Graduate School ETD Form 9
(Revised 01/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By RAHUL POTHARAJU

Entitled **Data-Driven Approaches to Improve Dependability of Cloud Services**

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Cristina Nita-Rotaru	Navendu Jain
Sonia Fahmy	
Jennifer Neville	
Ninghui Li	

To the best of my knowledge and as understood by the student in the *Thesis/Dissertation Agreement. Publication Delay, and Certification/Disclaimer (Graduate School Form 32)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Cristina-Nita-Rotaru

Approved by: Sunil Prabhakar 04/30/2014

Head of the Department Graduate Program Date

DATA-DRIVEN APPROACHES TO IMPROVE DEPENDABILITY OF CLOUD SERVICES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Rahul Potharaju

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2014

Purdue University

West Lafayette, Indiana

In dedication to my mother, who has sacrificed so much for making me into who I am, my father, who has instilled in me the power of endurance, my brother, who has taught me to believe in hard work and for always looking out for me, my wife, who has given me the love and support that enabled the hours of contemplation necessary to complete this research, to my sister-in-law and nephew, who have provided the extra motivation to finish my Ph.D.

ACKNOWLEDGMENTS

There are a number of people without whom this dissertation might not have been written, and to whom I am greatly indebted. I take this opportunity to express my gratitude to the people who have been instrumental in my journey so far.

I feel highly privileged to take this opportunity to wish my profound gratitude with a deep sense of obligation to my doctoral advisor, **Professor Cristina Nita-Rotaru**, for her inspiring guidance, helping attitude, and above all for providing necessary support during the whole span of this research work. She has provided key guidance during my years at Purdue as a PhD candidate. The quality of this document is also largely due to her never ending stream of comments and suggestions for improvement – I am grateful to you for holding me to a high research standard and enforcing strict validations for each research result. Thank you for teaching me the noble arts of science, project management, technical writing, manuscripts preparation and publications. You have given me skills that would guide me in my research journey for years to come.

My mentor, **Dr. Navendu Jain** (Microsoft Research), inspired me to forge ahead into uncharted technological territory with his persistence, guidance, meticulous excitement and seemingly limitless drive. He has the attitude and substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and instilled in me a deep sense of discipline. Without his guidance and persistent help, this dissertation would not have been possible.

I would also like to express the deepest appreciation to my committee members, **Professor Sonia Fahmy**, **Professor Ninghui Li** and **Professor Jennifer Neville**, for their encouragement, insightful and constructive comments, and hard questions. I take this opportunity to record my sincere thanks to all the faculty members of the Department of Computer Science and CERIAS at Purdue University for their help. I would like to express my sincere gratitude to my thesis reader, **Dr. William Gorman**, for sharing his knowledge and constructive comments which helped a lot in improving the quality of this document.

I am thankful to **Endadul Hoque**, an extremely dear friend and an excellent lab partner for all the moral support, encouragement and bonding that I will forever treasure. Thank you for believing in me, even when I did not believe in myself. Another friend that requires special mention is **Andrew Newell**, a friend that I always admired – I thank you for all the intellectual debates we've had over the years.

Nothing happens all of a sudden but is the outcome of something preceding, and this dissertation is no exception. I would like to thank one person in particular, **Dr. Bogdan Carbunar**. Although you have not been directly involved in this dissertation, I would like you to know that you have been a great source of motivation for me. Thank you for supporting me and believing in me. Without your initial support, constant guidance, motivation and encouragement, I could not have made it this far.

I thank all fellow labmates: **David Zage**, **Jeff Seibert**, **Jayaram Radhakrishnan**, **Aditi Gupta**, **Christopher Gates**, **Hyojeong Lee**, **Luojie Xiang**, **Salmin Sultana**, **Pawan Prakash**, **Advait Dixit**, **Rohit Ranchal**, **Wahbeh Qardaji**, **Ruchith Fernando**, **Nabeel Yousuf**, **Dong Su** and **Haining Chen**, for the stimulating discussions. Thank you for providing a very warm environment and being my family in the university. A million thanks for all the useful tips and encouragement.

My deepest thanks go to my long-time friends, now spread all over the world. They kept me sane and connected to an important part of my life. **Pradyumna Goli**, **Divya Goli**, **Alekhya Vemuri**, **Aditya Vemuri**, **Farzana Rahman**, **Sree Divya Vadlapudi**, **Jayashree Subramanian**, **Arpit Jain**, **Rohit Sehgal**, **Deepika Gopalakrishnan**, **Deepak Reddy**, **Avinash Reddy**, **Chatur Bhuja Mishra**, **Sandeep Chary**, **Suryanarayana Murty**, **Sankalp Sarma**, **Virajit Jalaparti**, **Charan Tej**, **Sankeerth Reddy**, **Amit Mondal**, **Olusanya Soyannwo**, **Ionut**

Trestian, **Mario Sanchez**, **John Otto**, **David Choffnes**, **Jiazhen Chen**, **Yinzhi Cao**, **Hongyu Gao** and **Hitesh Khandelwal**.

Great thanks to my favorite music artists such as **Linkin Park**, **Eminem**, **OneRepublic**, **Take That**, **Maroon 5**, **Bruno Mars**, **A.R. Rahman** and **Yanni**. I have gone through long periods of time where your music was my only companion. Thank you for creating wonderful music that inspired me during times of deep stress.

My special thanks to my colleagues from Microsoft Research: **Dr. Nachi Nagappan**, **Dr. Ishai Menache**, **Dr. Jim Larus** and **Dr. Vivek Narasayya** for their valuable guidance and encouragement. I am also very grateful to Microsoft Research: It was such an honor to have been given the opportunity to intern so many times at this wonderful place.

Thanks to my previous mentors and teachers: **Professor Yan Chen**, **Professor Fabian Bustamante**, **Professor Aleksandar Kuzmanovic**, **Professor Peter Dinda** from my time at Northwestern University, **Dr. Jehan Wickramasuriya**, **Dr. Shivajit Mohapatra**, **Dr. Nitya Narasimhan**, **Dr. Faisal Ishtiaq**, **Mike Needham**, **Michael Pearce**, and **Dr. Venu Vasudevan** from my time at Motorola Applied Research for preparing me well for the challenge of graduate school.

The beliefs that no problem is unsolvable, that an optimistic outlook precludes solving seemingly insurmountable tasks and that the right perspective will simplify all of my endeavors were instilled within me by my parents, **Dr. Kamala Devi** and **Dr. Nagabhushana Rao**. Their love and guidance has helped me tremendously. My father, especially, has been my role model throughout my academic career – I still continue learning how to make presentation decks from him. My brother, **Dr. Anil Kumar**, has been both a dependable friend and a mentor and has played a critical role in my professional development. I want to thank my sister-in-law, **Manasa Rao**, from the bottom of my heart for her continuous support, generosity and encouragement. I am also greatly indebted to my in-laws **Suthaja** and **TR Ramanujam** for their support and encouragement throughout. **Sindu**, as for you, you are my rock, my love, my life and my best friend, our life together beckons.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Potharaju, Rahul Ph.D., Purdue University, May 2014. Data-driven Approaches to Improve Dependability of Cloud Services. Major Professor: Cristina Nita-Rotaru.

The growing demand for always-on and low-latency cloud services is driving the creation of globally distributed datacenters. A major factor affecting service availability is reliability of the network, both inside the datacenters and wide-area links connecting them. While several research efforts focus on building scale-out datacenter networks, little has been reported on real network failures and how they impact geo-distributed services. Towards improving the dependability of the underlying datacenter networks, in this dissertation, we make one of the first attempts to characterize *intra-datacenter* and *inter-datacenter* network failures from a service perspective. Specifically, we make the following contributions:

1. Analysis Methodology for Structured Data: Our dataset includes multiple sources of structured network telemetry data spanning three years logged in monitoring servers of a large cloud provider comprising 100k+ servers, 10k+ core network devices, 2k+ middleboxes and 100k+ network links across 10+ datacenters. This dataset covers a wide range of network data sources, including syslog and SNMP alerts, and traffic carried by links. To this end, we describe a systematic methodology for analyzing this structured data based on event processing to extract events having service-level impact.

2. Analysis Methodology for Unstructured Data: Our dataset also includes an important piece of operational knowledge – network trouble tickets, which are diaries written by network operators to keep track of their troubleshooting efforts while fixing a problem. To this end, we take a *practical* step towards automatically analyzing natural language text in network trouble tickets to infer the problem symptoms,

troubleshooting activities and resolution actions. Our system, NetSieve combines statistical natural language processing (NLP), knowledge representation, and ontology modeling to achieve these goals.

3. Data-Driven Approaches to Deriving Actionable Insights: Our overarching goal in this dissertation is to enable operators to understand global problem trends instead of making decisions based on isolated incidents. We outline several analyses rooted in reliability analysis and applied statistics for characterizing network failures and deriving actionable insights from them. Our study reveals several important findings on (a) the failure characteristics of network elements, (b) the availability of network domains, (c) service impact, (d) causes of network failures, (e) effectiveness of repairs, and (f) modeling failures.

As part of this dissertation, we have built a broad range of systems including real-time network dashboards, a big data analytics system for analyzing network telemetry data, and an inference tool for root cause analysis in network troubleshooting. Several components of the dissertation work either have undergone a tech-transfer or are being used by multiple business groups inside Microsoft. NetWiser, a Microsoft Research project entailing this dissertation, was awarded the *Microsoft Trustworthy Computing Reliability Award* for 2013.

The problem inference system part of this dissertation, NetSieve, is currently being used across different teams within Microsoft to improve network management: the Network Architecture team for comparing device reliability across platforms and vendors, the Capacity Planning team for understanding why network redundancy is ineffective in masking failures, and the Incident Management and Operations team for finding the top-k problems and failing components while troubleshooting devices and determining whether past repairs were effective. Since its inception, NetSieve has also been used to automate root cause analysis of security incidents within Microsoft's datacenters and recently found its way into commercial use through Microsoft's System Center Advisor (http://www.systemcenteradvisor.com).

1 INTRODUCTION

Cloud services are growing rapidly to provide a fast-response and always-on experience to end users. Reliability is critically important for these services as failures not only hurt site availability and revenue, but also risk data loss. For instance, Dropbox experienced two recent widespread outages [1, 2] which prevented its users from synchronizing files or accessing its site. In another instance, the hurricane Sandy led to flooding of many datacenters in NYC causing service outages [3], and failures of many trans-atlantic fiber links peering from NYC significantly degrading capacity [4]. In 2011, the entire US East region of Amazon became unavailable due to a faulty fail-over during maintenance [5] impacting several popular services such as Dropbox, Foursquare, Instagram, Quora and Reddit.

To increase service availability in a cost-effective manner, cloud providers are deploying their services across geo-distributed datacenters [6], and building their networks based on a scale-out design using inexpensive commodity hardware [7–9]. However, as the number of devices and links in a datacenter grows, failures become the norm rather than the exception. Making things worse, the network infrastructure also comprises long-haul, inter-datacenter links to synchronize and replicate user data and application state [10]. These links comprise a variety of components such as cables, optical adapters, and complex software protocol stacks, whose failures can lead to reduced bandwidth capacity, stale data, or even service outages.

Despite the practical significance of cloud network failures, little is known about how they impact services. The research literature offers several real-world studies on failures of disk and storage systems [11–14], DRAM [15], personal computers [16], and errors in software configuration [17], but they do not consider network failures. Recent studies [18–20] study failures of network switches, *but they neither analyze* *their impact on cloud services nor do they examine inter-datacenter network failures*; we compare to prior work in detail in Chapter 6.

In the following, we will first study the types of service impact caused by network failures pertaining to the network core and middleboxes. Next, we will look at the diversity of services impacted by network failures.

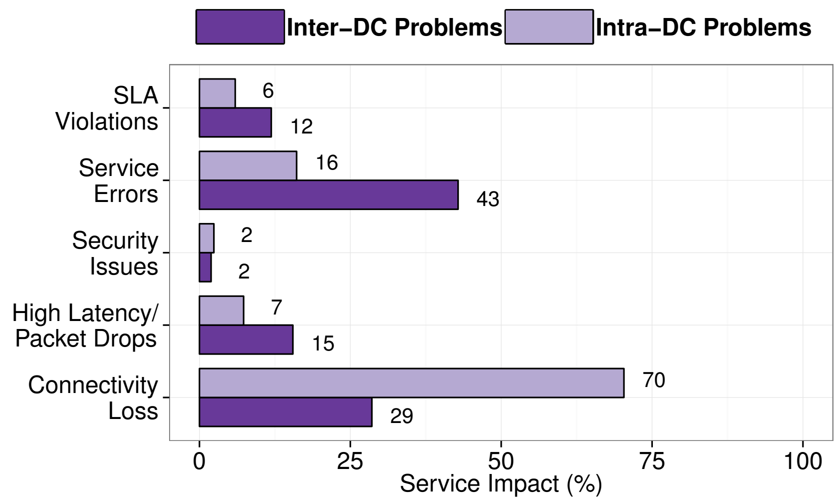


Figure 1.1. The top-5 categories of service impact observed from the high severity incidents attributed to Intra-Datacenter network devices (Layer-3 routers and Layer-2 switches) and Inter-Datacenter network links. Connectivity loss problems (70%) and Service Errors (43%) dominate the impact due to Intra-Datacenter and Inter-Datacenter network problems, respectively.

- **Types of Service Impact:** Figure 1.1 shows the top-5 categories of service impact caused by network failures based on our analysis of high severity incidents spanning five years (2008-13) in a cloud provider comprising dozens of datacenters; an incident is high severity if it causes high customer or business impact. The network elements comprise both Intra-Datacenter Layer-3 and Layer-2 devices (Access routers, Aggregation switches and Top-of-Rack switches) and Inter-Datacenter links. We observe that loss of connectivity and service errors (e.g., replication problems leading to stale data) dominate the service impact. Further, we observe that Inter-Datacenter network failures are caused due to link flapping

(36%), high link utilization (29%) and unplanned changes (6%). In comparison, the Intra-Datacenter network failures are dominated by connectivity errors (64%-78%), hardware failures (20%-73%) and software problems (7%-24%) across Layer-3 and Layer-2 devices.

