PreFix: Switch Failure Prediction in Datacenter Networks

SHENGLIN ZHANG*, Tsinghua University & Nankai University, China
YING LIU, WEIBIN MENG, Tsinghua University, China
ZHILING LUO, Zhejiang University, China
JIAHAO BU, Tsinghua University, China
SEN YANG, Georgia Institute of Technology, USA
PEIXIAN LIANG, University of Norte Dame, USA
DAN PEI†, Tsinghua University, China
JUN XU, Georgia Institute of Technology, USA
YUZHI ZHANG, Nankai University, China
YU CHEN, HUI DONG, XIANPING QU, LEI SONG, Baidu, Inc, China

In modern datacenter networks (DCNs), failures of network devices are the norm rather than the exception, and many research efforts have focused on dealing with failures *after* they happen. In this paper, we take a different approach by *predicting* failures, thus the operators can intervene and "fix" the potential failures *before* they happen. Specifically, in our proposed system, named *PreFix*, we aim to determine during runtime whether a switch failure will happen in the near future. The prediction is based on the measurements of the current switch system status and historical switch hardware failure cases that have been carefully labelled by network operators. Our key observation is that failures of the same switch model share some common syslog patterns before failures occur, and we can apply machine learning methods to extract the common patterns for predicting switch failures. Our novel set of features (*message template sequence, frequency, seasonality and surge*) for machine learning can efficiently deal with the challenges of noises, sample imbalance, and computation overhead. We evaluated PreFix on a data set collected from 9397 switches (3 different switch models) deployed in more than 20 datacenters owned by a top global search engine in a 2-year period. PreFix achieved an average of 61.81% recall and 1.84×10^{-5} false positive ratio, outperforming the other failure prediction methods for computers and ISP devices.

*Both Nankai University and Tsinghua University contributed equally to this paper.
†Dan Pei is the correspondence author.

Authors' addresses: Shenglin Zhang, Tsinghua University & Nankai University, College of Software, Nankai University, Tianjin, China, zhangsl@nankai.edu.cn; Ying Liu, Weibin Meng, Tsinghua University, Institute of Network Sciences and Cyberspace, Beijing, China, liuying@cernet.edu.cn,m_weibin@163.com; Zhiling Luo, Zhejiang University, advanCed Computing aNd sysTem Laboratory (CCNT), Hangzhou, Zhejiang, China, luozhiling@zju.edu.cn; Jiahao Bu, Tsinghua University, Institute of Network Sciences and Cyberspace, Tsinghua University, Beijing, China, bujh1994@foxmail.com; Sen Yang, Georgia Institute of Technology, School of Electrical and Computer Engineering, Atlanta, Georgia, USA, sen.yang@gatech.edu; Peixian Liang, University of Norte Dame, Department of Computer Science and Engineering, College of Engineering, Indiana, USA, pliang@nd.edu; Dan Pei, Tsinghua University, Department of Computer Science and Technology, Beijing, China, peidan@tsinghua.edu.cn; Jun Xu, Georgia Institute of Technology, School of Computer Science, Atlanta, Georgia, USA, jx@cc.gatech.edu; Yuzhi Zhang, Nankai University, College of Software, Tianjin, China, zyz@nankai.edu.cn; Yu Chen, Hui Dong, Xianping Qu, Lei Song, Baidu, Inc, Beijing, China, {chenyu07,dionghui02,quxianping,song_lei}@baidu.com.

2476-1249/2018/3-ART2 $15.00
https://doi.org/10.1145/3179405

CCS Concepts: • **Networks → Network performance modeling**; • **Computing methodologies → Supervised learning by classification**;

Additional Key Words and Phrases: Failure prediction, Operations, Datacenter, Machine learning

ACM Reference Format:
Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, and Yu Chen, Hui Dong, Xianping Qu, Lei Song. 2018. PreFix: Switch Failure Prediction in Datacenter Networks. *Proc. ACM Meas. Anal. Comput. Syst.* 2, 1, Article 2 (March 2018), 29 pages. https://doi.org/10.1145/3179405

1 INTRODUCTION

A modern datacenter employs a large number of servers and network devices [52]. For example, tens of thousands of switches are deployed in Microsoft's datacenter networks to connect hundreds of thousands to millions of servers [22]. Switches, including top-of-rack (ToR) switches and aggregation switches, receive and aggregate traffic from servers and forward it to higher level routers, and hence play a fundamental role in the network. Despite switches' importance, switch failures are the norm rather than the exception in a modern data center [19, 22, 52]. Among all network device failures, switch failures are the dominant type in terms of both downtime (∼ 74%) and the number of occurrences (∼ 23%) [19]. Switch failures, if not dealt with promptly and gracefully (*i.e.*, with no or little service interruption during the replacement of a failed switch), can lead to service performance degradation or even outages [1–3].

Existing datacenter fault-tolerance solutions (to switch failures), such as [9, 34, 35, 57], focus on changing the protocols and network topologies such that the datacenter network can automatically failover. However, not all the switch failures can be dealt with using these approaches. For example, the ToR switches, which dominate other types of switches in number, typically do not have hot backups (see later in Figure 1). Hence when ToR switches fail, operators need to quickly diagnose and locate the failed switches and mitigate the effects of the failures. Indeed, several approaches were proposed to do so [23, 60, 61, 65, 66]. However, these approaches either face deployment challenges or take a nontrivial amount of time to locate and fix the failed switches while the application performance is being degraded. What makes the matter worse is that a switch that is about to fail can drop packets [22] silently (*i.e.*, without logging or reporting packet drops). Such silent failures are hard to diagnose but their negative impact on system performance is very noticeable to end users. As a result, today's datacenters are still suffering from the performance issues caused by switch failures.

1.1 Predicting Switches Using PreFix

In this work, we take a different approach: instead of dealing with failures *after* they happen as done in the previous works, we proactively *predict* switch failures so that the operators can intervene and "fix" the looming failures *before* they happen. For example, if a ToR switch failure is predicted to happen, operators can employ standard load balancing mechanisms to shift the network traffic from the servers in the corresponding rack to those servers (that provide the same services) in other racks, and then replace the failure-looming switch. Using such an approach, performance degradations are avoided, or "fixed" beforehand, which is why we name our proposed system "PreFix".

Our approach is based on machine learning: we aim to learn a failure prediction model from various monitoring logs. This approach is motivated by the following factors. First, there are typically a massive number of switch syslog messages in datacenters, which record events such as state changes of interfaces, configuration changes, powering down of devices, plugging in/out

of a line card, and operational maintenance. Second, past switch failure instances, likely due to the (considerable) damages they had caused, were often carefully recorded (labelled) by network operators. Third, our preliminary investigations suggest that failures of the switches of a certain type often exhibit some common pathological behavior patterns (omens) before they occurred (see an example later in Table 2). Fourth, the approach of mining system logs has been successfully applied to various "sister problems" such as predicting failures of ISP networks [25, 48, 49], of computers [15–17, 33, 47, 71], of virtual machines [58], and of online ad services [51].

In PreFix, we aim to determine *during runtime* (*i.e.*, online) whether a switch failure in datacenter networks will happen *in the near future*. To do so, we first train a machine learning model offline using the syslog messages associated with historical switch failure instances. We then apply the failure patterns (omens) learned through this offline process to the feature values extracted from the runtime syslogs for predicting future failures. Since hardware failures account for the vast majority of all switch failures in terms of downtime and the number of occurrences [19], we focus on *hardware failures* (referred to as *failures* hereafter) in this work. To the best of our knowledge, this is the first work on predicting switch failures in datacenters.

1.2 The Challenges of This Prediction Problem

PreFix faces several interesting challenges. We note that although most of these challenges are common in log-based failure predictions [15–17, 25, 33, 47–49, 51, 58, 71], they become even harder to overcome in switch failure prediction to the extent that existing failure prediction algorithms do not perform well. In the following we list two most significant challenges.

(1) **Noisy signals in syslog data.** Operational switches generate many different types of syslogs. Events recorded by syslogs can be operators' normal log in/out, a PING session to another network device or a server, interface up/down, configuration synchronization to a standby switch, DDoS attack alerts, line cards plugging in/out, *etc.* While these logs are rich in information, the signals are very noisy in the following sense. Syslogs rarely contain *explicit* failure omens (referred to as *omens* hereafter); rather, omens are scattered across many syslog messages of diverse types. Those approaches [15, 18] that rely on omen messages being consecutive in the message sequence cannot deal with such noises effectively. Rather, it is necessary to extract "clean" omen patterns from which most of noises (irrelevant syslog messages) are filtered out.

(2) **Sample imbalance.** Although the frequency of switch failures across a datacenter can be high, the failure frequency per switch is very low, especially for a single TOR switch [19]. Therefore, syslog messages in the vast majority of the monitoring intervals contain no omens. More specifically, in our dataset, the monitoring intervals that do not contain omens outnumber those that do by $72500:1$ (see later in Table 7). That is, the omen samples and non-omen samples are severely imbalanced, which poses a significant challenge to the usual objective of simultaneously achieving high precision (low false positive ratio) and high recall (low false negative ratio) for machine learning based failure prediction.

The offline learning procedure of PreFix works as follows. For each model of switches, PreFix first extracts message templates from historical syslog messages and convert this failure prediction problem into a template sequence classification problem. Then using four template features that were carefully selected based on domain knowledge, PreFix applies Random Forest (RF) to the template sequences converted from the syslog messages associated with the historical switch failure tickets to obtain the model. The online prediction procedure of PreFix works similarly: PreFix converts the real-time syslogs generated by a switch to syslog templates, extracts the values therein of these four template features, and uses the trained model to determine whether the switch is failure-looming or not.

The two aforementioned challenges are tackled by PreFix as follows.

(1) To learn the sequential feature, one of the four aforementioned template features, from syslogs, we proposed a novel, simple, yet effective *two fold longest common subsequence (LCS^2)* method. The method not only efficiently measures the similarity between a given syslog sequence and omen syslog sequences, but also effectively filters out noises in the omen syslog sequences.

(2) Different from previous approaches which rely solely on the sequence feature [15, 18, 48], we propose to extract three additional features from syslogs, *namely*, frequency, seasonality (quasi-periodic happening of a certain event) and surge (a sudden burst). These four features complement each other very well, resulting in relatively high precision and recall at the same time, as will be shown in Section 4.2. For example, while the combination of sequence and surge features allows PreFix to effectively catch (predict) most of the true failures, the false alarms can also be high. The combination of frequency and seasonality features, on the other hand, can be used to filter out frequent and/or quasi-periodic syslog messages (which are usually noises) respectively, and hence are very effective at suppressing false alarms. This filtering also effectively mitigates the aforementioned sample imbalance problem by removing most of the irrelevant samples that could otherwise be mistakenly considered omen samples. We will further elaborate the complementary nature of these four features in **Section 4.3**.

We have conducted extensive evaluation experiments using data collected from *9,397 switches* (that belong to 3 different switch models) deployed in *more than 20 datacenters* owned by a top-tier global search engine over a *2-year period*. Our results demonstrate the superior performance of PreFix, as measured by the combination of precision and recall, compared to that of spectrum kernel support vector machine (SKSVM) [15, 18] and hidden semi-Markov model (HSMM) [48], two machine-learning models that were used for predicting failures of computers and ISP devices, respectively. In particular, PreFix can achieve very low false alarm rates (*i.e.,* high recall rates) while maintaining relatively high precision rates, making it well suited for practical use in a large datacenter. For example, for a single switch, the mean time between two consecutive false alarms is 8,494 *days*, and for a large data center with 10,000 switches, on average 1.2 false alarms are generated per day. This false alarm rate is quite acceptable to operators according to our in-person surveys.

The rest of the paper is organized as follows. We provide an introduction to datacenter network architecture, switch failures, and switch syslogs in Section 2, and describe the design of PreFix in Section 3. The evaluation of PreFix is presented in Section 4, followed by related works in Section 5. Finally, we conclude our paper in Section 6.

2 BACKGROUND

In this section, we first introduce the architecture of datacenter networks, and highlight the important role switches play in datacenter networks (Section 2.1). We then describe switch failures and how they are labeled (Section 2.2). Finally, we describe switch syslogs (Section 2.3), which we use for predicting switch failures.

