Drift-Bottle: A Lightweight and Distributed Approach to Failure Localization in General Networks

Xudong Zuo
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory
Shenzhen, China
zuoxd20@mails.tsinghua.edu.cn

Qing Li[*]
Peng Cheng Laboratory
Shenzhen, China
liq@pcl.ac.cn

Jingyu Xiao
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory
Shenzhen, China
jy-xiao21@mails.tsinghua.edu.cn

Dan Zhao
Peng Cheng Laboratory
Shenzhen, China
zhaod01@pcl.ac.cn

Jiang Yong
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory
Shenzhen, China
jiangy@sz.tsinghua.edu.cn

ABSTRACT

Network failure severely impairs network performance, affecting latency and throughput of data transmission. Existing failure localization solutions for general networks face problems such as difficulty in acquiring data from end hosts, need for extra infrastructure, and excessive resource consumption. Meanwhile, solutions designed for data center networks are hard to apply in general networks, as they usually rely on the topology regularity of DCNs. In this paper, we propose Drift-Bottle, a lightweight and distributed approach to failure localization in general networks. In Drift-Bottle, each switch judges the status of flows and makes a local inference for suspicious links. We design a distributed localization scheme where each normal packet is used as a "drift-bottle" that carries a "letter", i.e., a lightweight inference header, while traversing the network. Each switch along the path updates the inference header by aggregating it with its local inference. Whenever the inference is evident enough to identify the culprit links of failures, a warning is sent to the operator immediately. Drift-Bottle implements its function mainly on the data plane of programmable switches and thus reduce the overhead brought to switches significantly. Evaluation based on simulation on different topologies demonstrates that Drift-Bottle provides fast, precise and lightweight failure localization to operators of general networks.

CCS CONCEPTS

• **Networks → Network monitoring**.

[*]Corresponding Author: Qing Li

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CoNEXT '22, December 6–9, 2022, Roma, Italy
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9508-3/22/12…$15.00
https://doi.org/10.1145/3555050.3569137

KEYWORDS

Failure Localization, General Network, Programmable Switch, In-network Intelligence

ACM Reference Format:
Xudong Zuo, Qing Li, Jingyu Xiao, Dan Zhao, and Jiang Yong. 2022. Drift-Bottle: A Lightweight and Distributed Approach to Failure Localization in General Networks. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3555050.3569137

1 INTRODUCTION

Are our networks robust enough? The short answer may be no. In the past decade, modern networks are growing at an incredible rate in both scale and complexity. To deliver a vast variety of applications and contents to millions of users in a fast and reliable way, ICPs and ISPs have upgraded their network infrastructures with thousands of servers and switches. However, the booming scale of networks inevitably harms its robustness while facing various failures. A production network may suffer from hundreds of failures per annual [24]. Failure of several switches can cause an immediate packet blackhole which can last for hours [7].

Link failures and corruptions are among the most common and frequent failures in networks. A failed link will drop all packets on it, whereas a corrupted link will drop packets at a considerable rate. Packet losses caused by link failures and corruptions severely impair network performance, affecting both latency and throughput of data transmissions. Therefore, it is essential for network operators to detect and localize the failed or corrupted links to mitigate the damage. Many monitoring tools have been designed to find the culprits of packet loss caused by link failure/corruptions in general networks. However, failure detection and localization in general networks is not easy. Existing solutions fall into two main categories: host-based solutions and switch-based solutions.

Host-based solutions rely on information collected from end hosts to localize failures. Network tomography [1, 5, 6, 18] gathers status (e.g., retransmission packets and surviving rates of packets) and path information of data flows from end hosts together, which are further analyzed using linear algebra or stochastic approaches to pinpoint failures. However, ISP network operators may not have the authority to visit users' end hosts. The heterogeneity of host models (e.g., PC, mobile phone and IoT device) also brings additional complexity to deployment of monitoring modules.

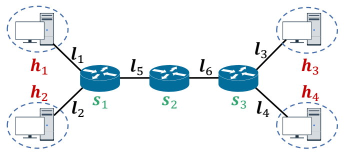
Switch-based solutions deploy their monitoring modules on switches. PreFix [27] extracts abnormal patterns from switch syslogs, which causes frequent CPU interrupts of switch and reduces the packet forwarding rate. SyNDB [13] and DynaFL [28] capture real-time snapshots of switches and submit them to a Data Collector and Analyst (DCA) periodically, which exerts network bandwidth and brings expenditure to operators for deploying extra infrastructures. Everflow [31] sets matching rules on switches and mirrors matched packets to analysts, which requires a considerable number of servers for analysis. NetSight [10] sends postcards of packets to servers for packet history reconstruction, which brings overhead to both switches and bandwidth. To detect inter-switch packet drops, NetSeer [30] records the sequence numbers of packets in each switch by a ring buffer. If discrete sequence numbers of received packets are detected by a switch, it will inform the upstream switch of the missing numbers, which will then check its ring buffer to raise a packet drop event. However, since the memory of the buffer is limited, the sequence numbers in the ring buffer may be overridden before the notification arrives if the delay of the link is too long. Therefore, NetSeer may not be operational in general networks, where there often exist links with long delays.

Exploiting the topology regularity of data center networks (DCNs), some solutions have been proposed to localize failures in DCNs. Pingmesh [9] utilizes the symmetry of DCN topology to reduce the overhead of real-time active probings. Relying on the hierarchical routing architecture of DCNs, NetBouncer [23] and 007 [2] localize failure by monitoring different data paths of more or less equal lengths. However, the strong dependency on topology regularity of DCN-based solutions hinders their application in general networks, which usually have much more complex and irregular topologies.

In this paper, we propose Drift-Bottle, a lightweight and distributed approach to fast failure localization in general networks. Drift-Bottle is designed to detect link failures and corruptions, and localize the links that are responsible for packet losses in the network.

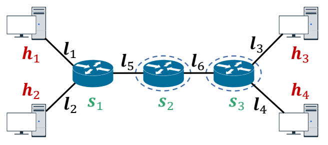
To avoid the inaccessibility of end hosts in general networks, we deploy its monitoring modules on switches. Each switch records flow-level features of data flows and uses a lightweight machine learning model deployed on its programmable data plane to periodically infer whether its monitored flows are normal or not. Then, each switch generates a *local inference* for the locations of suspected failures based on the flow status using a weight assignment scheme.

To achieve fast response to failures while inducing minimum bandwidth overhead, instead of submitting all local inferences to a DCA, we design a distributed mechanism to aggregate inferences from switches along data paths: we consider normal packets in the network as drift bottles that transport letters about failure inferences between monitors by flows. In detail, we add a special fixed-length lightweight inference header to each normal packet, which carries an aggregation of inference results made by all upstream switches along the data path. As the inference header "drifts" in the network along with a packet, each switch on its path would



$$A_{host} = \begin{array}{c} \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccc} l_1 & l_2 & l_3 & l_4 & l_5 & l_6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ h_1 \leftrightarrow h_2 \\ h_1 \leftrightarrow h_3 \\ h_2 \leftrightarrow h_3 \\ h_3 \leftrightarrow h_4 \end{array}$$

(a) Host-based Monitoring



$$A_{switch} = \begin{array}{c} \begin{array}{cccccc} l_1 & l_2 & l_3 & l_4 & l_5 & l_6 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ h_1 \leftrightarrow s_2 \\ h_2 \leftrightarrow s_2 \\ h_3 \leftrightarrow s_2 \\ h_4 \leftrightarrow s_2 \\ h_1 \leftrightarrow s_3 \\ h_4 \leftrightarrow s_3 \end{array}$$

(b) Switch-based Monitoring

Figure 1: Boolean path-link algebra of flow monitoring

update it by aggregating it with its local inference result before forwarding it to the next hop. In particular, if the updated inference is strong enough to identify the culprits of failures, a warning is raised immediately to the network operator. It is worth noting that operators need no extra physical links or servers to deploy Drift-Bottle on their networks, nor do they need to worry about bandwidth overhead caused by this system.

