AAsclepius: Monitoring, Diagnosing, and Detouring at the Internet Peering Edge

Kaicheng Yang[†,§], Yuanpeng Li[†,§], Sheng Long[†,¶], Tong Yang[†,§], Ruijie Miao[†], Yikai Zhao[†], Chaoyang Ji[¶], Penghui Mi[¶], Guodong Yang[¶], Qiong Xie[¶], Hao Wang[¶], Yinhua Wang[¶], Bo Deng[¶], Zhiqiang Liao[¶], Chengqiang Huang[¶], Yongqiang Yang[¶], Xiang Huang[¶], Wei Sun[¶], Xiaoping Zhu[¶]

[†]National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University

[§]Peng Cheng Laboratory, Shenzhen, China [¶]Huawei Cloud Computing Technologies Co., Ltd., China

Abstract

[1] Network faults occur frequently in the Internet. From the perspective of cloud service providers, network faults can be classified into three categories: cloud faults, client faults, and middle faults. This paper mainly focuses on middle faults. To minimize the harm of middle faults, we build a fully automatic system in Huawei Cloud, namely AAsclepius, which consists of a monitoring subsystem, a diagnosing subsystem, and a detouring subsystem. Through the collaboration of the three subsystems, AAsclepius monitors network faults, diagnoses network faults, and detours the traffic to circumvent middle faults at the Internet peering edge. The key innovation of AAsclepius is to identify the fault direction with a novel technique, namely PathDebugging. AAsclepius has been running for two years stable, protecting Huawei Cloud from major accidents in 2021 and 2022. Our evaluation on three major points of presence in December 2021 shows that AAsclepius can efficiently and safely detour the traffic to circumvent outbound faults within a few minutes.

1 Introduction

Network faults, including congestion, link failures, BGP misconfigurations, *etc.*, occur frequently in the Internet [1–8]. Obviously, network faults could degrade the network performance, and even lead to outages [9, 10], threatening the connectivity of the Internet. As a cloud service provider (CSP) which serves users at the Internet peering edge, the quality of service (QoS) for users is significantly harmed by the frequent network faults.

As pointed by BlameIt [11], from the perspective of CSPs, network faults can be classified into three categories based on where they occur (see Figure 1): 1) *cloud faults* which occur in the *cloud network* (cloud AS [2]); 2) *client faults* which occur in the *client network* (client AS); 3) *middle faults* which occur in the *middle network* (all AS'es between the cloud AS and the client AS). First, the cloud network is fully controlled by CSPs. We have already deployed a battle-tested system in Huawei Cloud to heal cloud faults, and this paper does not focus on cloud faults. Second, the client network is neither controlled by nor directly connected to CSPs. Therefore, when client faults occur, what we can do is to request the corresponding Internet service providers (ISPs) [11] for fault healing. Third, the middle network is not controlled by but directly connected to CSPs. Same as other CSPs, Huawei Cloud is connected to the middle network through dozens of points of presence[3] (PoPs). All PoPs and datacenters in Huawei Cloud are interconnected by a private backbone. Leveraging the private backbone, the traffic between clients and Huawei Cloud can go through different PoPs. This implies that we can choose different middle paths (paths in the middle network) by choosing different PoPs, so as to handle middle faults. In

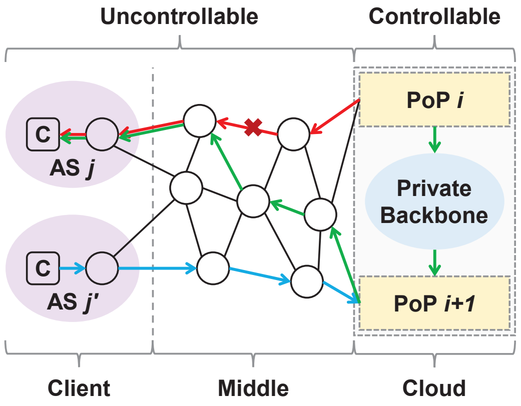

Figure 1: Three categories of network faults. "C" refers to a client in the client AS, and "×" refers to a middle fault. When a middle fault occurs in the red path, we can detour the traffic to egress at PoP$_{i+1}$, so as to route the traffic along the green path to circumvent the fault.

[1]Co-primary authors: Kaicheng Yang and Yuanpeng Li. Corresponding author: Tong Yang (yangtongemail@gmail.com).

[2]AS refers to autonomous system, which is a very large network or group of networks with a single routing policy.

[3]Point of presence is the point where Huawei Cloud accesses the Internet, see more details in Section 2.1.

summary, this paper focuses on middle faults, and identifies client faults as well.

The design goal of this paper is to design a fully automatic system to minimize the harm of middle faults. The system should have three main functions: fault monitoring, fault diagnosis, and traffic detouring. First, the system should persistently monitor the middle and client networks in a lightweight manner to detect and report faults. Second, the system should accurately diagnose the reported faults at fine granularity. Third, for a middle fault, the system should efficiently and safely detour the traffic to a healthy path, and detour the traffic back immediately when the fault disappears.

Towards the design goal, the most important and challenging issue is to identify the fault direction, since no existing works handle this problem. Suppose we find a middle fault between a PoP_i and a client AS_j. If a fault occurs in the path from PoP_i to AS_j, we define the direction of this fault as outbound direction, and similarly we define inbound direction. In this case, fortunately, we have many PoPs, and thus outbound faults can be quickly circumvented: detouring the traffic through another PoP_{i+1} to the client (see Figure 1). This action is quick because it only needs to update the routing table of PoP_i. If a fault occurs in the inbound direction from AS_j to PoP_i, unfortunately available solutions to change the traffic path need to change the routing tables of all routers in the middle network, which is obviously slow. The above observations pose great importance on *identifying the fault direction.* Further, it is challenging to identify the fault direction. On the one hand, the middle network where faults occur is completely out of our control. On the other hand, our monitoring results do not include direction information.

Researchers and engineers have proposed various solutions for network faults at Internet scale [1, 11–24]. Among them, BlameIt [11], Edge Fabric [12], Espresso [13], Entact [14], and CPR [15] are most related to our application scenarios. However, as shown in Table 1, these works only have two of the three main functions, and none of them identifies the fault direction. BlameIt monitors and diagnoses network faults in the cloud environment. However, it does not support traffic detouring and fault direction identification. Edge Fabric, Espresso, Entact, and CPR support traffic detouring, but they do not diagnose network faults. In summary, all existing works do not meet our design goal.

Aiming at the design goal, we design a system, namely **AAsclepius** (**A**uto**Asclepius**). AAsclepius consists of three subsystems: a monitoring subsystem, a diagnosing subsystem, and a detouring subsystem. Each subsystem is responsible for implementing a main function. Below we only show how the diagnosing subsystem addresses the above challenge of identifying fault direction.

Diagnosing subsystem: AAsclepius uses a decision tree with intuitive design to achieve the **accuracy** and **fine granularity** of diagnosis. Our experienced experts have spent a long time configuring the thresholds (§ 5) used in the decision tree. These thresholds have proven to work excellently after two years of validation. Our key innovation is to propose a technique, namely *PathDebugging*, to address the most important and challenging issue, *i.e.*, identifying the fault direction. For a middle fault between a PoP_i and a client AS_j, the idea of PathDebugging is to replace the path from AS_j to PoP_i with a zero-fault path. In spite of the simple idea, the implementation procedure is rather complicated, and the details are provided in Section 5.3.

To date, AAsclepius has been running for two years, keeping Huawei Cloud free of major accidents. In December 2021, we conducted an evaluation on three major PoPs. The results show that for all outbound faults, AAsclepius can efficiently and safely detour the traffic to circumvent them within a few minutes.

2 Settings

2.1 Cloud Infrastructure

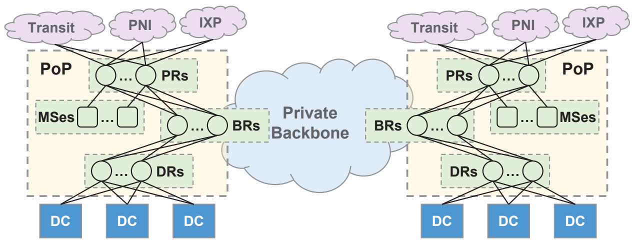

Figure 2: A PoP houses multiple peering routers (PRs) to access the Internet, backbone routers (BRs) to access the private backbone, and datacenter routers (DRs) to access datacenters. Each peering router is connected to a monitoring server (MS) cluster (currently we only deploy one server per PR) used for monitoring QoS (§ 4.2).

In order to serve approximately one billion geo-distributed users, Huawei has deployed a cloud consisting of dozens of PoPs and datacenters globally, as well as a private backbone that interconnects all PoPs and datacenters. To access the Internet, as shown in Figure 2, each PoP houses multiple peering routers (PRs) as edge routers, exchanging BGP routes and traffic with transit providers, private network interconnects (PNIs), and Internet exchange points (IXPs) outside the PoP. The interconnectivity between all PoPs and datacenters provided by the private backbone greatly improves the flexibility of traffic detouring. Traffic from any datacenter is first routed to the backbone routers (BRs) via datacenter routers (DRs). Through the private backbone, the traffic can then be routed to egress at any PoP. Similarly, the inbound traffic can also ingress at any PoP. Such flexibility of traffic detouring is the basis of AAsclepius (§ 6).