- **Categories of Impacted Services:** Our analysis of the high severity incidents revealed that network failures impact a broad range of services. The Intra-Datacenter network failures mainly affected messaging (e.g., email, IM services, SMS) services in 38.9% of the incidents. SaaS applications (e.g., web hosting, CDN, data analytics) were affected in 32% of the incidents equally due to Intra- and Inter-Datacenter problems where the geo-distributed services were disrupted for up to several hours. Thus, understanding network failures at both the intra- and inter-datacenter level is important to deliver high availability for cloud services.
- **Impact of Middlebox Failures**: Next, we provide empirical evidence supporting the prominence of middleboxes in datacenters and their significant contribution to network outages. We utilize incident troubleshooting data over a three year period (2009-2011) comprising *high priority incidents*: an event is classified as high priority if: (i) it significantly impacts datacenter traffic (monitored using traffic monitors placed near the edge of the datacenter) or (ii) a customer registers a complaint because one or more of his online services becomes inaccessible. Since these incidents denote a significant impact from both customer and financial standpoint, they have the highest priority to be resolved.

We analyzed the diary entries from network trouble tickets of these incidents and for each we inferred the problematic device based on the diary description. Fig 1.1 shows the % contribution by middleboxes and other network devices to these incidents along with their population; absolute device counts omitted due to confidentiality reasons. We observe that middleboxes constitute only 11.4% of the population, yet, they contributed to 43.1% of the high priority incidents.

1.1 Goal

This dissertation centers around the problem of improving dependability of cloud services. We address a variety of problems along the following dimensions:

- **Characterizing the impact of network failures on services**: To increase service availability amidst multiple failure sources such as hardware, software, and human errors, cloud providers are deploying their services across geo-distributed datacenters [6], and building their networks based on a scale-out design using inexpensive commodity hardware [7–9]. To this end, we characterize the impact of network failures on geo-distributed services.
- **Determining the pain points in managing network infrastructure**: To achieve high availability, operators need to focus on fixing the most unreliable devices and links in the network. Therefore, we study in detail the failure characteristics of the network core and other network appliances such as middleboxes.
- **Analyzing the effectiveness of network redundancy**: To provide fault tolerance, network elements are typically deployed in 1:1 redundancy inside the network to allow traffic to flow along an alternate route when a device or link becomes unavailable. However, this redundancy comes at a high cost — both financial expenses and management overheads — as the network operators have to now maintain a large number of network devices and links in the datacenter network. A fundamental question that we tackle in this dissertation is to determine whether network redundancy is 100% effective in masking failures and if not, what is the root cause for its ineffectiveness.
- **Importance of operational knowledge in understanding root causes of network failures**: Network trouble tickets form an importance piece of operational knowledge inside datacenter networks and are diaries written by network operators to document the steps they are taking towards debugging a problem. While there is rich information available within these trouble tickets pertaining to the underlying network (e.g., root causes of network failures), this information

is hidden within natural text and therefore, it becomes challenging to extract anything meaningful. This dissertation describes the design and implementation of a *practical* system that combines natural language processing, knowledge representation, and ontology modeling to produce concise representation of trouble tickets which can then be leveraged for problem inference.

Towards addressing problems in the above dimensions, we perform one of the first characteristic studies of cloud network failures from a service perspective. Our study using real-world data focuses on understanding the failure modes and correlation of network failure logs and how they impact cloud services. Specifically, we take an operator-centric approach in this work and aim to answer the following questions:

Q1 **Failure Characterization**: What are the failure characteristics of the network core and other network appliances such as middleboxes inside a datacenter? What are the failure characteristics of the network stamp (set of all network elements rooted at a pair of Layer-3 Access routers) of a service inside a datacenter as well as across geo- distributed datacenters? What device types fail the most? How long does it take to repair their failures? How often do they fail?

Q2 **Failure Modeling**: How to model failures of network components? Are failures recurrent? Are they bursty? Are device-level repairs effective? Is fiber length correlated to failures?

Q3 **Effectiveness of Network Redundancy**: How effective are redundancy mechanisms in handling intra- and inter-datacenter network failures?

Q4 **Network stamp availability of a service**: How many independent network stamps are needed to meet an a service-level-agreement (SLA) (e.g., to achieve certain guarantees on the effectiveness of network redundancy) of a cloud service?

Q5 **Capacity vs. Availability**: For commodity Layer-2 switches, how do their port capacity in terms of connected devices affect their availability in operation?

Q6 **Exploiting Operational Knowledge**: How to extract the root causes of network failures from the operational knowledge available within datacenters? How can we learn from our mistakes to improve network management?

1.2 Contribution

In this dissertation, we focus on *big data analytics* in cloud datacenters to gain actionable insights to address a broad class of real-world problems outlined in Section 1.1. The data in these systems comprises a variety of structured information sources such as network event logs and performance counters, and unstructured text such as trouble tickets and security incident reports. To make actionable decisions, however, we face several key challenges.

C1 **Structured data is noisy.** Network event data is very high volume spanning tens of thousands of switches and routers across geo-distributed datacenters, and noisy due to false alarms and redundant events making it hard to study device or link failure characteristics.

C2 **Unstructured data is ambiguous.** The unstructured data within network trouble tickets is descriptive and ambiguous: it has domain-specific words and synonyms mixed with regular dictionary words, spelling and grammar errors, and writings from different network operators.

C3 **Deriving actionable insights requires field-knowledge.** Measuring and analyzing the data logs is not enough. We need to derive implications from the analysis so that operators can incorporate them in their day-to-day operations.

Towards addressing these challenges, this dissertation makes the following contributions:

1. **Analysis Methodology for Structured Data:** Our dataset includes multiple sources of structured network telemetry data spanning three years logged in monitoring servers of a large cloud provider comprising 100k+ servers, 10k+ core

network devices, 2k+ middleboxes and 100k+ network links across 10+ datacenters. This data covers a wide range of network data sources, including syslog and SNMP alerts, maintenance tracking and revision control system, and traffic carried by links. To this end, we describe a systematic methodology for analyzing this structured data based on event processing to extract events having service-level impact.

2. **Analysis Methodology for Unstructured Data:** Our dataset also includes an important piece of operational knowledge — network trouble tickets. Network trouble tickets are diaries written by operators to keep track of their troubleshooting efforts while fixing a problem. To this end, we take a *practical* step towards automatically analyzing natural language text in network tickets to infer the problem symtoms, troubleshooting activities and resolution actions. Our system, NetSieve combines statistical natural language processing (NLP), knowledge representation, and ontology modeling to achieve these goals. To cope with ambiguity in free-form text, NetSieve leverages learning from human guidance to improve its inference accuracy.
3. **Data-Drive Approaches to Deriving Actionable Insights:** Our overarching goal is to enable operators to understand global problem trends instead of making decisions based on isolated incidents. Towards this, we first apply our methodology for structured and unstructured data to obtain a relatively noiseless data. Next, we outline several analyses rooted in reliability analysis and applied statistics for characterizing network failures and deriving actionable insights from them.

This dissertation reveals many key findings that can provide useful guidelines to improve network reliability for geo-distributed services. Our major findings are:

1. **Characteristics of network failures**: Network failures cause significant impact to cloud services, dominated by connectivity loss problems (70%) and service errors (43%) due to Intra-Datacenter and Inter-Datacenter network problems, respectively. Middlebox failures prominent and can greatly degrade service per-

formance and availability. Network device failures, in general, are not memoryless and exhibit the “few bad apples” effect.

2. **Network redundancy**: Network redundancy in core devices (e.g., Access routers, Aggregation switches, Top-of-rack switches etc.) is ineffective in 40% of the cases. In cases of middleboxes, network redundancy is ineffective in 33% of the cases for firewalls and load balancers; nearly 100% effective for Intrusion Prevention and Detection Systems and VPN devices. In addition, we observed Network redundancy to be least effective at the Access router-Aggregation switch layer and to be most effective at the Inter-datacenter level.
3. **Availability of network domains:** We find that a service hosted on a single large (L2) network domain risks low reliability. We make the empirical observation that the number of independent network domains for a desired network redundancy effectiveness SLA of 3 9’s is three and for 4 9’s is four.
4. **Causes of network failures:** We found that most failures are grey, dominated by connectivity errors and link flaps that exhibit intermittent connectivity; fail-stop device failures occur, but they are not as often. Hardware faults, misconfigurations, and overload problems are present, they are not in majority in contrast to common perception [19]. We also observed a variety of misconfigurations such as incorrect rules, VLAN misallocation and mismatched keys.

1.3 Real World Impact

As part of this dissertation, we have built a broad range of systems including real-time network dashboards, a big data analytics system for analyzing network telemetry data, and an inference tool for root cause analysis in network troubleshooting. Several components of the dissertation work either have undergone a tech-transfer or are being used by multiple business groups inside Microsoft. NetWiser, a Microsoft Research project entailing this dissertation work, was awarded the *Microsoft Trustworthy Computing Reliability Award* for 2013.

The problem inference system part of this dissertation, NetSieve, is currently being used across different teams within Microsoft to improve network management: the Network Architecture team for comparing device reliability across platforms and vendors, Capacity Planning for understanding why network redundancy is ineffective in masking failures, and Incident Management and Operations team for finding the top-k problems and failing components while troubleshooting devices and determining if past repairs were effective. Since its inception, NetSieve has also been used to automate root cause analysis of security incidents within Microsoft's datacenters and recently found its way into commercial use through Microsoft's System Center Advisor (http://www.systemcenteradvisor.com).

1.4 Dissertation Outline

The rest of this dissertation is organized as follows. Chapter 2 provides an overview of the datacenter network architecture and long-haul links connecting geo-distributed datacenters. Chapter 3 describes a methodology based on event processing for filtering network events that have caused service-level impact. In Chapter 4, we propose a new practical approach towards finding the root causes of network failures using unstructured data such as network trouble tickets and describe the design of a system that analyzes natural language text in network tickets to infer the problem symptoms, troubleshooting activities and resolution actions. We conduct a detailed reliability analysis of datacenter networks in Chapter 5. Specifically, we analyze the failure characteristics of several types of core network elements and middleboxes and provide insights into the effectiveness of network redundancy in masking failures and causes behind network failures. We discuss related work in Chapter 6 and conclude in Chapter 7.

2 DATACENTER ARCHITECTURES

In this chapter, we present an overview of the datacenter network architecture and long-haul links connecting geo-distributed datacenters.

2.1 Intra-Datacenter Topology

A datacenter network is typically set up as a multi-root spanning tree topology comprising different types of devices such as routers, switches, load balancers, and firewalls. Figure 2.1 illustrates an example topology of a datacenter network based on the functional separation of Layer-2 (trunking, VLANs, etc.) and Layer-3 (routing) responsibilities. Top-of-Rack (ToR) switches connect servers hosting applications via 10/100/1000 Ethernet links, with the uplinks being either 1GE or 10GE ports. The ToRs are connected upstream to Aggregation switches (AGG) which serve as an aggregation point for the Layer-2 traffic. Traffic from AGGs is forwarded to Access routers (ARs) that use Virtual Routing and Forwarding (VRF) to create a virtual, Layer-3 environment for each tenant. The ARs aggregate traffic from up to several thousand servers and route it to core routers that connect to other datacenters and the Internet. To provide fault tolerance, network devices are typically deployed in 1:1 redundancy pairs or larger groups.

Figure 2.1 illustrates the placement[1] of middleboxes such as firewalls, IDPS, load balancers, and VPNs. Firewalls (FW) protect applications from unwanted traffic (e.g., DoS attacks) by examining packet fields at IP, transport and application layers against a specified set of rules. Network-based IDPS devices analyze traffic to safeguard against attacks such as malware and intrusions. Typically, hardware FWs and IDPSes are deployed to handle high traffic rates [21]. Load Balancers (LB) distribute

[1]Other configurations are possible e.g., firewalls at the network edge.

application requests across hosted servers. Redundant pairs of LBs typically connect to each aggregation switch (AGG) and perform mapping between static (exposed to clients through DNS) and dynamic IP addresses of the servers. Recent hardware LBs also provide other features such as NAT, SSL acceleration, and data caching. VPNs are typically used to facilitate secure remote access for web and client/server applications. Some commercial offerings such as Cisco Catalyst 6509-E provide VPN functionality through IPSec and SSL. Due to the low population of other middleboxes such as NATs and media converters, our study did not include them.

A **network stamp** (shown in dashed red in Figure 2.1) is the set of all network elements that are rooted at a pair of Layer-3 ARs, comprising multiple Layer-2 AGG domains in the underlying subtrees.

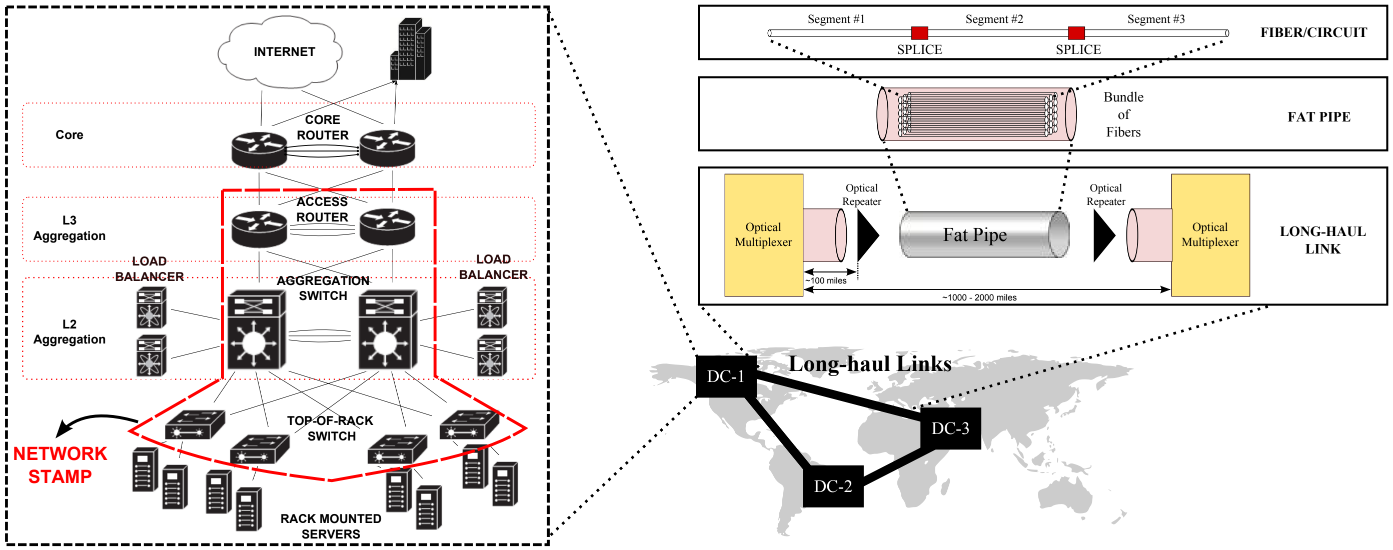


Figure 2.1. Example of a datacenter network architecture. Long-haul links (typically, optical fibers) connect geo-graphically distributed datacenters and can span thousands of miles.