2.1 Datacenter Network Architecture

Today's datacenter networks are built with commodity Ethernet switches and routers [6, 21, 22]. Figure 1 shows a typical datacenter network architecture [19, 22, 43]. It is comprised of several layers. In the bottom layer, servers are mounted on racks and connected to a ToR switch via Ethernet NICs. Tens of ToRs in turn are connected to a primary aggregation switch and a backup one for redundancy purposes (L2). The two aggregation switches are then connected to access routers (L3), which further aggregate the traffic and are connected to core routers. A datacenter network is connected to other datacenter networks and the Internet via core routers. Several types of middleboxes, such as load balancers, VPNs, firewalls, and intrusion detection and prevention

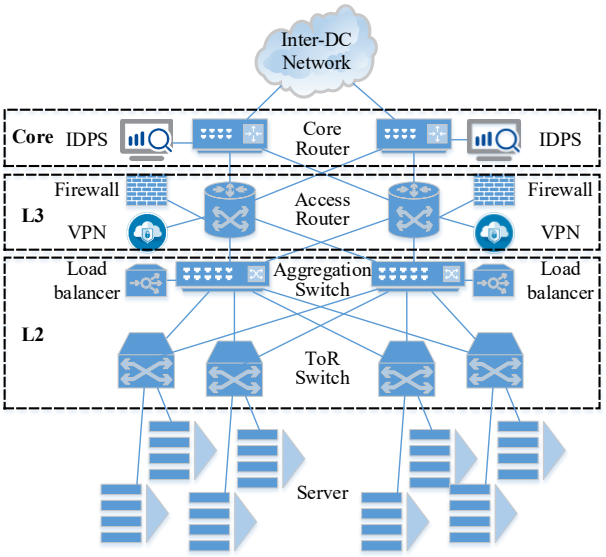


Fig. 1. Typical datacenter network architecture

systems (IDPSes), are usually deployed across the datacenter networks to improve performance and enhance security.

Switches, including ToRs and aggregation switches, outnumber all other network devices in a datacenter by a wide margin. With a rapid increase in traffic volumes in datacenter networks [52], the number of switches in a data center has also grown rapidly over the years. For example, Guo *et al.* reported that there were tens of thousands of switches in Microsoft's datacenters [22]. Similarly, the datacenter we studied also deploys tens of thousands of switches supplied by several different vendors.

2.2 Hardware Failure of Switches

When studying switch failures, it is important to first determine what types of logged events should be labeled as "failures". Here, we use the definition of switch failure in [19, 47]: a switch failure occurs when the service provided by a switch (namely traffic forwarding) deviates from the correct service behavior. Note this deviation should be observable (*e.g.*, by a human, a server, a router, or another switch) for the event to be considered a failure. Switch failures can be roughly classified into several different types, including (1) external problems such as power supply down; (2) configuration problems such as VPN tunneling errors; (3) hardware failures such as the crash, induced by hardware errors, of a line card or the entire switch; and (4) software crash due to bugs. In general, external problems and configuration problems are exogenously generated by operators or other devices and hence cannot be predicted using the measurement data of switches. In addition, switches can usually automatically recover from software crashes (typically via a reboot) [19] so predicting software crashes on switches is unnecessary in our context. Therefore, in this work we focus on *hardware failures*, which dominate all types of the switch failures in terms of the number of occurrences (~ 33% of all failures) and the total downtime (~ 72% of that of all failures) [19].

In our work, the switch (hardware) failures are labeled based on the following three types of observed raw data.

Table 1. Examples of switch syslog messages

Vendor	Time stamp	Switch ID	Message type	Detailed message
V1	Jan 23 14:24:41 2016	Switch 1	SIF	Interface te-1/1/8, changed state to up
V1	Mar 19 15:04:11 2016	Switch 4	OSPF	A single neighbour should be configured
V1	Apr 16 08:07:19 2016	Switch 4	lacp	Attempt to send lacpdu on port(38) from lag failed,Transport failed
V2	Apr 21 14:53:05 2016	Switch 11	10DEVM/2/POWER_FAILED	Power PowerSupply1 failed
V2	Sep 23 00:10:39 2015	Switch 13	10IFNET/3/LINK_UPDOWN	GigabitEthernet1/0/18 link status is DOWN
V2	Nov 8 07:29:06 2015	Switch 17	10CFM/5/CFM_SAVE CONFIG_SUCCESSFULLY	Configuration is saved successfully

(1) Failure tickets. When a service anomaly event is detected (*e.g.*, the average response time of the search engine deteriorates significantly for more than a minute or two) and a switch failure is suspected as the root cause (*e.g.*, the total amount of traffic to all servers connected to a switch sees a significant drop during the same time period), a failure ticket [64] is generated (by the data center performance monitoring system) and sent to the *network operators*. The *network operators* will then look into this ticket and determine whether the service deterioration was indeed caused by the suspected switch failure, and record this event in a *failure ticket*.
(2) Proactive switch failure detection via SNMP polling. In the tier-1 search engine we are working with, the real-time status of a switch, such as its CPU and memory utilizations and the traffic rate at each interface, is constantly monitored by the *network operators* via SNMP polling. If a problem in packet forwarding occurs at one or more interfaces, network operators will research the root cause of the failure and record this event accordingly.
(3) Proactive switch failure detection via syslogs. The operational instructions that vendors provide for their switch products usually include a list of syslog keywords that may indicate a switch failure. Regular expression queries are executed regularly to match the syslog messages against these failure-indicating keywords, and network operators are alerted if there is a hit. When alerted, network operators will verify whether the corresponding event is indeed a switch failure, and if so, identify the root cause of the failure. However, such regular expression matching can only detect a small portion of switch failures (*i.e.*, high false negative rates) in practice, and can also have false positives [19].

Since every switch failure event (of one of the above three types) was *manually verified* by the network operators, this failure data set can serve as the ground truth for our evaluations.

2.3 Switch Syslogs

Switch syslogs record hardware and software conditions observed by switches. They can be state changes of interfaces, links, or neighbors (*e.g.,* the state of an interface changes to "down"); operational maintenance (*e.g.,* operators log "in/out"); environmental condition alerts (*e.g.,* high temperature); *etc.* Although syslog messages are designed mainly for debugging and tracking software and hardware problems, they can also be used in root cause analysis for network incidents. Hence network operators usually deploy dedicated servers for collecting syslogs from all switches across datacenter networks.

Table 1 shows several examples of switch syslog messages. As the table shows, a syslog message usually has a primitive structure containing several fields, including a timestamp field recording

Table 2. An example of syslog sequence before a switch failure

Message ID	Time stamp	Message type	Detailed Message
D1	15:41:27	SIF	Interface ae0, changed state to down
D2	15:44:30	SIF	Interface ae3, changed state to down
D3	15:45:51	SIF	Vlan-interface vlan22, changed state to down
D4	15:46:59	SIF	Interface ae3, changed state to up
D5	15:47:21	SIF	Vlan-interface vlan22, changed state to up
D6	15:48:30	OSPF	Neighbour(rid:10.231.0.42, addr:10.231.38.85) on vlan22, changed state from Full to Down
D7	15:49:35	SIF	Interface ae2, changed state to down
D8	15:49:45	SIF	Vlan-interface vlan18, changed state to down
D9	15:50:42	SIF	Interface ae2, changed state to up
D10	15:50:59	SIF	Vlan-interface vlan18, changed state to up
D11	15:51:22	OSPF	A single neighbour should be configured
D12	15:51:52	OSPF	A single neighbour should be configured
D13	15:52:46	SIF	Interface ae1, changed state to down
D14	15:53:24	SIF	Vlan-interface vlan20, changed state to down
D15	15:54:31	OSPF	Neighbour(rid:10.231.0.40, addr:10.231.36.85) on vlan20, changed state from Full to Down
D16	15:55:12	SIF	Interface ae1, changed state to up
D17	15:56:47	SIF	Vlan-interface vlan20, changed state to up
D18	15:59:01	OSPF	A single neighbour should be configured

when the syslog message was generated, a switch ID field identifying the switch that generated the message, a message type field providing the rough characteristics of the message, and a detailed message field describing the details of the event. In general, switch failures of *the same switch model* share many common omen patterns, but switch failures of *different switch models* share few common omen patterns. That is because the syntax and semantics of the *message type* field and the *detailed message* field often vary significantly from one switch vendor/model to another. As with other failure prediction work [8], the failure prediction model for each switch model needs to be individually retrained using the failure data of the switch model. Therefore, for each switch model, we predict failures of switches of this model based on historical failures and syslogs *of this model.*

3 DESIGN OF PREFIX

The objective of PreFix is, for every switch in a datacenter, to predict during runtime whether there will be a hardware failure in the near future. The key insight behind the design of PreFix is that failures of the switches of a certain model share some common syslog patterns that can be extracted for predicting such failures. In this section, we first model the problem of syslog-based switch failure prediction in Section 3.1, and then provide an overview of PreFix's architecture in Section 3.2. They are followed by the details of the components of PreFix, including template learning and matching (Section 3.3), feature extraction (Section 3.4), and training and prediction (Section 3.5).

3.1 Model of Syslog-based Switch Failure Prediction

In the following, we first formulate the problem of switch failure prediction in Section 3.1.1. Then we describe the intuition behind, and the model used for, the syslog-based failure prediction problem in Section 3.1.2.

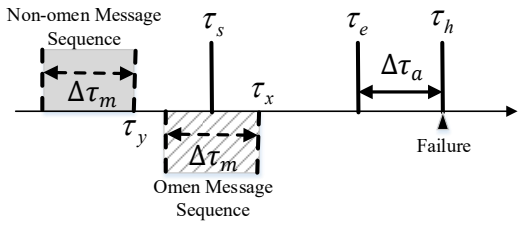


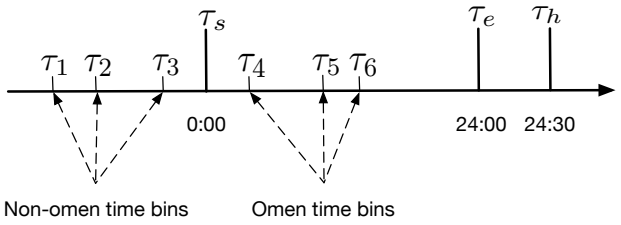
Fig. 2. The model of switch failure prediction. For a given switch failure that occurred at τ_h, our objective is to predict the failure during $[\tau_s, \tau_e]$. τ_e is $\Delta\tau_a$ before τ_h because network operators need no more than $\Delta\tau_a$ time to react to a positive failure prediction. In the offline learning procedure, given the failure at τ_h, for any τ_x in $[\tau_s, \tau_e]$, the syslog message sequence in $[\tau_x - \Delta\tau_m, \tau_x]$ is labeled as an omen message sequence, while the syslog message sequence in $[\tau_y - \Delta\tau_m, \tau_y]$ is labeled as a non-omen message sequence when $\tau_y \notin [\tau_s, \tau_h]$.

3.1.1 Problem Formulation of Switch Failure Prediction. We first discretize time into (relatively) short equal-length time bins with a length ξ (say $\xi = 15\ min$), and refer to each time bin by its starting time. Figure 2 shows the objective of switch failure prediction. Suppose that a failure occurs at τ_h. Our objective is to make the positive failure prediction at any time bin τ_x within $[\tau_s, \tau_e]$ (say $\tau_e - \tau_s = 24\ h$), where τ_e is $\Delta\tau_a$ (say $\Delta\tau_a = 30\ min$) ahead of τ_h, as network operators need no more than $\Delta\tau_a$ time to react to a positive failure prediction (*e.g.*, by shifting the traffic and replacing the failure-looming switch). We emphasize that a positive failure prediction implies the following situational assessment: a switch failure is imminent and the switch needs to be replaced right away. Hence if this "imminent failure" prediction is made too early, say at a time τ_x that is much earlier than τ_s, the prediction will be considered incorrect (*i.e.,* a false alarm). We emphasize that this "imminent failure" prediction is very different from being able to estimate the exact time of failure τ_h very early on, which we believe is impossible.