To serve users, each PoP announces a distinct set of IP prefixes as its *dominating prefixes*. Obviously, the PoP at which the inbound traffic will be routed to ingress is only determined

Desired functions	AAsclepius	BlameIt [11]	EdgeFabric [12]	Espresso [13]	Entact [14]	CPR [15]
Fault monitoring	✓	✓	✓	✓	✓	✓
Fault diagnosis	✓	✓	×	×	×	×
Direction identification	✓	×	×	×	×	×
Traffic detouring	✓	×	✓	✓	✓	✓

Table 1: Comparison with prior solutions.

by its IP destination address. Unfortunately, if a PoP fails to announce its dominating prefixes, the inbound traffic with IP destination address in those prefixes will not be routed to Huawei Cloud. To achieve fault resilience, each PoP additionally announces the dominating prefixes of the other PoPs with multiple duplicate AS'es prepended to their `AS_path`[4] (*e.g.*, 12345, 12345, 12345). In this configuration, the traffic is normally routed as before. If a PoP fails to announce its dominating prefixes, after BGP converges, the traffic that originally ingresses at that PoP will be routed to ingress at another PoP. Therefore, the availability of Huawei Cloud service is guaranteed as long as one PoP remains operational.

We have built a traffic monitoring system in Huawei Cloud, passively counting the sizes of flows entering or exiting Huawei Cloud at <Source IP, Destination IP> granularity for billing using programmable switches. This system not only helps AAsclepius with fault monitoring as it can determine the AS'es and IP /24 prefixes which contain clients of Huawei Cloud, but also helps AAsclepius with traffic detouring as it can provide visibility to the traffic volume between any PoP and AS. While it is cost-efficient to passively monitor stateless traffic volume, it is too expensive to maintain per-flow state for a large cloud to monitor network faults, and therefore AAsclepius still uses *active probing*.

2.2 Domestic Network Infrastructure

China's network is mainly controlled by three top-tier transit-providers. To serve geo-distributed users, each top-tier transit provider builds its own large-scale backbone network with sufficient intra-bandwidth interconnecting with all its client networks. By comparison, the inter-bandwidth across different transit providers is usually limited. Such infrastructure guarantees that from a PoP's perspective, the IPs in the same AS share similar middle paths. It also motivates AAsclepius' design (further shown in § 6): for traffic suffering from network faults, AAsclepius detours it across different PoPs within the same transit-provider for better performance, instead of detouring it across different transit-providers in the same PoP.

3 AAsclepius Overview

Currently, because most traffic of Huawei Cloud exits from PoPs deployed in our country, we have deployed AAsclepius on a large scale in our domestic infrastructure. As mentioned above, AAsclepius has three subsystems: a monitoring subsystem, a diagnosing subsystem, and a detouring subsystem.

[4]The BGP AS path attribute sequentially lists the AS numbers comprising the path to the destination.

Below we briefly present the modules and workflow of these subsystems (see Figure 3). Because most of Huawei Cloud traffic is still IPv4 traffic, AAsclepius targets at only IPv4 traffic currently. The terminologies frequently used in this paper are shown in Table 2.

Table 2: Terminologies frequently used in this paper.

Terminology	Meaning
Active IP address	An IP address which will respond to ICMP probes. (§ 4.1)
IP*	A representative IP address in a prefix for probing. (§ 4.1)
Victim-AS	An AS which is identified as suffering from faults. (§ 4.3)
Fake fault	A network anomaly that causes a healthy-AS to be identified as a victim-AS. (§ 5.1)
Backup PoP	PoP_{i+1} is the backup PoP of PoP_i. (§ 5.3)
Victim traffic	The traffic that is suffering from faults. (§ 6)

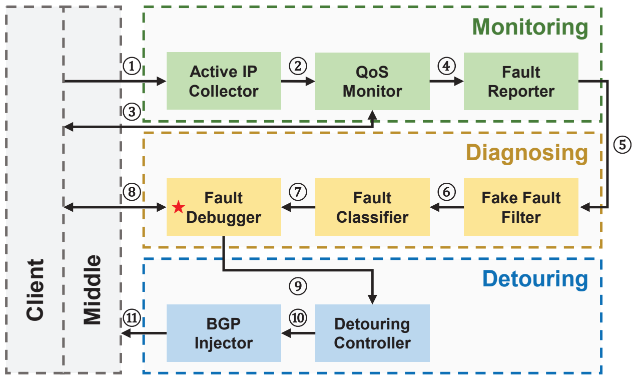

Figure 3: The workflow of AAsclepius.

Monitoring subsystem (Figure 3 top). This subsystem consists of three modules: an *active IP collector*, a *QoS monitor*, and a *fault reporter*. The QoS monitor and the fault reporter are deployed on a per-PoP basis.

① The active IP collector monitors all IP /24 prefixes in the AS'es which contain clients of Huawei Cloud, and collects all active IP addresses.

② The active IP collector selects representative IP addresses for each IP /24 prefix and informs the QoS monitor.

③ The QoS monitor in each PoP receives representative IP addresses, and runs QoS agents to send ICMP probes to representative IP addresses in order to measure their packet loss rates.

④ The QoS monitor aggregates packet loss rates at AS-granularity, and passes them to the fault reporter.

⑤ The fault reporter in each PoP receives the aggregated packet loss data from the QoS monitor in the same PoP, then identifies and reports victim-AS'es to the diagnosing subsystem.

Diagnosing subsystem (Figure 3 middle). This subsystem consists of three modules: a *fake fault filter*, a *fault classifier*, and a *fault debugger*. The fake fault filter is deployed on a per-PoP basis.

⑥ The fake fault filter receives reported victim-AS'es from the fault reporter in the same PoP, identifies and filters fake faults, and outputs the other faults to the fault classifier.

⑦ The fault classifier receives the faults from the fake fault filter in many PoPs, classifies the faults into three categories, and passes middle faults to the fault debugger.

⑧ The fault debugger receives middle faults from the fault classifier, and runs debugging agents to monitor a second path for each middle fault.

⑨ The fault debugger then compares the monitored packet loss rates of QoS agents and debugging agents to identify the direction for each middle fault.

Detouring subsystem (Figure 3 bottom). The detouring subsystem consists of two modules: a *detouring controller* and a *BGP injector*. The BGP injector is deployed on a per-PoP basis.

⑩ The detouring controller receives middle faults from the fault debugger, generates detouring strategy for each middle fault, and outputs the strategy to the BGP injector.

⑪ The BGP injector receives detouring strategy from the detouring controller, generates corresponding BGP routes, and announces them to PRs or DRs to detour the victim traffic.

4 Monitoring Network Faults

The monitoring subsystem consists of the active IP collector (§ 4.1), the QoS monitor (§ 4.2), and the fault reporter (§ 4.3). Each module will be introduced in one subsection.

4.1 Collector: Selecting Representative IPs

The first module in the monitoring subsystem is the active IP collector (collector for short). The collector, deployed in a VM in Huawei Cloud, is used to collect and select active IP addresses. It proceeds in two steps. 1) For only IP /24 prefixes in the AS'es which contain clients of Huawei Cloud, we decide to actively send ICMP probes to IP addresses in them, and regard the monitored performance as the QoS for users in this prefix. To guarantee the accuracy of active monitoring, we need to select active IP addresses as targets for probing. Therefore, the **first step** is to collect active IP addresses from the AS'es which contain clients of Huawei Cloud. 2) Considering that the active IP addresses in each IP /24 prefix share similar middle paths and client paths[5], it is of no value to monitor all of them. To achieve lightweight monitoring, we need to further reduce the overhead by selecting representative IP addresses (IP*s for short) in each prefix for probing. Therefore, the **second step** is to select IP*s in each IP /24 prefix, and then passes them to the second module, *i.e.*, the QoS monitor. Below, we describe the two steps of the collector in detail.

The first step of the collector proceeds as follows. The collector maintains *health points* as the indicator of activeness for each IP address, and only scans IP /24 prefixes in the AS'es which contain clients of Huawei Cloud by sending ICMP probes to each IP address in them periodically. In each round of scanning, if an IP address responds to the ICMP probes, its health points will increase; otherwise, its health points will decrease. The health points will decay over time, so as to indicate recent health status. After each round of scanning, the collector selects the IP addresses whose health points exceed a predefined threshold as active IP addresses.

The second step of the collector proceeds as follows. After each round of scanning, for each IP prefix, the collector sorts the active IP addresses in it based on the health points from the highest to the lowest. The IP addresses that usually belong to gateways (*e.g.*, .1, .254) will be given bonus health points before sorting. Then, for each IP /24 prefix, the collector selects the top-k active IP addresses in it as IP*s, and passes them to the QoS monitor. By default, we set k to 5. For those prefixes that contain clients of Huawei Cloud, we set a larger k to better reflect the QoS. Note that the information of the AS'es and IP /24 prefixes containing clients of Huawei Cloud is provided by the traffic monitoring system built in Huawei Cloud (§ 2.1).