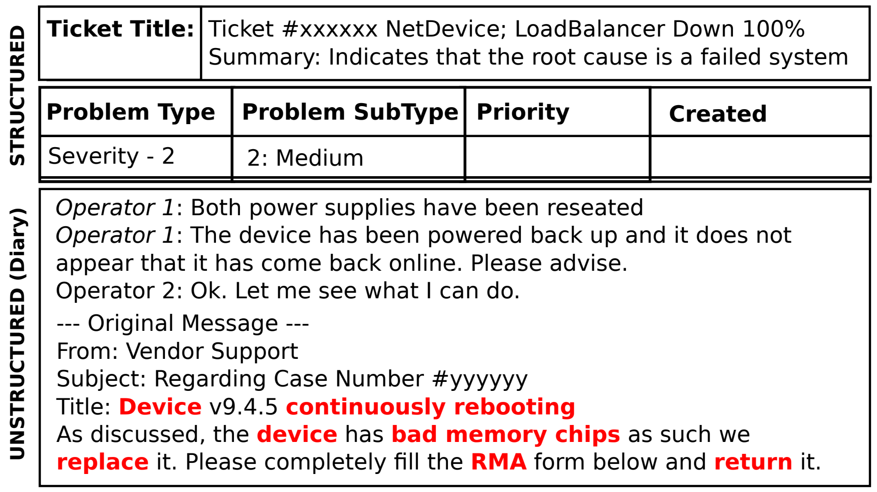
2.2 Inter-Datacenter Connectivity

Geographically distributed datacenters are connected to each other and to the Internet typically using long-haul WDM (wave division multiplexing) optical transport networks spanning about 3000 miles between two endpoints. WDM is usually operated either in a coarse (CWDM) or dense (DWDM) multiplexing manner. While the former utilizes multiple wavelengths spaced at 20nm and operates in the 1271-1611nm spectrum range, the latter utilizes many wavelengths spaced narrowly at 0.8nm and operates in the 1530-1565nm spectrum range (C-band). Modern coherent receivers use polarization multiplexed quaternary phase shift keying (PM-QPSK) modulation and can achieve 100Gbps transmission on 50GHz ITU channel grid, and a total capacity of 8Tbps in the C-band.

Long-haul fiber resources are scarce, expensive and time consuming to construct as well as to fix as engineers may have to travel to the remote physical location. As shown in Figure 2.1, unlike traditional telecommunication networks that require a lot of intermediate add/drop points (e.g., optical multiplexers, signal repeaters), Inter-Datacenter links are mostly point-to-point fat pipe connections with few intermediate add/drops. Fat pipes contain multiple *segments* spliced together to form an optical *circuit* also known as a long-haul fiber.

2.3 Operational Knowledge

Our focus is on datacenter environments comprising thousands of routers and switches. In this setup, when a link connecting two devices goes down, an event is logged by a network monitoring system which, in turn, triggers an alarm to an Operator Console, monitored by network operators. To track network troubleshooting and maintenance dealing with these alarms, network operators typically deploy a trouble ticket system which logs all the steps from opening a ticket (e.g., customer complaint, SNMP alarm) till its resolution [22].


STRUCTURED
UNSTRUCTURED (Diary)

Ticket Title:	Ticket #xxxxxx NetDevice; LoadBalancer Down 100% Summary: Indicates that the root cause is a failed system

Problem Type	**Problem SubType**	**Priority**	**Created**
Severity - 2	2: Medium		

Operator 1: Both power supplies have been reseated
Operator 1: The device has been powered back up and it does not appear that it has come back online. Please advise.
Operator 2: Ok. Let me see what I can do.
--- Original Message ---
From: Vendor Support
Subject: Regarding Case Number #yyyyyy
Title: **Device** v9.4.5 **continuously rebooting**
As discussed, the **device** has **bad memory chips** as such we **replace** it. Please completely fill the **RMA** form below and **return** it.

Figure 2.2. An example network trouble ticket.

Trouble tickets comprise two types of fields: (a) structured data often generated automatically by alarm systems such as ticket id, time of alert, and syslog error, and (b) free-form text written by operators to record the diagnosis steps and communication (e.g., via IM, email) with the customer or other technicians while mitigating the problem. Even though the free-form field is less regular and precise compared to the fixed text, it usually provides a *detailed* view of the problem: what happened? what troubleshooting was done? and what was the resolution? Figure 2.2 shows a ticket describing continuous reboots of a load balancer even after reseating its power supply units; bad memory as the root cause; and memory replacement as the fix; which would be hard to infer from coarse- grained fixed data.

3 ANALYSIS METHODOLOGY FOR STRUCTURED DATA

Our dataset includes multiple sources of network failure data spanning three years logged in monitoring servers of a large cloud provider comprising 100k+ servers and 10k+ Layer-2 and Layer-3 devices across 10+ datacenters. These datacenters host a variety of applications ranging from customer facing ones such as web services, video streaming, data stores, and enterprise applications to data intensive applications such as search indexing and MapReduce jobs.

Defining a failure is challenging due to three key reasons:

1. **Redundancy**: The presence of redundancy at different layers in the network and application stack makes it difficult to assess the impact of a failure. For instance, even if network redundancy could not mask a failure, application-level redundancy could have masked it. However, determining if a failure has been masked across different layers requires that we have access to both network- and application-level logs.
2. **Diversity of failures**: The presence of different kinds of failures such as fail-stop and grey makes it difficult to classify an event as a failure. While fail-stop failures are relatively easier to detect, we observed in practice that grey failures (i.e., short-lived failures) can be among the most difficult types of failures to diagnose. The device or link is seemingly working but its output contains more errors than usual.
3. **Diversity of devices**: Precisely defining a failure is complicated because its impact is dependent on the device type. For instance, a faulty load balancer or a router may cause a loss in throughput while a misconfigured firewall may incorrectly forward or drop legitimate traffic.

To address these challenges, we consider all events logged by a network monitoring system inside the datacenters as failures that can either cause (i) traffic impact (in terms of loss in traffic volume) or (ii) functional impact (device functions incorrectly in routing or processing traffic). For the former, we leverage the network traffic logs to estimate loss in traffic volume per failure event. For the latter, we extract functional impact based on problems observed in network trouble tickets associated with the failure event.

While these definitions are simple and intuitive, there are several key challenges in utilizing the network event logs for studying device/link failure characteristics:

1. How can we differentiate noisy events from *meaningful* failures? For instance, syslog messages can be spurious where a device may log multiple 'down' events (even when it is operational) or when neighbors may send redundant notifications for the same device. Similarly, how to handle link flaps generating multiple and possibly overlapping down/up messages which get logged as separate events?
2. Is it accurate to consider all failures to characterize reliability? In particular, some down events are expected due to routine and scheduled maintenance e.g., code update. Operators aim to limit the planned downtime, and at the same time, prioritize detection and mitigation of unexpected failures. While prior efforts [20, 23] did not differentiate them, "lumping" all events together risks inflating failure rates.
3. How to handle *redundant failures* from devices that keep logging errors while undergoing repairs or being scheduled for replacement? Figure 3.1 shows the CCDF for different types of LBs in our dataset with some devices logging more than 1000 down messages over a few hours as the monitoring system did not suppress them during triage. Such events may bias measurement of device failures.

In this chapter, we outline several techniques rooted in applied statistics to extract events causing service-level impact.

Chapter Organization: The rest of this chapter is organized as follows. Section 3.1 describes the multiple sources of data collected by network operators, comprising our large-scale dataset spanning three years (July 24, 2010 - June 24, 2013). Section 3.2

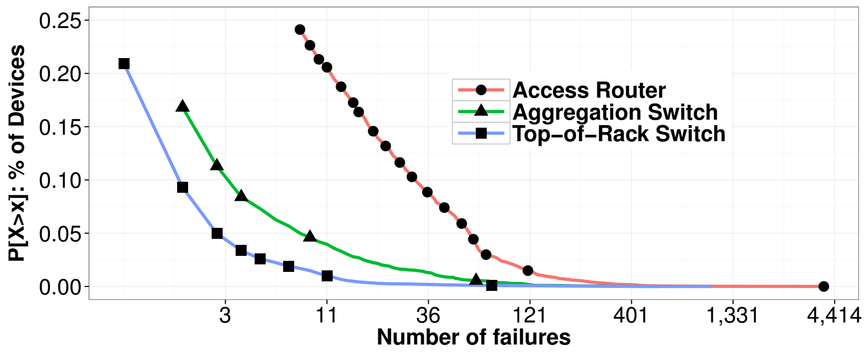


Figure 3.1. Complementary cumulative distribution (CCDF) plot of the number of failures logged by devices. The tail indicates devices that log thousands of failures.

describes a systematic methodology based on event processing to extract failures causing service-level impact. Finally, Section 3.3 outlines our validation methodology.

3.1 Network Datasets

Network Event Logs: Network failures are typically detected from monitoring alarms such as syslog and SNMP traps and tracking the health of each device/link via ping and SNMP polling. These logs contain information about the network element experiencing the event, the event type, the other end- point of this device/link, and a short machine-generated description of the event.

High Severity Incidents: To analyze impactful incidents where service outages occur and customers get impacted, operators keep details of each high severity incident. Similar to the trouble ticket data, each high severity incident has a unique ticket identifier and contains both structured and unstructured information which we leverage for problem inference. We use this dataset over a period of five years (2008-13).

Trouble Tickets: To track network faults during troubleshooting, a ticketing system is used typically based on the NOC RFC [22]. This system coordinates tasks among network engineers working on an incident. Tickets have a unique identifier and contain both structured information about the failure (such as when and how a failure was discovered) and a diary of steps taken by operators to resolve the problem. As we will empirically validate in Chapter 4, we cannot use the structured information inside network trouble tickets for any problem inference due to their high inaccuracy. Therefore, we leverage NetSieve [24] on the network support tickets to infer root causes for (1) high severity incidents and (2) failure events including maintenance-related network changes.

Maintenance Data: To track activities such as device repairs/provisioning, configuration changes, and software upgrades throughout the network, operators use a maintenance tracking and revision control system. It serves as a repository of syslog information and includes comments from network engineers about when and why changes were performed. Before debugging an outage, an engineer checks this repository for on-going maintenance and verifies any recent changes to the device configuration. We also obtained maintenance tracking information for inter-datacenter long-haul links where the operators recorded the expected duration of a fiber or segment to be down. To avoid skewing the failure distributions due to maintenance events, we compute the downtime of devices/links by removing the downtime from planned changes.

Network Traffic Data: We utilize traffic averages observed every five minutes on network interfaces logged using SNMP [25] polling. Traffic monitoring systems use the MIB [26] format to store the data that includes fields such as the interface type (token ring, ethernet etc.), other end of the interface, interface status (up/down), number of bytes sent/received. We correlate this traffic data with failure events to extract failures impacting network traffic, and to reverse-engineer the topology using active link-level connectivity.

3.2 Analysis Methodology

We build upon the methodology of Turner et al. [18] and Gill et al. [20] on utilizing low-level network events, but differentiate in four important ways. First, we apply a pipeline of event processing steps to analyze and correlate network event data sources (Figure 3.2). Second, we identify redundant events and analyze how they contribute towards measurement noise. Third, we remove events generated due to inactive links and planned maintenance to identify unexpected failures. Finally, we extract *impactful failures* by correlating network events with traffic loss, and infer their problem root causes from trouble tickets. Figure 3.2 shows the effectiveness of each processing step.

Step 1: The goal of the first step is to fix various timing inconsistencies. First, it groups all events with the same start and end time originating on the same interface with the same event description (thereby removing duplicate events). Next, by picking the earliest start and end times, multiple events within a 60 second time window on the same interface are grouped into a single event. This is done to avoid problems due to clock skews and log buffering. Finally, if two events originating on the same interface contain the same event description and have the same start time but different end times, they are grouped into a single event and assigned the earlier of the end times. We take the earliest end times as events may not be marked as cleared long after their resolution.

Step 2: The second step filters all planned network changes based on a maintenance tracking system. Each network change is annotated with the time window, the device name and the type of maintenance being carried out. Network operators likely have a good understanding of problems being handled by scheduled maintenance and thus, we focus on analyzing device and link-level reliability due to unexpected outages.

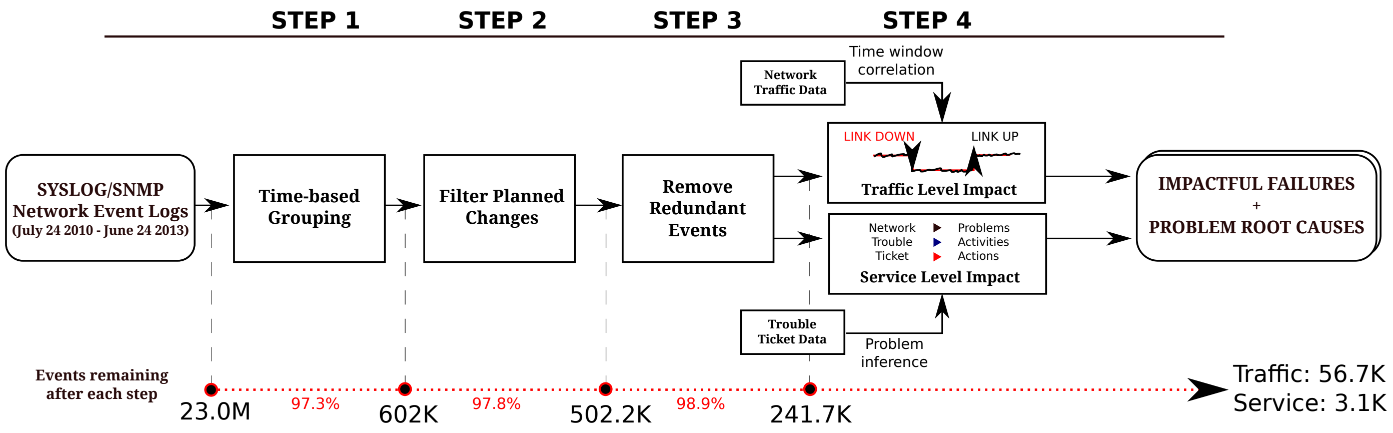


Figure 3.2. A pipeline of event processing steps to analyze and extract impactful failures from network data sources.

