, switch outages, and node outages.

(1) Multiple factors contribute to lost mirrored packets, including ERSPAN rate limiting, congestion at HMuxes, and probe packet loss. The overall loss rate fluctuates between $O(10^{-6})$ and $O(10^{-3})$ across the three regions. This loss leads to underestimated Layer-3 coverage counters, necessitating more Phase-II tasks. Fortunately, this does not create coverage blind spots – the affected interfaces are slightly over-covered. During epochs with a loss rate of $O(10^{-3})$, the total task count increases by less than 1.3% compared to normal epochs. The difference in planning time is negligible.

(2) Switch outages cause inevitable coverage blind spots, although they occur occasionally. RD-Probe ensures that other devices are unaffected if they still connect to end hosts and the control plane. Specifically, coverage counters will be zero for compromised interfaces, and IP-in-IP tasks specifying these interfaces as intermediate hops in Phase II will get no response. If a Phase-II task receives no response throughout its lifetime, the source agent will immediately alert the operator to the potential faulty interface. Note that we have not tested RD-Probe under massive device outages; in such cases, monitoring might not be the primary concern.

(3) Node outages, while rare, have different impacts compared to switch outages. For an outage in the generation cluster, we can easily migrate the two-phase generation algorithm to other active nodes. This is because the only stateful information RD-Probe preserves across epochs is the de facto topology, which can be reconstructed with assistance from CMDB. For an outage in HMuxes and the streaming cluster, it effectively results in the loss of mirrored packets. To prevent excessive Phase-II tasks, we allow Phase II to raise alarms if too many coverage counters are zero (e.g., over 1% of all interfaces) and skip those interfaces. Once the nodes are back, the system may leave blind spots for the current epoch but will function normally in the next epoch.

5.3 Benefits of RD-Probe

In the first month since RD-Probe went online, it spotted many previously unnoticed issues. Most of them only caused fail-slow symptoms or intermittent packet drops, making them hard to locate via manual inspection. Fortunately, given sufficient port-level coverage under continuous monitoring, we found issues before users' perception. Next, we present three example cases in production found by RD-Probe, which escaped the eyes of other deployed tools.

Case 1: Switch LPU silent drops. One task executing object storage service in a region witnessed frequency drops. After obtaining the rate difference Δ (i.e., the difference between the switch's Tx and Rx rate) along its route, a core switch showed abnormal data (Figure 6(a)). Before this failure, the switch was already sub-healthy, producing a positive rate difference. During the failure, one chip in the line processing unit (LPU) associated with the buggy inbound interface ran into soft errors. It silently dropped some incoming packets and could not report its monitoring data to the control plane. Thus, the rate difference decreased to −880 Mbps. We isolated the faulty chip 10 minutes after failure, and this rate gradually restored to the normal value, i.e., zero.

This case highlights the importance of sufficient coverage. If we missed this inbound interface, the error would continue to affect business traffic that was hashed to this chip. Furthermore, white-box methods often fail to detect such failures for several reasons. (i) Methods relying on device self-reporting (e.g., Syslog) can be compromised by LPU soft errors, as the signaling or reporting

mechanism is affected. (ii) Silent drops occur even before packets reach detection points, such as before extracting in-packet telemetry data in INT methods and before capturing traffic in ERSPAN.

Case 2: Unexpected Layer-2 load imbalance. Some tasks going through one uplink on a stacked switch suffered from high loss rates and latency. This uplink aggregated two ports (p_1, p_2) from different member switches. Figure 6(b) shows the ratio of uplink throughput of p_1 over p_2 in an hour. The ideal ratio should be 1 for perfect Layer-2 load balance, but it was $O(10^6)$! This failure was due to flawed modules in preferential local traffic forwarding, which caused traffic to go only through the local port instead of stack cables. Thus, if the load of one member switch exceeds its capacity, the stack switch will be congested. We settled this issue by resetting the module to enable forwarding through stack cables. The throughput ratio dropped to $O(1)$.

Prior to RD-Probe, we detected such Layer-2 load imbalance by comparing the throughput of p_1 and p_2 obtained using telemetry methods. However, this approach generated an excessive number of false positives (over 90 out of every 100 alarms), making it hardly usable. The false alarms were caused by uneven allocation of large flows within an uplink LAG. For most of the time, this uneven allocation does not cause issues as long as the combined throughput remains within the switch's capacity. Eventually, we adopted the black-box solution, which collects traffic metrics from the same perspective as our customers and raises alarms only when such imbalance causes undesirable packet loss and latency.

This case shows the importance of sufficient port-level coverage. If we only require coverage of the aggregated interface, tasks may go through p_2 and experience no issues, causing false negatives.

Case 3: Detour routing. Some tasks executing TCP ping among two pods suddenly experienced increasing latency. After contacting recent configuration updates, we found the root cause was detour routing caused by the wrong route policy. Specifically, operators specified wrong preference values for some updated BGP routes. This error led the traffic to a slow path containing firewalls and two more hops. We rolled back the switch configurations to resolve this issue.

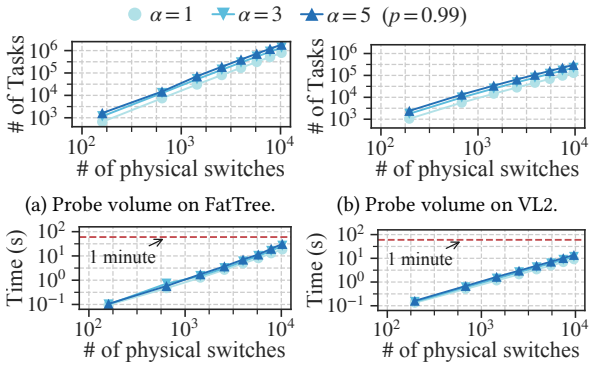
This case shows the importance of black-box monitoring. All the deployed white-box tools in the region failed to raise alarms as switch components were still working as expected. Verification tools can detect such issues but with higher latency, as they need to retrieve the latest FIBs and RIBs from switches and verify a large number of entries.

5.4 Future Efforts

Although RD-Probe proves successful in our datacenters, two practical issues need further investigation.