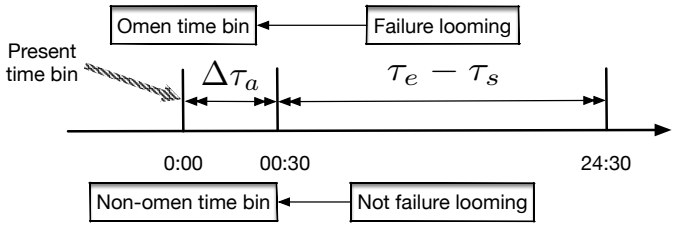
3.1.2 Syslog-Based Switch Failure Prediction. After investigating dozens of switch failure cases, we observe that the syslogs before switch failures often have similar patterns. For example, Table 2 shows a series of syslog messages half an hour before a switch failure. From the table we can see that the interfaces or the vlan-interfaces of the switch became unstable and they switched on and off frequently in a short period (*i.e.*, a sudden burst of on-off oscillations). We observe that similar sudden bursts occurred before 30+ other switch failures. Furthermore, the syslog sequences before these failures are all quite similar to the one shown in Table 2. Hence, intuitively if we can learn this similarity, we can predict switch failures before they occur and reduce the loss caused by switch failures.

As shown in Figure 2 and explained earlier, the failure prediction at any time (bin) τ_x is made based on the assessment of the syslog messages within $[\tau_x - \Delta\tau_m, \tau_x]$ (say $\Delta\tau_m = 2\ h$). Hereafter, we refer to the syslog message sequence within $[\tau_x - \Delta\tau_m, \tau_x]$ as τ_x's corresponding message sequence. Note that, if the message sequence contains too few messages (say containing less than θ messages), it is virtually impossible to extract either the omen pattern or the non-omen pattern from it; In PreFix, we set this threshold θ based on the domain knowledge of operators. Note that all message sequences discussed hereafter each has at least θ messages in it.

In the offline learning procedure, a message sequence is considered an omen message sequence (*i.e.*, an omen to a switch failure) if the prediction time τ_x is within the desired prediction time window $[\tau_s, \tau_e]$, and a non-omen message sequence otherwise. We label this time bin τ_x an omen or non-omen time bin accordingly. Note that an omen time bin thus defined always occurs before, and not during, a failure. We use the labels, and time bin's corresponding message sequences



(a) Offline learning procedure: τ_4, τ_5 and τ_6 during $[\tau_s, \tau_e]$ are labeled as omen time bins, and τ_1, τ_2 and τ_3 beyond $[\tau_s, \tau_e]$ are labeled as non-omen time bins.



(b) Online prediction procedure: at present time bin $(0:00)$, if we predict that there will occur a switch failure during $[00:30, 24:30]$, we will classify the present time bin as an omen one; otherwise, we will classify the present time bin as a non-omen one.

Fig. 3. A toy example of the switch failure prediction problem, in the offline learning procedure and in the online prediction procedure.

to train PreFix. For example, as shown in Figure 3 (a), suppose that $\tau_s = 0:00$, $\tau_e = 24:00$, $\tau_h = 24:30$. Then τ_4, τ_5 and τ_6 (inside $[\tau_s, \tau_e]$) are omen time bins, and τ_1, τ_2 and τ_3 (outside $[\tau_s, \tau_e]$) are non-omen time bins.

In the online prediction procedure, the failure prediction problem is converted into a time bin classification problem: to classify a time bin as either an omen time bin or a non-omen time bin based on its corresponding message sequence. Figure 3 (b) shows a toy example of the switch failure prediction problem in the online prediction procedure. Suppose that $\Delta\tau_a = 30\ min, \tau_e - \tau_s = 24\ h$. At present time bin $(0:00)$, if we predict that there will occur a switch failure during $[00:30, 24:30]$ based on the present time bin's corresponding message sequence, we classify the present time bin as an omen time bin; otherwise, we classify it as a non-omen time bin.

3.2 PreFix Framework

Figure 4 shows the architectural framework of PreFix. PreFix is composed of an offline learning component and an online prediction component. The offline learning component automatically learns the syntax/semantics of syslog messages, extracts omen patterns, and trains the model using historical switch data. The online prediction component predicts whether the current syslog sequence (*i.e.*, that in the immediate past) indicates an omen (for a switch failure) according to the trained model.

In the offline learning component, the first step is to learn message templates from historical syslog messages automatically. This step is accomplished in PreFix using an accurate and incrementally retrainable approach developed earlier in [70]. Armed with the message templates thus learned, we then match each historical syslog message to a specific message template. This way, a syslog

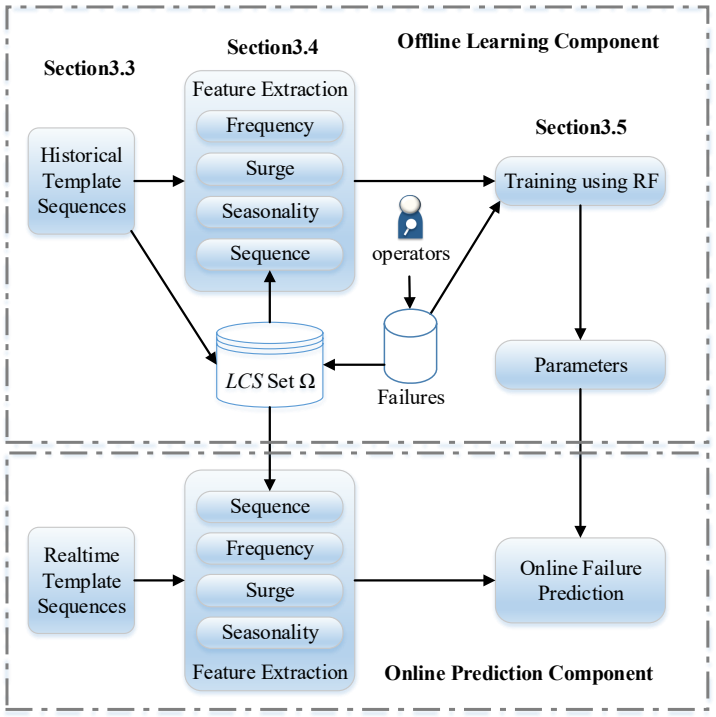


Fig. 4. The framework of PreFix

message sequence is converted to a template sequence. From this template sequence we then extract the values of the aforementioned four features (namely sequence, surge, seasonality, and frequency) that to the best of our knowledge can best tell apart omen message sequences and non-omen ones. Finally, we train PreFix to learn omen patterns using the template sequences of historical switch failures. This training is performed regularly (*e.g.*, daily) to keep the model up to date.

In the online prediction component, PreFix first matches real-time syslog messages to message templates, then extracts the values of the aforementioned four features from the template sequence, and predicts whether the realtime syslog messages are indicative of a failure based on the omen patterns learned in the offline learning component.

3.3 Template Learning and Matching

As mentioned in Section 2.3, switch syslog messages have various formats and the detailed messages are unstructured. Although there is a message type field (see Table 1) that describes the schematic characteristics in each syslog message, each message type can include quite different syslog messages. For example, although there are 13 syslog messages in Table 2 that belong to the message type "SIF" (system interconnect fabric) and describe the switch status changes collected using SIF technology, the detailed messages are quite different.

When we mask the variables of the detailed message field, *i.e.,* interface number or vlan-interface number, using the same symbol, *e.g.*, an asterisk as shown in Table 3, there are only four different syslog message structures, or *subtypes*. However, manually obtaining all *subtypes* without domain knowledge is almost infeasible because not every part that should be masked in the detailed message is obviously characterized such as the interface number [44]. In addition, although *part* of the

Table 3. Syslog message subtypes of SIF

Subtype No.	Subtype (template) structure	Corresponding Message ID in Table 2
$N1$	Interface *, changed state to down	D1, D2, D7, D13
$N2$	Interface *, changed state to up	D4, D9, D16
$N3$	Vlan-interface *, changed state to down	D3, D8, D14
$N4$	Vlan-interface *, changed state to up	D5, D10, D17

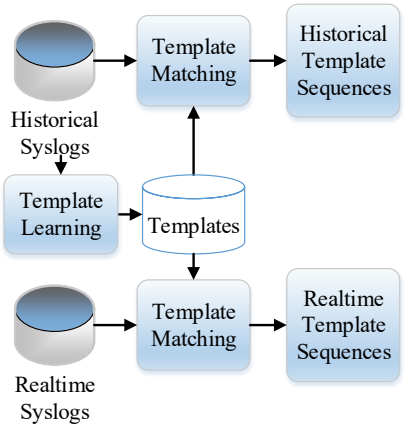


Fig. 5. PreFix learns message templates from historical syslog messages, after which historical syslog messages and realtime ones are matched to historical template sequences and realtime ones, respectively

syslog *subtypes* can be obtained from vendor support, these *subtypes* may *change* owing to software upgrade. Therefore, our objective here is to automatically obtain *message templates* in which the need-masked parts are removed and the message *subtypes* are retained without relying on any domain knowledge.

Syslog parsing techniques for *network devices including routers and switches* have been well studied, and extracting message templates from syslog messages is a common practice [27, 28, 44, 70]. For this purpose, PreFix adopts FT-tree, an accurate and incrementally retrainable template extraction method proposed by Zhang *et al.* [44]. We omit the details of FT-tree here, which can be found in [44], since they are not necessary for understanding PreFix. Through this template extraction (using FT-tree), we can match a historical syslog message or realtime one to a specific message template. Given a syslog message sequence $(s_1, s_2, ..., s_n)$, for each syslog message s_i, we denote $t_{(s_i)}$ as the message template that s_i matches to and call $(t_{(s_1)}, t_{(s_2)}, ..., t_{(s_n)})$ the template sequence of $(s_1, s_2, ..., s_n)$. As mentioned in Section 3.1, the failure prediction problem is to classify a time bin into either an omen time bin or a non-omen time bin based on its corresponding message (template) sequence.

3.4 Feature Extraction

After converting syslog messages to templates, we then try to capture the features of each template sequence that can best tell apart omen message sequences and non-omen ones. The most straightforward method of constructing the feature vector is the bag-of-words algorithm used in natural language processing and information retrieval [53]. Similar to the keyword based method, in this algorithm the occurrence or the frequency of each word in a message template constitutes

a feature vector. However, some keywords that may sound "suspicious", such as "loss", "down", and "failed", are not always indicative of a failure. For instance, if a switch tries to PING another network device, the *loss ratio* will be recorded in the syslog message for this PING session. Since this type of syslog messages and hence their corresponding message templates occur frequently under normal operations, they are not indicative of failures. Therefore in PreFix, we focus not on *words in a single message template*, but on the patterns of template sequences.

After a careful examination of syslog messages leading up to all switch failures, we identify the following four features that we believe can best tell apart omen message sequences and non-omen ones.

- **Sequence.** Sequence is apparently an essential feature, as we found that syslog messages leading up to switch failures often follow similar sequential patterns. For example, in our dataset, we found syslog messages with similar sequential patterns as the one shown in Table 2 preceded more than 30 other switch failures.
- **Frequency.** Frequency is an essential feature in most of machine learning applications, including this one: some types of syslog messages, such as a switch pinging other network devices, occur frequently under normal operations and hence are generally "innocuous" (not indicative of looming switch failures).
- **Surge.** Surge is an essential feature in this application because the occasional occurrences of syslog messages of a certain type are usually not indicative of switch failures, but a sudden burst of such messages usually are. For example, an occasional interface flap (up/down) may "heal by itself" because of switch's failover mechanism, but a sudden burst of them likely indicates an imminent failure of this switch.
- **Seasonality.** It appears necessary to include seasonality as a feature for the following reason. Usually, syslog messages (of a certain type) that occur periodically or quasi-periodically are "innocuous". An example of such quasi-periodic syslog messages is those corresponding to operators logging into switches for routine inspections.

As explained above, intuitively each of these four features appears necessary for the prediction problem. The necessity of each feature is further confirmed by the node impurity tests that we will describe in Section 4.4. Empirically the combination of these four features appear sufficient to capture the failure omen patterns for all the switch failures (415 of them in total) in our data set, as we will elaborate in Section 4.3.

This said, there is really no way for us to tell whether we have missed any informational (but very subtle) feature that may improve the (failure) prediction accuracy significantly. However, since the raw dataset we use is now publicly available [5], no knowledge or insights will be permanently lost. In the future other researchers can experiment with any new feature(s) on this dataset and hopefully discover some useful ones.

In the following, we first describe how to extract the aforementioned four features, namely frequency (Section 3.4.1), seasonality (Section 3.4.2), sequence(Section 3.4.3), and surge (Section 3.4.4), from each template sequence. Then we explain in Sections 3.4.5 and 3.4.6 how these four features are aggregated and justified, respectively .