Step 3: The third processing step removes redundant events due to devices that continue logging error messages when they are being troubleshooted or where the events were not suppressed even after the problem had been identified. Figure 3.1 shows the CCDF plot for different types of devices in our dataset. Observe that a small fraction of devices log up to thousands of failure events. To filter them, we apply the following technique based on discussion with operators: merge all events that have the same ticket identifier as events with the same ticket ID are likely to have the same symptoms.

Step 4: The final step aims to identify events causing service impact based on two rules: the event caused (i) loss of traffic (i.e., a drop in the median traffic on the device/link during a failure compared to its median value in the recent past e.g., preceding 2-hour window), or (ii) *noticeable* application-level impact.

A. Traffic Impact: Since the network is shared and we did not have access to application logs, we estimate the failure impact by leveraging network traffic data and computing the ratio of the median traffic on a failed device/link during a failure and its value in the recent past (preceding a two hour window): a failure has traffic impact if this ratio is less than one [20]. Note that this filter is applied *after* a network alarm has been logged and a ticket opened to resolve the issue. Hence, it will not misclassify failures due to the inherent time-varying nature of traffic e.g., time of day effects. We perform hypothesis testing to validate this approach. Specifically, we use the Mann-Whitney-Wilcoxon (MWW) [27] test for validation, which is a non-parametric significance test to compare two populations. The *null hypothesis* to test is that the traffic values observed before a failure and during a failure have identical distribution functions. We randomly sampled 7k middlebox failure events and 7k time points from periods during which no known failures occurred. For the former, we obtained traffic values up to eight hours before the failure and during the failure, and up to eight hours before the event and up to two hours afterwards for the latter.

Figure 3.3 plots the distribution of the p-value to quantify the fraction of cases where we can reject the null hypothesis at a significance level of 0.05. The top figure shows that in 99.3% of the failures, p-value is less than 0.05; the remaining cases had insufficient traffic data points (e.g., due to best-effort logging). Hence the null hypothesis can be rejected in 99.3% of the cases. Similarly, the bottom figure shows that the distributions are identical during periods when no failures occurred. Further, a time window size of two hours and eight hours of traffic before the event yields the highest and lowest accuracy, respectively. Indeed, under more strict assumptions, a significant MWW test can be interpreted as showing a difference in medians [28]. Gill et al. [20] also correlated failures with traffic deviations, but it did not evaluate its accuracy or specify the time window size before a failure to compute the traffic ratio with highest accuracy. As Figure 3.3 (top) shows, the accuracy is clearly dependent on the time window size.

B. Device Malfunction Impact: To measure impact due to incorrect device function, we leverage the information logged by operators in trouble tickets to determine the problem root causes when middleboxes fail. Specifically, we apply NetSieve [24], an automated problem inference system that analyzes the free-form text in a trouble ticket to generate its synopsis: (1) the problems observed e.g., link down, misconfigured rule, (2) troubleshooting performed e.g., check configuration, verify routes, and (3) actions taken for resolution e.g., replaced line card, cleaned fiber.

3.3 Validation

We performed ground truth validation to evaluate the fidelity of our failure analysis methodology. Specifically, we validate our methodology along two dimensions: (1) accuracy i.e., are all the processed events actionable? and (2) completeness i.e., did it miss any events from the ground truth data?

Our approach towards validation of our event processing pipeline is as follows (see Figure 3.5): For the former, we ensure that our result set includes *all* events deemed

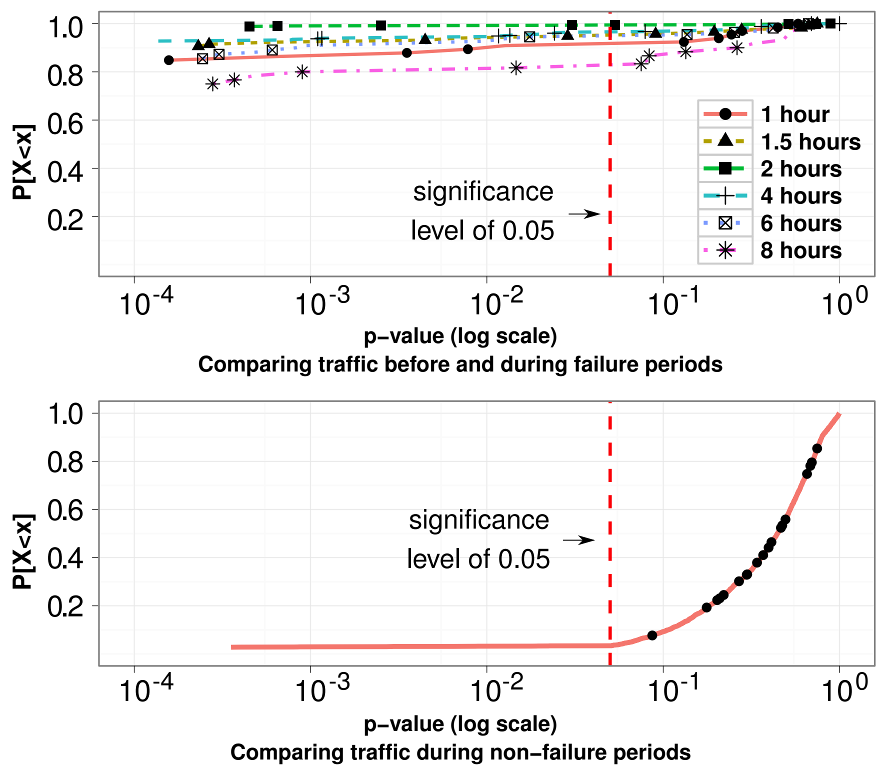


Figure 3.3. MWW test: p-value < 0.05 in 99.3% of the failures (top), and p-value > 0.05 in 99.2% of the events during no failure periods (bottom). Hence the null hypothesis can be rejected.

"actionable" by operators — we can recognize these actionable events by verifying if an operator attached a trouble ticket to it implying that the event was troubleshooted. More precisely:

- Take all the events X that have a ticket associated with them *i.e.*, the network operators have already marked these events as *actionable*. We assume that any event not in the set X was not actionable (e.g., no visible impact by the network analyst or an end-user) and hence there are no *false negatives* in our case.

- Compute the set of events Y which have had some form of impact (e.g., traffic impact) from the raw events being generated by the network monitoring system.

- Verify that $Y \subseteq X$

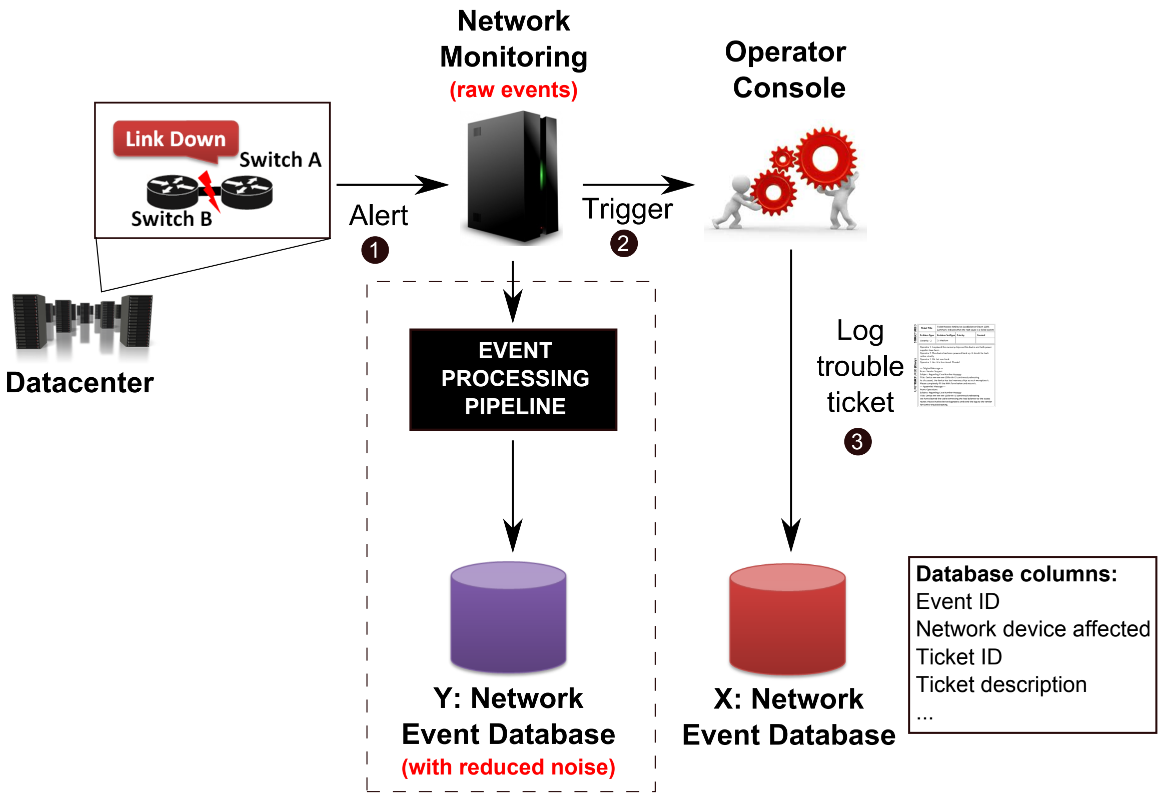


Figure 3.4. Our goal is to attach the *event processing pipeline* to the raw events getting generated by the network monitoring system so as to minimize the burden on the operator.

Note: Quantifying $X — Y$ (which could be construed as *false positives* in this case) is subjective because a network operator may not take an action (e.g., network trouble ticket is empty) if the problem is short-lived or a grey failure e.g., link congestion, link flapping, device reboot due to a failed assertion. Therefore, we verify that $X' = Y$ where $X' \subseteq X$ and X' contains all events which are not grey failures.

As an additional step of validation, we leverage the high severity incident database described in *Section* 1. Because each such incident caused a service impact where the network redundancy was ineffective, we use this database as the ground truth. We compared our result set against the high severity incidents, and verified that (1) none of the events from this incident list were missed (i.e., no false negatives) and (2) in each case, network redundancy was in fact, unsuccessful in masking the failure.

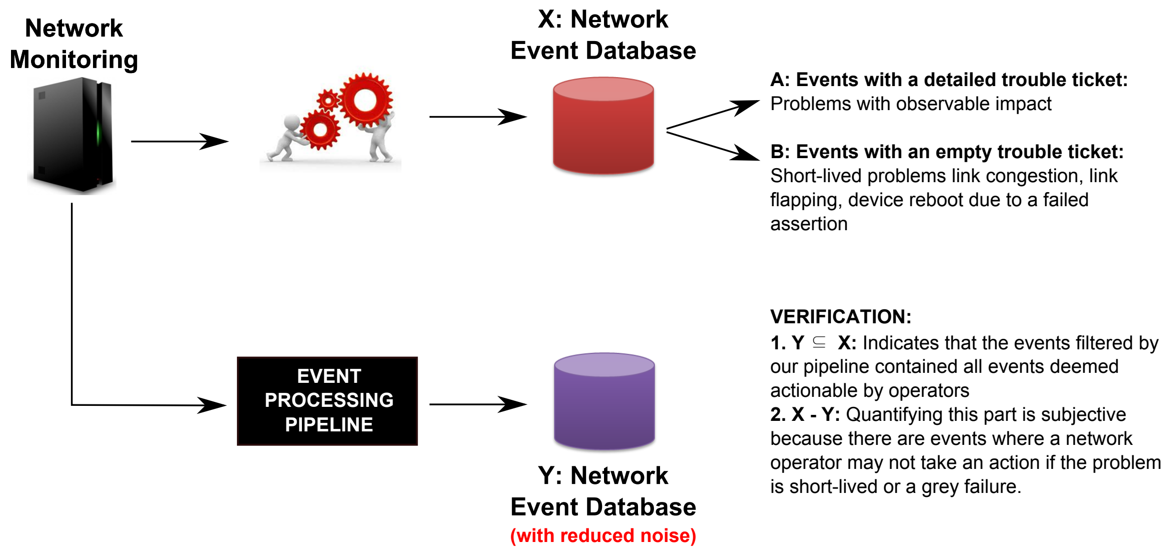

Network
Monitoring
X: Network
Event Database
A: Events with a detailed trouble ticket:
Problems with observable impact
B: Events with an empty trouble ticket:
Short-lived problems link congestion, link flapping, device reboot due to a failed assertion
EVENT PROCESSING PIPELINE
Y: Network
Event Database
(with reduced noise)
VERIFICATION:
1. Y ⊆ X: Indicates that the events filtered by our pipeline contained all events deemed actionable by operators
2. X - Y: Quantifying this part is subjective because there are events where a network operator may not take an action if the problem is short-lived or a grey failure.

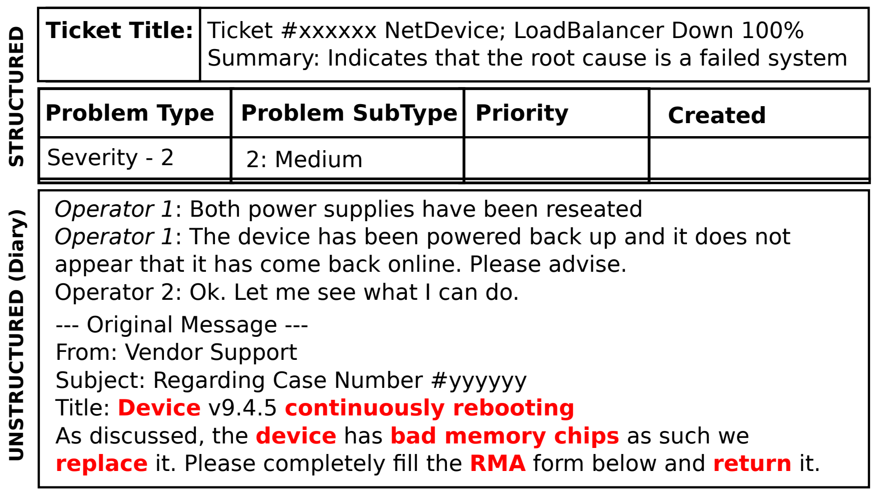
Figure 3.5. Verifying the events filtered by the *event processing pipeline*

3.4 Summary

In large-scale distributed systems, failures become a norm rather than an exception. Using several network data sources, this chapter presented (i) techniques rooted in applied statistics to achieve noise reduction and (ii) methodology for correlating different source of data. In Chapter 5, we will leverage this data to derive actionable insights that network operators can incorporate into their day-to-day operations.