Sufficient coverage of north-south traffic. North-south traffic refers to traffic from the Internet to the datacenter and vice versa. RD-Probe aims at sufficient port-level coverage within datacenters. It does not consider north-south traffic. However, virtualization and randomness techniques applied to such traffic bring new challenges to sufficient Layer-2 coverage, such as elastic IP (EIP), NAT, and software load balancer (SLB).

Filtering out server-side failures. Server-side `iptables` updates, restarts, migrations, power-downs, and other factors lead to numerous false alarms in the daily operation of datacenters. It is challenging to determine whether a failure is a genuine network anomaly. Currently, we combine ring-ping (§5.1), health checks, and assistance from downstream interfaces to filter out server-side failures. However, mistakes often occur. We need a better approach to identify network-side failures.



(a) Probe volume on FatTree. (b) Probe volume on VL2.

(c) Planning time on FatTree. (d) Planning time on VL2.

Figure 7: Scalability of RD-Probe.

6 RD-PROBE AT SCALE

6.1 Simulation Setup

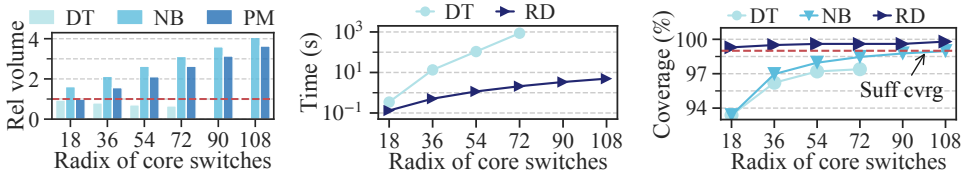
Behavioral-level simulator. We build a behavioral-level simulator on top of NS-3 [52] in about 7,000 lines of C++ code and 2,000 lines of Python code. It aims to evaluate the performance of RD-Probe under massive scales. We run simulations on a machine with two 10-core 2.4 GHz CPUs and 126 GB of RAM. Although we extend the NS-3's native architecture to simulate the monitoring architecture (§2.1), there are still two differences between them. Fortunately, they will not significantly change RD-Probe's performance.

(1) *Omitted modules:* The simulator omits the analysis and health check modules. It only implements the coverage counter logic in the data processing module. Such simplification is reasonable since RD-Probe does not rely on the omitted modules. RD-Probe will show a similar performance given the same input as in the production environment.

(2) *Simplified techniques:* We simplify LACP and switch stacking by only implementing their packet-level forwarding behaviors. For example, the simulated LACP takes each LAG as given, while the real LACP has a negotiation mechanism to detect and enable a LAG. We carefully choose simplification that does not affect port-level coverage in the simulator.

Simulator configuration. We configure techniques and noise in the simulator as follows. As for LACP, each interface aggregates two ports by default. As for switch stacking, each Layer-3 switch comprises two physical switches connected by one stack cable. As for VRF, we adopt two planes: one for intra-pod routing and another for inter-pod routing. As for ECMP, we enable per-flow load balancing based on a flow's five-tuple. Regarding noise, we inject a packet drop probability in $[0, 10^{-3}]$ to each physical link.

Topologies. We use two vanilla topologies: FatTree [2, 69] and VL2 [27, 54]. The largest FatTree and VL2 are 64-ary and 144-ary (for core switches), respectively. The largest de facto topology (transformed based on simulator configuration) for Fattree contains 10.2k

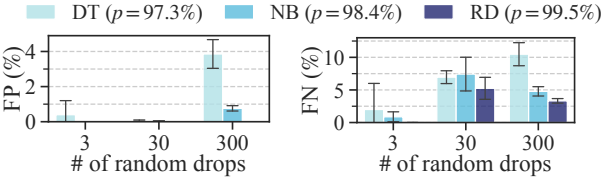


(a) Relative volume. (b) Planning time. (c) Achieved coverage.

System	Volume	Time	Coverage
deTector [54]	✔	✗	✗
NetBouncer [69]	✗	✔	Conditional
Pingmesh [30]	✗	✔	✗
RD-Probe	✔	✔	✔

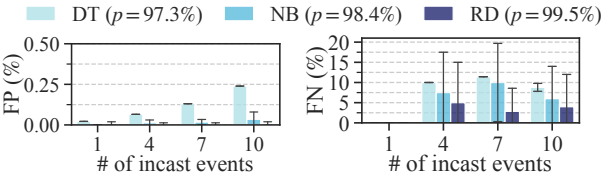
(d) Summary of results.

Figure 8: Monitoring performance of existing systems.



(a) False positive ratio. (b) False negative ratio.

Figure 9: Detection accuracy on random drops.



(a) False positive ratio. (b) False negative ratio.

Figure 10: Detection accuracy on congestion.

switches and 267.2k links, and contains 9.6k switches and 41.7k links for VL2.

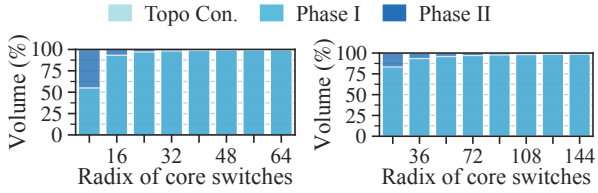
Failures. We inject random drops and congestion to evaluate how sufficient coverage enhances failure detection. A random drop occurs by setting a physical link to drop packets with a probability between 0.01 and 0.5. Congestion is simulated via incast [3, 44], where five servers generate many concurrent GET requests to an HTTP server. To create bottleneck links reliably, we enable preferential local traffic forwarding (§5.3) on ToR switches of HTTP servers. We inject up to ten incast events per simulation run.

Existing systems. We compare RD-Probe with Pingmesh (PM) [30], NetBouncer (NB) [69], and deTector (DT) [54]. To the best of our knowledge, they are state-of-the-art black-box monitoring systems that have large-scale deployment. To incorporate low-level techniques, we build their probe generation methods on Layer-2 components rather than their intended Layer-3 components. Thus, we eliminate their difference in the intended coverage granularity.

6.2 Performance of RD-Probe

We evaluate the achieved coverage and scalability of RD-Probe. Specifically, we fix $p = 0.99$ and vary $\alpha \in [1, 5]$. We follow the same procedure as in §5 to obtain results except for one thing: Here, we construct the de facto topology from scratch instead of modifying the one in the previous epoch.