4.2 Monitor: Monitoring QoS

The second module in the monitoring subsystem is the QoS monitor (monitor for short). The monitor is used to monitor the QoS for users in different AS'es. It is deployed based on the following considerations. 1) As Huawei Cloud can serve users from any PoP, we need to monitor the QoS by monitoring the performance of IP*s from every PoP. 2) To avoid the disturbance of cloud faults and traffic detouring (§ 6), we should start monitoring as close to the PR as possible. 3) Considering that the cloud traffic can egress at any PR, we should monitor all PRs to achieve full coverage of monitoring, *i.e.*, send probes to each IP* through all PRs. Therefore, AAsclepius deploys the monitor on a per-PoP basis as follows. As shown in Figure 2, for each PR, AAsclepius deploys a server directly connected to it, namely *monitoring server*. On each monitoring server, the monitor runs a QoS agent to send ICMP probes, which will egress at the PR connected to the monitoring server, so as to achieve full coverage. This deployment can easily scale out from a monitoring server to a cluster consisting of multiple monitoring servers, so as to improve the capability of the monitor.

The monitor proceeds in two steps. First, the monitor monitors the performance of each IP* received from the active IP collector. Specifically, **every minute**, the monitor executes a round of monitoring: for each IP*, each QoS agent sends ICMP probes to it, and computes its average packet loss rate

[5]Client paths refer to the paths in the client AS.

among all QoS agents, so as to achieve full coverage. We call this process **QoS monitoring**.

Second, the monitor aggregates the performance of IP*s at AS-granularity, and then passes the aggregated performance data to the third module, *i.e.*, the fault reporter. Specifically, after each round of monitoring, for each AS, the monitor computes the average packet loss rate of all IP*s in it as its packet loss rate. The reason behind is as follows. It is expected that we can monitor the QoS in every IP /24 prefix to achieve fault monitoring at prefix-granularity, but not all IP /24 prefixes contain active IP address. The network infrastructure in our country can ensure that the IP*s in the same AS share similar middle paths, and thus share similar middle faults (discussed in Section 2.2). Therefore, monitoring the QoS at AS-granularity will not degrade the sensitivity to middle faults. In our deployment, we find the above attribute also applies to the network of every ISP in every province. Therefore, we call the network of every ISP in every province a *pseudo AS*, and aggregate the performance of IP*s at pseudo-AS-granularity to ease maintenance. *In the rest of this paper, we always use AS to refer to pseudo AS.*

4.3 Reporter: Reporting Victim-AS'es

The third module in the monitoring subsystem is the fault reporter (reporter for short). The reporter, deployed on a per-PoP basis, is used to identify victim-AS'es and filter victim-AS'es suffering from transient faults. The reporter proceeds in two steps.

In the **first step**, the reporter receives the aggregated performance data from the QoS monitor deployed in the same PoP, and identifies whether an AS is suffering from faults based on three observations. The first observation is that the packet loss rate of each AS remains relatively stable when no fault occurs; once a fault occurs, the packet loss rates of the AS'es suffering from the fault suddenly increase. Therefore, for each AS, we decide to monitor the variation of its packet loss rate, and use a *fault threshold* to identify whether it is suffering from faults. The second observation is that different AS'es have different patterns of packet loss rates, because different AS'es point to different middle paths and client paths, and are maintained by different ISPs. Therefore, we should use an AS-specific fault threshold rather than a unified one. The third observation is that the condition of the Internet varies over time, and therefore the fault threshold should dynamically evolve as time goes by.

Based on the above considerations, the first step proceeds as follows. The reporter considers that each AS is in *healthy-state*, *i.e.*, not *victim-state* (suffering from faults), at the beginning. Here, we use "victim" instead of "faulty" to avoid confusion, as a client AS suffering from faults may have no faults occurring in its own network (the faults can occur in the middle path between the AS and the PoP to affect the traffic). The reporter maintains packet loss rates for each AS in recent W minutes. For AS$_j$, the reporter computes the average ($l_{avg}(j,t)$) and standard deviation ($\sigma(j,t)$) of packet loss rates, to characterize its current pattern, where t (Minute) is the current time. Let $\mathcal{T}_v(j,t)$ be the fault threshold used to identify whether AS$_j$ is suffering from faults at time t. We set

$$\mathcal{T}_v(j,t) = l_{avg}(j,t-1) + \max(\tau\sigma(j,t-1), \delta)$$

where δ is a constant (by default 3%) to filter minor faults. For AS$_j$, every minute, the reporter receives its current packet loss rate $l_{cur}(j,t)$ from the QoS monitor, and then compares $l_{cur}(j,t)$ with $\mathcal{T}_v(j,t)$. If $l_{cur}(j,t)$ exceeds $\mathcal{T}_v(j,t)$, the reporter identifies that AS$_j$ is suffering from faults, and transits AS$_j$ to victim-state. We call an AS in victim-state as a *victim-AS*, and a time period in which an AS is continuously in victim-state a *victim-period*. For a victim-AS, to avoid the disturbance of high packet loss rate caused by faults, its fault threshold will stop to update when it transits to victim-state from healthy-state. When the packet loss rate of a victim-AS stays below its fault threshold for W minutes, we consider that all its packet loss rates in the sliding window are not affected by faults. In this case, the victim-AS will transit back to healthy-state and restart to update its fault threshold. In our deployment, we set the window size W to 60 and τ to 3. Here, we provide some insights into their settings. We set window size W to 60 (1 hour) because it can stably present the recent condition of Internet. As shown in Figure 10, the average packet loss rate varies smoothly per hour (with less than 0.1% variation). We set τ to 3 because *3σ-rule* is widely used in outlier detection. When there is no fault, the loss rate of each IP* can be regarded as independent and identically distributed. Therefore, the average loss rate of IP*s ($l_{avg}(j,t)$) follows normal distribution according to central limit theorem, and applies to 3σ-rule.

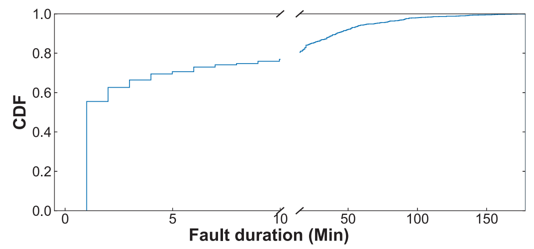

Figure 4: CDF of fault duration of victim-AS'es.

In the **second step**, based on another observation, the reporter filters victim-AS'es suffering from transient faults, and reports the remaining victim-AS'es to the diagnosing subsystem. The observation is that transient faults disappear quickly, and thus have limited harm to our cloud service. Therefore, we would like to ignore transient faults. Based on the above consideration, the second step proceeds as follows. Suppose the current time is t. The reporter only reports the victim-AS'es that satisfy the following two requirements to the diagnosing subsystem: 1) the victim-AS is identified as suffering from faults at time t; 2) the victim-AS has ever been identified as suffering from faults for at least M minutes continuously in the current victim-period. In our deployment, we set M to

3. We select M based on an analysis of the distribution of the fault duration[6] of victim-AS'es in three major PoPs in December 2021. As shown in Figure 4, the fault duration of almost 2/3 of victim-AS'es is below 3 minutes. Therefore, setting M to 3 can efficiently filter transient faults without compromising much timeliness.

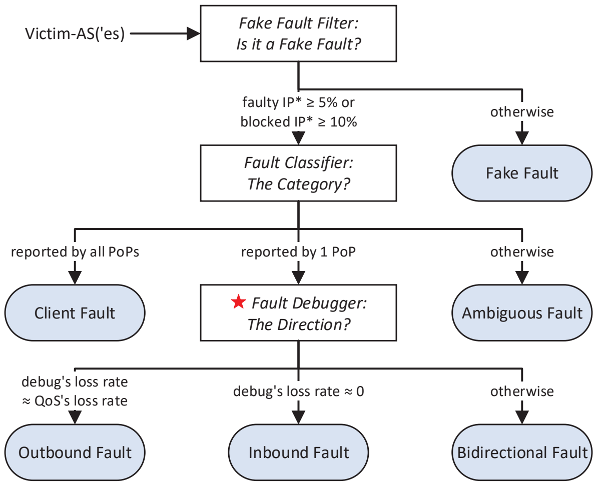

Figure 5: Decision tree.

5 Diagnosing Network Faults

In this section, we show how to use a decision tree in the diagnosing subsystem to achieve accurate and fine-grained diagnosis. As shown in Figure 5, there are three critical decision nodes in the decision tree: the fake fault filter (§ 5.1), the fault classifier (§ 5.2), and the fault debugger (§ 5.3). The design of the structure of the decision tree is quite intuitive. With the reported victim-AS'es, first, the fake fault filter first filters those fake faults that lead to misreported victim-AS'es. Second, the fault classifier pick out the middle faults that AAsclepius may circumvent from the true faults. Third, the fault debugger classifies the middle faults into inbound/outbound/bidirectional faults to guide the subsequent traffic detouring. Each decision node will be introduced in one subsection.

5.1 Fake-filter: Filtering Fake Faults

Motivation: The fault reporter in the monitoring subsystem reports a victim-AS when the average packet loss rate of all IP*s in the AS increase. However, the increase of average packet loss rate does not mean that a real fault occurs. For example, when a router that hosts an IP* is updating its operating system, it may not be able to respond to ICMP probes. Thus the packet loss rate of this IP* will suddenly increase to 100%, which also leads to the increase of average packet loss rate. If a victim-AS is actually healthy, we say it is suffering from **fake faults**. Therefore, it is desired for each PoP to filter all victim-AS'es with fake faults.

[6]The fault duration of a victim-AS refers to the interval between the first time and the last time it is identified as suffering from faults in the same victim-period.