3.4.1 Frequency Extraction. Some *subtypes* of syslog messages, and the corresponding message templates, occur frequently throughout the deployment of a switch. Examples of such syslog messages include those recording the packet loss rate of PING sessions, and those denoting that the switch successfully sends control packets to another device. Such messages are usually considered normal, even if they occur frequently before failures.

For each time bin, a vector is used to record such frequency information: each scalar in the vector corresponds to a distinct template and records the number of occurrences (*i.e.*, the frequency) of this template. For example, suppose that at a time bin τ_k, $\mathbf{s}^k = (s_1^k, s_2^k, ..., s_n^k)$ are τ_k's corresponding message sequence (see Section 3.1.2 for the definition), and s_i^k is mapped to template $t_{(s_i^k)}$. Then $\mathbf{t}^k = (t_{(s_1^k)}, t_{(s_2^k)}, ..., t_{(s_n^k)})$ is the template sequence during time interval $[\tau_k - \Delta\tau_m, \tau_k]$. For each message template $t_j, j \in \{1, 2, ..., N\}$, where N is the number of different templates ($N \approx 240$ in our syslog data set), we denote $C_\Delta^k(t_j)$ as its number of occurrences during $[\tau_k - \Delta\tau_m, \tau_k]$. This way, for each template t_j, we obtain its frequency vector $\mathbf{C}_\Delta(t_j) = (C_\Delta^1(t_j), C_\Delta^2(t_j), ..., C_\Delta^M(t_j))$ of time bins $(\tau_1, \tau_2, ..., \tau_M)$.

3.4.2 Seasonality Extraction. As mentioned earlier, some subtypes of syslog messages occur quasi-periodically. That is, the same subtype (template) of syslog messages occur at a similar time every hour/day/week/year. Let H_j be the time interval during which syslog messages of template (subtype) t_j were observed. For example, suppose the syslog messages of subtype $t_{j'}$ are produced only by a certain version of switch firmware that was in use for six months; then $H_{j'}$ is precisely this six-month period. Then for t_j we obtain a time series $\mathbf{C}_\xi(t_j) = (C_\xi^1(t_j), C_\xi^2(t_j), ..., C_\xi^h(t_j))$, where $C_\xi^k(t_j)$ is the number of occurrences of t_j in the corresponding message sequence of the kth time bin, and h is the number of time bins in H_j. Clearly, if t_j is seasonal, then $\mathbf{C}_\xi(t_j)$ is a periodic time series. Hence our objective becomes determining whether $\mathbf{C}_\xi(t_j)$ is seasonal/periodic.

Determining whether a time series is seasonal has been well studied. There are two primary types of seasonality determination methods: autocorrelation based time domain methods and discrete Fourier transform (DFT) based frequency domain methods [42]. Since DFT based methods perform well only for short and medium periodicity (say an hour or a day), and the accuracy deteriorates for large periodicity (say a week or a month) [56], they are not appropriate in our scenario because the periodicity of some message templates is on the order of a week or even a month. Instead, we use YIN, a simple yet efficient autocorrelation-based method that has significantly improved upon the accuracy of the original autocorrelation method [12]. In YIN, a difference function is calculated as follows,

$$D'(\rho, t_j) = \frac{D(\rho, t_j)}{\frac{1}{\rho}\sum_{k=1}^{\rho} D(k, t_j)} \tag{1}$$

where

$$D(\rho, t_j) = \sum_{k=1}^{h-\rho} (C_\xi^k(t_j) - C_\xi^{k+\rho}(t_j))^2 \tag{2}$$

and $\rho \in \{hour, day, week, month\}$ is the set of candidate periodicity values obtained using domain knowledge. Since a smaller value of $D'(\rho, t_j)$ implies stronger seasonality, for message template $t_j (j \in \{1, 2, ..., N\})$, we use $\alpha_j = \min_\rho D'(\rho, t_j)$ as the final seasonality value. Please note that α_j is calculated based on H_j, the time interval during which syslog messages of template (subtype) t_j were observed. Therefore, for a given template t_j, there is only one α_j value in a training cycle.

We use YIN rather than autocorrelation for determining whether a syslog template sequence is periodic or quasi-periodic, also for the following reason. Even when there is periodicity in the occurrences of syslog messages of a certain type, this periodicity is never strict (i.e., is noisy). For example, suppose an operator logs into a specific switch for routine inspection between 9:00 am and 10:00 am daily. We consider the syslog messages corresponding to this daily routine periodic, but this periodicity is clearly noisy (unless the operator logs in at exactly the same time every day, sat at 9:01am). YIN can better handle the noisy nature of such periodicity than autocorrelation, by avoiding zero-lag bias through normalization [12].

Table 4. Omen template sequences before three failures

Sequence No.	template sequence
T^1	**48 48 49** 46 47 63 48 **49** 46 47 62 62 48 49 **63 46 47 62**
T^2	**48 48 49 49 63** 63 **46** 46 **47** 47 **62** 62 56 56 57 57 58 58 59 59
T^3	50 62 **48** 49 46 47 62 **48 49** 63 46 47 62 56 57 58 59 48 **49 63 46 47 62** 48 49 46 47 48 49 63 51 46 47 50 62 62

3.4.3 Sequence Extraction. Table 4 lists the omen template sequences of three different switch failures, and the first one is the template sequence corresponding to the syslog sequence shown in Table 2. We can see that these omen template sequences share common subsequences (bold numbers in Table 4), and extracting this subsequence (feature) can clearly help with failure prediction. However, irrelevant templates, or *noise templates* (non-bold numbers in Table 4), also exist abundantly in omen template sequences. These *noise templates*, if not mostly filtered out, will interfere with the failure prediction.

Extracting the sequential feature for classification has been used in a broad range of applications such as genomic analysis [13], workflow resource management [37], health informatics [59], anomaly detection [30], information retrieval [50], and query log behavior determination [55]. In our problem, however, there are two challenges in extracting the sequential feature. First, classification methods such as SVM, logistic regression (LR) and RF usually take a vector of features as the input data, but there are hardly any *explicit* features in message template sequences. Second, even if we can transform a message template sequence into a set of features, this set will be massive, making it prohibitively computationally expensive to select a right (small) subset for the classification [63].

We have developed a novel method called *two fold longest common subsequence (LCS)*, or LCS^2 in short, to extract effective sequential features efficiently from message template sequences. LCS^2 can effectively filter out *noise templates* in omen template sequences, thereby allowing for more accurate predictions of switch failures. LCS [14] is the classic algorithmic problem of finding the longest common subsequence among a set of sequences, whose computation is extremely fast and can be parallelized [7, 10]. LCS^2 is a two-fold LCS: the first fold is the LCS of two omen template sequences, and the second fold is the LCS of the LCS resulting from the first fold and a given syslog template sequence. The intuition behind LCS^2 is that if two omen template sequences share some common sequential pattern, the LCS of them not only inherits this common pattern, but also filters out *noise templates* in both template sequences. Hence, comparing a given template sequence to the LCS of two omen template sequences, instead of an omen template sequence, can better determine whether this given template sequence is omen (indicative of an imminent switch failure) or not.

Here we use a toy example to illustrate how LCS^2 works. Suppose that for a certain switch model, three switch failures occurred in history. The three message template sequences shown in Table 4, namely T^1, T^2, and T^3, are the template sequences of these three failures, respectively. To extract common patterns and filter out noise templates (non-bold numbers in Table 4), we calculate the *LCS* of T^1 and T^2 (LCS_{12}), that of T^1 and T^3 (LCS_{13}), and that of T^2 and T^3 (LCS_{23}), respectively, as shown in the second column of Table 5. This way, we can see that most of noise templates have been filtered out. LCS_{12}, LCS_{13} and LCS_{23} constitute the *LCS* set Ω. Here, we get the *first fold LCS*.

For a given template sequence (48 49 46 66 63 48 80 49 46 47 62 62 48 49 63 46 47) at time bin τ^k (*i.e.,* $\mathbf{t}^k$), we calculate its LCS with LCS_{12}, LCS_{13}, and LCS_{23}, respectively. These three second-fold LCSes are denoted as $LCS(LCS_{12}, \mathbf{t}^k)$, $LCS(LCS_{13}, \mathbf{t}^k)$, and $LCS(LCS_{23}, \mathbf{t}^k)$, respectively, and are shown in the fourth column of Table 5.

Table 5. $LCS_{i\lambda}$ for the omen template sequences in Table 4, $LCS(LCS_{i\lambda}, \mathbf{t}^k)$, $\mathbf{t}^k)$), and $R^k_{i\lambda}$. The given template sequence is (48 49 46 66 63 48 80 49 46 47 62 62 48 49 63 46 47), and $R^k_{max} = 0.875$

$i\lambda$	$LCS_{i\lambda}$	$LCS(LCS_{i\lambda}, \mathbf{t}^k)$	$R^k_{i\lambda}$
12	48 48 49 49 63 46 47 62	48 48 49 49 63 46 47	0.875
13	48 49 46 47 63 48 49 46 47 62 48 49 63 46 47 62	48 49 46 63 48 49 46 47 62 48 49 63 46 47	0.875
23	48 48 49 49 63 46 46 47 47 62 62	48 48 49 49 63 46 47	0.636

Then, we calculate the ratio (denoted as R^k_{12}) of the length of $LCS(LCS_{12}, \mathbf{t}^k)$ to that of LCS_{12}, the ratio (denoted as R^k_{13}) of the length of $LCS(LCS_{13}, \mathbf{t}^k)$ to that of LCS_{13}, and the ratio (denoted as R^k_{23}) of the length of $LCS(LCS_{23}, \mathbf{t}^k)$ to that of LCS_{23}, respectively. These three ratios are shown in the sixth column of Table 5. In general, the maximum among these ratios (equal to 0.875 here) is defined as the sequence feature score of the given template sequence (is (48 49 46 66 63 48 80 49 46 47 62 62 48 49 63 46 47) here). This value measures how closely the given template sequence resembles those of omen template sequences.

Now we describe how LCS^2 works in general. For a given *omen time bin* τ_i, suppose that $\mathbf{s}^i = (s^i_1, s^i_2, ..., s^i_n)$ is the syslog message sequence within $[\tau_i - \Delta\tau_m, \tau_i]$, and $\mathbf{t}^i = (t_{(s^i_1)}, t_{(s^i_2)}, ..., t_{(s^i_n)})$ is the template sequence of $\mathbf{s}^i$. For any $\mathbf{t}^i, i \in \{1, 2, ..., L\}$, where L is the number of *omen time bins*, we calculate the $LCS_{i\lambda}$ of $\mathbf{t}^i$ and $\mathbf{t}^\lambda$, where $\lambda \in \{1, ..., L\} - i$. If the length of $LCS_{i\lambda}$ is greater than β (β is set to 8 based on domain knowledge), we add $LCS_{i\lambda}$ to the LCS set Ω. If $\forall\lambda \in \{1, ..., L\} - i$, the length of $LCS_{i\lambda}$ is smaller than β, we will add $\mathbf{t}^i$ to Ω, in case that the sequential feature of $\mathbf{t}^i$ is different from those of other template message sequences. Now we obtain the *first fold LCSes*, which are precisely the *LCSes* of any *two* omen template sequences as in the above toy example.

If two omen template sequences $\mathbf{t}^i$ and $\mathbf{t}^\lambda$, share a common sequence pattern, then $LCS_{i\lambda}$ will represent the common sequence pattern. In this case, we call $LCS_{i\lambda}$ an *omen LCS*. However, if $\mathbf{t}^i$ and $\mathbf{t}^\lambda$ have different sequence patterns, then $LCS_{i\lambda}$ will represent a *noise template*. In this case, we call this $LCS_{i\lambda}$ a *noise LCS*. Because it is almost impossible to distinguish an *omen LCS* from a *noise LCS*, we have to include both in the LCS set Ω, and instead tackle the problem caused by *noise LCS* using frequency and seasonality features.