4 ANALYSIS METHODOLOGY FOR UNSTRUCTURED DATA

Network failures are a significant contributor to system downtime and service unavailability [18, 20, 29]. To track network troubleshooting and maintenance, operators typically deploy a trouble ticket system which logs all the steps from opening a ticket (e.g., customer complaint, SNMP alarm) till its resolution [22]. Trouble tickets comprise two types of fields: (a) structured data often generated automatically by alarm systems such as ticket id, time of alert, and syslog error, and (b) free-form text written by operators to record the diagnosis steps and communication (e.g., via IM, email) with the customer or other technicians while mitigating the problem. Even though the free-form field is less regular and precise compared to the fixed text, it usually provides a *detailed* view of the problem: what happened? what troubleshooting was done? and what was the resolution? Figure 4.1 shows a ticket describing continuous reboots of a load balancer even after reseating its power supply units; bad memory as the root cause; and memory replacement as the fix; which would be hard to infer from coarse-grained fixed data.



STRUCTURED

Ticket Title:	Ticket #xxxxxx NetDevice; LoadBalancer Down 100% Summary: Indicates that the root cause is a failed system		
Problem Type	**Problem SubType**	**Priority**	**Created**
Severity - 2	2: Medium		

UNSTRUCTURED (Diary)

Operator 1: Both power supplies have been reseated
Operator 1: The device has been powered back up and it does not appear that it has come back online. Please advise.
Operator 2: Ok. Let me see what I can do.
--- Original Message ---
From: Vendor Support
Subject: Regarding Case Number #yyyyyy
Title: **Device** v9.4.5 **continuously rebooting**
As discussed, the **device** has **bad memory chips** as such we **replace** it. Please completely fill the **RMA** form below and **return** it.

Figure 4.1. An example network trouble ticket.

Unfortunately, while tickets contain valuable information to infer problem trends and improve network management, mining them automatically is extremely hard. On one hand, the fixed fields are often inaccurate or incomplete [30]. Our analysis (Section 4.1.1) on a large ticket dataset shows that the designated problem type and subtype fields had incorrect or inconclusive information in 69% and 75% of the tickets, respectively. On the other hand, since the free-form text is written in natural language, it is often ambiguous and contains typos, grammatical errors, and words (e.g., "cable", "line card", "power supply") having domain-specific meanings different from the dictionary.

Given these fundamental challenges, it becomes difficult to automatically extract *meaning* from raw ticket text even with advanced NLP techniques, which are designed to process well-written text (e.g., news articles) [31]. Most prior work on mining trouble tickets use either keyword search and manual processing of free-form content [32–34], predefined rule set from ticket history [35], or document clustering based on manual keyword selection [30]. While these approaches are simple to implement and can help narrow down the types of problems to examine, they risk (1) inaccuracy as they consider only the presence of a keyword regardless of where it appears (e.g., "do not replace the cable" specifies a negation) and its relationship to other words (e.g., "checking for maintenance" does not clarify whether the ticket was *actually* due to maintenance), (2) a significant human effort to build the keyword list and repeating the process for new tickets, and (3) inflexibility due to pre-defined rule sets as they do not cover unexpected incidents or become outdated as the network evolves.

Table 4.1
Examples of network trouble tickets and their inference output from NetSieve.

		Inference output from NetSieve		
	Ticket Title	**Problems**	**Activities**	**Actions**
1	SNMPTrap LogAlert 100%: Internal link 4.8 is unavailable.	link down, failover, bad sectors	swap cable, upgrade fiber, run fsck, verify HDD	replace cable, HDD
2	HSRPEndpoint SwitchOver 100%: The status of HSRP endpoint has changed since last polling.	firmware error, interface failure	verify and break-fix supervisor engine	replace supervisor engine, reboot switch
3	StandbyFail: Failover condition, this standby will not be able to go active.	unexpected reboot, performance degraded	verify load balancer, run config script	rma power supply unit
4	The machine can no longer reach internet resources. Gateway is set to load balancer float IP.	verify static route	reboot server, invoke failover, packet capture	rehome server, reboot top-of-rack switch
5	Device console is generating a lot of log messages and not authenticating users to login.	sync error, no redundancy	power down device, verify maintenance	replace load balancer
6	Kernel panic 100%: CPU context corrupt.	load balancer reboot, firmware bug	check performance, break-fix upgrade	upgrade BIOS, reboot load balancer
7	Content Delivery Network: Load balancer is in bad state, failing majority of keep-alive requests.	standby dead, misconfigured route	upgrade devices	replace standby and active, deploy hot-fix
8	OSPFNeighborRelationship Down 100%: This OSPF link between neighboring endpoints is down.	connectivity failure, packet errors	verify for known maintenance	replace network card
9	HighErrorRate: Summary: http://domain/characteristics.cgi?<device>.	packet errors	verify interface	cable and xenpak module replaced
10	AllComponentsDown: Summary: Indicates that all components in the redundancy group are down;	down alerts	verify for decommissioned devices	decommission load balancer

In this chapter, we present NetSieve, a problem inference system that aims to automatically analyze ticket text written in natural language to infer the problem symptoms, troubleshooting activities, and resolution actions. Since it is nearly impractical to understand any arbitrary text, NetSieve adopts a domain-specific approach to first build a knowledge base using existing tickets, automatically to the extent possible, and then use it to do problem inference. While a ticket may contain multiple pieces of useful information, NetSieve focuses on inferring three key features for summarization as shown in Table 4.1:

1. **Problems** denote the network entity (e.g., router, link, power supply unit) and its associated state, condition or symptoms (e.g., crash, defective, reboot) as identified by an operator e.g., bad memory, line card failure, crash of a load balancer.
2. **Activities** indicate the steps performed on the network entity during troubleshooting e.g., clean and swap cables, verify hard disk drive, run configuration script.
3. **Actions** represent the resolution action(s) performed on the network entity to mitigate the problem e.g., upgrade BIOS, rehome servers, reseat power supply.

To achieve this functionality, NetSieve combines techniques from several areas in a novel way to perform problem inference over three phases. First, it constructs a domain-specific knowledge base and an ontology model to interpret the free-form text using pattern mining and statistical NLP. In particular, it finds important domain-specific words and phrases (e.g., "supervisor engine", "kernel", "configuration") and then maps them onto the ontology model to specify relationships between them. Second, it applies this knowledge base to infer problems, activities and actions from tickets and exports the inference output for summarization and trend analysis. Third, to improve the inference accuracy, NetSieve performs incremental learning to incorporate human feedback.

Our evaluation on 10k+ network tickets from a large cloud provider shows that NetSieve performs automated problem inference with 89%-100% accuracy, and several network teams in that cloud provider have used its inference output to learn global problem trends: (1) compare device reliability across platforms and vendors, (2) analyze cases when network redundancy failover is ineffective, and (3) prioritize checking for the top-k problems and failing components during network troubleshooting.

This chapter describes the following:

- A large-scale measurement study (Section 4.1) to highlight the challenges in analyzing structured data and free-form text in network trouble tickets.
- Design and implementation (Section 4.2) of NetSieve, an automated inference system that analyzes free-form text in tickets to extract the problem symptoms, troubleshooting activities and resolution actions.
- Evaluation (Section 4.3) of NetSieve using expert review, study with network operators and vendor data, and showing its applicability (Section 5.8) to improve network management.

Limitations: NetSieve is based on analyzing free-form text written by operators. Thus, its accuracy is dependent on (a) fidelity of the operators' input and (b) tickets containing sufficient information for inference. NetSieve leverages NLP techniques, and hence is subject to their well-known limitations such as ambiguities caused by *anaphoras* (*e.g.*, referring to a router as *this*), complex negations (*e.g.*, "device gets replaced" but later in the ticket, the action is negated by the use of an anaphora) and truth conditions (*e.g.*, "please replace the unit once you get more in stock" does not clarify whether the unit has been replaced). NetSieve inference rules may be specific to our ticket data and may not apply to other networks. While we cannot establish representativeness, this concern is alleviated to some extent by the size and diversity of our dataset. Finally, our ontology model represents one way of building a knowledge base, based on discussions with operators. Given that the ticket system is subjective and domain-specific, alternative approaches may work better for other systems.

Chapter Organization: The rest of this chapter is organized as follows. Section 4.1 describes the challenges associated with analyzing unstructured data. In Section 4.2,

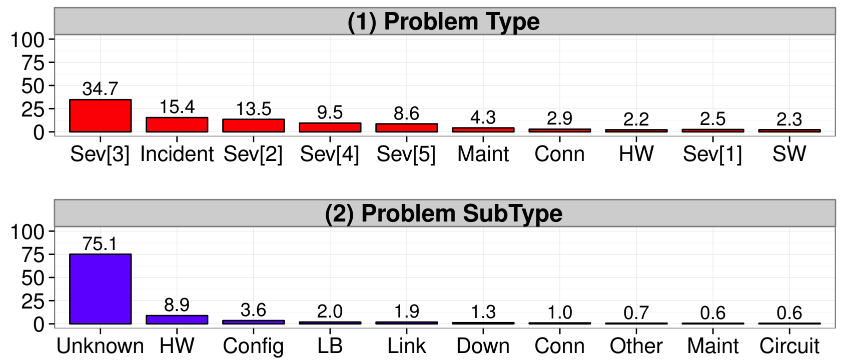


Figure 4.2. Problem types and subtypes listed in the tickets.

we describe the design and implementation of NetSieve that takes a practical step towards automatically analyzing natural language text in network tickets to infer the problem symptoms, troubleshooting activities, and resolution actions. Section 4.3 describes the system evaluation. Finally, we describe the deployment and impact of NetSieve in Section 5.8.

4.1 Challenges in Analyzing Network Trouble Tickets

In this section, we present a measurement study to highlight the key challenges in automated problem inference from network tickets. The dataset comprises 10K+ (absolute counts omitted due to confidentiality reasons) network tickets logged during April 2010-2012 from a large cloud provider. Next, we describe the challenges in analyzing fixed fields and free-form text in trouble tickets.

4.1.1 Challenges: Analyzing Fixed Fields

C1: Coarse granularity. The fixed fields in tickets contain attributes such as 'ProblemType' and 'ProblemSubType', which are either pre-populated by alarm systems or filled in by operators. Figure 4.2 shows the top-10 problem types and sub-types along-with the fraction of tickets. *Sev* denotes problem severity assigned based on SLAs with the customer. We observe that while problem types such as *Software*, *Hardware*, *Maintenance*, and *Incident* provide coarse granularity information about the problem type, other types e.g., *Sev[1-5]* are highly subjective reflecting operator's judgement and they account for 68.8% of the tickets. As a result, these fields are not useful to precisely infer the observed problems.

C2: Inaccuracy or Incompleteness. Figure 4.3 shows the problem categorization for a randomly selected subset of tickets labeled by a domain expert (top) and the field values from the tickets (bottom) for three different types of devices: (1) Access Routers (AR), (2) Firewalls, and (3) Load balancers (LB); the number of tickets is 300, 42, and 299, respectively.

We make two key observations. First, the Problem SubType field (in the bottom row) is *Unknown* in about 79%-87% of the tickets. As a result, we may incorrectly infer that devices failed due to unknown problems, whereas the problems were precisely reported in the expert labeled set based on the same ticket data. Second, the categories annotated by the expert and ticket fields for each device type have little overlap, and even when there is a common category, there is a significant difference in the fraction of tickets attributed to that category e.g., 'Cable' accounts for 0.6% of the LB tickets whereas the ground truth shows their contribution to be 9.7%.

The reason that these fields are inaccurate or incomplete is that operators work under a tight time schedule, and they usually have a narrow focus of mitigating a problem rather than analyzing failure trends. Thus, they may not have the time, may not be motivated, or simply forget to input precise data for these fields after closing the tickets. Further, some fixed fields have a drop-down menu of pre-defined labels and every problem may not be easily described using them.

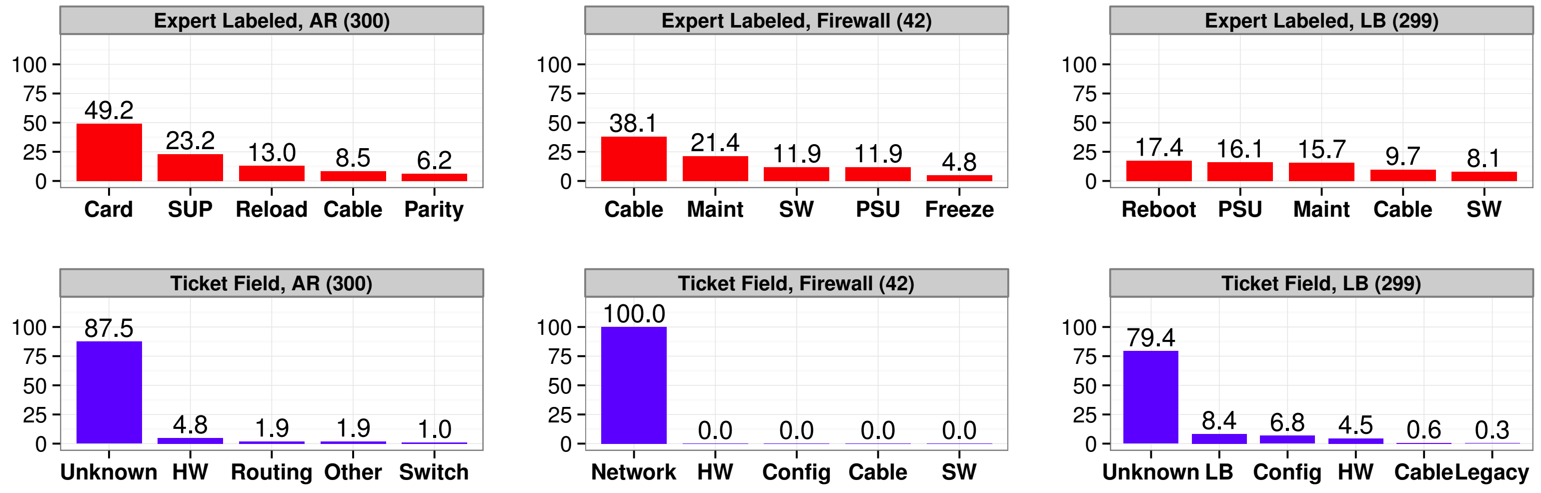


Figure 4.3. Categorization of the 'Problem SubType' field in tickets for (a) Access Routers (AR), (b) Firewalls, and (c) Load balancers (LB). The top and bottom rows show the major problem subtypes as labeled by an expert and the ticket field, respectively.