The main contributions of this paper are summarized as follow:

- We propose Drift-Bottle, a lightweight and distributed approach to failure localization in general networks. Drift-Bottle works on programmable switches to provide always-on and host-irrelevant failure localization.
- We design a distributed mechanism to aggregate failure inferences from different switches equipped with Drift-Bottle. This mechanism avoids excessive overhead and infrastructural change to the network, which ensures the deployability and scalability of our system.
- We evaluate Drift-Bottle by simulations in different failure scenarios and network topologies. The result shows that our system performs well under different topologies within a small time (about 0.1s) after the occurrence of failures.
- We build a prototype of Drift-Bottle by P4 language [4] and deploy it on Tofino [12]. The evaluation shows the overhead brought by our system is tiny and acceptable.

The rest of the paper is organized as follows. In Section 2, we explain the motivation of our design. In Section 3 and 4, we describe the overview of our system and details of each modules respectively. Then we describe the implementation in Section 5. In Section 6, we show the evaluation results of our system. In section 7, we introduce some works related to Drift-Bottle. At last, we make the conclusion of our paper in Section 8.

2 MOTIVATION

2.1 Switch-based Flow Monitoring

Network failures have adverse impacts on normal data transmission and drop packets from running flows. To detect and localize failures, host-based solutions deploy modules on end hosts to perform real-time monitoring. In general networks, however, the information obtained from monitoring is usually not sufficient to pinpoint the specific links to blame if we consider an end-to-end flow as the minimum monitoring unit. Take the boolean system from [15] as an example. We abstract a routing matrix **A** from the network where $A_{ij} = 1$ if the i-th path contains the j-th link and $A_{ij} = 0$ otherwise. Then we represent the connectivity of each path with a vector **b**, with $b_i = 0, 1$ indicates abnormal/normal status of the i-th path. The connectivity vector of all links x is defined in the same manner. After monitoring, we solve the binary inequality $\mathbf{A}x \geq \mathbf{b}$ with respect to x to find links that may drop packets. Unfortunately, due to the linear relativity between paths and monitoring limits in practice, the routing matrix **A** can be rank deficient, which leads to indefiniteness of the inequality. As shown in Figure 1, if either one of link l_5 and l_6 fails, the operator cannot tell which one of them to blame for, as all paths of monitored flows contain either both of them or neither of them.

Drift-Bottle enhances the capability of flow monitoring by deploying modules on switches rather than on end hosts. Switch-based monitoring modules can perceive the status of flows at each hop whereas host-based ones cannot, which will help operators to localize failures in fine granularity. Reconsider the topology in Figure 1. If s_3 detects anomaly from $h_1 - h_4$ but s_2 does not, the operator can infer that l_6 is the culprit of packet loss. Described in algebraic way, switch-based monitoring modules extend the dimension of row vector space of **A** by taking upstream/downstream part of flows as the minimum monitoring units, and reduce the number of unrestrained variables in x.

Moreover, in general networks, users may not authorize operators to deploy monitoring modules on their end hosts due to privacy consideration, which impeding network operators from deploying host-based monitoring. Also, the fact that end hosts are not always-on and their devices often come with a variety of models further hinders the effectiveness and feasibility of deploying monitoring modules on end hosts.

2.2 Flow Anomaly Detection

Controlled by the application and transport layers, an active flow will reach a steady state with stable transmission rate and RTT. A failure occurred in the path of the flow causes packet loss and breaks its steady state, which inspires us to detect and localize potential failures by finding out abnormal flows and their data paths. To take advantages of switch-based monitoring, we regard unidirectional flows determined by $<IP_{src}, IP_{dst}>$ as monitoring targets instead of bidirectional flows used by flow monitoring methods such as [2, 5, 6, 15, 18, 22, 29].

Figure 2 shows how unidirectional flows behave while encountering failures. When link l breaks down, packets of flow $<h_2, h_1>$ are dropped, which will be detected by the monitoring module on switch s. Although the broken link also drops packets of the opposite flow $<h_1, h_2>$, s can still receive its packets normally in a short time, as the upper layer (e.g. TCP in Transport layer) of h_1 needs to wait for certain time, such as Retransmission Timeout (RTO), before reacting to the absence of h_2.

To detect anomalies of monitored unidirectional flows, we collect some flow-level metrics periodically, such as the number of received

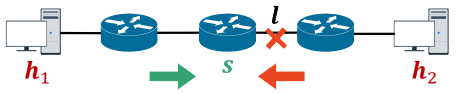


Figure 2: Behavior of Unidirectional Flows while Encountering Failures

packets and the amount of received bytes, which are easy to access on data plane. Significant changes in these metrics strongly indicates an anomaly of monitored flow. The most straightforward way to detect anomalies according to changes in these metrics is to adopt a threshold-based method. However, it is hard for such approach to distinguish the subtle difference between changes caused by potential failures and by normal events like the end of transmission.

In Drift-Bottle, we turn to a machine learning based anomaly detection approach. Machine learning models are good at mining the implicit correlation among high-dimensional data and abstracting the dynamic relationship between input and output, which well suits our requirement. Supported by in-network intelligence techniques, we deploy our ML-based models in pipelines of programmable switches to achieve fast and precise flow anomaly detection. Inference of failure can be made by jointly considering the status of monitored flows and their data paths.

2.3 Distributed Inference Aggregation

Information obtained from a single monitor is usually not sufficient to localize potential failures. To provide credible failure localization, most existing methods utilize a centralized mechanism for inference aggregation, such as introducing a DCA and gathering failure inferences from all monitors periodically. However, aggregating inferences from all monitors is not necessary in most cases. Intuitively, a failure can be detected and localized by aggregating just inferences from monitors nearby. Recall the example in Figure 1. When link l_6 fails, the status of flows perceived by s_2 and s_3 is enough to localize the culprit. This inspires us to design a distributed mechanism that aggregates inferences only from monitors "locally" to detect potential failures nearby.

A naive design for distributed aggregation is dividing the network into multiple zones statically and aggregating inferences in each zone. However, the aggregation based on static division does not fully exploit the status of cross-zone flows. As it is difficult to divide a general network into zones with few inter-zone links, we can not be optimistic about the performance of distributed aggregation based on static division.

In Drift-Bottle, we utilize normal packets of existing flows in the network to carry the inferences. Carried by a packet, the inference traverses each hop of the corresponding data path. Every time the packet is forwarded, the inference is aggregated. After multiple times forwarding, the inference will aggregate multiple inferences from different switches and may be "strong" enough to localize potential failures in the data path.

3 SYSTEM OVERVIEW

In this paper, we aim to design a solution to practical failure localization in general networks with following design goals.

Real-time monitoring. The monitoring system should be always-on in order to ensure quick reactions to potential network failures.

Hosts irrelevance. It should not require any information from end hosts, as operators may not be permitted by users to deploy monitoring modules on their devices.

Balanced Performance. It should make a reasonable trade-off between accuracy and sensitivity, as frequent false warnings are unacceptable to network operators.

Low overhead and favorable scalability. The system should not introduce excessive overhead and structural modification to networks.

To this end, we present Drift-Bottle, a lightweight and distributed approach to failure localization in general networks. Drift-Bottle meets the design goals with three key ideas:

Switch-based monitoring and inference generation. Drift-Bottle performs failure detection and localization by monitoring modules deployed on switches. The switch traces the status of flows passing through it by maintaining a group of registers in the data plane. If an anomaly is detected from a monitored flow, the switch considers all links belonging to its upstream data path as suspects. Oppositely, links corresponding to normal flows are innocent to potential network failures. In each time window, the switch separates abnormal flows from normal ones and generates local failure inferences according to their data paths.