For a given template sequence, compared to the similarity (measured by LCS) between it and the original omen template sequences, the similarity between it and the *omen LCSes* is much less susceptible to the noises contained in the omen template sequences for the following reason. In the latter case, for a noise template sequence to be mistaken for omen, it would have to be present in all three sequences: the given template sequence and the two omen template sequences whose LCS contains it. In contrast, in the former case, this noise template sequence would only need to be present in two sequences: the given template sequence and an omen LCS that contains it. The probability for the latter to happen is much smaller than that for the former to happen, even when this noise template sequence appears very frequently (*e.g.,* with a high probability of 0.1 in each sequence). This is the main motivation behind our two fold design, which we now specify precisely.

For a given time bin τ_k, suppose that the template sequence within $[\tau_k - \Delta\tau_m, \tau_k]$ is $\mathbf{t}^k = (t_{(s^k_1)}, t_{(s^k_2)}, ..., t_{(s^k_n)})$. As explained earlier in the toy example, we calculate the sequence feature score $R^k_{\max}$ as

$$R^k_{\max} = \max_x R^k_{i\lambda} \tag{3}$$

where

$$R_{i\lambda}^{k} = \frac{Len(LCS(LCS_{i\lambda}, \tau^{k}))}{Len(LCS_{i\lambda})} \tag{4}$$

where $LCS_{i\lambda} \in \Omega$, $LCS(LCS_{i\lambda}, \tau^k)$ is the LCS of $LCS_{i\lambda}$ and $\mathbf{t}^k$ (*i.e.,* τ_k's corresponding template sequence), and $Len(\cdot)$ is the length of a template sequence. We call $LCS_{i\lambda}$ the corresponding *LCS* of $R_{i\lambda}^{k}$. For instance, in the above examples shown in Table 5, $R_{\max}^{k} = 0.875$, and the corresponding *LCS* of $R_{\max}^{k}$ is LCS_{12} or LCS_{13}. Now, we get the *second fold LCS*.

We use the ratio of the length of $LCS(LCS_{i\lambda}, \tau^k)$ to that of $LCS_{i\lambda}$, rather than the ratio of the length of $LCS(LCS_{i\lambda}, \tau^k)$ to that of $\mathbf{t}^k$, because if $LCS_{i\lambda}$ is an *omen LCS*, in general there are less *noise templates* in $LCS_{i\lambda}$ than in $\mathbf{t}^k$, and the former ratio is more accurate in measuring how "ominous" $\mathbf{t}^k$ is. We use the maximum $R_{i\lambda}^{k}$ value rather than the mean or the median as the sequence feature score, because the omen pattern that a given template sequence matches to may appear in only one omen LCS, and in this case, only the maximum $R_{i\lambda}^{k}$ value fully reflects this matching. Finally, we obtain the sequence feature vector $\mathbf{R} = (R_{\max}^{1}, R_{\max}^{2}, ..., R_{\max}^{M})$ for time bins $(\tau_1, \tau_2, ..., \tau_M)$.

If LCS_x is $R_{\max}^{k}$'s corresponding *LCS*, and it is a *noise LCS* which contains *multiple noise templates*, then $\mathbf{t}^k$ also contains *multiple noise templates*. However, since *noise templates* are usually frequent or seasonal, $\mathbf{t}^k$ can usually be filtered out by the frequency or the seasonality features. In other words, the frequency and the seasonality features can effectively deal with the problem caused by *noise LCSes*, as will be demonstrated by the experiments to be described in Section 4.3.

3.4.4 Surge Extraction. Although some message templates are not indicative of switch failures when they occur occasionally, they likely are when they occur in a sudden surge. For example, if an interface experiences an up/down (flap), it will be easily fixed by switch's in-built failover technologies (see Cisco's support page [4]), and hence is not indicative of a failure. However, as shown in Table 2, if the interface flap occurs much more frequently than usual, a switch failure may occur, and thus should be replaced. Hence, the surge of message templates is an important feature for predicting switch failures.

Now we describe how to extract the surge feature from a syslog message sequence. Again, suppose that $\mathbf{s}^k = (s_1^k, s_2^k, ..., s_n^k)$ is time bin τ_k's corresponding message sequence. Then $\mathbf{t}^k = (t_{(s_1^k)}, t_{(s_2^k)}, ..., t_{(s_n^k)})$ is the template sequence for the time interval $[\tau_k - \Delta\tau_m, \tau_k]$. For a given message template t_j, our objective here is to find whether there are one or more sudden surges in the number of occurrences of t_j within $[\tau_k - \Delta\tau_m, \tau_k]$. By splitting $[\tau_k - \Delta\tau_m, \tau_k]$ into w smaller time intervals with equal duration $\delta (\delta \ll Duration(\mathbf{s}^k)$ and we set $\delta = 2$ *mins* in our scenario) and counting the number of occurrences of t_j in each time interval, we obtain a time series $\mathbf{C}_\delta(t_j) = C_\delta^1(t_j), C_\delta^2(t_j), ..., C_\delta^w(t_j)$, where $C_\delta^i(t_j)$ is the number of occurrences of t_j in the ith time interval. Intuitively, one or more level shifts or spikes will appear in $\mathbf{C}_\delta(t_j)$ if there are one or more sudden surges in the occurrences of t_j. To measure such level shifts or spikes, we use a behavior change detection method similar to that used in [68, 69] for assessing software changes in large Internet-based services. It is based on singular spectrum transform. For each template t_j, it computes the *change score* of $\mathbf{C}_\delta(t_j)$ in $[\tau_k - \Delta\tau_m, \tau_k]$, which we denote as cs_j^k. Whenever there is any spike or level shift in $\mathbf{C}_\delta(t_j)$, this change score cs_j^k will become much larger than usual. This way, for each syslog template t_j, we obtain a surge feature vector $\mathbf{cs}_j = (cs_j^1, cs_j^2, ..., cs_j^M)$ for time bins $(\tau_1, \tau_2, ..., \tau_M)$.

3.4.5 Feature Aggregation. As summarized in Table 6, for all time bins $(\tau_1, \tau_2, ..., \tau_M)$, after extracting the above four features, we obtain $\mathbf{C}_\Delta(t_j) = (C_\Delta^1(t_j), C_\Delta^2(t_j), ..., C_\Delta^M(t_j))$, $\mathbf{cs}_j = (cs_j^1, cs_j^2, ..., cs_j^M)$ and α_j for message template t_j, and $\mathbf{R} = (R_{\max}^{1}, R_{\max}^{2}, ..., R_{\max}^{M})$ for all templates. Because α_j for message template t_j, there is only one α_j, and a seasonal message template *cannot* be indicative of

Table 6. Variables/Vectors associated with these four features

Feature	For time bin τ_k	For template t_j
Frequency	$C_\Delta^k(t_j)$	$\mathbf{C_\Delta}(t_j) = (C_\Delta^1(t_j), C_\Delta^2(t_j), ..., C_\Delta^M(t_j))$
Seasonality	None	α_j
Sequence	$R_{\max}^k$	None
Surge	cs_j^k	$\mathbf{cs}_j = (cs_j^1, cs_j^2, ..., cs_j^M)$

failures, we assign weights to the other features based on α_j. More specifically, α_j as well as $\mathbf{C_\Delta}(t_j)$ and $\mathbf{cs}_j$ all describe the features for message template t_j. A lower α_j means stronger seasonality, and hence implies that the switch is less likely to be failure looming. Therefore, we assign weights to $\mathbf{C_\Delta}(t_j)$ and $\mathbf{cs}_j$ and get the weighted vectors as $\mathbf{C_\Delta}(t_j)' = \mathbf{C_\Delta}(t_j) \times \alpha_j$ and $\mathbf{cs}_j{}' = \mathbf{cs}_j \times \alpha_j$. Finally, we obtain the integrated feature matrix $\mathbf{A} = (\mathbf{R}, \mathbf{C_\Delta}(1)', ..., \mathbf{C_\Delta}(N)', \mathbf{cs}_1{}', ..., \mathbf{cs}_N{}')^{\mathrm{T}} = (\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_M})$ for time bins $(\tau_1, \tau_2, ..., \tau_M)$.

3.4.6 Feature Justification. The feature extraction part is the most critical component of PreFix. As Section 4.2 shows, the combination of the four features achieves relatively high precision and recall at the same time. These features all have clear physical significance in our context.

For the following three reasons, we did not adopt the end-to-end model that uses the time series of raw data (*i.e.*, time series of the template sequences extracted from the raw syslog messages) as features.

First, although taking raw data as input and employing an end-to-end system, *e.g.*, deep neural networks (DNN), to extract feature automatically is possible, we can hardly gain any insights from the results [20], mainly because the automatically extracted and integrated features cannot be directly observed and associated with domain knowledge. In contrast, PreFix uses four features with physical significance which can be easily understood intuitively by operators.

Second, usually the end-to-end model is not as generalizable as the one that extracts features using domain knowledge [41]. For example, if the data used for training and that used for prediction do not follow the same probability distribution, the end-to-end model will suffer from lower accuracy than the one with domain knowledge-based feature engineering, which is the approach in PreFix.

Third, in our scenario, the number of positive labels (failures) is relatively small (averagely 585 omen time bins for each switch model) and much smaller than that of negative ones (roughly 72,500 times more). In addition, there are about 240 different templates, which correspond to a 240-dimensional learning space. Should an end-to-end training model be applied, the small number of positive labels would have been far from sufficient to extract all the "right" features, which degrades the accuracy. More specifically, in this case the learning problem (using raw data) corresponds to putting 585 data points into a 240-dimensional space, from which hardly anything can be learned using such an end-to-end approach. Furthermore, standard dimensionality-reduction techniques do not seem to help in this case. For example, the benchmark experiments (see Section 4.2 for more details) show that SKSVM, which is a typical end-to-end model that encodes the time series of raw data into lower-dimension matrix, suffers from very low accuracy in our scenario.

3.5 Training and Predicting

In the final step of PreFix, we apply a supervised classification method, RF [24], to determine whether a given time bin's corresponding message sequence is indicative of failures (*i.e.,* omen). RF is a well-known effective ensemble learning algorithm for binary classification. RF employs multiple binary decision trees (*i.e.,* forest) during the training stage. In each binary decision tree, the class (in our case either 1 for omen or 0 for non-omen) that maximizes the information gain

(from the training data) as measured by the Gini index is selected as a candidate "vote" (for 1 or for 0). Then among these candidate votes, the majority vote is output as the final vote.

In the offline learning component, for each omen time bin, we label it 1 (positive), and similarly for each non-omen time bin, we label it 0 (negative). This way, we construct a label vector v for $(\tau_1, \tau_2, ..., \tau_M)$. Then we input feature matrix $\mathbf{A}$ as samples and v as labels into the RF model, train the model, and obtain the parameters used for online failure prediction. In the online prediction component, when a time bin's corresponding message sequence is generated, PreFix will convert the syslog messages to templates, extract features from the template sequence, and determine whether the template sequence is indicative of a failure based on the RF parameters learned in the offline learning component.

We choose RF rather than LR or SVM as the machine learning method in our scenario for the following two reasons. First, since the number of dimensions for training features is relatively high ($2 \times N + 1 \approx 600$) and the number of positive samples (omen time bins) is relatively small (as shown in Table 7) and much smaller than that of negative samples (non-omen time bins), the training models for LR and SVM are prone to overfitting and their accuracies are prone to being weighted down by this data sparsity. Second, since RF makes decisions based on the outcome of the majority voting by its many decision trees, this imbalance between the numbers of positive and negative samples has much less impact on the accuracy of RF [11]. Note we do not consider the use of the well-known RF method for this classification problem as a contribution of this work.

4 EVALUATION

In this section, we evaluate PreFix's performance. We use switch syslogs and failure tickets that are collected from several real-world datacenters owned by a top global search engine. The data set is described in details in Section 4.1. As aforementioned, all of the switch hardware failures have been manually confirmed by network operators, which are used as the ground truth for the evaluation.

To the best of our knowledge, in the literature there is no previous work on syslog-based switch failure prediction in datacenter networks, and thus in Section 4.2 we compare PreFix with two previous log based failure prediction methods, *i.e.,* SKSVM [15] applied in computers and HSMM [48] applied in ISP devices. We implement PreFix, HSMM, and SKSVM with Python 2.7.

The evaluation experiments are designed to demonstrate main contributions of this work: learning omen patterns and filtering out irrelevant templates by extracting the four features, *i.e.,* sequence, frequency, surge and seasonality, and developing the novel LCS^2 method to extract the sequence feature. To demonstrate the benefits of LCS^2 method and those of the combination of the above four features in Section 4.3, we compare the performances of PreFix when all the four features are extracted, when only the sequence feature is extracted, when all the features but the sequence feature are extracted, and when only the sequence, frequency and seasonality features are extracted.