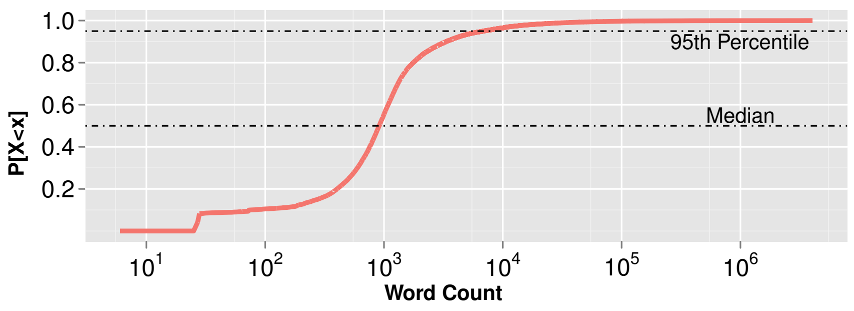


Figure 4.4. Distribution of word count for the ticket dataset.

4.1.2 Challenges: Analyzing Free-Form Text

Figure 4.4 shows the distribution of word count in free-form text across our dataset consisting of 231 million words and 3.2 million sentences. We observe that the median word count is 1,266, but the distribution has a long tail with tickets comprising about a million words. Ziefle et al. [36] indicate that the average reading speed of a human is about 180 words/minute, but they assume that the text is direct, regular and easy-to-understand, without any errors. In comparison to structured data, the free-form text in network tickets is descriptive and ambiguous: it has *domain-specific* words and synonyms mixed with regular dictionary words, spelling and grammar errors, and writings from different operators.

Specifically, we highlight the following challenges in mining free-form text in trouble tickets:

C1: Diversity of content. A ticket may contain a variety of semantic elements such as emails, IMs, device debug logs, devices names, and operator notes.

C2: Domain-specific words. Without a prior list of domain-specific keywords, training spell checkers can be hard e.g., DMZ and DNS are both valid technical keywords, but they cannot be found in the dictionary.

C3: Redundant text. Tickets often contain text fragments that appear with high frequency. We observe three types of frequently occurring fragments (see Figure 4.1): templates, emails and device logs. Templates are text fragments added by operators to meet triage guidelines, but they often do not contain any problem-specific information. Many emails are asynchronous replies to a previous message and thus, it may be hard to reconstruct the message order for inference. Log messages are usually appended to a ticket in progress. Therefore, text mining using metrics such as term frequency may incorrectly give more weightage to terms that appear in these logs.

Overall, these challenges highlight the difficulty in automatically inferring problems from tickets. While we studied only our ticket dataset, our conversation with operators (having a broader industry view and some having worked at other networks), suggests that these challenges are similar to those of many other systems.

4.2 NetSieve Design and Implementation

In this section, we first give an overview of NetSieve and then describe its design and implementation.

4.2.1 Design Goals

To automatically analyze free-form text, NetSieve should meet the following design goals:

1. *Accuracy*: The inference system needs to be accurate as incorrect inference can lead to bad operator decisions, and wasted time and effort in validating inference output for each ticket, thus limiting practicality.
2. *Automation*: Although we cannot completely eliminate humans from the loop, the system should be able to operate as autonomously as possible.
3. *Adaptation*: As the network evolves, the system should be able to analyze new types of problems and leverage human feedback to acquire new knowledge for continuously improving the inference accuracy.
4. *Scalability*: The system should be scalable to process a large number of tickets where each ticket may comprise up to a million characters, in a reasonable time.

5. *Usability*: The output from the inference system should provide a user-friendly interface (e.g., visualization, REST, plaintext) to allow the operator to browse, filter and process the inference output.

4.2.2 Overview

NetSieve infers three key features from network trouble tickets: (1) Problem symptoms indicating what problem occurred, (2) Troubleshooting activities describing the diagnostic steps, and (3) Resolution actions denoting the fix applied to mitigate the problem.

Figure 4.5 shows an overview of the NetSieve architecture. NetSieve operates in three phases. First, the knowledge building phase constructs a domain-specific knowledge base and an ontology model using existing tickets and input from a domain-expert. Second, the operational phase uses the knowledge base to make problem inference from tickets. Third, the incremental learning phase improves the accuracy of knowledge base using human guidance. We next give a brief description of each of these phases.

Knowledge Building Phase: The goal of this phase is to analyze free-form text to extract important domain-specific phrases such as “power supply unit” and “load balancer” using repeated pattern mining (Section 4.2.3) and statistical NLP (Section 4.2.3). These domain-specific phrases are then mapped onto an ontology model (Section 4.2.3) that formally represents the relationships between network entities and stores them in a knowledge base. This phase is executed either when NetSieve is bootstrapped or to re-train the system using expert feedback.

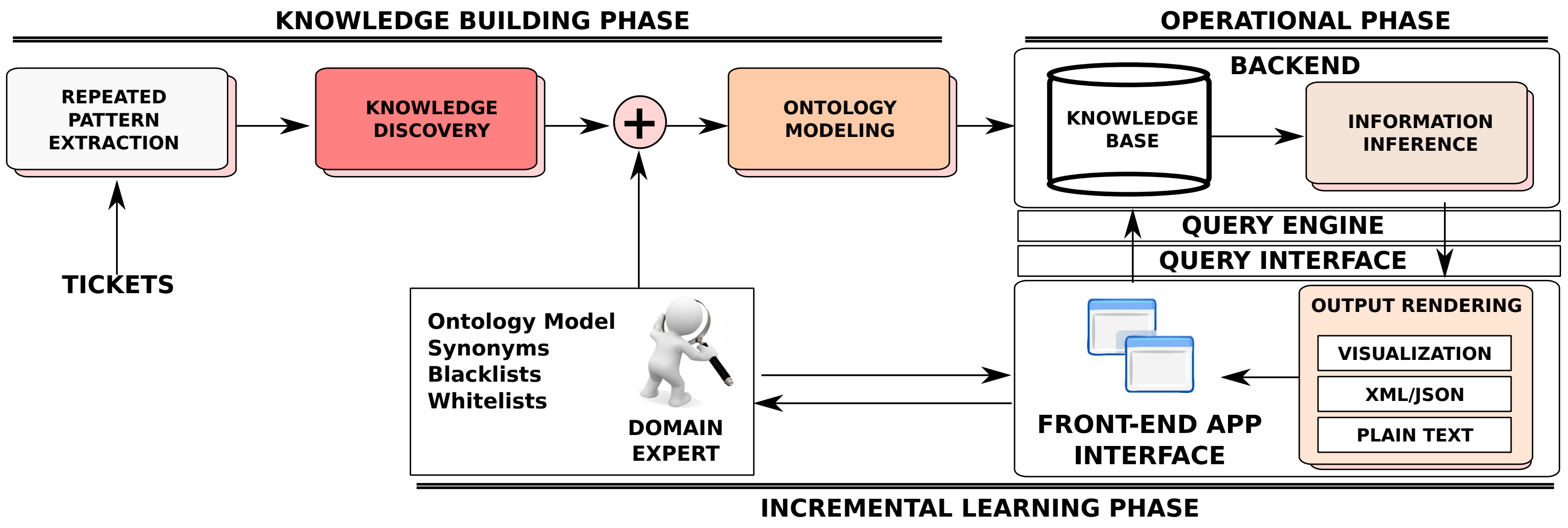


Figure 4.5. NetSieve Architecture: The first phase builds a domain-specific knowledge base using existing tickets. The second phase uses the knowledge base to make problem inference. The third phase leverages human guidance to improve the inference accuracy.

Operational Phase: The goal of this phase is to perform problem inference (Section 4.2.4) from a ticket using the knowledge base. To export the inference output, NetSieve supports SQL (through the *Query Engine*) and HTTP GET requests (through a *Query Interface* such as REST [37]) and outputs results in a variety of data formats such as XML/JSON, and through data visualization for ticket summarization and trend analysis.

Incremental Learning Phase: To improve inference accuracy, it is important to continuously update the knowledge base to incorporate any new domain-specific terminologies. NetSieve provides an interface to allow a domain-expert to give feedback for improving the ontology model, synonyms, blacklists and whitelists. After each learning session, NetSieve performs problem inference using the updated knowledge base.

4.2.3 Knowledge Building Phase

Building a domain-specific knowledge phase requires addressing three key questions. First, what type of information should be extracted from the free-form text to enable problem inference? Second, how do we extract this information in a scalable manner from a large ticket corpus? Third, how do we model the relationships in the extracted information to infer *meaning* from the ticket content. Next we describe solutions to these questions.

Repeated Pattern Extraction

Intuitively, the phrases that would be most useful to build a knowledge base should capture domain-specific information and be related to *hot* (common) and important problem types. As mining arbitrary ticket text is extremely hard (Section 4.1), we first extract hot phrases and later apply filters (Section 4.2.3) to select the important ones.

DESIGN: To find the hot phrases from ticket text, we initially applied conventional text mining techniques for n-gram extraction. N-grams are arbitrary and recurrent word combinations [38] that are repeated in a given context [39]. Since network tickets have no inherent linguistic model, we extracted n-grams of arbitrary length for comprehensive analysis without limiting to bi-grams or tri-grams. We implemented several advanced techniques [39–41] from computational linguistics and NLP, and observed the following challenges:

1. Extracting all possible n-grams can be computationally expensive for a large n and is heavily dependent on the size of the corpus. We investigated using a popular technique by Nagao et al. [41] based on extracting word co-locations, implemented in C [42]. On our dataset, this algorithm did not terminate on a 100K word document after 36 hours of CPU time on a Xeon 2.67 GHz eight-core server with 48 GB RAM, as also observed by others [43].
2. Determining and fine-tuning the numerous thresholds and parameters used by statistical techniques [39, 41, 44] is difficult when the corpus size is large.
3. Not all n-grams are *useful* due to their semantic context. For instance, n-grams such as "showing green" (LED status) and "unracking the" (unmounting the server) occurred frequently together but they do not contribute to the domain knowledge.

To address these challenges, we trade completeness in n-gram extraction for scalability and speedup. Our idea to extract hot patterns is to use a data compression algorithm, typically used to compress files by finding recurring patterns in the data and encoding them. A dictionary is maintained to map the repeated patterns to their output codes. Clearly, these dictionaries do not include all possible n-grams, but they contain hot patterns that are frequent enough to bootstrap the knowledge base.

Data compression algorithms typically operate at a byte or character level, and they do not output the frequency of patterns in their dictionary. To address these issues, NetSieve performs pattern extraction in two phases (Figure 4.6). First, it tokenizes input into sentences and leverages LZW [45] to develop a word-level LZW

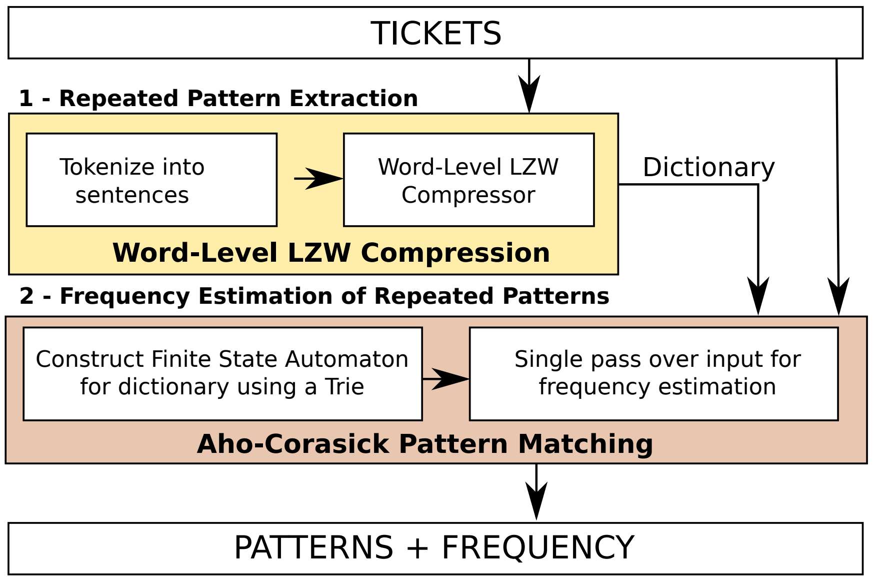

TICKETS
1 - Repeated Pattern Extraction
Tokenize into sentences
Word-Level LZW Compressor
Word-Level LZW Compression
Dictionary
2 - Frequency Estimation of Repeated Patterns
Construct Finite State Automaton for dictionary using a Trie
Single pass over input for frequency estimation
Aho-Corasick Pattern Matching
PATTERNS + FREQUENCY

Figure 4.6. Two Phase Pattern Extraction. First, NetSieve tokenizes input into sentences and applies WLZW to build a dictionary of repeated patterns. Second, it uses the Aho-Corasick pattern matching algorithm to calculate their frequency.