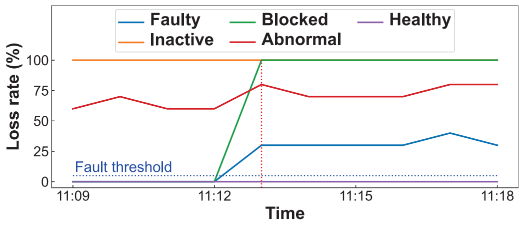

Figure 6: IP* Classification.

Workflow: To prevent fake faults from interfering with our diagnosis, AAsclepius deploys the first module in this subsystem: a fake fault filter for each PoP (*Fake-filter* for short). In each PoP, Fake-filter performs analysis for each IP* to identify whether a reported victim-AS is caused by a fake fault, and we will only diagnose real faults in next two modules. The input of Fake-filter includes: a victim-AS$_j$ with its IP*s, and the historical loss rate of each IP*. Then we analyze why the average packet loss rate increases a lot. Fake-filter divides the IP*s into 5 categories according to their loss rate (see Figure 6), but only use three categories (faulty IP*s, blocked IP*s, and healthy IP*s) to identify fake faults. The 5 categories of IP*s are detailed below.

- Symbols: Suppose at time t_{fault}, victim-AS$_j$ transits to victim-state, lasts for $M = 3$ minutes, and then is reported to Fake-filter. Note that the setting of M is discussed in Section 4.3. Let $T_{pre}[3Min]$ be the 3 minutes before t_{fault}, and let $T_{post}[3min]$ be the 3 minutes after t_{fault}.
- **Faulty IP*s:** We call an IP* a faulty IP* if it satisfies the three conditions: 1) its average loss rate in $T_{pre}[3Min]$ is lower than fault threshold $\mathcal{T}_v(j,t_{fault})$; 2) its average loss rate in $T_{post}[3Min]$ is in $[\mathcal{T}_v(j,t_{fault}), 100\%)$; 3) its highest loss rate in $T_{post}[3min]$ is lower than 100%. We consider that faulty IP*s are affected by the fault occurring at t_{fault}.
- **Blocked IP*s:** We call an IP* a blocked IP* if it satisfies the three conditions: 1) its average loss rate in $T_{pre}[3Min]$ is lower than $\mathcal{T}_v(j,t_{fault})$; 2) its average loss rate in $T_{post}[3Min]$ is higher than $\mathcal{T}_v(j,t_{fault})$; 3) its highest loss rate in $T_{post}[3Min]$ reaches 100%. These IP*s are often blocked in two cases. First, they are added into a blocklist by some network devices (IP blocking). Second, with a small probability, they suffer from serious faults.
- **Healthy IP*s:** We call an IP* a healthy IP* if its average loss rates in both $T_{pre}[3Min]$ and $T_{post}[3Min]$ are lower than $\mathcal{T}_v(j,t_{fault})$. We consider that healthy IP*s are not affected by the fault occurring at t_{fault}.
- **Inactive IP*s:** We call an IP* an inactive IP* if its average loss rate in $T_{pre}[3Min]$ is 100%. Inactive IP*s previously responded to ICMP probes, but stop responses before $T_{pre}[3Min]$.
- **Abnormal IP*s:** We call an IP* an abnormal IP* if its average loss rate in $T_{pre}[3Min]$ is in $[\mathcal{T}_v(j,t_{fault}), 100\%)$. We suspect that abnormal IP*s have suffered from other network anomalies or network faults occurring before $T_{pre}[3Min]$.

Identifying fake faults: After classifying faults into the above five categories, we analyze whether it is a fake fault. Obviously, the last two categories cannot be used for identification. We define two metrics using the first three categories. We define *faulty IP* ratio* as $\frac{\text{\# faulty IP*s}}{\text{\# total}}$, and define *blocked IP* ratio* as $\frac{\text{\# blocked IP*s}}{\text{\# total}}$, where # total = # faulty IP*s + # blocked IP*s + # healthy IP*s. We set two thresholds for the two metrics respectively. According to long time maintenance, we find when If the faulty IP* ratio is no less than 5%, or the blocked IP* ratio is no less than 10%, Fake-filter reports the fault as real; otherwise, Fake-filter reports the fault as fake.

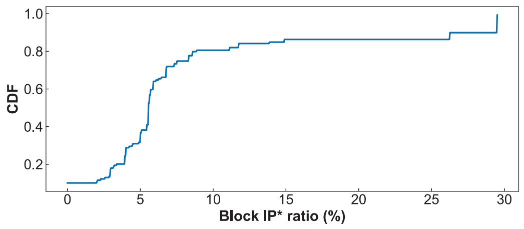

Figure 7: CDF of block IP* ratio of victim-AS'es without faulty IP*s.

While the threshold for faulty IP* ratio is mainly set according to long-time operational experience, we provide some insights in the setting of the threshold for block IP* ratio. When real faults occur, an affected IP* will be either faulty IP* or block IP*. When there is no real fault, the faulty IP* ratio keeps close to 0, but the blocked IP* ratio often reaches a little larger than 0 (*e.g.*, 5%) due to IP blocking. Therefore, we should set a larger threshold for blocked IP* ratio to filter the fake faults caused by IP blocking. We analyze the distribution of the block IP* ratio of all reported victim-AS'es without faulty IP*s in May 10th, 2023. These reported faults are of high probability to be fake faults as no faulty IP* is reported, and a fault with a larger block IP* ratio should have a larger probability of being real fault. As shown in Figure 7, about 80% reported victim-AS'es have less than 10% block IP* ratio. Therefore, setting the threshold for blocked IP* ratio to 10% may efficiently filter most fake faults while not missing serious faults.

5.2 Classifier: Identifying Fault Category

Overview: To identify fault category, AAsclepius deploys the second module in this subsystem: the fault classifier (classifier for short). Recalling in the previous module, Fake-filter in each PoP filters fake faults, and reports the other faults to the classifier. The input is the received faults from many PoPs in every minute: <PoP$_1$,victim-AS list$_1$> ... <PoP$_n$,victim-AS list$_n$>. For each victim-AS, the classifier outputs its fault category: client fault, middle fault, or *ambiguous fault*. For each middle fault, the classifier will report the corresponding PoP and victim-AS to the third module, *i.e.*, the fault debugger.

Workflow: Our observation is the same as the prior work [11]: most of the time the victim-AS is caused by either client faults or middle faults. We use the number of PoPs that report each victim-AS to identify fault category, and there are three cases.

- Case 1: If a victim-AS is reported from only one PoP, the classifier identifies the fault as a middle fault.
- Case 2: If a victim-AS is reported from all PoPs, the classifier identifies the fault as a client fault.
- Case 3: Otherwise, it is too difficult to identify, and we have to concede, and the classifier identifies the fault as an ambiguous fault.

The reason of the above classification is as follows. Considering a client in an AS, for different PoPs, the middle paths are different, while the client paths are usually similar. Therefore, 1) that a victim-AS is reported from 1 PoP is incurred by middle faults with high probability; 2) that a victim-AS is reported from all PoPs is incurred by client faults with high probability; 3) that a victim-AS is reported from multiple but not all PoPs, and the above probabilities of middle faults and client faults both decrease a lot. Our maintenance results (see Figure 8) show that the probability of case 3 is around 7%, and thus currently we just ignore case 3.

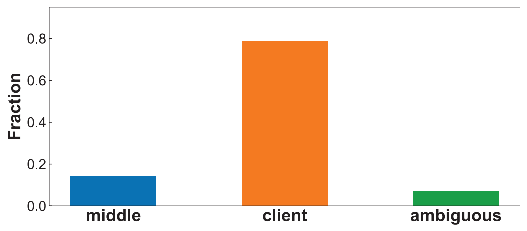

Figure 8: Fraction of each category of faults.

Operational experience: In our daily maintenance, we find a situation that will degrade the accuracy of our diagnosis: although multiple PoPs report the same victim-AS, the time they first report is not always the same, but sometimes with a one-minute or two-minute difference. A potential reason behind is as follows. In the early stage of a network fault, there may be only a small amount of congested links. Due to the randomness of the routing inside the victim-AS, the ICMP probes may go through paths with slight difference, and thus the results measured at different PoP points may be slightly different. For example, suggest that ICMP probes sent from both PoP$_i$ and PoP$_{i+1}$ reach a router using equal cost multi-path strategy (ECMP) in a client AS, and there are two equal-cost paths towards the destination. One of the path first becomes congested due to the randomness of hash functions, while the other one remains uncongested. Suggest that the ICMP probes from PoP$_i$ are forwarded to the congested link, and those from PoP$_{i+1}$ are forwarded to the uncongested link, then only PoP$_i$ reports this AS as a victim-AS. As the congestion further evolves (which may take one or two minutes), both PoP$_i$ and PoP$_{i+1}$ report this AS as a victim-AS. In this case, if the classifier diagnoses a victim-AS once it is reported, the victim-AS may be misdiagnosed because there could be some PoPs that have not reported this victim-AS in time. To address this issue, for each reported victim-AS, we decide to delay the diagnosis for two minutes. Specifically, only when a victim-AS is continuously reported from a PoP for three minutes, the classifier starts to diagnose it. The classifier diagnoses the victim-AS based on the number of PoPs that report it in the past three minutes, instead of the number in the current minute.