Flow anomaly detection via in-network intelligence. In order to perform fast flow anomaly detection, Drift-Bottle utilizes in-network intelligence. In detail, we train a classifier offline and deploy it on the programmable data plane of a switch. The classifier takes flow-level features extracted during passive monitoring as inputs, and determines whether a data flow is normal or not. We implement the classifier by a decision tree due to two reasons. First, the decision tree is lightweight enough to be deployed on data plane under limited computing and storage resources. Second, the decision tree only relies on a group of classification rules to complete the classification, which can be easily converted into flow table rules in the data plane.

Distributed inferences aggregation. After the generation of failure inferences locally on each switch, we propose a distributed mechanism to aggregate them with low overhead. Drift-Bottle represents an inference by a special packet header with a fixed length of bytes. When a packet enters a switch, the data plane retrieves the inference stored in its header and aggregates it with the local inference to obtain an updated inference. If this updated inference is credible enough, a failure handling process will be triggered. Otherwise, the switch updates the header with the updated inference and forwards the packet as normal.

The overall architecture and workflow of Drift-Bottle are illustrated in Figure 3. At each Drift-Bottle enabled switch, in each time window, its **Flow Monitoring Module** collects flow-level features and feeds them to a decision-tree based classifier to separate normal and abnormal flows. Then, the **Inference Generation Module** uses path information of the monitored flows to generate a local inference, which essentially assigns a weight for each link, indicating how likely it is to be blamed for flow anomalies. Drift-Bottle represents an inference (containing only links with the top k highest weights) by a fixed length of bytes and stores it into a special

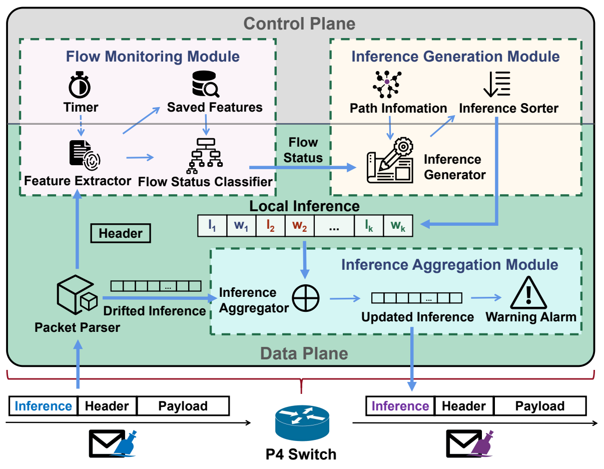


Figure 3: Overview of Drift-Bottle

header of packets. Once a packet arrives, the **Inference Aggregation Module** extracts the inference stored in its header, aggregates it with the local one, and updates the header with the aggregated inference. A warning about failures will be raised if the credibility of the new inference exceeds a threshold set by operators. Finally, the switch forwards the packet as normal.

To meet our design goals, Drift-Bottle performs real-time monitoring by switches and does not require any information from end hosts. Drift-Bottle utilizes the in-network intelligence technique and a distributed aggregation mechanism to provide credible inferences about network failures. The overhead introduced by our system is small as Drift-Bottle confines its main functions to the data plane, thus avoiding excessive data exchange between the data plane and the control plane, and adopts an in-packet method for inference aggregation. Last but not least, the deployment of Drift-Bottle requires no extra physical links or servers, which ensures the scalability of our system.

4 DESIGN DETAILS

4.1 Flow Monitoring Module

The flow monitoring module works based on two observations which are the product of transmission control logic on application and transport layers: a flow will reach a relative steady state if there is no failure in the network, such as self-similarity [32] and other statistical characteristics; once a failure (e.g., a link failure or link corruption) occurs, the relatively steady state of some flows will be broken, like a drastic drop of the transmission rate. As discussed in 2.2, a unidirectional flow will suffer from a failure located in the upstream part of its data path in spite of the transport layer protocol, which inspires us to localize potential failures by detecting anomalies of unidirectional flows passing monitoring switches.

We perform flow monitoring with the help of in-network intelligence. We utilize the technique from [20] to plant a decision tree on the programmable data plane, which transforms the classification rules of decision trees into the entries of match-action tables. The decision tree takes flow-level features as input and infers whether a unidirectional flow is determined by $<IP_{src}, IP_{dst}>$ is normal or not during the recent past period of time. To better capture the changes of monitored flows in the time dimension, we extract some measures from several sampling time intervals and formulate feature vectors by windows sliding on sampling intervals. The length of sampling intervals and sliding windows are configured by operators and are consistent across the network for the sake of scalability and deployability. To extract the behavior patterns of most flows, we set the length of sliding windows to the 90^{th} percentile of RTTs of all data paths in the network.

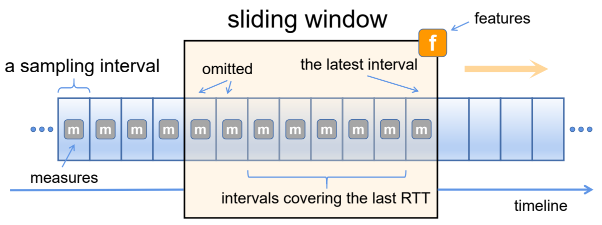


Figure 4: Sampling intervals and sliding windows

Measures in each sampling interval. Table 1 shows the measures extracted in each sampling interval. n_{packet} and len_{all} summary the amount of received data. The size of the biggest packet len_{max} and the size of the latest packet len_{last} are also recorded because a normal ending flow may transmit packets much smaller than MTU. We utilize these two measures, the number of bursts n_{burst} and the position of the last burst pos_{burst}, to depict detailed behaviors of monitored flows in each sampling interval. To obtain them, we divide each sampling interval into sub-intervals with serial numbers. A sub-interval will be labeled as a burst if the switch receives at least one packet from the monitored flow during it.

Table 1: Measures in Each Sampling Interval

Measure	Definition
n_{packet}	number of received packets
len_{all}	total size of received packets
len_{max}	size of the largest packet
len_{last}	size of the last packet
n_{burst}	number of bursts
pos_{burst}	position of the last burst

Features in each sliding window. The Feature Extractor generates a feature vector $x = (f_{flow}, f_{avg}, f_{last})$ for a monitored flow in each sliding window. Features in f_{flow} describe the characteristics of the monitored flow, such as RTT, length of the corresponding path, and the number of sampling intervals to cover its RTT. Features in f_{avg} describe the average behavior of the monitored flow in the last RTT, i.e., measures averaged over all sampling intervals in its last RTT. Features in f_{last} describe the behavior of the monitored flow in the last sampling interval. During offline training, we label a record of features as abnormal if the packets from corresponding unidirectional flow cannot reach the monitor at the time due to failures. Otherwise, it is labeled as normal. The details of features are summarized in Table 2.

Table 2: Features in Each Sliding Window

Type	Feature	Definition
f_{flow}	RTT	RTT of monitored flow
	len_{path}	length of data path of flow
	$n_{interval}$	n of intervals to cover a RTT
f_{avg}	avg_n_{packet}	avg. n_{packet} of intervals in last RTT
	avg_len_{all}	avg. len_{all} of intervals in last RTT
	avg_len_{max}	avg. len_{max} of intervals in last RTT
	avg_len_{last}	avg. len_{last} of intervals in last RTT
	avg_n_{burst}	avg. n_{burst} of intervals in last RTT
	avg_pos_{burst}	avg. pos_{burst} of intervals in last RTT
f_{last}	$last\_n_{packet}$	n_{packet} in last interval
	$last\_len_{all}$	len_{all} in last interval
	$last\_len_{max}$	len_{max} in last interval
	$last\_len_{last}$	len_{last} in last interval
	$last\_n_{burst}$	n_{burst} in last interval
	$last\_pos_{burst}$	pos_{burst} in last interval

During online monitoring, the module updates its measure registers when receiving packets of monitored flows. The data plane only maintains the measures of the latest sampling interval. We set a timer with a length of a sampling interval on the control plane. Once the timer expires, the control plane replicates measures of the latest measures from the data plane, calculates f_{avg} based on measures of former intervals, then sends f_{flow} and f_{avg} to the data plane. The data plane concatenates all types of features to form feature vectors of the current sliding window and feeds them to the decision tree to obtain the status of monitored flows.