Coverage results. Port-level coverage stays between 99.4-99.8%, 99.1-99.6%, and 99.0-99.3% for $\alpha = 1, 3, 5$, respectively. Thus, it achieves sufficient coverage as Theorem 2 dictates. Moreover, the coverage approaches p from above as the scale or α increases. This fact indicates that we need significantly more tasks to nudge p even higher (e.g., 99.9%).



(a) Volume breakdown on FatTree. (b) Volume breakdown on VL2.
(c) Time breakdown on FatTree. (d) Time breakdown on VL2.

Figure 11: Overhead breakdown when $(\alpha, p) = (1, 99\%)$.

Scalability results. We plot the overall probe volume on FatTree and VL2 in Figure 7(a) and (b), respectively. We obtain two findings. (i) The task number increases slowly in the network scale. For example, the task number is about the switch number to the power of 1.53 on FatTree when $\alpha = 5$ (obtained by applying least-squares linear regression to that curve). Note that a k-ary FatTree has $O(k^2)$ switches and $O(k^3) = O((k^2)^{1.5})$ links. Thus, the required task number is at least the switch number to the power of 1.5. This additional 0.03 in the power stems from the coupon collector process when we model randomness techniques (§4.2), which is pretty small. (ii) Achieving sufficient coverage for $\alpha = 1$ is the most expensive. Once we reach that for $\alpha = 1$, we only need to add 69.7-76.7% of tasks to reach that for $\alpha = 3$, and another 61.2-69.8% of tasks for $\alpha = 5$.

We plot the overall planning time on FatTree and VL2 in Figure 7(c) and (d), respectively. We obtain two findings. (i) RD-Probe finishes within 32 seconds, even at the largest scale. If we exclude the time spent on topology construction, it reduces to within 16 seconds. (ii) Time differences given different αs are negligible. This is because α only affects the time on Phase I and II. The dominating factor in their time complexity is the network scale rather than α.

6.3 Comparison with Existing Systems

Monitoring performance. We compare the probe volume, planning time, and port-level coverage of existing systems given $\alpha = 1$ and the VL2 topology. We omit the results of DT since a radix of 90 because it runs out of memory when constructing the routing matrix. Figure 8 shows the results.

(1) Figure 8(a) shows the volume of DT, NB, and PM relative to that of RD-Probe. The volume of NB and PM increases rapidly due to full-mesh pinging. It is four times that of RD-Probe at only 5k switches, leading to an additional 180 GBs of probing data daily in

one region (excluding other rising resource demands). Thus, they exhaust limited probing resources at large scales.
(2) Figure 8(b) shows the planning time of DT and RD-Probe, as NB and PM do not contain a separate probe planning phase. While RD-Probe finishes within 10 seconds at the largest scale, DT can run for hours at a radix of 90 as the penalty for pursuing the minimal volume. It violates the time requirement when reacting to network updates. Thus, DT also has scalability issues at large scales.
(3) Figure 8(c) shows the coverage of NB, DT, and RD-Probe. It omits the results of PM since its coverage stays within 62.8-90.3%, too far away from sufficient coverage. The coverage gap of PM mainly lies in core switches. Their ports comprise 33.3% of all ports, but only 26.7% of tasks go through them. As for DT, its coverage stops increasing at 97.5%. The reason is that every task in DT exercises IP-in-IP ping to control paths. However, it cannot control which port it travels to within an interface. Given a larger aggregation degree, the attainable coverage of DT is even lower. As for NB, its coverage slowly grows from 93.3% to 99.2% at the cost of incurring excessive volume. This growth is owing to the increasing likeliness of covering stack ports. As the volume rapidly grows, switches tend to forward traffic via stack ports for Layer-2 load balance.

We summarize these results in Figure 8(d). RD-Probe is the only system satisfying coverage and scalability requirements, even when other systems are also provided with the de facto topology.

Detection accuracy. To show how sufficient coverage assists failure detection, we fix the VL2 topology with the core radix as 72 and inject two types of failures into the network: random drops and congestion. To eliminate the impact of the detection algorithm on accuracy, we only take the tasks from different systems. They all detect failures using the same coordinate descend method from NB, which takes the loss rate of each task as input and estimates the loss probability of each interface. If one member port in the detected interface has injected failures, we count it as a true positive. All reported results are averages of multiple simulation runs.

As for random drops, Figure 9(a) and (b) show the false positive (FP) and false negative (FN) ratios achieved by tasks from different systems, respectively. We obtain three findings. (i) Higher coverage entails higher accuracy. When the coverage grows from 97.3% to 99.5%, FP drops from 3.86% to almost zero, and FN drops from 10.5% to 3.3%. These ratios are higher than the portion of uncovered ports. The reason is that the detection method does not know the status of uncovered ports. Such information loss causes it to make wrong judgments on even covered ports. (ii) 97.3% coverage is not good enough. It cannot achieve zero FP nor FN, given only three failures. Manual checks of the detection logs show that it often misjudges uplink ports as faulty, even if the actual loss is on a covered stack port. (iii) 99.5% coverage is already satisfying, as it attains an almost zero FP and less than 4% FN. We need far more tasks to further improve coverage, whose benefit-cost ratio is too low.

As for congestion, Figure 10(a) and (b) show the false positive (FP) and false negative (FN) ratios achieved by tasks from different systems, respectively. Regardless of coverage and failure numbers, the FP ratio remains close to zero, because the detection algorithm seldom blames a non-congested interface for packet loss. However, a 2.2% increase in coverage reduces the FN ratio from above 10% to below 4%. The FN ratio is high for other systems since they tend to overlook congestion caused by Layer-2 load imbalance at ToR switches. Without sufficient coverage, probe tasks from other systems are prone to missing many ports involved in link aggregation and switch stacking. On the other hand, RD-Probe still has a non-zero FN ratio, since the detecting algorithm we use is not free from false negatives [69].

6.4 Microbenchmark

We evaluate how each step in RD-Probe contributes to the overall overhead. We present the results given $(\alpha, p) = (1, 99\%)$ and defer those given stricter requirements (i.e., when $\alpha = 3$ and 5) to Appendix C. All these results show a similar trend.