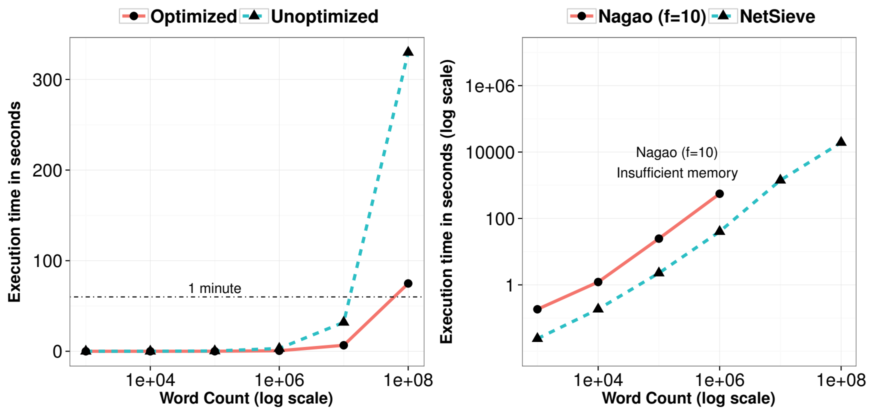


Figure 4.7. Performance of WLZW: (a) Optimized implementation using Cython gives a performance boost of up to 5x-20x over a Python based solution as expected. Comparing NetSieve with N-gram extraction of Nagao et al. [41] (b) NetSieve is able to scale well beyond a million words in comparison to Nagao(f=10), where *f* is the phrase frequency.

Table 4.2
Examples of phrases extracted using the Two Phase Pattern Extraction algorithm.

Phrase Type	Phrase Pattern
Frequent messages	team this is to inform you that there has been a device down alarm reported on
Debug messages	errors 0 collisions 0 interface resets 0 babbles 0 late collision 0 deferred <device> sup 1a
Email snippets	if you need assistance outside of these hours please call into the htts toll free number 1 800

encoder (WLZW) that builds a dictionary of repeated patterns at the word-level. In the second phase, NetSieve applies the Aho-Corasick algorithm [46] to output frequency of the repeated phrases. Aho-Corasick is a string matching algorithm that runs in a single-pass and has a complexity linear in the pattern length, input size and the number of output matches.

IMPLEMENTATION: We implemented the two phase pattern extraction algorithm in Cython [47], that allows translating Python into optimized C/C++ code. To optimize performance, we implemented the Aho-Corasick algorithm using suffix-trees [48] as opposed to the conventional suffix-arrays [49]. As expected, we achieved a 5x-20x performance improvement using Cython compared to a Python-based solution (Figure 4.7(a)).

Figure 4.7(b) shows the performance comparison of WLZW to *Nagao (f=10)* [41] which extracts all n-grams that occur at least 10 times. The latter terminated due to insufficient memory for a million word document. In comparison, NetSieve is able to process documents containing 100 million words in under 2.7 hours.

Note that WLZW is one way to extract hot patterns; we will explore other methods [43, 50, 51] in the future.

Knowledge Discovery

The goal of the knowledge discovery phase is to filter important domain-specific patterns from the extracted set in the previous phase; Table 4.2 shows examples of the extracted phrases. We define a pattern as *important* when it contributes to understanding of the "central topic" in a ticket. Consider the following excerpt from a ticket:

We found that the **device <name> Power LED** is **amber** and it is in **hung state**. This device has **silver power supply**. We need to change the **silver power supply** to **black**. We will let you know once the **power supply** is **changed**.

The central topic of the above excerpt is "device failure that requires a power supply unit to be changed" and the phrases in bold are relevant. While the pattern extractor outputs these phrases as repeated patterns, the key challenge is how to distinguish them from noisy patterns.

DESIGN: An intuitive method is to select the most frequently occurring patterns as important. However, we observed that many of them were warning messages which did not contribute to the central topic. Therefore, we apply a pipeline of three filters to identify the important domain-specific patterns.

Phrase Length/Frequency Filter: The idea behind applying this filter is that both the length and frequency of a phrase can act as good indicators of its importance. Intuitively, we are interested in phrases of short-length, but having a high-frequency. The rationale is that the other length-frequency combinations are either noise, occur due to typos, or can be constructed using short phrases.

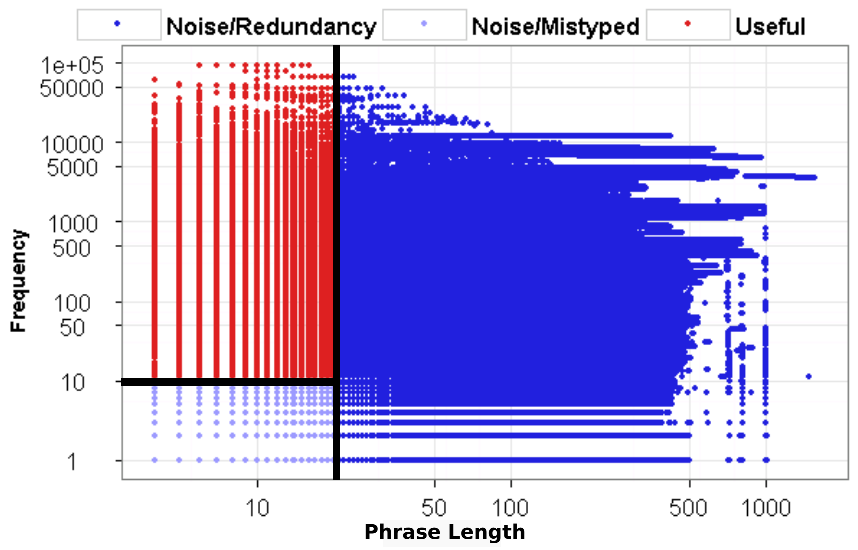


Figure 4.8. Filtering phrases using phrase length and frequency.

We did not use a spell checker because an untrained or an undertrained one may incorrectly modify domain-specific words such as DNS and DMZ, and the probability of an important domain-specific phrase having typos in a large fraction of tickets is likely small. We plot the distribution of length and frequency of phrases (Figure 4.8) and then manually inspect a random subset in each quartile to derive heuristics for threshold-based filtering (Table 4.3).

Part-Of-Speech (PoS) Filter: The second filter is based on the seminal work of Justeson *et al.* [52]. They postulate that technical terms or domain-specific phrases have no satisfactory formal definition and can only be intuitively characterized: they generally occur only in specialized types of usage and are often specific to subsets of domains. Specifically, they conclude that most technical phrases contain only nouns and adjectives after analyzing four major technical dictionaries and subsequently provide a set of seven patterns outlined in Table 4.4. We build upon these patterns and map them to state-of-the-art Penn Treebank tagset [53], a simplified part-of-speech tagset for English, using regular expressions (Table 4.4). Further, this mapping allows

Table 4.3
Thresholds for Phrase Length/Frequency filter.

Filtering Rule	**Reason**
Length > 20 words	Likely Templates (long repeated patterns)
Single word phrases *i.e.*, unigrams	Important unigrams occur as part of a bi-gram or a tri-gram.
Frequency < 10 i.e., rare words or phrases	Likely an isolated incident and not a frequently occurring problem trend
Contain numbers	Domain-specific phrases rarely contain numbers

Table 4.4
NetSieve converts the Justeson-Katz PoS patterns to Penn Treebank PoS patterns to filter technical phrases.

Justeson-Katz Patterns	**NetSieve Patterns**	**Example**
Adjective Noun	JJ NN[PS]*	mobile network
Noun Noun	NN[PS]* NN[PS]*	demo phase
Adjective Adjective Noun	JJ JJ NN[PS]*	fast mobile network
Adjective Noun Noun	JJ NN[PS]* NN[PS]*	accessible device logs
Noun Adjective Noun	NN[PS]* JJ NN[PS]*	browser based authentication
Noun Noun Noun	NN[PS]* NN[PS]* NN[PS]*	power supply unit
Noun Preposition Noun	NN[PS]* IN NN[PS]*	device down alert
JJ: Adjective; **NN**: Singular Noun; **NNP**: Proper singular noun; **NNPS**: Proper plural noun; **IN**: Preposition		

our implementation to leverage existing part-of-speech taggers of natural language toolkits such as NLTK [54] and SharpNLP [55]. Filtering takes place in two steps: (1) each input phrase is tagged with its associated part-of-speech tags, and (2) the part-of-speech pattern is discarded if it fails to match a pattern.

Entropy Filter: The third filter uses information theory to filter statistically insignificant phrases, and sorts them based on importance to aid manual labeling. We achieve this by computing two metrics for each phrase [43]:

1. **Mutual Information (MI)**: $MI(x, y)$ compares the probability of observing word x and word y together (the joint probability) with the probabilities of observing x and y independently. For a phrase pattern, MI is computed by the following formula:
$$MI(xYz) = \log\left(\frac{tf(xYz) * tf(Y)}{tf(xY) * tf(Yz)}\right) \tag{4.1}$$
where xYz is a phrase pattern, x and z are a word/character and Y is a sub-phrase or sub-string, tf denotes the *term-frequency* of a word or phrase in the corpus.

2. **Residual Inverse Document Frequency (RIDF)**: RIDF is the difference between the observed IDF and what would be expected under a Poisson model for a random word or phrase with comparable frequency. RIDF of a phrase is computed as follows:
$$RIDF = -\log\left(\frac{df}{D}\right) + \log\left(1 - \exp\left(\frac{-tf}{D}\right)\right) \tag{4.2}$$
where *df* denotes the document-frequency (the number of tickets which contain the phrase) and D as the total number of tickets.

Phrases with high RIDF or MI have distributions that cannot be attributed to chance [43]. In particular, MI aims to pick vocabulary expected in a dictionary, while RIDF aims to select domain-specific keywords, not likely to exist in a general dictionary. We investigated both metrics as they are orthogonal and that each tends to separately pick interesting phrases [43].

IMPLEMENTATION: The three filters are applied as a sequential pipeline and are implemented in Python. For PoS tagging, we utilize the Stanford Log-Linear PoS Tagger [56] as an add-on module to the Natural Language Toolkit [54] and implement a multi-threaded tagger that uses the phrase length/frequency filter to first filter a list of candidate phrases for tagging.

After applying the threshold-based filtering and PoS filters on the input 18.85M phrases, RIDF and MI are computed for the remaining 187K phrases. This step significantly reduced the computational cost compared to prior work [39–41, 43], which aim to compute statistics for all n-grams. Similar to [43], we did not observe strong correlation between RIDF and MI (Figure 4.9 (top)). We relied solely on RIDF because most phrases with high MI were already filtered by RIDF and the remaining ones contained terms not useful in our context.

The bottom graph of Figure 4.9 shows the CCDF plot of RIDF which can be used to set a threshold to narrow down the phrase list for the next stage of human review. Determining the threshold poses a trade off between the completeness of the domain-dictionary and the human effort required to analyze the extracted patterns. In our prototype, we set the threshold based on RIDF such that 3% (5.6K) of the total phrases (187K) are preserved. Further, we sort these phrase patterns based on RIDF for expert review as phrases with higher values get labeled quickly. An expert sifted through the 5.6K phrases (Figure 4.10) and selected 1.6K phrase patterns that we consider as ground truth, in less than four hours. We observed that most of the discarded patterns were redundant as they can be constructed from the ground truth patterns.

While we leverage manual review to obtain the ground truth, this is a necessary step for any supervised technique. We plan to explore other techniques such as Named-Entity Recognition [57] and using domain experts for crowdsourcing [58] to automate this step.

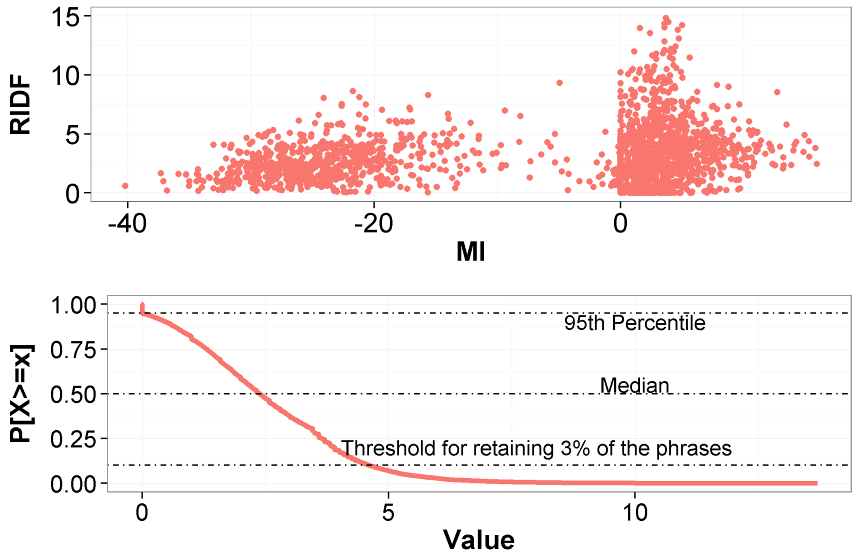


Figure 4.9. Absence of correlation between RIDF and MI metrics (top). Using a CCDF plot of RIDF to determine a threshold for filtering the phrases for expert review (bottom).

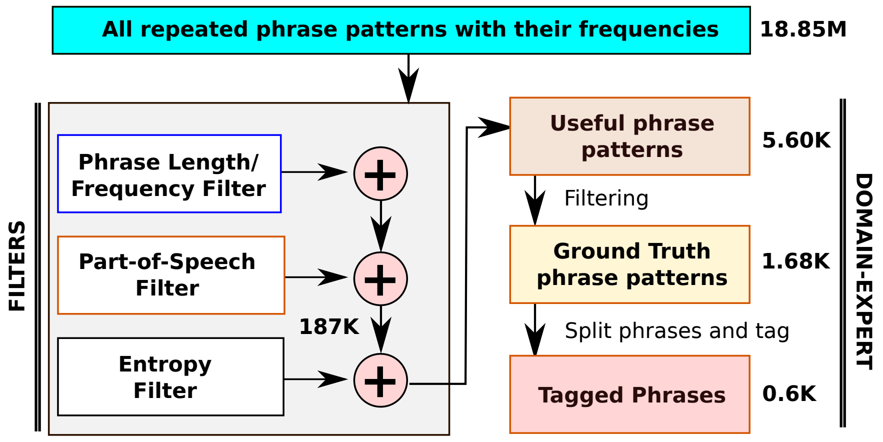


Figure 4.10. The pipeline of filtering phrases to determine a list of ground truth phrase patterns which are then split and tagged manually with the NetSieve-Ontology classes.

Ontology Modeling

The goal of building ontology models is to determine *semantic interpretation* of important domain-specific phrases generated by the knowledge discovery stage. For instance, between the terms *slot* and *memory slot*, we are looking for the latter term with high specificity. Intuitively, we need to model an ontology where domain-specific phrases have a concrete meaning and can be combined together to enable semantic interpretation.