4.2 Inference Generation Module

With results given by the Flow Monitoring module, the switch supposes to generate a local inference about the location of potential failures. As the failure model in 2.2 describes, an abnormal unidirectional flow indicates potential failures in the upstream part of its data path with respect to the monitoring switch. Naturally, the operator may narrow down the scope of potential failure locations to the intersection of upstream data paths of all abnormal unidirectional flows.

We propose a weight assignment scheme to measure the likelihood of corruption of each link. A weight counter w_i is set for each link l_i. Once l_i appears in the upstream data path of an abnormal flow, the corresponding counter w_i is incremented by 1. The link with the highest counter is blamed for existing flow anomalies.

However, the scheme above is not good enough as it does not utilize the information from the "innocent" part of data paths. Consider the situation shown in Figure 5 where unidirectional flows $<h_1, h_9>$, $<h_2, h_9>$, $\cdots$, $<h_8, h_9>$(marked by brown arrows) and also $<h_9, h_1>$, $<h_{10}, h_1>$ (marked by purple arrows) are active and monitored by switch s. The actual failure locates in link l_2, which leads to anomalies of unidirectional flows $<h_9, h_1>$ and $<h_{10}, h_1>$. Suppose the Flow Monitoring module on s correctly infers the status of <h_4, h_9>, ..., <h_8, h_9> as normal flows and <h_9, h_1>, <h_{10}, h_1> as abnormal flows but misclassifies <h_1, h_9>, <h_2, h_9> and <h_3, h_9> as abnormal flows. Using the method above, the operator finds that link l_1 gets a weight of 3 while link l_2 gets a weight of 2, and mistakenly believes l_1 is more responsible for the failure although actually most of the flows passing it stays normal.

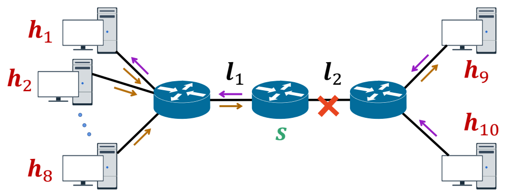


Figure 5: An example of false warning due to the absence of information from normal flows

It is worth noting that, for a normal flow inferred by the Flow Monitoring module, we tend to believe its upstream part of data path is less guilty of any failures. To exploit the information from normal flows, whenever a link l_i appears in the upstream data path of a normal flow, we decrement its corresponding counter by 1. Reconsider the toy situation in Figure 5 with the information from normal flows. The operator will find that l_2 is the most suspicious link with a weight of 2 while l_1 gets a weight of -2. Thus, the failure will be localized accurately.

The workflow of the Inference Generation module is outlined in Algorithm 1. We represent an inference by a set containing pairs formed by links and their weights, as $I = \{(l_i, w_i)\}$. Then, we define the aggregation operator $\bigoplus$, which simply aggregates inference $I_1 = \{(l_i, w_{1i})\}$ and $I_2 = \{(l_i, w_{2i})\}$ as $I_1 \bigoplus I_2 = \{(l_i, w_{1i} + w_{2i})\}$. During monitoring, the module generates an inference for each monitored flow based on its status and upstream part of data path. The control plane takes these inferences of each link away and aggregates them one by one to generate the local inference. Then, the control plane sorts the links in the local inference by their weights, and saves the top k links with the highest non-zero weights, where k is the length of inference. At last, the control plane sends the local inference to the data plane. Sorted by their weights, links belonging to more upstream data paths of abnormal flows but less upstream data paths of normal flows will rank higher, which indicates higher suspicion for causing failures.

4.3 Inference Aggregation Module

After the generation of local inferences on each monitoring switch, we propose a distributed mechanism to aggregate them and raise warnings about failures. Drift-Bottle utilizes normal packets in the network for inference aggregation. When a packet is received by the first switch from an end host, the Inference Aggregator module on the switch's data plane packs the local inference in a special header of the packet and lets the inference drift with the packet. Then, as the packet travels along its data path towards the destination, whenever it is received by a switch, the switch's Inference Aggregation module extracts the inference from the packet header, aggregates the received inference with its local inference, checks whether the new inference is strong enough to raise a warning

Algorithm 1: Local Inference Generation

```
Input: $F$ - set of monitored flows, $P$ - upstream data paths
       of flows, $L$ - set of links, $S$ - status of monitored
       flows, $k$ - length of inference
Output: $I$ - local inference about failures
1  $IF \leftarrow \emptyset$ on the data plane;
2  for $f \in F$ do
3      $path_f \leftarrow$ upstream data path of $f$ from $P$;
4      $status_f \leftarrow$ status of $f$ from $S$;
5      if $status_f = abnormal$ then
6          $I_f \leftarrow \{(l_i, 1) \mid \forall l_i \in path_f\}$;
7      else
8          $I_f \leftarrow \{(l_i, -1) \mid \forall l_i \in path_f\}$;
9      end if
10     $IF \leftarrow IF \cup \{I_f\}$;
11 end for
12 Upload $IF$ to the control plane;
13 $I \leftarrow \{(l_i, 0) \mid \forall l_i \in L\}$ on the control plane;
14 for $I_f \in IF$ do
15     $I \leftarrow I \bigoplus I_f$;
16 end for
17 Remove $(l_i, w_i)$ from $I$ if $w_i = 0$;
18 Sort $I = (l_i, w_i)$ in descending order by $w_i$;
19 Truncate $I$ to the $k$-th $(l_i, w_i)$;
20 Send $I$ to the data plane;
21 return $I = \{(l_i, w_i)\}$
```

about failures, updates the header with the new inference, and then forward the packet as normal. The process repeats until the packet reaches the last switch before its destination, where the switch deletes the inference from the header before forwarding the packet to the destination host.

Intuitively, after multiple times of aggregations, the weight of the culprit will far exceed that of normal links in drifted inferences. Thus, we propose a threshold-based mechanism for monitors to raise warnings about potential failures. After the aggregation and update of the drifted inference, the monitor checks whether the inference meets the conditions below:

$$\begin{aligned} hop_{now} &\geq hop_{min}\,, \\ w_0 &\geq \alpha \times hop_{now}\,, \\ w_0 &\geq \beta \times w_1\,, \end{aligned} \tag{1}$$

where hop_{now} is the number of times the drifted inference is received and aggregated; w_0 is the highest weight of accused links in the inference, and w_1 is the second highest one; hop_{min} and α are preset thresholds related to the scale of the network. The selection of β will be discussed in Section 6.7. The link with the highest weight will be labeled as the culprit when an inference exceeds those thresholds. Notice that multiple links can be reported by different drifted inferences, which ensures the capability of Drift-Bottle to handle concurrent failure scenarios.

As can be observed from equation (1), Drift-Bottle will not raise a warning unless the drifted inference has aggregated local inferences from at least hop_{min} switches, and at least α abnormal flows are detected by each switch on average. In practice, network operators can make a trade-off between sensitivity to potential failures and acceptable FPR (false positive rate) by adjusting the thresholds. With lower hop_{min} and α, Drift-Bottle is more sensitive when detecting network anomalies, but is also more prone to classification error caused by the flow status classifier. With higher hop_{min} and α, Drift-Bottle will be more tolerant to network "jitters" but may also miss out network failures which cause dropped packets from few flows.

The handling of warnings is up to network operators, according to their specific scenarios, and is out of the scope of this paper. For example, switches may perform local re-routing after a warning is raised, or activate probings towards accused links for more precise information of failures.