Volume breakdown. Figure 11(a) and (b) show the volume breakdown on FatTree and VL2, respectively. Only Phase I and II contribute to the volume. 55.0% of volume comes from Phase I on the 8-ary FatTree. This percentage rapidly rises above 99% when the radix is above 40, indicating that Phase I sufficiently covers most ports. This observation matches the coverage guarantee of Phase I (i.e., Theorem 1), which also becomes stricter as the radix grows.

Time breakdown. Figure 11(c) and (d) show the time breakdown on FatTree and VL2, respectively. We derive three findings. (i) Topology construction consumes over 63% of the total time. This is because it issues at least one `SplitEdge` operation per link (for LACP) and two `SplitVertex` operations per vertex (for switch stacking and VRF). Fortunately, as switches rarely reconfigure these techniques, this time is only incurred once during the initial epoch of RD-Probe. (ii) Phase I takes a larger percentage of time at larger scales. Its consumption is 21.6% and 32.6% on the largest FatTree and VL2, respectively. The reason is that finding the proper number of tasks becomes computation-intensive as the number of ECMP groups and switches increases. (iii) Phase II is the fastest step. Its consumption is below 15.4% on FatTree and 4.0% on VL2, as most interfaces are sufficiently covered.

7 CONCLUSION

RD-Probe is a complexity-aware and scalable monitoring system with the guarantee of sufficient port-level coverage. It leverages topology construction to expose low-level complexity and two-phase generation to fulfill coverage and scalability requirements. Deployment experience and simulation study show that RD-Probe addresses prior challenges and outperforms existing deployed systems. RD-Probe has been running robustly in three Huawei Cloud's datacenter regions for over six months and enduring various faults in production environments. It has brought immediate benefits to our daily operation.

Ethical issues. This work does not raise any ethical issues.

ACKNOWLEDGEMENT

We thank our shepherd, Maria Apostolaki, and the anonymous reviewers for their insightful comments. We also thank Lijun Zhao, Liyan Huang, and Hai Liu from Huawei Cloud for their continuous support and timely feedback. Additionally, we appreciate Shujie Han, Jintao He, and Zirui Ou for their help in polishing an earlier version of this paper. This work is supported by National Key Research and Development Program of China (2023YFB2904600), National Natural Science Foundation of China (62172007), and Research Grants Council of Hong Kong (GRF 14201523).

REFERENCES

[1] Jung Ho Ahn, Nathan Binkert, Al Davis, Moray McLaren, and Robert S Schreiber. 2009. HyperX: topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 1–11.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.

[3] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *Proc. of SIGCOMM*. 63–74.

[4] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically finding the cause of packet drops. In *Proc. of NSDI*. 419–435.

[5] François Aubry, David Lebrun, Stefano Vissicchio, Minh Thanh Khong, Yves Deville, and Olivier Bonaventure. 2016. SCMon: Leveraging segment routing to improve network monitoring. In *Proc. of INFOCOM*. IEEE, 1–9.

[6] Novella Bartolini, Ting He, and Hana Khamfroush. 2017. Fundamental limits of failure identifiability by Boolean network tomography. In *Proc. of INFOCOM*. IEEE, 1–9.

[7] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *Proc. of SIGCOMM*. 155–168.

[8] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic in-band network telemetry. In *Proc. of SIGCOMM*. 662–680.

[9] Robert Birke, Lydia Y Chen, and Evgenia Smirni. 2012. Data centers in the cloud: A large scale performance study. In *2012 IEEE Fifth international conference on cloud computing*. IEEE, 336–343.

[10] Matt Brown, Ari Fogel, Daniel Halperin, Victor Heorhiadi, Ratul Mahajan, and Todd Millstein. 2023. Lessons from the evolution of the Batfish configuration analysis tool. In *Proc. of SIGCOMM*. 122–135.

[11] Ramón Cáceres, Nick G Duffield, Joseph Horowitz, and Donald F Towsley. 1999. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information theory* 45, 7 (1999), 2462–2480.

[12] Jeffrey D Case, Mark Fedor, Martin Lee Schoffstall, and James Davin. 1990. Rfc1157: Simple network management protocol (snmp).

[13] Benoit Claise. 2004. Cisco Systems NetFlow Services Export Version 9. https://www.ietf.org/rfc/rfc3954.txt.

[14] Ítalo Cunha, Renata Teixeira, Nick Feamster, and Christophe Diot. 2009. Measurement methods for fast and accurate blackhole identification with binary tomography. In *Proc. of SIGCOMM IMC*. 254–266.

[15] Amogh Dhamdhere, David D Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky KP Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C Snoeren, and Kc Claffy. 2018. Inferring persistent interdomain congestion. In *Proc. of SIGCOMM*. 1–15.

[16] Amogh Dhamdhere, Renata Teixeira, Constantine Dovrolis, and Christophe Diot. 2007. NetDiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data. In *Proc. of CoNEXT*. 1–12.

[17] Nick G Duffield and Matthias Grossglauser. 2001. Trajectory sampling for direct traffic observation. *IEEE/ACM transactions on networking* 9, 3 (2001), 280–292.

[18] John Evans and Fabien Chraim. 2023. No packet left behind. https://storage.googleapis.com/site-media-prod/meetings/NANOG88/4790/20230613_Evans_No_Packet_Left_v1.pdf.

[19] Chongrong Fang, Haoyu Liu, Mao Miao, Jie Ye, Lei Wang, Wansheng Zhang, Daxiang Kang, Biao Lyv, Peng Cheng, and Jiming Chen. 2020. VTrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network. In *Proc. of SIGCOMM*. 31–43.

[20] Marco Ferrante and Monica Saltalamacchia. 2014. The coupon collector's problem. *Materials matemàtics* (2014), 0001–35.

[21] M Foschiano et al. 2015. Cisco systems' encapsulated remote switch port analyzer (erspan). *draft-foschiano-erspan-00. txt, Tech. Rep* (2015).

[22] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2014. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 27–38.

[23] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks.. In *Proc. of NSDI*. 549–564.

[24] Rainer Gerhards. 2009. *The syslog protocol*. Technical Report.

[25] Denisa Ghita, Hung Nguyen, Maciej Kurant, Katerina Argyraki, and Patrick Thiran. 2010. Netscope: Practical network loss tomography. In *Proc. of INFOCOM*. IEEE, 1–9.