5.3 Debugger: Identifying Fault Direction

Overview: Recalling that the previous module reports a PoP and a victim-AS to this module. Suggest PoP$_i$ is reported to find that victim-AS is suffering from middle faults. This module, namely fault debugger (debugger for short), is used to further identify and output the direction of the middle fault. There are three faults with different directions: outbound faults, inbound faults, and bidirectional faults. We propose a novel technique, namely *PathDebugging*, to perform the debugging process. This technique is the key novelty of AAsclepius.

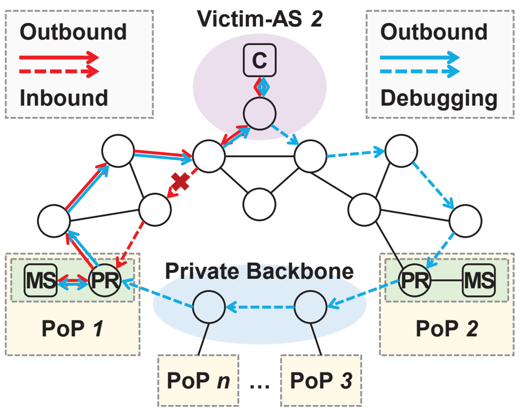

Figure 9: PathDebugging. "**MS**" refers to the monitor server in the PoP, "**C**" refers to a client in the victim-AS$_2$, and "×" refers to a middle fault.

Statement of fault direction: As shown in Figure 9, there are four paths between the monitor server (MS) and the client (C): two outbound paths (solid lines with arrows), one inbound path (dash red lines with arrows), and one debugging path (dash blue lines with arrows including the part across the private backbone). In this Figure, the reported PoP and victim-AS from the previous module are PoP$_1$ and AS$_2$. This means that we only know there is a fault between PoP$_1$ and AS$_2$, but do not know the fault direction. Outbound faults, inbound faults, and bidirectional faults point to different directions. For different fault directions, we will use different detouring strategies in the detouring subsystem (§ 6).

Rationale: To identify whether the fault is in the outbound path or the inbound path, the idea of our key technique PathDebugging is to replace inbound path with a zero-fault path, which is named the **debugging path**. After replacement, we monitor the packet loss rate of the ICMP probes between the reported PoP and victim-AS: 1) if the loss rate does not change, it means it is an outbound fault; 2) if the loss rate decreases to near 0, it is an inbound fault; 3) if the loss rate decreases but not reach 0, it is a bidirectional fault. Next we show how to set a debugging path and route the ICMP reply packets along the debugging path.

Workflow: Recall that each PoP has multiple PRs (peering routers), each PR is connected to a monitoring server, and each monitoring server runs a QoS agent. We deploy another agent named *debugging agent* in each monitoring server. These two agents are very similar except that they use different source IP addresses. By leveraging the debugging agent and BGP prefix announcement, next we show how to set the debugging path and let the ICMP reply packets follow the debugging path.

Phase 1: announcing BGP prefixes. We have deployed many PoPs: PoP$_1$, PoP$_2$, ..., PoP$_n$. We associate every two adjacent PoPs for debugging and detouring. We call PoP$_{i+1}$ the *backup* of PoP$_i$. For PoP$_i$, we assign a unique IP /24 prefix to the debugging agents in it, and the prefix is announced by all PRs in the backup PoP$_{i+1}$. For example in Figure 9, the PRs in PoP$_2$ announce the unique prefix of the debugging agent in PoP$_1$.

Phase 2: activating the debugging path. This phase aims to let the ICMP reply packets follow the debugging path, and observe the packet loss rate. Take Figure 9 as an example. The debugging agent in PoP$_1$ chooses a source IP address from its unique IP /24 prefix Pre_1, and then sends ICMP packets along the outbound path (the solid line) to the client. The ICMP reply packets will follow the debugging path (the dash blue line crossing the private backbone), because the PRs in PoP$_2$ announce the unique prefix Pre_1. Note that there is no fault in the debugging path for the following two reasons. First, as the middle fault is identified when only PoP$_1$ reports the fault, there must be no fault in the path from the victim-AS to PoP$_2$. Second, the private backbone is adequately provisioned, and thus can be considered as faultless. To save monitoring overhead, the debugging path is inactive by default, and will be activated when the module Fake-filter starts to report a victim-AS and a PoP.

Phase 3: identifying fault direction. Let l_{QoS} be the average packet loss that is monitored by QoS agents. Let l_{debug} be the average packet loss rate that is monitored by debugging agents. The fault debugger then identifies the fault direction according to the following formula.

$$Direction = \begin{cases} Inbound & \texttt{Cond}_1 \\ Outbound & \neg\texttt{Cond}_1 \wedge \texttt{Cond}_2 \\ Bidirectional & \neg\texttt{Cond}_1 \wedge \neg\texttt{Cond}_2 \end{cases}$$

$$\texttt{Cond}_1 = \left[l_{debug} \leqslant 3\%\right]$$
$$\texttt{Cond}_2 = \left[|l_{QoS} - l_{debug}| \leqslant 3\%\right]$$

The rationale behind the formula is as follows. 1) If it is an inbound fault, there should be no fault in outbound path and debugging path, and therefore l_{debug} should be small, which corresponds to $\texttt{Cond}_1 = \left[l_{debug} \leqslant 3\%\right]$. Here, 3% equals to the constant δ (Section 4.3) that we use to filter minor faults. 2) If it is an outbound fault, both debugging agent and QoS agent should detect the fault, and therefore they must not meet $\texttt{Cond}_1$. Further, considering there is no fault in inbound path and debugging path, the difference between l_{debug} and l_{QoS} should also be small, which corresponds to $\texttt{Cond}_2 = \left[|l_{QoS} - l_{debug}| \leqslant 3\%\right]$. 3) If it is an bidirectional fault, as discussed in 1) and 2), it should not meet $\texttt{Cond}_1$ and $\texttt{Cond}_2$.

6 Detouring Victim Traffic

The detouring subsystem consists of a detouring controller and a BGP injector deployed on a per-PoP basis. We describe how the detouring controller and the BGP injector cooperate to detour traffic suffering from faults (so called victim traffic), circumventing outbound faults (§ 6.1) and inbound faults (§ 6.2), respectively. For bidirectional faults, we can split them into outbound faults and inbound faults, and then circumvent them separately. Therefore, we will not discuss how to circumvent bidirectional faults.

6.1 Circumventing Outbound Faults

Rationale: For every outbound fault associated with one victim-AS$_j$ and one reported PoP$_i$, the traffic from PoP$_i$ to victim-AS$_j$ is the victim traffic. To circumvent the outbound fault, considering its backup PoP$_{i+1}$ not reporting victim-AS$_j$, we decide to detour the victim traffic to egress at PoP$_{i+1}$. As the cloud network is fully under control, to achieve this, we can inject BGP routes to DRs in PoP$_i$. Because traffic detouring will inevitably degrade the latency when there is no fault, we need to detour the victim traffic back as soon as possible after the fault disappears. As AAsclepius deploys the QoS monitor on monitoring servers directly connected to PRs, the traffic detouring at DRs will not interfere with QoS monitoring. Therefore, the monitoring subsystem can continuously identify whether victim-AS$_j$ is suffering from faults after detouring, and thus we can detour the victim traffic back when we have high confidence that the fault has already disappeared.

Workflow: The workflow of detouring victim traffic proceeds as follows. First, to ensure safety, before detouring the victim traffic, the detouring controller checks whether the PRs in PoP$_{i+1}$ and the private backbone will exceed 80% load rate after this detouring. Here, AAsclepius can easily determine the load rate of the PRs and private backbone after traffic detouring because the traffic monitoring system built in Huawei Cloud (§ 2.1) shares its visibility to traffic volume between any PoP and any AS to AAsclepius. Second, if the checking result is safe, the detouring controller then collects all IP prefixes of victim-AS$_j$ from PRs in PoP$_{i+1}$ rather than PoP$_i$, so as to guarantee that the PRs in PoP$_{i+1}$ can route the detoured traffic to the destination IP address. Third, the detouring controller passes the collected IP prefixes of victim-AS$_j$ and the IP addresses of PRs in PoP$_{i+1}$ to the BGP injector in PoP$_i$. In PoP$_i$, the BGP injector maintains a BGP connection with each DR. Fourth, the BGP injector generates corresponding BGP routes for the received IP prefixes of victim-AS$_j$. For these routes, the `local_pref`[7] is set to a very large value (*e.g.*, 1000), and the `next_hop`[8] is set to the received IP addresses of PRs in in PoP$_{i+1}$. Fifth, in PoP$_i$, the BGP injector announces the generated BGP routes to all DRs. By setting `local_pref` to a large value, the generated routes can override the original routes, and then the victim traffic will be detoured to egress at PoP$_{i+1}$. Once victim-AS$_j$ has been identified as not suffering from faults continuously for **10 minutes** by the fault reporter in PoP$_i$ (§ 4.3), the detouring controller will then inform the BGP injector in PoP$_i$ to withdraw the corresponding routes, so as to detour the victim traffic back.

6.2 Circumventing Inbound Faults

Rationale: For every inbound fault associated with one victim-AS$_j$ and PoP$_i$, the traffic from victim-AS$_j$ to PoP$_i$ is the victim traffic. Similarly, we decide to detour the victim traffic to ingress at PoP$_{i+1}$. However, the PoP at which the victim traffic ingresses is directly selected by ISPs, not the CSP. To address this issue, we can change the BGP announcement of PoP$_i$, and leverage BGP to detour the victim traffic. In order for the QoS monitor to continuously monitor existing faults to provide guidance on when to detour the victim traffic back, the change of the BGP announcement should not involve the prefixes assigned to the QoS monitor.