4.1 Data Sets

In cooperation with network operators, we pick three switch models *with the most switch hardware failures* from two vendors (hereafter, we collectively use M1, M2 and M3 to represent the three switch models), and we collect data from *all* the switches of each model, and analyze *all* the hardware failures and syslogs of the switches over a *2-year period.* Table 7 lists the number of hardware failures, the number of failed switches, the number of switches in total, the number of omen time bins, and the number of non-omen time bins.

As described in Section 2.2, switch hardware failures are collected from three sources: 1) failure tickets, 2) proactive switch failure detection via SNMP polling, and 3) proactive switch failure detection via syslogs. The switch hardware failures from the three observations are *all manually*

Table 7. Detailed information for the three models of switches

Switch model	# failures	# failed switches	# switches in total	# Omen time bins	# Non-omen time bins
M1	228	131	2223	1273	5,516,435
M2	48	30	3288	317	22, 997, 509
M3	139	31	3886	164	660, 736

verified by network operators (more than 10 senior operators in the studied company are responsible for the manual verification work), and thus they can be used as the ground truth for our evaluation.

To simulate the online prediction procedure, in the evaluation experiments we slide $\Delta\tau_m$ every time bin, and obtain omen time bins and non-omen time bins following the definitions in Section 3.1.

Similar to [70], in the evaluation experiments we set $\tau_e - \tau_s = 24\ h$, $\Delta\tau_m = 2\ h$, $\Delta\tau_a = 30\ min$, $\theta = 5$ and $\xi = 15\ min$. These settings are justified as follows. Based on empirical experience, the network operators believe that syslogs within 24 *hours* before a failure are indicative of the failure, and thus can be used for failure prediction. After analyzing the syslogs before dozens of failures, we found that, in most cases (more than 90%), syslogs in any two hours among the syslogs that are within 24 hours before a given failure can capture the omen pattern. In addition, monitoring syslogs of a 24 hours' period for the online prediction procedure is computationally too intensive. The operators need at most 30 minutes to react to a positive failure prediction, such as shifting the traffic and replacing the switch. Furthermore, if there are less than $five$ syslog messages in a time bin's corresponding message sequence, the message sequence would be too short to capture the omen pattern, and thus cannot be used for failure prediction [1]. Operators' reaction time to a positive failure prediction, *i.e.,* $\Delta\tau_a$ should be divided by the length of each time bin, *i.e.,* ξ. Since $\Delta\tau_a = 30\ min$, ξ can be $30, 15, 10, 5, 2, 1$. On the one hand, if ξ is too small, *e.g.,* $\xi \in \{1\ min, 2\ min, 5\ min, 10\ min\}$, there will be too few time bins with more than $\theta = 5$ syslog messages. On the other hand, if ξ is too large, *e.g.,* $\xi = 30\ min$, the number of omen time bins will drop by more than than 50%, and thus impact greatly on the training of PreFix. Consequently, we set $\xi = 15\ min$ in this work.

To ensure that our experiments are reproducible, we have built a website which contains all the data applied in the evaluation, including the historical switch failures, message templates and message template sequences of all the three switch models [5].

Please note that for a specific switch model, the switches are usually uniformly deployed in multiple data centers. For example, the three models used in our evaluation experiments are uniformly deployed in more than 20 data centers. As a result, for a specific switch model in a specific data center, the failure cases are too few to train a failure prediction model. Therefore, it is almost infeasible to compare the performance of PreFix among different data centers. Consequently, for each switch model, we collect the data from switches across *all data centers*.

4.2 Evaluation of The Overall System

To the best of our knowledge, in the literature there are no previous works on failure prediction for switches in data center networks. That is, we do not have benchmark methods *in this domain*. Consequently, to demonstrate the performance of PreFix, we compare PreFix with *popular* log-based failure prediction methods *in other domains*. Specifically, we compare PreFix with SKSVM, which

[1] About 50% of switch failures do not have any omen time bin, *i.e.,* a time bin which contains no less than 5 syslog messages within 24 hours before the failure. We do not consider these failures in the evaluation.

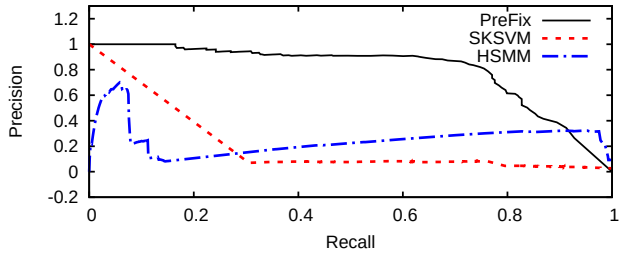
is a log-based failure prediction system used in computers [15, 17], and HSMM, which is applied for log-based failure prediction in ISP devices [48]. Please note that the parameters of SKSVM and HSMM are all set *as the ones with the best accuracy*, respectively.

A method's capability to predict failure is usually assessed by four metrics that have intuitive interpretation, *i.e.*, precision, recall, F1 measure and false positive ratio (FPR) [47, 48], and thus we use these metrics to evaluate the performance of each method. For a time bin, based on the aforementioned failure tickets, we know the outcome as either an omen time bin or not. For each method, we label its outcome as a true positive (TP), true negative (TN), false positive (FP), and false negative (FN). True positives are omen time bins that are accurately determined as such by the method, and true negatives are time bins that are accurately determined as non-omen. If the method determine that a time bin is an omen one when, in fact, it is actually non-omen, we then label the outcome as a false positive. False negatives are omen time bins that are incorrectly missed by the method. We calculate Precision, Recall and FPR as follows: $Precision = \frac{TP}{TP+FP}$, $Recall = \frac{TP}{TP+FN}$, $F1\ mesure = \frac{2*Precision*Recall}{Precision+Recall}$ and $FPR = \frac{FP}{FP+TN}$. Following [39] we also use the mean time between false alarms (MTBFA) to evaluate failure prediction methods. A method's MTBFA is the averaged elapsed time between false alarms of a switch during the deployment of the method. In our scenario, generally about 1/15 of time bins contain no less than 5 syslogs, and thus can be used for switch failure prediction. Suppose that for a switch, if the latest time bin has more than 5 syslogs, each method predicts whether there will be a failure every time bin (15 minutes), and then MTBFA can be calculated as $\frac{15\ min}{1/15 \times FPR}$.

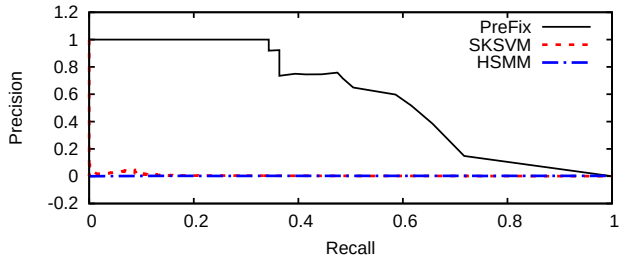
We use a *10-fold* cross validation model to evaluate the three methods (PreFix, HSMM and SKSVM). *K-fold* cross validation is a model validation technique that provides an insight on how a prediction model will generalize to an independent dataset [29], and it has been demonstrated to minimize over-fitting [40]. Since *10-fold* cross-validation is commonly used, we also apply it in our evaluation [38].

Figure 6 shows the precision recall curves (PRCs) for PreFix, SKSVM and HSMM across all the three models of switches. In PreFix, the precision and recall change as we adjust the threshold of the ratio of trees that vote for a decision (omen time bin or non-omen time bin), and we thus plot the PRCs for PreFix by adjusting this threshold. Similarly, since the threshold for classification can be customized based on Equation 10 in [48], we plot the PRCs for HSMM by varying the threshold. Please note that in a PRC to the upper right means good performance, while to the left bottom means bad performance. From the PRCs we can see that PreFix outperforms the other two methods across all the three models of switches. The performance of HSMM is not good in our scenario for it only extracts the sequence feature, while frequency, surge and seasonality of syslogs are also important as described in 3.4.1, 3.4.4, and 3.4.2, respectively. SKSVM does not perform well either, because it only extracts the sequence feature, and the method relies too much on the consecutiveness of sequences, and thus the accuracy greatly degrades when noises exist in sequences.

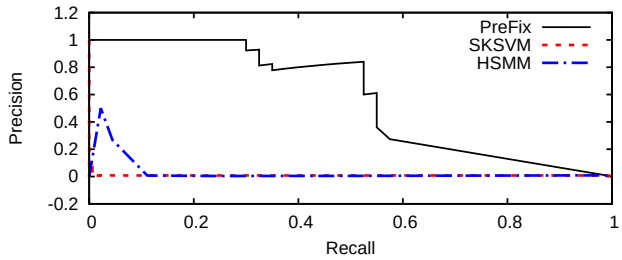
Table 8 compares the precision, recall, F1 measure and FPR of PreFix, SKSVM and HSMM across all the three models of switches when the each of the three methods achieves the best F1 measure, respectively. Although SKSVM and HSMM achieve close or even better recalls than PreFix in some cases, the precisions of SKSVM and HSMM are much more lower than that of PreFix, which in turn means that the FPRs of SKSVM and HSMM are higher than that of PreFix. If too many false positives, or false alarms are generated by a failure prediction system, network operators will be overwhelmed by the large amount of false alarms, and thus they will not like to deploy such a system. Specifically, Table 9 shows the averaged FPR and MRBFA for PreFix, SKSVM and HSMM. From the table we can see that the MTBFA of PreFix is much longer than that of SKSVM and HSMM.



(a) M1



(b) M2



(c) M3

Fig. 6. Comparison of PRCs among PreFix, SKSVM and HSMM across different models of switches

Table 8. The precision, recall, F1 measure and FPR for PreFix, SKSVM and HSMM across three models of switches

Switch model	Method	Precision	Recall	$F1$	FPR
M1	PreFix	87.35%	74.36%	80.33%	2.49×10^{-5}
	SKSVM	8.25%	76.09%	14.89%	1.96×10^{-3}
	HSMM	32.27%	95.3%	48.21%	4.63×10^{-4}
M2	PreFix	59.79%	58.59%	59.18%	5.43×10^{-6}
	SKSVM	4.47%	8.72%	5.91%	2.57×10^{-5}
	HSMM	0.28%	60.58%	0.56%	2.94×10^{-3}
M3	PreFix	84.00%	52.50%	64.61%	2.48×10^{-5}
	SKSVM	0.79%	91.91%	1.58%	2.85×10^{-2}
	HSMM	26.32%	11.11%	15.63%	7.72×10^{-5}

Table 9. The averaged FPR, MTBFA, False positives for 10, 000 switches per day for PreFix, SKSVM and HSMM

Method	Averaged FPR	MTBFA	False positives for 10, 000 switches per day
PreFix	1.84×10^{-5}	8494 days	1.2
SKSVM	1.01×10^{-2}	15 days	650.2
HSMM	1.16×10^{-3}	134 days	74.3

As aforementioned, there are tens of thousands of switches in today's large datacenter networks. Therefore, we also list the number of false alarms per day for datacenter networks including 10, 000 switches in Table 9. If PreFix is deployed for failure prediction in datacenter networks that have 10, 000 switches, an averaged 1.2 false alarms will be generated everyday, which is quite acceptable for network operators. While the averaged false alarms per day are 650.2 and 74.3 for SKSVM and HSMM, respectively. The large number of false alarms is totally unacceptable for network operators, and thus SKSVM and HSMM cannot be used for switch failure prediction in our scenario.

As shown in Appendix A, 150 decision trees are enough for the RF method of PreFix. In addition, even if there is only 30% of all the data shown used in the evaluation, PreFix will still achieve good accuracy.