[26] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *Proc. of SIGCOMM*. 350–361.

[27] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *Proc. of SIGCOMM*. 51–62.

[28] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. 2009. BCube: a high performance, server-centric network architecture for modular data centers. In *Proc. of SIGCOMM*. 63–74.

[29] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. 2008. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proc. of SIGCOMM*. 75–86.

[30] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proc. of SIGCOMM*. 139–152.

[31] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proc. of SIGCOMM*. 357–371.

[32] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proc. of NSDI*. 71–85.

[33] Vipul Harsh, Sangeetha Abdu Jyothi, and P Brighten Godfrey. 2020. Spineless data centers. In *Proc. of HotNets*. 67–73.

[34] Leo Huang and Arya Reais-Parsi. 2013. Localizing packet loss in a large complex network. https://apricot.net/apricot2013/assets/Google_Monitoring_APRICOT_2013.pdf.

[35] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proc. of SIGCOMM*. 404–421.

[36] Michael Isard. 2007. Autopilot: automatic data center management. *ACM SIGOPS Review* 41, 2 (2007), 60–67.

[37] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, et al. 2019. Validating datacenters at scale. In *Proc. of SIGCOMM*. 200–213.

[38] Vimalkumar Jeyakumar, Mohammad Alizadeh, Yilong Geng, Changhoon Kim, and David Mazières. 2014. Millions of little minions: Using packets for low latency network programming and visibility. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 3–14.

[39] Pravein Govindan Kannan, Nishant Budhdev, Raj Joshi, and Mun Choon Chan. 2021. Debugging Transient Faults in Data Centers using Synchronized Network-wide Packet Histories.. In *Proc. of NSDI*. 253–268.

[40] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *Proc. of SIGCOMM*, Vol. 15.

[41] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Computer Architecture News* 36, 3 (2008), 77–88.

[42] Petr Lapukhov and Aijay Adams. 2016. Netnorad: Troubleshooting networks via end-to-end probing. https://engineering.fb.com/2016/02/18/core-data/netnorad-troubleshooting-networks-via-end-to-end-probing/.

[43] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. Flowradar: A better netflow for data centers. In *Proc. of NSDI*. 311–324.

[44] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. Lossradar: Fast detection of lost packets in data center networks. In *Proc. of CoNEXT*. 481–495.

[45] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. 2018. A survey on network verification and testing with formal methods: Approaches and challenges. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 940–969.

[46] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

[47] Tal Mizrahi, Gidi Navon, Giuseppe Fioccola, Mauro Cociglio, Mach Chen, and Greg Mirsky. 2019. AM-PM: Efficient network telemetry using alternate marking. *IEEE Network* 33, 4 (2019), 155–161.

[48] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. 2022. FAst in-network GraY failure detection for ISPs. In *Proc. of SIGCOMM*. 677–692.

[49] Al Morton. 2016. Rfc7799: Active and Passive Metrics and Methods (with Hybrid Types In-Between).

[50] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proc. of SIGCOMM*. 85–98.

[51] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. 2009. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proc. of SIGCOMM*. 39–50.

[52] ns3 2022. Network Simulator 3. https://www.nsnam.org/.

[53] Tian Pan, Enge Song, Zizheng Bian, Xingchen Lin, Xiaoyu Peng, Jiao Zhang, Tao Huang, Bin Liu, and Yunjie Liu. 2019. Int-path: Towards optimal path planning for in-band network-wide telemetry. In *Proc. of INFOCOM*. IEEE, 487–495.