Workflow: The workflow of detouring victim traffic proceeds as follows. First, the detouring controller performs the safety checking similar to that in circumventing outbound faults. Second, In PoP$_i$, the detouring controller informs the BGP injector to announce the dominating prefixes of PoP$_i$ to all PRs. Note that the BGP injector needs to prepend multiple duplicate AS'es to the `AS_path` of these prefixes. In this way, after BGP converges, the victim traffic will be routed to ingress at PoP$_{i+1}$. Note that the dominating prefixes of PoP$_i$ are also announced by the other PoPs with multiple duplicate AS'es prepended to their `AS_path` (§ 2.1). We should ensure that PoP$_{i+1}$ prepends relatively less duplicate AS'es to the `AS_path` of these prefixes.

Discussion: A major risk of changing the BGP announcement at PRs is that it detours not only the victim traffic, but also all the other inbound traffic of PoP$_i$ (we call them innocent traffic). The latency of innocent traffic will inevitably degrade. Currently, considering the inevitable side effects, we currently disable AAsclepius to execute automatic detouring for inbound faults. AAsclepius only notifies the network operators of inbound faults, and provides an API for traffic detouring.

[7] The BGP local preference attribute is the second BGP attribute that can be used to choose the exit path for an AS.

[8] The BGP next hop attribute is the next hop IP address that is used to reach a certain destination.

6.3 Discussion

We first discuss the benefits of identifying fault direction, which is the key novelty of AAsclepius. Then, we discuss the potential downsides in the additional path asymmetry introduced by AAsclepius.

Benefits of identifying fault direction: Identifying fault direction can help minimize impacted traffic during traffic detouring. Because the start point of outbound path (cloud → client) is under control, to circumvent outbound faults, we can accurately determine outbound traffic requiring traffic detouring. Because the start point of inbound path (client → cloud) is beyond control, to circumvent inbound faults, we must change BGP announcement, and all inbound traffic is impacted. Therefore, with fault direction, we can reroute traffic accordingly to minimize impacted traffic.

Negligible downsides in introducing path asymmetry: According to RFC 3349 [25], the additional path asymmetry introduced by AAsclepius during detouring victim traffic may degrade performance of TCP traffic. Nevertheless, path asymmetry is a common phenomenon in the Internet (87% path-tuples show path asymmetry [26]). AAsclepius has been running for years, without customers complaining about performance degradation. Further, some prior works (Meta Edge-Fabric [12]/ Microsoft CASCARA [27]) also introduce additional path asymmetry as they switch transit-providers for better performance or cost-effectiveness. Therefore, we conclude that the additional path asymmetry should have negligible impact on traffic performance.

7 Evaluation

We first present the deployment status of AAsclepius (§ 7.1). Then, we present the performance of monitoring subsystem, diagnosing subsystem, and detouring subsystem (§ 7.2-7.4). In addition, we select several typical faults as case studies to illustrate the workflow of AAsclepius (see Appendix A).

7.1 Deployment Status

We have fully deployed AAsclepius on a large scale in our country. In August 2020, we start to run AAsclepius for just some provinces. After a month of testing, we start to run AAsclepius for the whole country. So far, AAsclepius has been running stable for two years. AAsclepius has diagnosed thousands of faults and circumvented more than two hundred middle faults. In 2021 and 2022, AAsclepius protects Huawei Cloud from major accidents. Our SRE team identifies network faults that cause more than five VIP customers to experience more than 5% packet loss rate for 10 minutes as major accidents. Major accidents typically last several hours, involving tens of AS'es, with (i) construction-related optical cable cuts, (ii) router failures, and (iii) traffic congestion being main root causes. For example, in August 2022, an outbound fault (may lead to major accident) affecting three provinces began at 20:57, resulting in a packet loss rate of up to 40%. AAsclepius executes traffic detouring at 21:04 (within 8 minutes) to circumvent the middle fault, and the fault finally ended at 22:30.

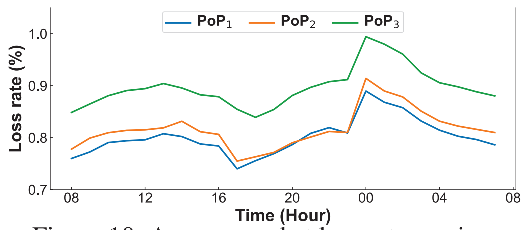

Figure 10: Average packet loss rate vs. time.

7.2 Performance of Monitoring Subsystem

We present the performance of the monitoring subsystem in three major PoPs in December 2021. First, we present the trend of packet loss rate and the distribution of victim-AS'es in different hours. Then, we present the distribution of fault duration of victim-AS'es. The following figures present data aggregated over 31 days in December.

Average packet loss rate vs. time (Figure 10): For the three major PoPs, we calculate the average packet loss rate of all IP*s in different hours. First, we find that the trend of the packet loss rate in each PoP is similar. This is possibly because the middle network in our country is adequately provisioned and well engineered, and thus the packet losses are mainly contributed by the client network. Because each IP* shares similar client paths in different PoPs, its packet loss rates in different PoPs are also similar, and thus the average packet loss rate in each PoP shares similar trends. Second, we find that the packet loss rate sharply increases to the peak at 0:00/24:00. It is possibly because ISPs in our country usually update routes and maintain network devices at this time, which is usually accompanied by network faults such as BGP misconfigurations, leading to the increase of average packet loss rate. Third, we find that the average packet loss rate in PoP_1 is slightly higher than that in PoP_2 and PoP_3. We suggest this is because that the middle network PoP_1 connected to usually has a relatively higher load.

Distribution of victim-AS'es vs. time (Figure 11): For each PoP, we count the distribution of victim-AS'es occurring in different hours. First, similar to the packet loss rate, we find that the distribution of the occurrence of victim-AS'es in each PoP is similar. Second, we also find that victim-AS'es are more likely to occur at 0:00/24:00. Third, we find that the distribution of the occurrence of victim-AS'es is positively correlated with the trend of packet loss rate. We suggest this is because a higher packet loss rate implies poorer network quality, which means more faults and thus more victim-AS'es.

Fault duration of victim-AS'es (Figure 12): Similar to Figure 4, we further present the distribution of fault duration of victim-AS'es in each PoP. We also find that the distribution is quite similar in each PoP. Considering that each PoP shares similar network quality, we can set unified parameters for each PoP to ease maintenance.

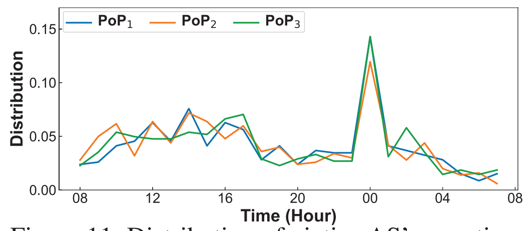

Figure 11: Distribution of victim-AS'es vs. time.

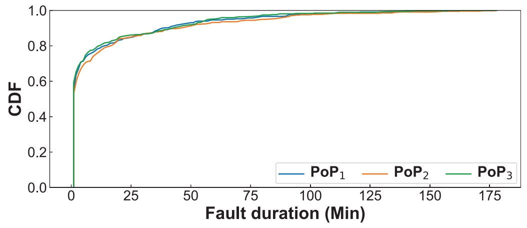

Figure 12: CDF of fault duration of victim-AS'es.

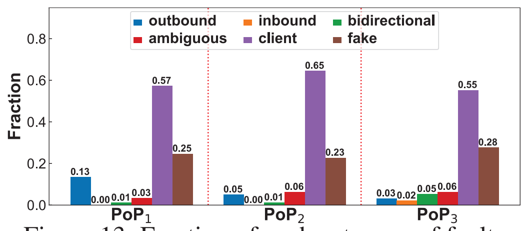

Figure 13: Fraction of each category of faults.

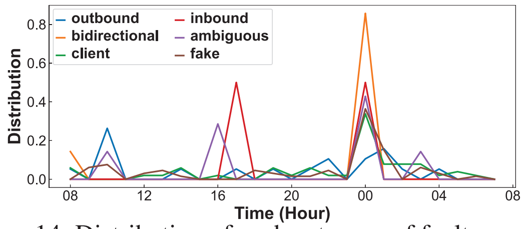

Figure 14: Distribution of each category of faults vs. time.

7.3 Performance of Diagnosing Subsystem

We present the performance of the diagnosing subsystem in three major PoPs in December 2021. First, we present the fraction of each category of faults. Then, we present the distribution of each category of faults in different hours. The following figures present data aggregated over 31 days in December.

Fraction of each category of faults (Figure 13): We count the fraction of each category of faults in each PoP. We find that the fractions in each PoP are similar: more than 50% of the faults are client faults, about 25% are fake faults, less than 15% are middle faults. It is possibly because the middle network in our country is well engineered and adequately provisioned, so that middle faults occur less frequently. We find that the fraction of outbound faults is far larger than that of inbound faults. The results show that outbound faults, inbound faults, and bidirectional faults account for 7%, 1%, and 2%, respectfully. We suspect this is because there is much more user download traffic than user upload traffic in the middle and client network, and thus the outbound paths are more likely to be congested.