4.3 Evaluation of The LCS^2 Method

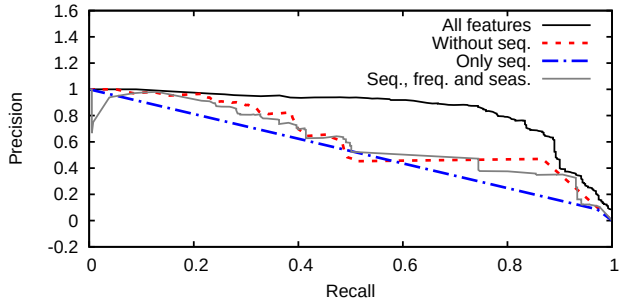
As described in Section 3.4.3, we propose a novel, simple yet effective method, LCS^2, to extract the sequence feature from template sequences. To demonstrate the performance of the method, we compare the performance of PreFix when all the four features are extracted, and all the features but the sequence feature are extracted ($\mathbf{A} = (\mathbf{C}_\Delta(1)', ..., \mathbf{C}_\Delta(N)', \mathbf{cs}_1', ..., \mathbf{cs}_N')^{\mathrm{T}}$).

Figure 7 shows the comparison results in terms of PRCs. We can see that when all the features but the sequence feature are extracted, the prediction model did not perform as well as when all the four features are extracted across all the three models of switches, especially for M1 and M3 switches. It demonstrates that the sequence feature is important for failure prediction, and that the LCS^2 method successfully extracts the useful omen sequence features for failure prediction.

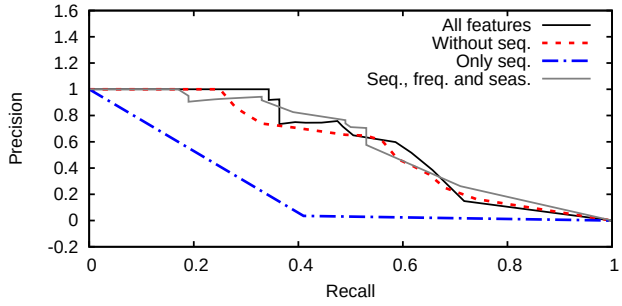
As discussed in Section 3.4.3, the frequency and seasonality features can tackle the problems posed by *noise LCSes*. Specifically, *noise LCSes* can degrade the LCS^2 method's accuracy. If $R^k_{\max}$'s corresponding *LCS* (see Section 3.4.3 for definition) is a *noise LCS*, and it contains *multiple noise templates*, we can infer that the template sequence in τ_k, *i.e.,* $\mathbf{t}^k$, also contains *multiple noise templates*. For the reason that a *noise template* should be frequent or seasonal, the frequency or seasonality feature can easily filter out $\mathbf{t}^k$ from omen template sequences. This way, the frequency and seasonality features tackle the problem induced by *noise LCSes*. To confirm above inferences, we evaluate the performance of PreFix when only the sequence feature is extracted (the integrated matrix $\mathbf{A} = \mathbf{R}$), and when the sequence, frequency and seasonality features are extracted ($\mathbf{A} = (\mathbf{R}, \mathbf{C}_\Delta(1)', ..., \mathbf{C}_\Delta(N)')^{\mathrm{T}}$).

As Figure 7 shows, if only the sequence feature is extracted, the prediction model does not perform as well as when the sequence, frequency and seasonality features are extracted. It is especially obvious when the input data is collected from M2 switches. The comparison results strongly verify the above analysis.

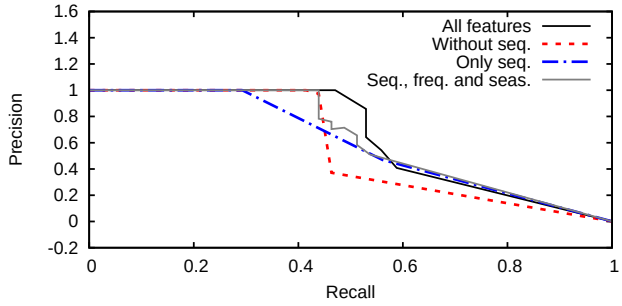
In a nutshell, the LCS^2 method which extracts the sequence feature is very important for failure prediction. In addition, the frequency and seasonality features mitigate the problem introduced by *noise LCSes*. The combination of the four features is really necessary in predicting failures for switches.



(a) M1



(b) M2



(c) M3

Fig. 7. PRCs of PreFix when all the four features are extracted, only the sequence feature is extracted, all the features but the sequence feature are extracted, and only the sequence, frequency, and seasonality features are extracted

4.4 Comparison of the Importance of the Four Features

To demonstrate the importance of the features, we calculate the *node impurity decrease* of the features in the RF model.

Random forest is comprised of a number of decision trees. Every node in the decision trees denotes a condition of a single feature. The node is mainly for splitting the dataset into two, and hence similar values will be classified into the same set. In general, decision trees choose the (locally) optimal condition based on *node impurity*, which is typically either Gini impurity or

Table 10. Normalized node impurity decrease of the features in the RF model

Switch model	Sequence	Frequency&Seasonality	Surge&Seasonality
M1	22.29%	51.14%	26.57%
M2	19.09%	50.25%	30.65%
M3	42.81%	36.86%	20.33%

information gain/entropy. Therefore, we can compute by how much each feature decreases the weighted impurity in a tree to figure out the importance of the feature. Similarly, in a random forest we can average the *node impurity decrease* in different trees, and thus quantitatively measure the importance of each feature.

As described in Section 3.4.5, in the integrated matrix $\mathbf{A}$ we combine the frequency feature with the seasonality feature ($\mathbf{C}_\Delta(t_j)' = \mathbf{C}_\Delta(t_j) \times \alpha_j$) and combine the surge feature with the seasonality feature ($\mathbf{cs}_j{}' = \mathbf{cs}_j \times \alpha_j$). Since *node impurity decrease* is calculated based on $\mathbf{A}$, we calculate the *node impurity decrease* of the combination of frequency and seasonality feature, and that of the combination of the surge and seasonality feature.

The normalized *node impurity decrease* of each feature (or combination of features) across different switch models is shown in Table 10. From the table we can see that sequence, the normalized node impurity of the combination of frequency and seasonality, and the combination of surge and seasonality are all greater than 19%, which proves that they are all important to training PreFix. That is, any of the above three contributes significantly to PreFix, which is consistent with the conclusion drown in Section 4.3.

5 RELATED WORK

Using log files for failure prediction has been widely applied in in ISP networks [25, 48, 49], computers [15–17, 33, 47, 71], virtual machines [58], and online ad service [51]. Liang *et al.* investigated the RAS event logs, and developed three simple failure prediction methods based on the characteristics of failure events, as well as the correlation between failure events and non-failure events [33]. Realizing the the importance of the sequential feature of log files to failure prediction, Fronza *et al.* used random indexing (RI) to represent the sequence of operations extracted from logs, and then applied weighted SVM to associate sequences to a class of failures or non-failures [16]. RI is not applied in our scenario because unlike the strictly structured software logs [16], the syslog messages of network devices in our scenario are usually unstructured. Salfner *et al.* applied HSMM to recognize the patterns of logs that indicate an imminent failure directly [48]. The drawback of using HSMM for failure prediction is that HSMM cares only about the sequential feature of logs while ignoring the other features like frequency, seasonality and surge, and thus it is not suitable in our scenario.

Several works has been conducted on analyzing syslogs of network devices for failure detection or prediction purpose [27, 28, 44, 67]. For example, motivated by the signature abstraction method used in spam detection, Qiu *et al.* constructed breath-first search tree which learned templates from syslogs, based on the frequency of words in syslogs [44]. Considering the difficulty of determining the root of the tree in the case of general logs for the above method, Kimura *et al.* presented a statistical template extraction (STE) method using a statistical clustering algorithm which consisted of two parts: statistical word scoring and score clustering [27]. Noticing that the format of syslog messages can dynamically change over time, Kimura *et al.* developed an online template extraction method to learn templates incrementally [44]. The method includes classification of words based on

the words' tendency to belong to a log template, and message clustering based on the log similarity between template clusters and new arrival messages.

There are two categories of sequence feature extraction methods for classification in previous works: feature based methods and sequence distance based methods [63]. The feature based method, *i.e.,* SKSVM, uses the *k-spectrum kernel* to transform a sequence into a feature vector and then apply conventional classification methods such as SVM and decision tree [31, 32, 36, 54]. While SKSVM has been successfully used in the prediction of computer system failures [15, 18], it does not perform well in our scenario because: (1) If we apply the identifier of a syslog message's corresponding message template as the *tag* of the syslog message for SKSVM, the method will cost too much CPU and memory resources. (2) SKSVM relies much on the consecutiveness of sequence data, and the *noise templates* in failure omen template sequences can greatly degrade the accuracy of the method. The sequence distance based methods use local alignment distance [46] or dynamic time wrapping distance (DTW) [26, 45, 62] to measure the similarity between a pair of template sequences, and classify the template sequences data based on the similarity results. Both local alignment distance and DTW are based on calculating the distance of *key points*. However, if we directly use the local alignment distance or DTW to measure the similarity between the a template sequence and omen template sequences, the *noise templates* in the omen template sequences can shadow the *key points* and greatly degrade the accuracy of failure prediction. Therefore, the sequence distance based methods do not work well in our scenario.

6 CONCLUSION

We designed and implemented a new tool, PreFix, for accurately predicting whether there will be a switch hardware failure in the near future. For each model of switches, in the offline procedure PreFix learns templates from historical syslogs, converts syslogs to templates, extracts the frequency, surge, seasonality and sequence features from template sequences, and applies RF to learn omen and non-omen patterns. In the online procedure PreFix matches real-time syslogs to templates, extracts the four features, and uses RF to determine whether the syslog messages are omen or not. We evaluated the whole system of PreFix by comparing it with SKSVM and HSMM using 9397 switches that belong to three models of switches which are deployed in more than 20 datacenters, and demonstrated that PreFix greatly outperforms SKSVM and HSMM in accuracy. We also evaluated the LCS^2 method applied in PreFix using real-world data. As for future work, we plan to learn failure omens across different switch models. In addition, we will look into predicting other types of switch failures such as those caused by software bugs.

REFERENCES

[1] 2010. Switch failure causes outages at Hosting.com data center. http://www.datacenterdynamics.com/content-tracks/servers-storage/switch-failure-causes-outages-at-hostingcom-data-center/32344.fullarticle. (June 2010).
[2] 2011. Transfer switch failure causes outage at Colo4 data center. http://www.datacenterdynamics.com/content-tracks/power-cooling/transfer-switch-failure-causes-outage-at-colo4-data-center/32548.fullarticle. (Augest 2011).
[3] 2015. Data center failure downs Virginia State computer network. http://www.datacenterdynamics.com/content-tracks/security-risk/data-center-failure-downs-virginia-state-computer-network/96247.article. (May 2015).
[4] 2017. Cisco Nexus 7000 Series NX-OS Interfaces Configuration Guide, Release 5.x. http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/5_x/nx-os/interfaces/configuration/guide/if_cli/if_basic.html. (June 2017).
[5] 2017. Project URL. https://www.dropbox.com/sh/t2yw2stfnzlecb3/AACCh5sdaMF5RObD708xDcJca?dl=0. (June 2017).
[6] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *SIGCOMM*. Seattle, WA, USA.
[7] Lloyd Allison and Trevor I Dix. 1986. A bit-string longest-common-subsequence algorithm. *Inform. Process. Lett.* 23, 5 (1986), 305–310.
[8] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. 2016. Predicting Disk Replacement towards Reliable Data Centers. In *Proceedings of the 22nd ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining. ACM, 39–48.

[9] Guo Chen, Youjian Zhao, Dan Pei, and Dan Li. 2015. Rewiring 2 Links is Enough: Accelerating Failure Recovery in Production Data Center Networks. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*. IEEE, 569–578.

[10] Maxime Crochemore, Costas S Iliopoulos, Yoan J Pinzon, and James F Reid. 2001. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Inform. Process. Lett.* 80, 6 (2001), 279–285.

[11] Liu Dapeng, Zhao Youjian, Xu Haowen, Sun Yongqian, Pei Dan, Luo Jiao, Jing Xiaowei, and Feng Mei. 2015. Opprentice: Towards Practical and Automatic Anomaly Detection through Machine Learning. In *ACM IMC*. Tokyo, Japan.

[12] Alain De Cheveigné and Hideki Kawahara. 2002. YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America* 111, 4 (2002), 1917–1930.

[13] Mukund Deshpande and George Karypis. 2002. Evaluation of techniques for classifying biological sequences. In *Advances in Knowledge Discovery and Data Mining*. Springer, 417–431.

[14] Cees Elzinga, Sven Rahmann, and Hui Wang. 2008. Algorithms for subsequence combinatorics. *Theoretical Computer Science* 409, 3 (2008), 394–404.