[54] Yanghua Peng, Ji Yang, Chuan Wu, Chuanxiong Guo, Chengchen Hu, and Zongpeng Li. 2017. deTector: a topology-aware monitoring system for data center networks. In *Proc. of USENIX ATC*. The Advanced Computing Systems Association.
[55] Peter Phaal, Sonia Panchen, , and Neil McKee. 2001. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. https://www.ietf.org/rfc/rfc3176.html.
[56] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving datacenter performance and robustness with multipath TCP. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 266–277.
[57] Arjun Roy, Deepak Bansal, David Brumley, Harish Kumar Chandrappa, Parag Sharma, Rishabh Tewari, Behnaz Arzani, and Alex C Snoeren. 2018. Cloud datacenter sdn monitoring: Experiences and challenges. In *Proc. of IMC*. 464–470.
[58] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. 2017. Passive Realtime Datacenter Fault Detection and Localization.. In *Proc. of NSDI*. 595–612.
[59] John Scudder, Rex Fernando, and Stephen Stuart. 2016. Rfc7854: BGP Monitoring Protocol (BMP).
[60] Vyas Sekar, Michael K Reiter, Walter Willinger, Hui Zhang, Ramana Rao Kompella, and David G Andersen. 2008. cSamp: A system for network-wide flow monitoring. (2008).
[61] Vyas Sekar, Michael K Reiter, and Hui Zhang. 2010. Revisiting the case for a minimalist approach for network flow monitoring. In *Proc. of SIGCOMM IMC*. 328–341.
[62] Gregg Siegfried. 2018. Monitoring Modern Services and Infrastructure. https://www.gartner.com/en/documents/3868219.
[63] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *Proc. of SIGCOMM SOSR*. 164–176.
[64] Han Hee Song, Lili Qiu, and Yin Zhang. 2006. NetQuest: A flexible framework for large-scale network measurement. *ACM SIGMETRICS Performance Evaluation Review* 34, 1 (2006), 121–132.
[65] Junho Suh, Ted Taekyoung Kwon, Colin Dixon, Wes Felter, and John Carter. 2014. Opensample: A low-latency, sampling-based measurement platform for commodity sdn. In *Proc. of ICDCS*. IEEE, 228–237.
[66] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2015. Cherrypick: Tracing packet trajectory in software-defined datacenter networks. In *Proc. of SIGCOMM SOSR*. 1–7.
[67] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying Datacenter Network Debugging with PathDump.. In *Proc. of OSDI*. 233–248.
[68] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2018. Distributed network monitoring and debugging with switchpointer. In *Proc. of NSDI*. 453–456.
[69] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks.. In *Proc. of NSDI*. 599–614.
[70] Yunhong Xu, Keqiang He, Rui Wang, Minlan Yu, Nick Duffield, Hassan Wassel, Shidong Zhang, Leon Poutievski, Junlan Zhou, and Amin Vahdat. 2022. Hashing Design in Modern Networks: Challenges and Mitigation Techniques. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 805–818.
[71] Kaicheng Yang, Yuanpeng Li, Zirui Liu, Tong Yang, Yu Zhou, Jintao He, Tong Zhao, Zhengyi Jia, Yongqiang Yang, et al. 2021. SketchINT: Empowering INT with TowerSketch for per-flow per-switch measurement. In *Proc. of ICNP*. IEEE, 1–12.
[72] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, et al. 2020. Accuracy, scalability, coverage: A practical configuration verifier on a global WAN. In *Proc. of SIGCOMM*. 599–614.
[73] Minlan Yu, Albert G Greenberg, David A Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. 2011. Profiling Network Performance for Multi-tier Data Center Applications.. In *Proc. of NSDI*, Vol. 11. 5–5.
[74] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch.. In *Proc. of NSDI*, Vol. 13. 29–42.
[75] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic test packet generation. In *Proc. of CoNEXT*. 241–252.
[76] Hongyi Zeng, Ratul Mahajan, Nick McKeown, George Varghese, Lihua Yuan, and Ming Zhang. 2015. Measuring and troubleshooting large operational multipath networks with gray box testing. *Mountain Safety Res., Seattle, WA, USA, Rep. MSR-TR-2015-55* (2015).
[77] Hongyi Zeng, Shidong Zhang, Fei Ye, Vimalkumar Jeyakumar, Mickey Ju, Junda Liu, Nick McKeown, and Amin Vahdat. 2014. Libra: Divide and conquer to verify forwarding tables in huge networks. In *Proc. of NSDI*. 87–99.
[78] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. 2020. Check before you change: Preventing correlated failures in service updates. In *Proc. of NSDI*. 575–589.
[79] Kaiyuan Zhang, Danyang Zhuo, Aditya Akella, Arvind Krishnamurthy, and Xi Wang. 2020. Automated verification of customizable middlebox properties with gravel. In *Proc. of NSDI*. 221–239.
[80] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. 2021. Hashing linearity enables relative path control in data centers. In *Proc. of ATC*. 855–862.
[81] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets.. In *Proc. of NSDI*. 991–1010.
[82] Naiqian Zheng, Mengqi Liu, Ennan Zhai, Hongqiang Harry Liu, Yifan Li, Kaicheng Yang, Xuanzhe Liu, and Xin Jin. 2022. Meissa: scalable network testing for programmable data planes. In *Proc. of SIGCOMM*. 350–364.
[83] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. Flow event telemetry on programmable data plane. In *Proc. of SIGCOMM*. 76–89.
[84] Yu Zhou, Dai Zhang, Kai Gao, Chen Sun, Jiamin Cao, Yangyang Wang, Mingwei Xu, and Jianping Wu. 2020. Newton: Intent-driven network traffic monitoring. In *Proc. of CoNEXT*. 295–308.
[85] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proc. of SIGCOMM*. 479–491.

APPENDICES

Appendices are supporting material that has not been peer-reviewed.

APPENDIX A ANALYSIS OF RANDOMIZED GENERATION

We present three properties and their formal proofs regarding randomized generation. Recall that $p_\alpha(n, m)$ denotes the probability that we collect α copies of each coupon after m draws of n types of uniform coupons. In this section, we investigate the following properties.

(1) The explicit formula and evaluation of $p_\alpha(n, m)$ (Appendix A.1)
(2) The growth rate of minimum m satisfying $p_\alpha(n, m) \geq p$ with respect to n (Appendix A.2)
(3) The coverage guarantee (Appendix A.3)

A.1 Computation of $p_\alpha(n, m)$

Notation. We use the following notation.

- Let $f_\alpha(x) = \sum_{i=0}^{\alpha-1} \frac{x^i}{i!}$ be the truncated Taylor series of the exponential function.
- Given an analytic function f at the origin, let $f[m]$ denote its coefficient of the x^m term in the Taylor expansion. For example, $e^x[m] = \frac{1}{m!}$ for any $m \in \mathbb{N}$.

An explicit formula. We show that

$$p_\alpha(n, m) = \frac{m!}{n^m}(e^x - f_\alpha(x))^n[m]. \tag{1}$$

Let $x_1, \ldots, x_n$ denote the copies of each type of coupon after m draws. Then, by the formula of multinomial coefficient, there are $\frac{m!}{x_1! \cdots x_n!}$ number of ways to collect these copies. Since there are n^m possible sequences of m draws, we have

$$p_\alpha(n, m) = \frac{1}{n^m} \sum_{\min x_i \geq \alpha,\ \sum x_i = m} \frac{m!}{x_1! \cdots x_n!}.$$

Note that $e^x - f_\alpha(x) = \sum_{i=\alpha}^{\infty} \frac{x^i}{i!}$. It follows that

$$(e^x - f_\alpha(x))^n[m] = \sum_{\min x_i \geq \alpha,\ \sum x_i = m} \frac{1}{x_1! \cdots x_n!}.$$

Thus, we derive the explicit formula of $p_\alpha(n, m)$.

Evaluation of the formula. Directly evaluating (1) for any triplet (α, n, m) is not feasible due to the complex expansion of the power

series. Instead, we use dynamic programming to evaluate its value in $O(\alpha^2 m^2 n^3)$ time. The boundary condition is

$$p_0(n,m) = 1 \qquad \text{for any } n,m \in \mathbb{N}, \tag{2}$$
$$p_\alpha(n,m) = 0 \qquad \text{for } \alpha > 0 \text{ and } m < \alpha n. \tag{3}$$

The transition equation is

$$\begin{aligned} &p_\alpha(n,m) \\ =&\sum_{i=0}^{n}(-1)^i\binom{n}{i}\binom{m}{i(\alpha-1)}\cdot\frac{(i(\alpha-1))!}{(\alpha-1)!^i}\cdot\frac{(n-i)^{m-i(\alpha-1)}}{n^m}\cdot \\ &p_{\alpha-1}(n-i,m-i(\alpha-1)) \qquad \text{for } \alpha>0 \text{ and } m \geq \alpha n. \end{aligned} \tag{4}$$