It should be noted that the switch does not update its local inference to the aggregated one although the latter may contain more information. We let each switch keeps its local inference unchanged in order to avoid *over aggregation*. Consider a simple linear topology formed by three switches s_1, s_2, s_3 sequentially, with their local inferences I_1, I_2, I_3. s_1 is forwarding multiple packets to s_3 via s_2. If s_2 updates its local inference after aggregation, the drifted inference from the n-th packets received by s_3 will be $n \times I_1 \bigoplus I_2$, which leads to a strong bias towards I_1 that may cause an incorrect warning about failures.

5 IMPLEMENTATION

We use P4 [4] (750 lines of codes) to implement Drift-Bottle in Intel Tofino model [12]. The P4 implementation mainly includes feature extraction, anomaly detection, inference generation and aggregation.

Feature extraction. Drift-Bottle's traffic features are implemented on the register in P4. At first, we extract measures divided based on sampling intervals. When a packet enters the switch, we get the *ingress_global_timestamp* defined in P4 $standard\_metadata$ and calculate $flow_{id}$ by hashing the 5-tuple ($IP_{src}, IP_{dst}, port_{src}, port_{dst}, protocol$). Suppose there are W sampling intervals, the i-th measure of the flow is indexed by $flow_{id} \cdot W + i$. Topology features f_{flow} are sent from the controller to the flow table of the switch. To reduce the consumption of memory units, only the measures in the last sampling interval will be maintained on the data plane. Measures of other intervals will be uploaded to the control plane to calculate f_{avg}.

Anomaly detection. Drift-Bottle's anomaly detection is implemented by match-action tables in P4. The whole table matches binary features, which are generated by concatenating topology features and traffic features of monitored flows. The entries of the tables are transformed from the rules of decision-tree-based classifiers. At the end of each sliding window, control plane takes measures of the current sampling interval. Then, it sends features f_{avg} to data plane by a specially constructed packet. Combined with features in the current window, the packet enters the flow anomaly detection module in the pipeline to get the status of monitored flows.

Inference generation. After the generation of flow status, Drift-Bottle updates the weights of local inference, and finally saves the top k links with the highest weights. The control plane of the switch mainly takes charge of the process. Drift-Bottle feeds the status into a match-action table formed by path information to get the inference of each flow. Then, we encapsulate the result as a packet, upload the packet to the control plane through the PCI bus, aggregate inferences of all monitored flows, sort the links by their weights in CPU, and then write sorted inferences back to the register by another constructed packet. The delay of the whole process is on a millisecond level. Notice that the process only appears at the end of each sampling interval, and only one pipeline is occupied during inference generation. Hence, the influence on throughput is negligible.

Inference aggregation. Drift-Bottle performs inference aggregation when a packet arrives at a switch. A certain register can only be read and written once in each stage of the P4 pipeline. To read the inference in one stage, we divide the bits of one register into individual parts and store the weight of one link in one part. In detail, we allocate 2 bytes for each accused link in the failure inference. The higher 1B encodes the identity of the link, and the lower 1B records the corresponding weight (-15~241, 0 is omitted). Drifted inferences requires 1B in addition to record hop_{now}. When the drifted inference of the arrived packet is extracted by parser, we utilize specialized match-action tables to aggregate it with the local inference and select the top k links with the highest weights to form a new drifted inference.

6 EVALUATION

6.1 Experiment Setup

Topology setting. To evaluate the performance of Drift-Bottle in different scenarios, we choose several topologies from TopologyZoo [14] and Rocketfuel [21], which vary in scale and structure. Geant2012 is the topology of the European academic network. Chinanet is about the same scale as Geant2012, but contains some busy nodes whose degrees are obviously greater than others, which makes the variance and skewness of degrees of nodes in Chinanet much bigger than ones in Geant2012 (17.30 to 3.79 and 2.63 to 1.42). Tinet connects its two main subnets with several very long links. AS1221 is the topology of a ring-like AS network. The basic statistics of the chosen topologies are shown in table 3.

Table 3: Statistics of Chosen Topologies

Topology	Node	Link	VAR. of link latency
Geant2012	40	61	14.12
Chinanet	42	66	8.09
Tinet	53	89	247.64
AS1221	104	151	9.39

Dataset generation. To train our decision-tree-based models in the flow monitoring modules, we generate datasets of network failures by simulation with Mininet [16] on these chosen topologies. We run the simulation under the settings below to approximate failure scenarios in real networks: the flows between each pair of hosts are generated randomly based on the preset flow density; the total bytes transmitted by the generated flows obey long-tailed distribution; the packet-sending process on each host obeys PPBP model [32] in order to maintain self-similarity in statistics.

During simulation, we inject failures by setting the status of links and nodes to failure manually, then we capture pcap records from each monitor before and after the occurrence of failures. We extract and label feature records of unidirectional flows from raw pcap records as shown in Section 4.1. The generated dataset is divided into a training set and a testing set at the ratio of 3:1 after feature extraction and labeling. For scenarios of single failure, we break down each link or node in the chosen topologies and generate dataset individually. We also generate the dataset of multiple failures for a detailed evaluation of our system.

Simulation setup. We evaluate Drift-Bottle mainly by simulation. We generate random traffic in the chosen topologies by Mininet, and record the forwarding history of packets as described before. Then we implement a simulator using Python, which replays the recorded traffic and simulates the work flow of our system. The simulator loads our offline-trained flow status classifiers, then performs feature extraction, inference generation, inference aggregation, and warning raising with replayed flows in different failure scenarios. We also implement some baselines in the simulator for evaluation. The simulation lasts for the largest RTT of all flows in the topology, which is at the magnitude of 0.1 seconds.

6.2 Baselines and Metrics

Baselines. To evaluate the weight assignment scheme shown in Section 4.2, we compare our system against different schemes. *Non-Negative* scheme adds 1 to the corresponding counter for a link related to an abnormal flow and does nothing otherwise. *007-Drifted* scheme is a variant of the voting scheme in [2], which adds $1/n$ to the counter for a link related to an abnormal flow where n is the length of corresponding upstream data path. In addition, *007-Modified* adds $-1/n$ to the counter for a link related to a normal flow.

We also compare Drift-Bottle against several centralized mechanisms to evaluate our distributed aggregation mechanism. *DB-Centralized* and *007-Centralized* use the same weight assignment scheme as Drift-Bottle and 007-Drift, respectively. These centralized mechanisms aggregate local inferences from all monitors together periodically. Then they utilize the procedure from [2] to find problematic links: centralized mechanisms check whether the weight of 1*st* link is greater than a preset portion of the sum of weights of all links or not. If so, they report the first link as a culprit, then execute the procedure again to the links that remained until no link exceeds the threshold.

For all distributed and centralized mechanisms, we collect links reported within a sliding window after the occurrence of failures. Then we compare them with the ground truth for evaluation. Unless stated, the length of inference of all distributed mechanisms is set to 4.

Metrics. The main measurements used for evaluation are *precision*, *recall* and *F1*. Notice that Drift-Bottle regards a link as the basic failure unit. Thus, we calculate *precision* as the ratio of correctly reported links among the warnings, and *recall* as the ratio of correctly reported links among actually failed links. *F1* is the harmonic average of *precision* and *recall*. We also introduce *accuracy* as the ratio of correctly classified links among all links, and *FPR* as the ratio of incorrectly accused links among innocent links. For example, in a scenario with 4 failures among 10 links, if a failure localization system reports 5 accused links and 3 of them are correct, its *precision*, *recall*, *accuracy* and *FPR* would be 60%, 75%, 70% and 33.3%, respectively.

6.3 Flow Status Classifier

Before the evaluation of the whole system, we show the performance of trained flow status classifiers. We set the length of a sampling interval to 4ms for all classifiers. Figure 6 shows the results. With the significant imbalance between normal and abnormal samples, we mainly focus on the recall of the classifiers for each class, i.e, the ratio of correctly classified samples of normal/abnormal flows.