Distribution of each category of faults vs. time (Figure 14): We count the distribution of each category of faults occurring in different hours. Similar to the distribution of victim-AS'es occurring in different hours (see Figure 11), we find that faults are more likely to occur at 0:00/24:00.

Potential misclassifications: Misclassifications are unavoidable. When there is a false positive (a non-middle fault is classified as a middle fault), AAsclepius may execute useless traffic detouring and increase the latency. When there is a false negative (a middle fault is classified as a non-middle fault), AAsclepius may not reduce the packet loss rate of victim traffic and receive complaints from customers. Because most network faults occur beyond our control, we can hardly obtain ground truth and misclassification rate. Nevertheless, AAsclepius can reduce packet loss rate in most traffic detouring (see Figure 16) and has protected cloud from major accidents for years, indicating an extremely low misclassification rate.

7.4 Performance of Detouring Subsystem

We present the performance of the detouring subsystem in three major PoPs in December 2021. We first present the response time of AAsclepius to middle faults. We then present the effect of traffic detouring on the packet loss rate and latency of victim traffic. The results show that the detouring subsystem is fast and effective. Note that we currently disable AAsclepius to execute automatic detouring for inbound faults, and thus all traffic detouring in this section is for outbound faults.

Evaluation criteria: In order to evaluate the detouring subsystem, we deploy *VM agents* in VMs in Huawei Cloud to perform *VM monitoring*. Similar to QoS agents, VM agents also send ICMP probes to each IP*. Because the probes sent from VMs are routed the same as the cloud traffic, we regard the performance of VM monitoring as the QoS, and use it to evaluate the effect of traffic detouring.

Response time to middle faults (Figure 15): We define the response time to a middle fault as the interval between the time its associated victim-AS transits from healthy-state to victim-state and the time the detouring subsystem reports the traffic detouring is executed. We find that the response time to all middle faults is within 8 minutes. Typically, the monitoring subsystem takes 3$\sim$5 minutes to identify and report a victim-AS; the diagnosing subsystem takes 3$\sim$4 minutes to identify its category and direction; the detouring subsystem takes less than 30 seconds to execute the traffic detouring.

Packet loss rate optimization (Figure 16): For each middle fault, we compare the average packet loss rate of its associated victim-AS in VM monitoring within 5 minutes before and after the corresponding traffic detouring is executed. We find that each traffic detouring reduces the packet loss rate by 7.0% on average. There are only less than 10% traffic detouring degrading the packet loss rate by less than 0.5%.

This is possibly because that the middle faults have already disappeared when the traffic detouring is executed.

Latency variation (Figure 17): For each middle fault, we compare the average latency of its associated victim-AS in VM monitoring within 5 minutes before and after the corresponding traffic detouring is executed. We find that each traffic detouring slightly degrades the latency by 1.9ms on average. This is reasonable because traffic detouring usually degrades the latency when no fault occurs, and the degradation has already been weakened by the middle faults.

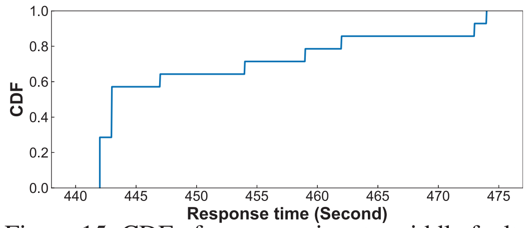

Figure 15: CDF of response time to middle faults.

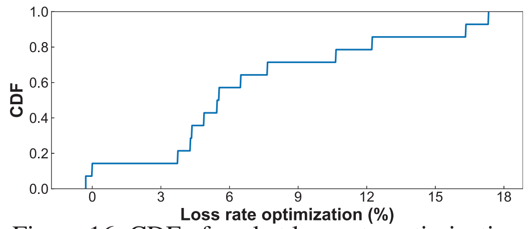

Figure 16: CDF of packet loss rate optimization.

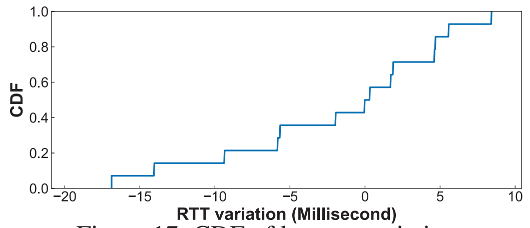

Figure 17: CDF of latency variation.

8 Related Work

Fault monitoring and diagnosis at Internet scale: Based on the measurement methods, existing fault monitoring and diagnosis solutions can be mainly classified into three categories: 1) active solutions which send probes to the Internet [14, 28–30]; 2) passive solutions which monitor ongoing connections [12, 13, 15, 17, 31, 32]; 3) hybrid solutions which combine active and passive solutions [1, 11, 16, 18, 19, 33]. Among these solutions, BlameIt [11] (hybrid), Entact [14] (active), Edge fabric [12] (passive), Espresso [13] (passive), and CPR [15] (passive) are most related to our application scenarios. BlameIt uses its passive measurement data to monitor faults and identify the fault category in the first phase, and further triggers impact-prioritized probes to localize the faulty AS for middle faults with the largest impacts in the second phase. However, BlameIt does not identify the fault direction, and is therefore distinguished from AAsclepius. The other solutions above do not diagnose faults but support traffic engineering, and we will cover them in the next paragraph.

Traffic engineering: There are substantial traffic engineering solutions, most of which are dedicated to optimizing CDN performance. A related kind of solutions select egress path and ingress point that a client should be directed to as a function of path performance [12–16, 34, 35]. Among them, Entact [14] measures path performance by sending probes to different IP /24 prefixes through alternate paths. Edge Fabric [12] and Espresso [13] passively measure path performance in different IP /24 prefixes by directing a small amount of flows to alternate paths and tracking their performance. Similar to AAsclepius, Espresso leverages Google's private backbone, B4 [36], and thus can route traffic to egress at distant PoPs. Through deployment in the kernel, CPR [15] even provides path failover at connection granularity. However, all the listed traffic engineering solutions optimize their traffic performance by selecting alternate outbound paths based on end-to-end measurement, and thus can only handle outbound faults. In contrast, AAsclepius detours traffic based on the category and the direction of the faults, and thus can handle both inbound and outbound faults. Therefore, these solutions are distinguished from AAsclepius. AAsclepius is also complementary to these solutions, as it can provide fine-grained fault category and fault direction information, which is useful for CDN performance optimization. Other solutions include IP anycast [37], co-located DNS and proxy servers [38], end-user mapping with EDNS [39], *etc.* [40].

Solutions in datacenters: Network faults in datacenters have been studied over decades, and researchers have provided various solutions [41–63]. These systems work excellently in data centers, but have not been extended to Internet scale.

9 Conclusion

Network fault is a widespread phenomenon in the Internet, which could harm the QoS of Huawei Cloud. Existing works do not identify fault direction. In this paper, we propose a fully automatic system, namely AAsclepius, to monitor and diagnose network faults, and detour victim traffic to circumvent middle faults. The key novelty of AAsclepius, PathDebugging, achieves identifying the directions of middle faults. AAsclepius has proven itself to be mature and reliable in two years of production deployment, and we consider extending AAsclepius to IPv6 network. Although the core methodology applied to IPv4 network can still be applied to IPv6 network, the main difficulty in the extension is how to efficiently find active IP addresses of high quality for QoS monitoring in the IPv6 address space which is much more larger than IPv4, and we are seeking for the solution.

Acknowledgment

We would like to thank the anonymous reviewers and shepherd Reto Achermann, for their help in improving this paper. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, National Natural Science Foundation of China (NSFC) (No. U20A20179).

References

[1] 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. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 1–15, 2018.

[2] Rodérick Fanou, Francisco Valera, and Amogh Dhamdhere. Investigating the causes of congestion on the african ixp substrate. In *Proceedings of the 2017 Internet Measurement Conference*, pages 57–63, 2017.

[3] Matthew Luckie, Amogh Dhamdhere, David Clark, Bradley Huffaker, and KC Claffy. Challenges in inferring internet interdomain congestion. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 15–22, 2014.

[4] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 350–361, 2011.

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

[6] Gianluca Iannaccone, Chen-nee Chuah, Richard Mortier, Supratik Bhattacharyya, and Christophe Diot. Analysis of link failures in an ip backbone. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 237–242, 2002.

[7] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. *ACM SIGCOMM Computer Communication Review*, 32(4):3–16, 2002.

[8] Nick Feamster and Hari Balakrishnan. Detecting bgp configuration faults with static analysis. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 43–56, 2005.

[9] Daniel Turner, Kirill Levchenko, Alex C Snoeren, and Stefan Savage. California fault lines: understanding the causes and impact of network failures. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 315–326, 2010.

[10] Monia Ghobadi and Ratul Mahajan. Optical layer failures in a large backbone. In *Proceedings of the 2016 Internet Measurement Conference*, pages 461–467, 2016.

[11] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 104–116. 2019.

[12] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431, 2017.

[13] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445, 2017.

[14] Zheng Zhang, Ming Zhang, Albert G Greenberg, Y Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, pages 33–48, 2010.

[15] Raul Landa, Lorenzo Saino, Lennert Buytenhek, and João Taveira Araújo. Staying alive: Connection path reselection at the edge. In *NSDI*, pages 233–251, 2021.

[16] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's scalable fault-tolerant cdn measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 501–517, 2018.

[17] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. Internet performance from facebook's edge. In *Proceedings of the Internet Measurement Conference*, pages 179–194, 2019.