Note that in (4), $p_\alpha(n,m)$ only relies on the values whose α, n, m are smaller, so dynamic programming is applicable. Since each coefficient can be computed in $O(\alpha n + m)$ multiplications, the time complexity is $O(\alpha mn \cdot n \cdot (\alpha n + m)) = O(\alpha^2 m^2 n^3)$. The boundary condition (3) is straightforward. Moreover, we impose the condition (2) to enable the transition equation to hold even when $\alpha = 1$. The following calculation shows how to derive the transition equation.

$$\begin{aligned} &p_\alpha(n,m) \\ =&\frac{m!}{n^m}\left(e^x - f_\alpha(x)\right)^n[m] \\ =&\frac{m!}{n^m}\left(e^x - f_{\alpha-1}(x) - \frac{x^{\alpha-1}}{(\alpha-1)!}\right)^n[m] \\ &\qquad\qquad \triangleright \text{ By definition of } f_\alpha(x) \\ =&\frac{m!}{n^m}\sum_{i=0}^{n}\binom{n}{i}\left(e^x - f_{\alpha-1}(x)\right)^{n-i}(-1)^i\frac{x^{i(\alpha-1)}}{(\alpha-1)!^i}[m] \\ &\qquad\qquad \triangleright \text{ By binomial theorem} \\ =&\frac{m!}{n^m}\sum_{i=0}^{n}\frac{(-1)^i}{(\alpha-1)!^i}\binom{n}{i}\left(e^x - f_{\alpha-1}(x)\right)^{n-i}[m-i(\alpha-1)] \\ &\qquad\qquad \triangleright \text{ By definition of } f[m] \\ =&\frac{m!}{n^m}\sum_{i=0}^{n}\frac{(-1)^i\binom{n}{i}}{(\alpha-1)!^i}\frac{(n-i)^{m-i(\alpha-1)}}{(m-i(\alpha-1))!}p_{\alpha-1}(n-i,m-i(\alpha-1)) \\ &\qquad\qquad \triangleright \text{ By the formula of } p_{\alpha-1}(\cdot,\cdot) \\ =&\sum_{i=0}^{n}(-1)^i\binom{n}{i}\cdot\frac{m!}{(m-i(\alpha-1))!}\cdot\frac{(n-i)^{m-i(\alpha-1)}}{n^m\cdot(\alpha-1)!^i}\cdot \\ &p_{\alpha-1}(n-i,m-i(\alpha-1)) \qquad \triangleright \text{ Reorganizing coefficients} \\ =&\sum_{i=0}^{n}(-1)^i\binom{n}{i}\binom{m}{i(\alpha-1)}\cdot\frac{(i(\alpha-1))!}{(\alpha-1)!^i}\cdot\frac{(n-i)^{m-i(\alpha-1)}}{n^m}\cdot \\ &p_{\alpha-1}(n-i,m-i(\alpha-1)) \end{aligned}$$

A.2 Growth Rate of m

Suppose we fix the coverage requirement (α, p). We are interested in the growth speed of the minimal m that satisfies $p_\alpha(N,m) \geq p$ in terms of N. Our result is that $m = \Theta(N \ln N)$. Thus, back to Algorithm 2, we know each port requires $O(\ln N)$ tasks, where N is the radix of a switch. Consequently, a switch only needs $O(N \ln N)$ tasks in the randomized generation phase.

Deduction of the result. We deduct it in three steps. First, we show that such a minimal m always exists. Note that $p_\alpha(N,m)$ is non-decreasing in m and approaches one when $m \to \infty$. Thus, given any (α, p) and $p \in (0,1)$, the existence of such an m is clear.

Second, we show $m = O(N \ln N)$. Let X be a random variable denoting the first time when N copies of each coupon are collected. A known result is that $\mathbb{E}X = N\ln N + (\alpha-1)N\ln\ln N + O(N)$ [20]. According to Markov inequality,

$$1 - p_\alpha(N,m) = \Pr[X > m] \leq \frac{\mathbb{E}X}{m}.$$

By letting $m = \frac{1}{1-p}\mathbb{E}X = O(N\ln N)$, we have $p_\alpha(N,m) \geq p$.

Third, we show $m = \Omega(N\ln N)$. It suffices to show that the limit inferior of the ratio $\frac{m}{N\ln N}$ is at least one when $N \to \infty$. We may assume $\alpha = 1$ for now since this will only make m lower. A beautiful result [46] by Pierre-Simon Laplace, Paul Erdös, and Alfréd Rényi states that when $\alpha = 1$, for any constant c we have

$$\Pr[X < N\ln N + cN] \to e^{-e^{-c}}, \text{ as } N \to \infty.$$

By choosing $c = -\ln(-\ln p)$, we know that $p_1(N, N\ln n + cn) \to p$. Thus, $\liminf \frac{m}{N\ln N} \geq 1$ and the claim is proved.

A.3 Coverage Guarantee of Phase I

We repeat the coverage guarantee of Phase I below for convenience.

Theorem 1. *After exercising Phase-I tasks, for a switch with N ports, at least $1 - \frac{1}{2N^{1/4}}$ fraction of ports has attained α-coverage on average when N is large.*

Proof. Let us focus on any Tier L. A first observation is that each task will contribute coverage to two switch ports in L. This is because a task will traverse each layer in sequence in multi-tier datacenters (see the first assumption in §2.4). It goes through one port on Tier L in the uplink direction and another in the downlink direction.

Thus, the mathematical model is that there are N types of coupons, and we make $2Nt$ uniform and independent draws. Our goal is to show that at most $\frac{1}{2N^{1/4}}$ types of coupons have less than t copies on average.

Let q denote the probability that one particular type of coupon has $< t$ copies. Since q is the same for all types of coupons, the types of coupons with $< t$ copies are exactly qN on average. Thus, it suffices to show that $q \leq \frac{1}{2N^{1/4}}$.