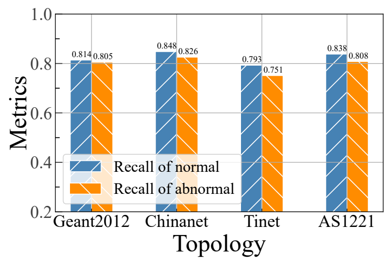


Figure 6: Flow status classifiers

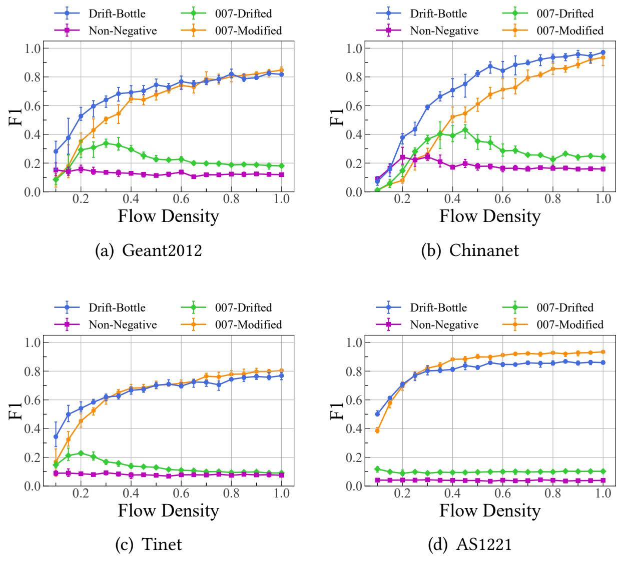


Figure 7: F1-score of different weight assignment schemes

6.4 Weight Assignment Scheme

In this experiment, we evaluate the performance of different weight assignment schemes in distributed aggregation mechanisms. We traverse every single link failure scenario on the chosen topologies and compare the metrics of our system with other weight assignment schemes under different flow densities (from 0.1 to 1.0). Figure 7 shows the F1-score of those weight assignment schemes. Drift-Bottle significantly outperforms Non-Negative and 007-Drifted schemes, as they do not utilize the information of normal flows. 007-Modified scheme reaches about the same performance as our system with the negative weights assigned to innocent links. However, 007-Modified scheme requires the representation and operation of float numbers, which is hard to implement on programmable data planes. Therefore, the weight assignment scheme of Drift-Bottle is still the optimal choice among these schemes.

6.5 Single Failure Scenario

Here we focus on the performance of our system when dealing with single link failure scenarios. We compare Drift-Bottle and 007-Drifted with their centralized versions, which aggregate failure inference from all switches together and raise warnings periodically. Figure 8 shows the results. Drift-Bottle clearly outperforms 007-Drifted and its corresponding centralized mechanism on all chosen topologies. It is not surprising that Drift-Bottle achieves better performances on Chinanet and AS1221. The star-like structure of Chinanet makes sure there are almost no unidentifiable links in its path-link algebra. So does the ring-link structure of AS1221. Oppositely, several long links carry most of inter-subnets flows in Tinet, which harms the performance of our system. The accuracy of our system is beyond 98.59%, while the FPR never exceeds 0.5%.

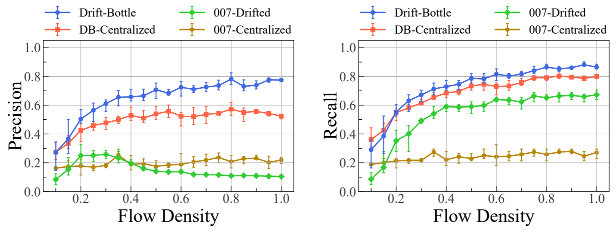
It should be noticed that Drift-Bottle outperforms its centralized version under high flow density. We find it reasonable as the centralized mechanism aggregates failure inferences from all switches and naturally introduces bias to innocent links. For example, a busy but normal link may get considerable weights in local inferences of several switches because of noise. Once aggregated by centralized mechanism, it may overshadow the signal of the real culprit, leading to an incorrect warning.

6.6 Multiple Failures Scenarios

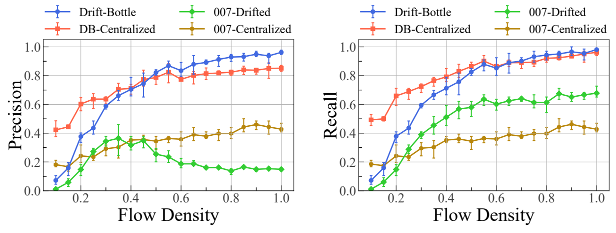
The next experiment aims to evaluate the ability of Drift-Bottle when facing multiple failures.

Multiple failures caused by single node failure. First, we introduce multiple failed links by traversing every single node failure scenario. A node failure is equivalent to failures of all connected links as our system regards a link as the basic failure unit. Figure 9 shows the results. Drift-Bottle still performs better than other schemes. Compared with itself in single link failure scenarios, our system gets lower recall as the number of failed links is much larger. However, operators still benefit from the high precision of Drift-Bottle, as the failed node will be located naturally once several connected links are reported. The accuracy of our system is beyond 97.76%, and the FPR is 0.5%.

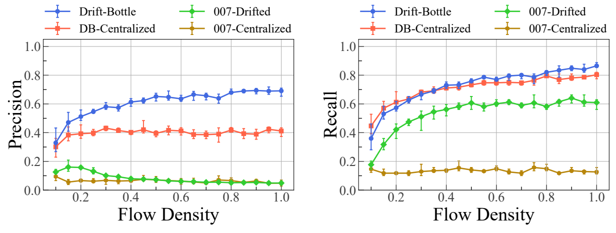
Random multiple failures. Then, we evaluate Drift-Bottle by injecting multiple failures randomly. We set failure units at each number randomly for 30 epochs and calculate the metrics of



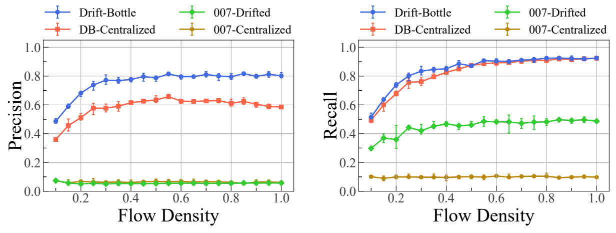
(a) Geant2012



(b) Chinanet



(c) Tinet



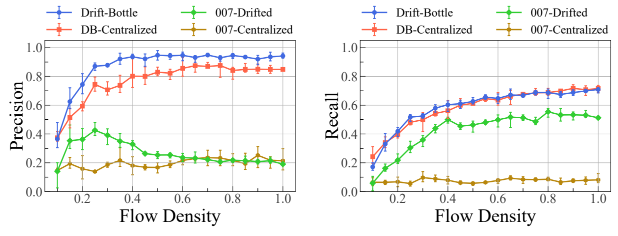
(d) AS1221

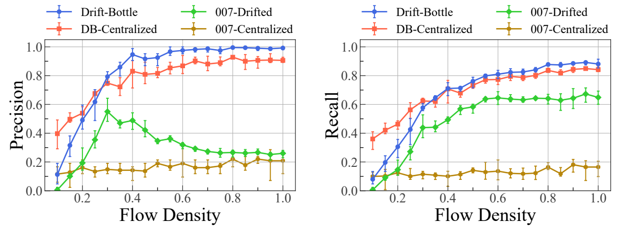
Figure 8: Single link failure scenarios

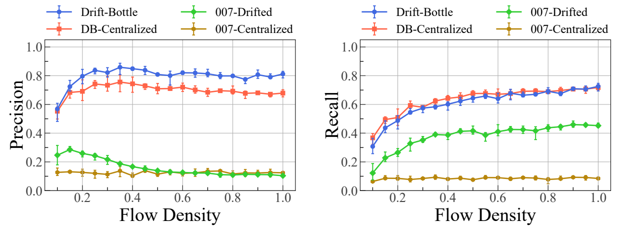
our system. Figure 10 shows the main metrics of Drift-Bottle on Chinanet under flow density of 1.0. Although the accuracy, recall and F1-score all drop inevitably as the number of failures increases, the precision of Drift-Bottle maintains at an considerable level. Our system performs similarly on other topologies.