[18] Rupa Krishnan, Harsha V Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize cdn performance. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 190–201, 2009.

[19] Ming Zhang, Chi Zhang, Vivek S Pai, Larry L Peterson, and Randolph Y Wang. Planetseer: Internet path failure monitoring and characterization in wide-area services. In *OSDI*, volume 4, pages 12–12, 2004.

[20] Anukool Lakhina, Mark Crovella, and Christiphe Diot. Characterization of network-wide anomalies in traffic flows. In *Proc. ACM IMC*, 2004.

[21] Ajay Anil Mahimkar, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Qi Zhao. Towards automated performance diagnosis in a large iptv network. *ACM SIGCOMM Computer Communication Review*, 39(4):231–242, 2009.

[22] Partha Kanuparthy and Constantine Dovrolis. Pythia: Diagnosing performance problems in wide area providers. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 371–382, 2014.

[23] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230, 2004.

[24] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 286–299, 2016.

[25] Hari Balakrishnan, V Padmanabhan, Godred Fairhurst, and Mahesh Sooriyabandara. Rfc3449: Tcp performance implications of network path asymmetry, 2002.

[26] Wouter De Vries, José Jair Santanna, Anna Sperotto, and Aiko Pras. How asymmetric is the internet? a study to support the use of traceroute. In *Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015, Ghent, Belgium, June 22-25, 2015. Proceedings 9*, pages 113–125. Springer, 2015.

[27] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-effective cloud edge traffic engineering with cascara. In *NSDI*, pages 201–216, 2021.

[28] Lin Quan, John Heidemann, and Yuri Pradkin. Trinocular: Understanding internet reliability through adaptive probing. *ACM SIGCOMM Computer Communication Review*, 43(4):255–266, 2013.

[29] Ítalo Cunha, Pietro Marchetta, Matt Calder, Yi-Ching Chiu, Bruno VA Machado, Antonio Pescapè, Vasileios Giotsas, Harsha V Madhyastha, and Ethan Katz-Bassett. Sibyl: a practical internet route oracle. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 325–344, 2016.

[30] Harsha V Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 367–380, 2006.

[31] Thomas Holterbach, Edgar Costa Molero, Maria Apostolaki, Alberto Dainotti, Stefano Vissicchio, and Laurent Vanbever. Blink: Fast connectivity recovery entirely in the data plane. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 161–176, 2019.

[32] Venkata N Padmanabhan, Sriram Ramabhadran, and Jitendra Padhye. Netprofiler: Profiling wide-area networks using peer cooperation. In *International Workshop on Peer-to-Peer Systems*, pages 80–92. Springer, 2005.

[33] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. Detecting peering infrastructure outages in the wild. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 446–459, 2017.

[34] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. Efficiently delivering online services over integrated infrastructure. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 77–90, 2016.

[35] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C Snoeren. Quantifying the benefits of joint content and network routing. In *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*, pages 243–254, 2013.

[36] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[37] Ted Hardie. Rfc3258: Distributing authoritative name servers via shared unicast addresses, 2002.

[38] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern cdns. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, 2015.

[39] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM Computer Communication Review*, 45(4):167–181, 2015.

[40] David Chou, Tianyin Xu, Kaushik Veeraraghavan, Andrew Newell, Sonia Margulis, Lin Xiao, Pol Mauri Ruiz, Justin Meza, Kiryong Ha, Shruti Padmanabha, et al. Taiji: managing global user traffic for large-scale internet services at the edge. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 430–446, 2019.

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

[42] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 662–680, 2020.

[43] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Passive realtime datacenter fault detection and localization. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 595–612, 2017.

[44] Arjun Roy, Rajdeep Das, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Understanding the limits of passive realtime datacenter fault detection and localization. *IEEE/ACM Transactions on Networking*, 27(5):2001–2014, 2019.

[45] Junzhi Gong, Yuliang Li, Bilal Anwer, Aman Shaikh, and Minlan Yu. Microscope: Queue-based performance diagnosis for network functions. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 390–403, 2020.

[46] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *NSDI*, 2016.

[47] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Lossradar: Fast detection of lost packets in data center networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 481–495, 2016.

[48] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 76–89, 2020.

[49] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016.

[50] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 ACM SIGCOMM Conference*. ACM, 2018.

[51] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *NSDI*, pages 991–1010, 2021.

[52] Kaicheng Yang, Yuhan Wu, Ruijie Miao, Tong Yang, Zirui Liu, Zicang Xu, Rui Qiu, Yikai Zhao, Hanglong Lv, Zhigang Ji, and Gaogang Xie. Chamelemon: Shifting measurement attention as network state changes. In *Proceedings of the 2023 ACM SIGCOMM Conference*. ACM, 2023.

[53] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 404–421, 2020.

[54] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *ACM SIGCOMM CCR*, volume 45. ACM, 2015.

[55] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016.

[56] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo,

and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 419–435, 2018.

[57] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. Taking the blame game out of data centers operations with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 440–453, 2016.

[58] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. Burstradar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*, pages 1–8, 2018.

[59] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. Netpilot: Automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 419–430, 2012.

[60] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 479–491, 2015.

[61] Cheng Tan, Ze Jin, Chuanxiong Guo, Tianrong Zhang, Haitao Wu, Karl Deng, Dongming Bi, and Dong Xiang. Netbouncer: Active device and link failure localization in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 599–614, 2019.

[62] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[63] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with pathdump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 233–248, 2016.

A Real-world Case Studies

To better illustrate the workflow of AAsclepius, we present several typical faults as case studies.

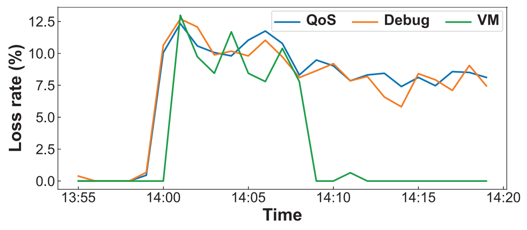

Figure 18: A typical outbound fault.

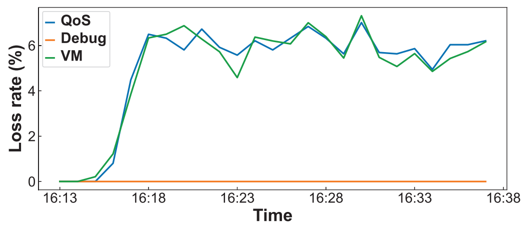

Figure 19: A typical inbound fault.

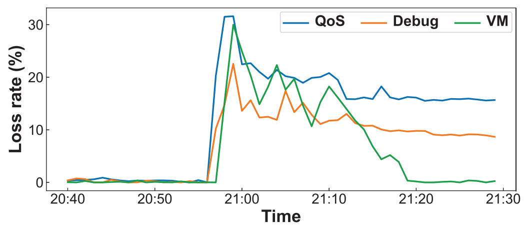

Figure 20: A typical bidirectional fault.

A typical outbound fault: Figure 18 plots the packet loss rates of QoS monitoring, debug monitoring[9], and VM monitoring during a typical outbound fault. The monitoring subsystem identifies a victim-AS at 14:00 at the first time, and reports the fault at 14:03. The diagnosing subsystem then identifies its fault category as a middle fault. The loss rate of debug monitoring keeps about 10%, which is similar to that of QoS monitoring. Therefore, the diagnosing subsystem identifies the middle fault as an outbound fault. The detouring subsystem detours the victim traffic at 14:08, and we can see the loss rate of VM monitoring suddenly decreases to almost 0%. In summary, the packet loss rate of VM monitoring decreases from up to 12.5% to almost 0% within 8 minutes. The outbound fault ends at 15:07, and the diagnosing subsystem detours the traffic back at 15:17, which is not plotted here.

A typical inbound fault: Figure 19 plots the packet loss rates of QoS monitoring, debug monitoring, and VM monitoring during a typical inbound fault. The monitoring subsystem identifies a victim-AS at 16:17 at the first time, and reports the fault at 16:20. The diagnosing subsystem then identifies its fault category as a middle fault. The packet loss rate of debug monitoring keeps less than 1%. Therefore, the diagnosing subsystem identifies the middle fault as an inbound fault. As we disable the automatic detouring for inbound faults, this fault is not circumvented, and the packet loss rate of VM monitoring keeps similar to QoS monitoring.

A typical bidirectional fault: Figure 20 plots the packet loss rates of QoS monitoring, debug monitoring, and VM monitoring during a typical bidirectional fault. This fault is a large-scale fault involving tens of victim-AS'es, and we just present one of them. The monitoring subsystem identifies the victim-AS at 20:57 at the first time, and reports the fault at 21:00. The diagnosing subsystem then identifies its fault category as a middle fault. The packet loss rate of the debug monitoring keeps around 15%, which is about 10% lower than that of the QoS monitoring. Therefore, the diagnosing subsystem identifies the middle fault as a bidirectional fault. The detouring subsystem detours the outbound traffic at 21:09, and the packet loss rate of VM monitoring decreases from about 20% to 10%. Due to the large scale of this fault, network operators manually detour the inbound traffic at 21:11, and the packet loss rate of VM monitoring decreases to less than 1% at 21:20. In summary, the packet loss rate of VM monitoring decreases from 30% to less than 1% within 20 minutes.

[9]We call the monitoring process performed by debugging agent in PathDebugging as debug monitoring for short.