[15] R Wesley Featherstun and Errin W Fulp. 2010. Using Syslog Message Sequences for Predicting Disk Failures. In *LISA*.

[16] Ilenia Fronza, Alberto Sillitti, Giancarlo Succi, Mikko Terho, and Jelena Vlasenko. 2013. Failure prediction based on log files using Random Indexing and Support Vector Machines. *Journal of Systems and Software* 86, 1 (2013), 2–11.

[17] Errin W Fulp, Glenn A Fink, and Jereme N Haack. 2008. Predicting Computer System Failures Using Support Vector Machines. *WASL* 8 (2008), 5–5.

[18] Errin W. Fulp, Glenn A. Fink, and Jereme N. Haack. 2008. Predicting Computer System Failures Using Support Vector Machines. In *Proceedings of the First USENIX Conference on Analysis of System Logs (WASL'08)*. 5–12.

[19] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *SIGCOMM*.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[21] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *SIGCOMM*. Barcelona, Spain.

[22] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, Zhi-Wei Lin, and Varugis Kurien. 2015. Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. 139–152.

[23] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. 71–85.

[24] Tin Kam Ho. 1995. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, Vol. 1. IEEE, 278–282.

[25] Guenther Hoffmann and Miroslaw Malek. 2006. Call availability prediction in a telecommunication system: A data driven empirical approach. In *Reliable Distributed Systems, 2006. SRDS'06. 25th IEEE Symposium on*. IEEE, 83–95.

[26] Eamonn J Keogh and Michael J Pazzani. 2000. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 285–289.

[27] Tomohiro Kimura, Koji Ishibashi, Takayoshi Mori, Hideyuki Sawada, Tsuyoshi Toyono, Ken Nishimatsu, Atsuyori Watanabe, Akihiro Shimoda, and Kohei Shiomoto. 2014. Spatio-temporal factorization of log data for understanding network events. In *INFOCOM, 2014 Proceedings IEEE*. IEEE, 610–618.

[28] Tatsuaki Kimura, Akio Watanabe, Tsuyoshi Toyono, and Keisuke Ishibashi. 2015. Proactive failure detection learning generation patterns of large-scale network logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 8–14.

[29] Ron Kohavi. 1995. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2 (IJCAI'95)*. 1137–1143.

[30] Terran Lane and Carla E Brodley. 1999. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security (TISSEC)* 2, 3 (1999), 295–331.

[31] Christina Leslie and Rui Kuang. 2004. Fast string kernels using inexact matching for protein sequences. *The Journal of Machine Learning Research* 5 (2004), 1435–1455.

[32] Christina S Leslie, Eleazar Eskin, and William Stafford Noble. 2002. The spectrum kernel: A string kernel for SVM protein classification.. In *Pacific symposium on biocomputing*, Vol. 7. 566–575.

[33] Yinglung Liang, Yanyong Zhang, Morris Jette, Anand Sivasubramaniam, and Ramendra Sahoo. 2006. Bluegene/l failure analysis and prediction models. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. IEEE,

425–434.
[34] Junda Liu, Aurojit Panda, Ankit Singla, Brighten Godfrey, Michael Schapira, and Scott Shenker. 2013. Ensuring Connectivity via Data Plane Mechanisms. In *NSDI*.
[35] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-tolerant Engineered Network. In *NSDI*.
[36] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *The Journal of Machine Learning Research* 2 (2002), 419–444.
[37] Zhiling Luo, Ying Li, Ruisheng Fu, and Jianwei Yin. 2016. Don't Fire Me, a Kernel Autoregressive Hybrid Model for Optimal Layoff Plan. In *Big Data (BigData Congress), 2016 IEEE International Congress on*. IEEE, 470–477.
[38] Geoffrey McLachlan, Kim-Anh Do, and Christophe Ambroise. 2005. *Analyzing microarray gene expression data*. Vol. 422. John Wiley & Sons.
[39] G Martin Milner. 2005. Detection/classification/quantification of chemical agents using an array of surface acoustic wave (SAW) devices. In *Proceedings of SPIE*, Vol. 5778. 305–316.
[40] Andrew W Moore. 2001. Cross-validation for detecting and preventing overfitting. *School of Computer Science Carneigie Mellon University* (2001).
[41] Nasser M. Nasrabadi. 2007. Pattern Recognition and Machine Learning. *Journal of Electronic Imaging* 16 (2007). https://doi.org/10.1117/1.2819119
[42] Srinivasan Parthasarathy, Sameep Mehta, and Soundararajan Srinivasan. 2006. Robust periodicity detection algorithms. In *Proceedings of the 15th ACM international conference on Information and knowledge management*. ACM, 874–875.
[43] Rahul Potharaju and Navendu Jain. 2013. Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC '13)*. 9–22.
[44] Tongqing Qiu, Zihui Ge, Dan Pei, Jia Wang, and Jun Xu. 2010. What Happened in My Network: Mining Network Events from Router Syslogs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. 472–484.
[45] Chotirat Ann Ratanamahatana and Eamonn Keogh. [n. d.]. *Making Time-series Classification More Accurate Using Learned Constraints*. 11–22.
[46] Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. 2004. Protein homology detection using string alignment kernels. *Bioinformatics* 20, 11 (2004), 1682–1689.
[47] Felix Salfner, Maren Lenk, and Miroslaw Malek. 2010. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)* 42, 3 (2010), 10.
[48] Felix Salfner and Miroslaw Malek. 2007. Using hidden semi-Markov models for effective online failure prediction. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE, 161–174.
[49] Felix Salfner and Steffen Tschirpke. 2008. Error Log Processing for Accurate Failure Prediction. In *Proceedings of the First USENIX Conference on Analysis of System Logs (WASL'08)*.
[50] Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *ACM computing surveys (CSUR)* 34, 1 (2002), 1–47.
[51] Mohammed Shatnawi and Mohamed Hefeeda. 2015. Real-time failure prediction in online services. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1391–1399.
[52] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. 183–197.
[53] Josef Sivic and Andrew Zisserman. 2009. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence* 31, 4 (2009), 591–606.
[54] Sören Sonnenburg, Gunnar Rätsch, and Bernhard Schölkopf. 2005. Large scale genomic sequence SVM classifiers. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 848–855.
[55] Pang-Ning Tan and Vipin Kumar. 2004. Discovery of web robot sessions based on their navigational patterns. In *Intelligent Technologies for Information Analysis*. Springer, 193–222.
[56] Michail Vlachos, S Yu Philip, and Vittorio Castelli. 2005. On Periodicity Detection and Structural Periodic Similarity.. In *SDM*, Vol. 5. SIAM, 449–460.
[57] Meg Walraed-Sullivan, Amin Vahdat, and Keith Marzullo. 2013. Aspen Trees: Balancing Data Center Fault Tolerance, Scalability and Cost. In *CoNEXT*.
[58] Yoshihiro Watanabe, Hiroyuki Otsuka, Masataka Sonoda, Shinji Kikuchi, and Yuki Matsumoto. 2012. Online failure prediction in cloud datacenters by real-time message pattern learning. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, 504–511.

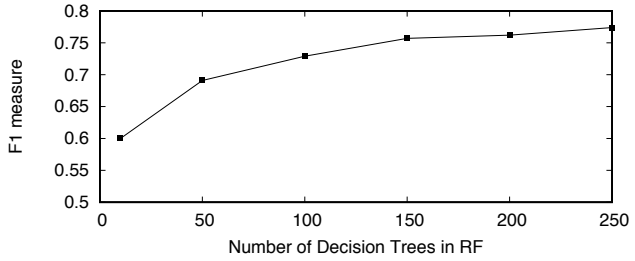
[59] Li Wei and Eamonn Keogh. 2006. Semi-supervised time series classification. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 748–753.
[60] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. 2012. NetPilot: automating datacenter network failure mitigation. In *Proceedings of the 2012 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '12)*. 419–430.
[61] Yang Wu, Mingchen Zhao, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. 2014. Diagnosing missing events in distributed systems with negative provenance. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 383–394.
[62] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. 2006. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 1033–1040.
[63] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. 2010. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter* 12, 1 (2010), 40–48.
[64] Ji Xue, Robert Birke, Lydia Y Chen, and Evgenia Smirni. 2016. Managing data center tickets: Prediction and active sizing. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 335–346.
[65] Minlan Yu, Albert G Greenberg, David A Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. 2011. Profiling Network Performance for Multi-tier Data Center Applications. In *NSDI*.
[66] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic test packet generation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. ACM, 241–252.
[67] Shenglin Zhang, Ying Liu, and Dan Pei. 2014. A measurement study on BGP AS path looping (BAPL) behavior. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 1–7.
[68] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. 2015. Rapid and Robust Impact Assessment of Software Changes in Large Internet-based Services. In *CONEXT*. Heidelberg, Germany.
[69] Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, Zhi Zang, Xiaowei Jing, and Mei Feng. 2016. FUNNEL: Assessing Software Changes in Web-based Services. *IEEE Transactions on Services Computing* (2016).
[70] Shenglin Zhang, Weibin Meng, Jiahao Bu, Sen Yang, Ying Liu, Dan Pei, Jun Xu, Yu Chen, Hui Dong, Xianping Qu, et al. 2017. Syslog processing for switch failure diagnosis and prediction in datacenter networks. In *Quality of Service (IWQoS), 2017 IEEE/ACM 25th International Symposium on*. IEEE, 1–10.
[71] Z. Zheng, Z. Lan, B. H. Park, and A. Geist. 2009. System log pre-processing to improve failure prediction. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*. 572–577.

A THE AMOUNT OF TRAINING AND DATA NECESSARY FOR PREFIX

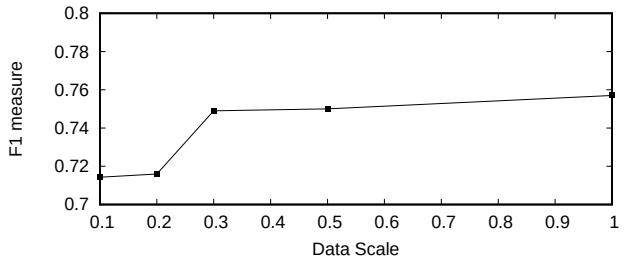
To demonstrate how much training is necessary for PreFix, we varied the number of decision trees in RF from 10 to 250, and calculated the F1 measure for each number using the 10-fold cross validation model. The number of trees is a key parameter for random forest: the more decision trees are in random forest, the more training the training model needs. Figure 8 (a) shows the evaluation results when PreFix is applied on M1 switch model. The F1 measure tends to be stable when the number of decision trees in RF is greater than 150. That is, 150 decision trees are enough for the RF method in PreFix to learn omen patterns of switch failures.

In addition, to understand how much data is necessary for PreFix, we sampled some proportion (say 0.1, 0.2, 0.3, 0.5) of all the data and calculated the F1 measure for each proportion. Specifically, for each proportion value, we randomly picked the proportion of omen samples and non-omen ones from all the omen samples and all the non-omen ones of M1 switch model, respectively. We then applied PreFix on the selected samples to calculate the F1 measure using the 10-fold cross validation model. Figure 8 (b) shows the result. When the proportion is greater than 0.3, the F1 measure becomes stable. Consequently, PreFix performs well even only 30% of samples are used.

To minimize over-fitting, we applied 10-fold cross validation to calculate the F1 measures in the above two experiments.



(a) F1 measure against the number of trees in RF



(b) F1 measure against the data scale

Fig. 8. F1 measures of PreFix on M1 switch model when the number of decision trees in RF, and data scale vary

ACKNOWLEDGMENTS

We thank our shepherd Evgenia Smirni and the anonymous reviewers for their insightful comments and feedbacks on the paper. We appreciate Yousef Azzabi and Juexing Liao for their helpful suggestions and elaborative proofreading.

The work was supported by National Natural Science Foundation of China (NSFC) under grant No.61402257, No. 61472214 and No. 61472210, US NSF under grant No. CNS 1218092 and No. CNS1423182, the National Key Basic Research Program of China (973 program) under grant No. 2013CB329105, the Global Talent Recruitment (Youth) Program, and the Cross-disciplinary Collaborative Teams Program for Science, Technology and Innovation, of Chinese Academy of Sciences-Network and system technologies for security monitoring and information interaction in smart grid.

Received November 2017, revised January 2018, accepted March 2018.