6.7 Threshold-based Warning Raising Mechanism

In this experiment, we evaluate the performance of our threshold-based warning raising mechanism. We mainly discuss the selection of β as other parameters are determined by the scale of the network. Recall the mechanism shown in 4.3, the first link will be reported as a culprit if $w_0 \geq \beta \times w_1$, where w_i is the weight of the i-th link l_i in a


(a) Geant2012


(b) Chinanet


(c) Tinet

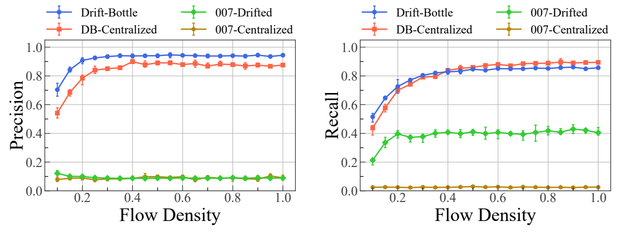

(d) AS1221

Figure 9: Multiple links failures caused by single node failure

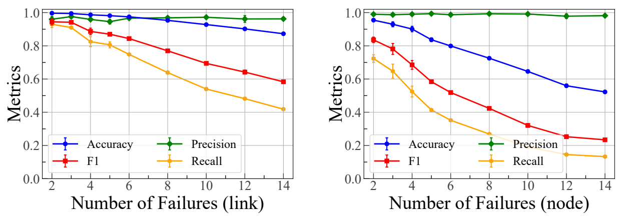
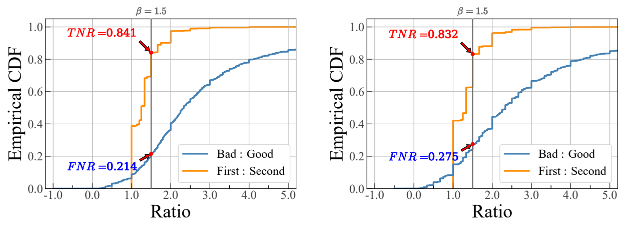

(a) Multiple links (b) Multiple nodes

Figure 10: Multiple failures scenarios (Chinanet)


(a) Geant2012 (b) Chinanet

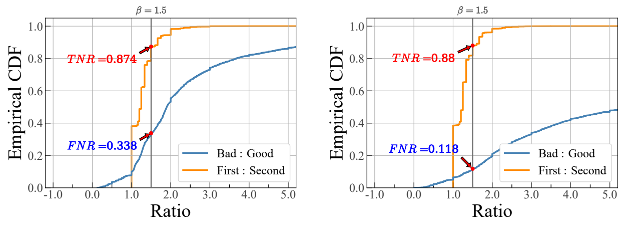

(c) Tinet (d) AS1221

Figure 11: CDFs of ratios of drifted inferences in single link failure scenarios

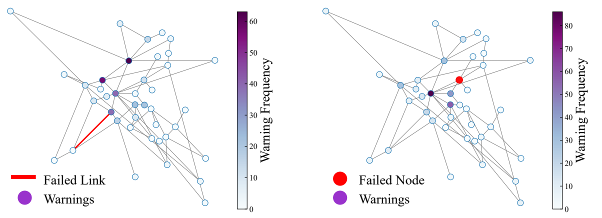
drifted inference. For inferences without a failed link, we expect the ratio of weights of the first and the second link to not exceed β; for inferences with a failed link, we expect the ratio of weights of the failed and the first innocent link to be beyond β. Figure 11 shows the CDFs of these two kinds of ratios of drifted inferences in single link failure scenarios. Our warning raising mechanism maintains good performance under the same β in different topologies.

6.8 Warning Locality

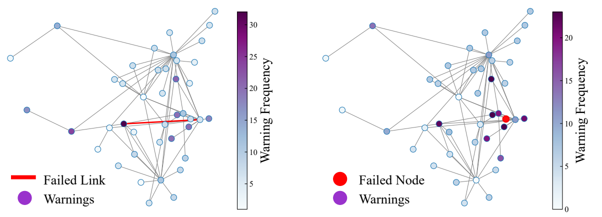
Here we focus on the perception scope of switches in Drift-Bottle. Figure 12 shows the warning frequency of each node while facing a failure. Generally, most of the warnings are raised by nodes in proximity to the failure unit, which verifies our motivation in 4.3. Notice the number of nodes which raises warnings about failed unit in Chinanet is larger than that in Geant2012. We attribute it to the highly star-like structure of topology of Chinanet.

6.9 Length of Inference

Figure 13 shows the performance of Drift-Bottle under different length of inference. As can be seen, the performance of Drift-Bottle improves significantly as the length of inference increases from 2 to 4. Extending the length beyond 4 only brings about very small (if any) improvement. It should be noted that implementing Drift-Bottle with long length of inference is resource-consuming. The P4 prototype of Drift-Bottle needs resubmits to complete inference aggregation with length of inference longer than 4. The selection of length of inference to 4 is a reasonable trade-off between performance and deployability.

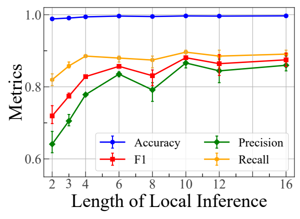


(a) Geant2012

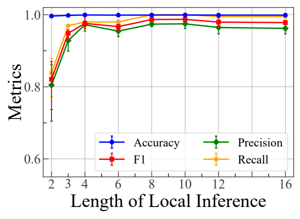


(b) Chinanet

Figure 12: Warning locality with respect to failures



(a) Geant2012



(b) Chinanet

Figure 13: Performance of Drift-Bottle under different length of inference

6.10 Resource Usage

Bandwidth. To represent and aggregate the drifted inferences (length of inference=4), we add a new header of 9B in each normal packet. Drift-Bottle only induces a negligible transmission amount of under 1% with respect to MTU of 1500B.

Switches. For switches equipped with P4 programs of Drift-Bottle, we evaluate their hardware resource usage and latency of packet processing. The P4 program of Drift-Bottle consumes 11 stages, 6.88% of SRAMs, 1.74% of TCAMs, 14.58% of meter ALUs and 13.54% of logical tables. While running our system, the average packet latency of switches increases from 732ns to 845ns under traffic of 100Gbps. The influence on throughput of switches is acceptable.

7 RELATED WORK

Failure localization in data center networks. Pingmesh from [9] implements a network measurement system to access real-time network latency between servers intra-pod, inter-pod, and inter-DC. NetBouncer [23] enables self-to-self proactive probing by bouncing probing packets received by top-level switches back to servers. 007 [2] uses a voting scheme to locate links that are guilty of packet retransmission. [19] utilizes a hypothesis test on TCP metrics to single out underperforming flows. NetPoirot [3] feeds various measures gathered from servers to a decision-tree-based model in order to discover the root cause of network anomaly.

Failure localization in general networks. In scenarios such as corporate networks, operators turn to host-based approaches for failure localization. [22] explores relativity between flow anomaly and link failures using belief networks, which is significantly time-consuming in large-scale networks. Network tomography [5, 6, 29] monitors multiple flows to access the packet loss rate of links via solving a path-link linear system, but fails to specify accurate failure location in the presence of high linear relativity between path vectors. To narrow down the scope of accused links, [18] introduces stochastic analysis after primary tomography, and MAX_COVERAGE [15] proposes a greedy solution to binary path-link algebra.