Fortunately, we have an explicit formula of q by binomial distribution:

$$q = \sum_{i=0}^{t-1}\binom{2Nt}{i}\frac{1}{N^i}\left(1-\frac{1}{N}\right)^{2Nt-i}. \tag{5}$$

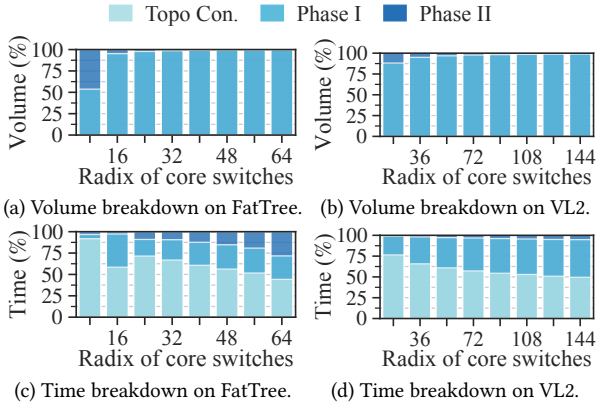
Now we apply two inequalities to simplify the term: $\binom{n}{x} \leq \frac{n^x}{x!}$ whenever $n \geq x$ and $1 - x < e^{-x}$. We have

$$q < \sum_{i=0}^{t-1}\frac{(2t)^i}{i!}\exp\left(-2t + \frac{i}{N}\right).$$

Recall from Appendix A.2 that $i < t = \Theta(\ln N)$. Thus, when N is large,

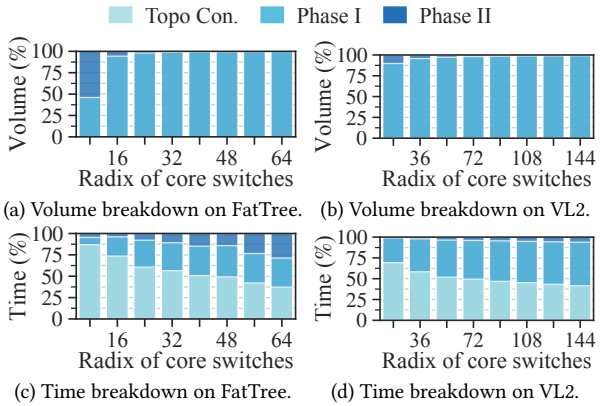
$$q \leq \sum_{i=0}^{t-1}\frac{(2t)^i}{i!}\exp\left(-2t\right).$$

Now, suppose X is a random variable following a Poisson distribution with mean $2t$. Then $q \leq \Pr[X < t]$. Meanwhile, we can view



(a) Volume breakdown on FatTree. (b) Volume breakdown on VL2.
(c) Time breakdown on FatTree. (d) Time breakdown on VL2.

Figure 12: Overhead breakdown when $(\alpha, p) = (3, 99\%)$.



(a) Volume breakdown on FatTree. (b) Volume breakdown on VL2.
(c) Time breakdown on FatTree. (d) Time breakdown on VL2.

Figure 13: Overhead breakdown when $(\alpha, p) = (5, 99\%)$.

X as the sum of t i.i.d. random variables, each following a Poisson distribution with mean 2. Therefore,

$$q \le \Pr\left[\sum_{i=1}^{t} X_i < t\right] = \Pr\left[\frac{\sum_{i=1}^{t} X_i - 2t}{\sqrt{2t}} < -\sqrt{\frac{t}{2}}\right].$$

When t is large, the central limit theorem says that $\frac{\sum_{i=1}^{t} X_i - 2t}{\sqrt{2t}}$ will converge in distribution to the standard normal distribution $\mathcal{N}(0,1)$. Let $l := \sqrt{t/2} > 0$. We have

$$\begin{aligned} q &\le \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-l} e^{-\frac{x^2}{2}} \mathrm{d}x = \frac{1}{\sqrt{2\pi}} \int_{l}^{\infty} e^{-\frac{x^2}{2}} \mathrm{d}x \\ &= \frac{1}{\sqrt{2\pi}} \int_{0}^{\infty} e^{-\frac{(x+l)^2}{2}} \mathrm{d}x \le \frac{e^{-\frac{l^2}{2}}}{\sqrt{2\pi}} \int_{0}^{\infty} e^{-\frac{x^2}{2}} \mathrm{d}x \\ &= \frac{1}{2} \exp\left(-\frac{l^2}{2}\right) = \frac{1}{2} \exp\left(-\frac{t}{4}\right). \end{aligned}$$

Recall from Appendix A.2 that $t \ge \ln N$. This implies $q \le \frac{1}{2N^{1/4}}$, which completes the proof. □

APPENDIX B ANALYSIS OF DETERMINISTIC GENERATION

The analysis of deterministic generation boils down to its coverage guarantee, which is stated in Theorem 2. We repeat the theorem here for convenience.

Theorem 2. *After exercising Phase II tasks, the coverage requirement (α, p) is achieved.*

Proof. After exercising Phase II tasks, we claim that for each interface $v.i$ containing n aggregated ports, there are at least $m^* := \min\{m : p_\alpha(n, m) \ge p\}$ tasks that go through it. This is valid for two reasons. (i) The coverage counter $C[v.i]$ never overestimates the number of Phase I tasks going through $v.i$ even in case of numerous faults (§5.2). (ii) Whenever $C[v.i] < m^*$, there will be $\delta = m^* - C[v.i]$ new tasks in Phase II going through this interface thanks to IP-in-IP ping. Thus, the interface will surely reach m^*-coverage.

Now, by definition of m^*, with $\ge p$ probability, each port in this interface will reach α-coverage. Let X_n denote the number of ports with α-coverage in this interface. We know $\mathbb{E}X_n \ge pn$. Summing over all interfaces, we know the fraction of ports in the whole datacenter that have α-coverage is

$$\frac{\sum_{\text{interface}} \mathbb{E}X_n}{\sum_{\text{interface}} n} \ge p,$$

which gives the desired result. □

It is noteworthy that Theorem 2 is topology-independent, as its proof does not assume any specific datacenter topology.

APPENDIX C MORE EXPERIMENTS

Microbenchmark. We show the results of volume and time breakdown given coverage requirement $(3, 99\%)$ and $(5, 99\%)$ in Figure 12 and 13, respectively. Similar to the results given $(1, 99\%)$, Phase I contributes to the majority of volume and 20-50% of time. Phase II contributes to less than 3% of volume and 20% of time. Topology construction consumes most of the time, but this overhead will likely be incurred only once (i.e., in the initial epoch).