Host-based approaches are not suitable for scenarios like ISP networks, as the deployment of modules on end hosts may not be permitted by their users. Switch-based approaches deploy monitoring modules on switches instead of hosts. Blink [11] detects retransmission from a set of selected flows on data plane, but does not provide the location of potential failures. Prefix [27] utilizes random forest to predict hardware failures of switches by mining abnormal patterns in syslog sequences. SyNDB [13] generates summaries of packets entering switches and submits them to the database for further analysis. DynaFL [28] compares fingerprinting and forwarding records of packets between adjacent switches to detect malicious behavior. Everflow [31] mirrors abnormal packets to analysts and diagnoses the root cause of network failures by injecting an incarnation of the abnormal packet to its first hop. NetSight [10] reconstructs histories of packets by recording and collecting postcards of passing packets. NetSeer [30] traces each step of packet processing in the data plane to extract event packets, then it sends the headers and metadata of distinct event packets to the analyst for further analysis.

Programmable data plane. Traditional switches process packets with fixed logic and fall short to meet new requirements brought by emerging network technologies and protocols one after another. To enhance the flexibility of network management and the ability to support new protocols, OpenFlow [17] decouples the control plane and the data plane of switches from each other, and provides standardized interfaces for management and configuration. [4] introduces protocol-independent packet processors and the corresponding programming language P4, which enables operators to customize processing logic in the data plane. Some monitoring tools such as Dapper [8] utilize the programmable data plane to perform failure detection in fine granularity.

In-network intelligence. In-network intelligence technique aims to deploy machine learning models on switches with the limit of computing and storage resources. [20, 25] plant trained decision trees on the data plane of programmable switches by translating classification rules into match-action entries in flow tables, which enables switches to deal with multiple kinds of classification tasks with a line rate of packets processing. PUFF [26] utilizes in-network intelligence to localize intra-domain network failures by extracting flow-level features periodically and feeding them to a pre-trained machine learning module in switches.

8 CONCLUSION

We introduce Drift-Bottle, a lightweight and distributed approach to failure localization in general networks. Drift-Bottle utilizes the in-network intelligence technique to detect flow-level anomalies on switches, then generates concise inferences about potential failures with information of data paths. Instead of a centralized mechanism, we design a distributed mechanism for inferences aggregation which avoids additional infrastructural modification in networks. Switches with the deployment of Drift-Bottle work at a line rate, as we implement its function mainly on the data plane. The evaluation shows that Drift-Bottle meets the demand of network operators by providing fast, precise, and lightweight failure localization in general networks.

ACKNOWLEDGEMENT

We would like to thank our shepherd (Aurojit Panda) and the anonymous reviewers for their constructive comments. This work is supported by the National Key Research and Development Program of China under grant No. 2020YFB1804704, the National Natural Science Foundation of China under grant No. 61972189, the Major Key Project of PCL under grant No. PCL2021A03-1, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

REFERENCES

[1] V. Arrigoni, N. Bartolini, A. Massini, and F. Trombetti. 2021. Failure Localization through Progressive Network Tomography. In *IEEE INFOCOM*.

[2] B. Arzani, S. Ciraci, L. Chamon, Y. Zhu, H. Liu, J. Padhye, B. T. Loo, and G. Outhred. 2018. 007: Democratically Finding the Cause of Packet Drops. In *USENIX NSDI*.

[3] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred. 2016. Taking the Blame Game out of Data Centers Operations with NetPoirot. In *ACM SIGCOMM*.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. Jul. 2014. P4: Programming Protocol-independent Packet Processors. *ACM SIGCOMM Computer Communication Review (SIGCOMM-CCR)* 44, 3 (Jul. 2014), 87–95.

[5] Y. Chen, D. Bindel, and R. H. Katz. 2004. Tomography-based Overlay Network Monitoring. In *ACM IMC*.

[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz. 2004. An Algebraic Approach to Practical and Scalable Overlay Network Monitoring. In *ACM SIGCOMM*.

[7] J. Dean. 2009. Designs, Lessons and Advice from Building Large Distributed Systems. In *LADIS keynote*.

[8] M. Ghasemi, T. Benson, and J. Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *ACM SOSR*.

[9] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z. Lin, and V. Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *ACM SIGCOMM*.

[10] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *USENIX NSDI*.

[11] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. 2018. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *USENIX NSDI*.

[12] Intel. 2020. Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.

[13] P. G. Kannan, N. Budhdev, R. Joshi, and M. C. Chan. 2021. Debugging Transient Faults in Data Centers using Synchronized Network-wide Packet Histories. In *USENIX NSDI*.

[14] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan. Oct. 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications (JSAC)* 29, 9 (Oct. 2011), 1765–1775.

[15] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. 2007. Detection and Localization of Network Black Holes. In *IEEE INFOCOM*.

[16] B. Lantz, B. Heller, and N.McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *ACM HotNets*.

[17] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Apr. 2008. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review (SIGCOMM-CCR)* 38, 2 (Apr. 2008), 69–74.

[18] H. X. Nguyen and P. Thiran. 2007. The Boolean Solution to the Congested IP Link Location Problem: Theory and Practice. In *IEEE INFOCOM*.

[19] A. Roy, S. Diego, H. Zeng, J. Bagga, and A. C. Snoeren. 2017. Passive Realtime Datacenter Fault Detection and Localization. In *USENIX NSDI*.

[20] J.-H. Lee K. Singh. 2020. SwitchTree: In-network Computing and Traffic Analyses with Random Forests. *Neural Computing and Applications* (2020).

[21] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. 2004. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Transactions on Networking (ToN)* 12, 1 (2004), 2–16.

[22] M. Steinder and A. S. Sethi. Oct. 2004. Probabilistic Fault Localization in Communication Systems Using Belief Networks. *IEEE/ACM Transactions on Networking (ToN)* 12, 5 (Oct. 2004), 809–822.

[23] C. Tan, Z. Jin, C. Guo, T. Zhang, H. Wu, K. Deng, D. Bi, and D. Xiang. 2019. NetBouncer: Active Device and Link Failure Localization in Data Center Networks. In *USENIX NSDI*.

[24] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage. 2010. California Fault Lines: Understanding the Causes and Impact of Network Failures. In *ACM SIGCOMM*.

[25] Z. Xiong and N. Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *ACM HotNets*.

[26] L. Ye, Q. Li, X. Zuo, J. Xiao, Y. Jiang, Z. Qi, and C. Zhu. 2021. PUFF: A Passive and Universal Learning-based Framework for Intra-domain Failure Detection. In *IEEE IPCCC*.

[27] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang, Y. Chen, H. Dong, X. Qu, and L. Song. 2018. PreFix: Switch Failure Prediction in Datacenter Networks. In *ACM SIGMETRICS*.

[28] X. Zhang, C. Lan, and A. Perrig. 2012. Secure and Scalable Fault Localization under Dynamic Traffic Patterns. In *IEEE S&P*.

[29] Y. Zhao, Y. Chen, and D. Bindel. Dec. 2009. Towards Unbiased End-to-End Network Diagnosis. *IEEE/ACM Transactions on Networking (ToN)* 17, 6 (Dec. 2009), 1724–1737.

[30] Y. Zhou, C. Sun, H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, P. Zhang, D. Cai, M. Zhang, and M. Xu. 2020. Flow Event Telemetry on Programmable Data Plane. In *ACM SIGCOMM*.

[31] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, and H. Zheng. 2015. Packet-Level Telemetry in Large Datacenter Networks. In *ACM SIGCOMM*.

[32] M. Zukerman, T. D. Neame, and R. G. Addie. 2003. Internet traffic modeling and future technology implications. In *IEEE INFOCOM*.