DESIGN: Developing an ontology model involves three steps [59, 60]: (1) defining classes in the ontology, (2) arranging the classes in a taxonomic (superclass, subclass) hierarchy and (3) defining interactions amongst the classes. Note that defining an ontology is highly domain-specific and depends on extensions anticipated by the domain-expert. We designed and embedded one such ontology into NetSieve based on discussion with operators. Below, we discuss the design rationale behind the classes, taxonomies and interactions in our model.

Classes and Taxonomies: A class describes a concept in a given domain. For example, a class of entities can represent all devices (*e.g.*, routers, load balancers and cables) and a class of conditions can represent all possible states of an entity (*e.g.*, bad, flapping, faulty). A domain-expert sifted through the 1.6K phrases from previous stage and after a few iterations, identified seven classes to describe the phrases, shown in Table 4.5.

Taxonomic Hierarchy: To enable fine-grained problem inference, *Entity* is divided into three sub-classes: *Replaceable* denoting entities that can be physically replaced, *Virtual* denoting entities that are intangible and *Maintenance* denoting entities that can "act" upon other entities. *Actions* are sub-classed in a similar way. The rest of the classes can be considered as qualifiers for *Entities* and *Actions*. Qualifiers, in general, act as adjectives or adverbs and give useful information about an *Entity* or *Action*. In the final iteration, our domain-expert split each of the 1.6K long phrase

Table 4.5
Classes in the NetSieve-Ontology model

Class	Sub-Class	Interpretation	Example
Entity	Replaceable	Tangible object that can be created/destroyed/replaced	Flash memory, Core router
	Virtual	Intangible object that can be created/destroyed/replaced	Port channel, configuration
	Maintenance	Tangible object that can act upon other entities	field engineer, technician
Action	Physical	Requires creating/destroying an entity	decommission, replace, rma
	Maintenance	Requires interacting with an entity and altering its state	clean, deploy, validate, verify
Condition	Problem	Describes condition that is known to have a negative effect	inoperative, reboot loop
	Maintenance	Describes condition that describes maintenance	break-fix
Incident	False Positive	State known to not have any problems	false positive, false alert
	Error	State known to cause a problem	error, exception
Quantity	Low	Describes the quantity of an entity/action	low, minor
	Medium		medium
	High		high, major
Negation	Synthetic	Uses verb or noun to negate a condition/incident/action	absence of, declined, denied
	Analytic	Uses ’not’ to negate a condition/incident/action	not
Sentiment	Positive	Adds strength/weakness to an action/incident	confirm, affirmative
	Neutral		not sure
	Negative		likely, potential

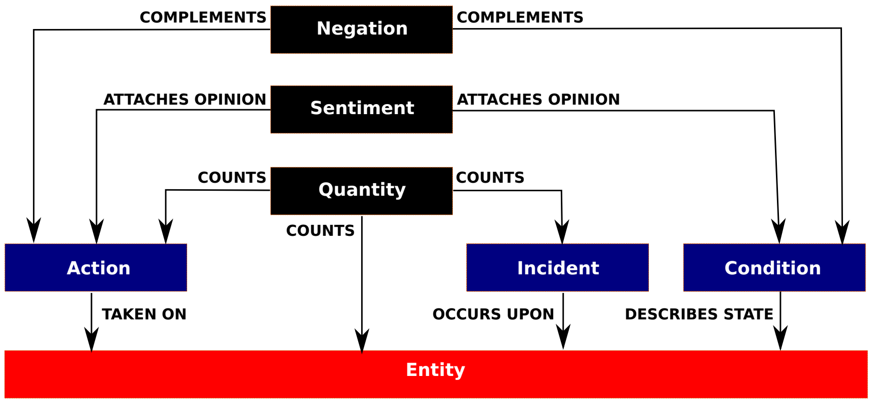


Figure 4.11. Ontology Model depicting interactions amongst the NetSieve-Ontology Classes.

patterns into their constituent small phrase patterns and tagged them with the most specific class that captured the phrase e.g., "and gbic replacement" $\rightarrow$ [(and, OMIT), (gbic, ReplaceableEntity), (replacement, PhysicalAction)].

Most of these tagged phrases are domain-specific multi-word phrases and are not found in a dictionary. While the words describing Entities were not ambiguous, we found a few cases where other classes were ambiguous. For instance, phrases such as "power supply" (hardware unit or power line), "bit errors" (memory or network link), "port channel" (logical link bundling or virtual link), "flash memory" (memory reset or type of memory), "device reload" and "interface reset" (unexpected or planned), and "key corruption" (crypto-key or license-key) were hard to understand without a proper context. To address this ambiguity, we use the text surrounding these phrases to infer their intent (*Section* 4.2.4).

Finally, using these mappings, NetSieve embeds a *ClassTagger* module that given an input, outputs tags for words that have an associated class mapping.

Interactions: An interaction describes relationships amongst the various classes in the ontology model. For instance, there are valid interactions (an *Action* can be caused upon an *Entity*) and invalid interactions (an *Entity* cannot describe a *Sentiment*). Figure 4.11 shows our model comprising interactions amongst the classes.

Table 4.6
Concepts for the NetSieve-Ontology

Concept	Pattern	Example
Problems	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by ProblemCondition	The (device) was (faulty)
Activities	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by MaintenanceAction	(check) (device) connectivity and (clean) the (fiber)
Actions	[Replaceable — Virtual — Maintenance] Entity preceded/succeeded by PhysicalAction	An (RMA) was initiated for the (load balancer)

IMPLEMENTATION: We obtained 0.6K phrases from the 1.6K phrases in Section 4.2.3 categorized into the seven classes. We implemented the *ClassTagger* using a trie constructed using NetSieve knowledge base of domain-specific phrases, and a dictionary of their ontology mappings. The tagging procedure works in three steps. First, the input is tokenized into sentences. Second, using the trie, a search is performed for the longest matching phrase in each sentence to build a list of domain-specific keywords e.g., in the sentence "the power supply is down", both "supply" and "power supply" are valid domain keywords, but the ClassTagger marks "power supply" as the relevant word. Finally, these keywords are mapped to their respective ontology classes using the dictionary. For instance, given the snippet from Section 4.2.3, the *ClassTagger* will produce the following output:

We found that the (device) / **ReplaceableEntity <name> (Power LED) / ReplaceableEntity** is **(amber) / Condition** and it is in **(hung state) / ProblemCondition**. This device has **(silver) / Condition (power supply) / ReplaceableEntity**. We need to change the **(silver) / Condition (power supply) / ReplaceableEntity** to **(black) / Condition**. We will let you know once the **(power supply) / ReplaceableEntity** is **(changed) / PhysicalAction**.

4.2.4 Operational Phase

The goal of this phase is to leverage the knowledge base to do automated problem inference on trouble tickets. A key challenge to address is how to establish a relationship between the ontology model and the physical world. In particular, we want to map certain interactions from our ontology model to concepts that allow summarizing a given ticket.

DESIGN: Our discussion with operators revealed a common ask to answer three main questions: (1) What was observed when a problem was logged?, (2) What activities were performed as part of troubleshooting? and (3) What was the final action taken to resolve the problem? Based on these requirements, we define three key concepts that can be extracted using our ontology model (Table 4.6): (1) *Problems* denote the state or condition of an entity, (2) *Activities* describe the troubleshooting steps, and (3) *Actions* capture the problem resolution.

The structure of concepts can be identified by sampling tickets describing different types of problems. We randomly sampled 500 tickets out of our expert-labeled ground truth data describing problems related to different device and link types. We pass these tickets through NetSieve's *ClassTagger* and get a total of 9.5K tagged snippets. We observed a common linguistic structure in them: in more than 90% of the cases, the action/condition that relates to an entity appears in the same sentence *i.e.,* information can be inferred about an entity based on its neighboring words. Based on this observation, we derived three patterns (Table 4.6) that capture all the cases of interest. Intuitively, we are interested in finding instances where an action or a condition precedes/succeeds an entity. Based on the fine granularity of the subclasses, the utility of the concepts extracted increases i.e., *PhysicalAction was taken on a ReplacableEntity* is more important than *Action was taken on an Entity.*

This type of a proximity-search is performed once for each of the three concepts. First, the *ClassTagger* produces a list of phrases along with their associated tags.

Second, we check to see if the list of tags contain an action/condition. Once such a match is found in a sentence, the phrase associated with the action/condition is added to a dictionary as a key and all entities within its neighborhood are added as corresponding values. We implemented several additional features like negation detection [61] and synonym substitution to remove any ambiguities in the inference output.

EXAMPLE: *"The load balancer was down. We checked the cables. This was due to a faulty power supply unit which was later replaced"*, is tagged as *"The (load balancer) / ReplaceableEntity was (down) / ProblemCondition. We (checked) / MaintenanceAction the (cables) / ReplaceableEntity. This was due to a (faulty) / ProblemCondition (power supply unit) / ReplaceableEntity which was later (replaced) / PhysicalAction"*. Next, a dictionary is built for each of the three concepts. Two classes are associated if they are direct neighbors. In this example, the output is the following:
[+] Problems - {down: load balancer, power supply unit}
[+] Activities - {checked: cable}
[+] Actions - {replaced: power supply unit}
In the final stage, the word "replaced" is changed into "replace" and "checked" into "check" using a dictionary of synonyms provided by the domain-expert to remove any ambiguities.

IMPLEMENTATION: We implemented our inference logic using Python. Each ticket is first tokenized into sentences and each sentence is then used for concept inference. After extracting the concepts and their associated entities, we store the results in a SQL table. Our implementation is able to do problem inference at the rate of 8 tickets/sec on average on our server, which scales to 28,800 tickets/hour; note that this time depends on the ticket size, and the overhead is mainly due to text processing and part-of-speech tagging.

Output Rendering

NetSieve by design is integrated into our data center troubleshooting architecture *i.e.*, it is able to access the NOC database in real-time. Therefore, any new ticket unseen by NetSieve is processed on-the-fly.

To export the inference output, NetSieve supports a variety of data formats such as XML [62] and JSON [63], and allows querying using its *Query Engine* or the API exposed through a REST [64] interface. NetSieve also supports graphical visualization of inference output to perform ticket summarization and trend analysis on multiple tickets.

DESIGN: During its *Operational Phase*, NetSieve expects either a single ticket (for summarization) or a collection of tickets (for trend analysis). The latter can be specific through a list of predicates (*e.g.*, device type, device model, data center, date range) based on which the collection of tickets will be narrowed down by the *Query Engine* inside NetSieve. Based on the type of input, there are two types of analysis of interest:

Sifting through long tickets poses two challenges. First, it takes significant effort to understand the ticket, especially if the ticket contains millions of words. Second, it introduces ambiguity - different network operators may arrive at different conclusions.
Ticket Summarization: NetSieve provides the ability to visually summarize a ticket to enable quick review by operators. We introduce *Concept Tree* to visually represent a ticket summary that logically maps the concepts extracted through the *sysname*-Ontology into a tree. Figure 4.12(a) shows an example.

: (1) At $depth = 1$, the node represents a unique ticket identifier for quick reference, (2) At $depth = 2$, there are three distinct nodes representing (Problems, Activities, Actions), (3) At $depth = 3$, each node represents an action/condition describing the entity and (4) At $depth = 4$, each node represents a single entity connected to the parent action/condition. When rendered, such a tree can be read bottom-up starting from entities. There are three requirements in rendering such a tree:

- **TS.R1**: Node placement should be non-overlapping
- **TS.R2**: The tree should be hierarchical with some notion of gravity that exerts a varying gravitational pull on nodes at different levels.
- **TS.R3**: It should be easy to examine any individual node.

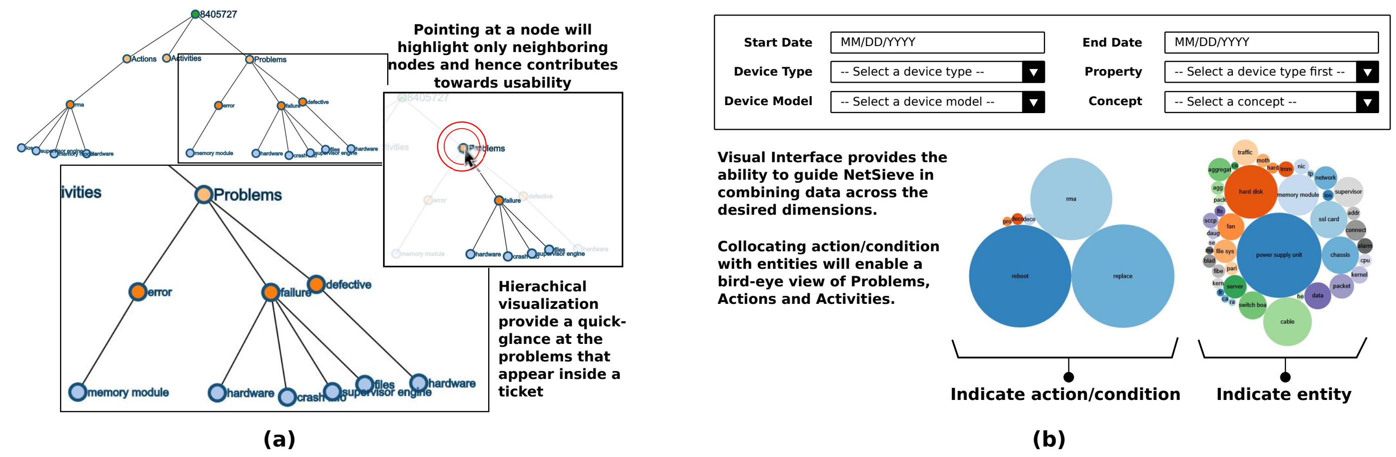
NetSieve satisfies **R1** by using Barnes-Hut approximation to calculate the layout of the nodes which are then placed onto an SVG canvas. Barnes-Hut approximation falls into the family of N-body simulations which simulate a dynamic system of particles, usually under the influence of physical forces, such as gravity. To satisfy **R2**, once the layout algorithm terminates, each node is re-aligned based on its depth in the tree by giving it an additional downward pull. We satisfy **R3** by ensuring that when the user moves a mouse cursor over any node, only the node and its direct parent and sibling are highlighted (shown in Figure 4.12). NetSieve is able to generate a *Concept Tree* in 1.8 seconds on an average in our experiments.

Trend Analysis: An integral part of network management involves decision making and inventory management. Specifically, a domain-expert may be interested in understanding the trend across certain dimensions. To help operators understand big problem trends, NetSieve performs trend analysis across specified dimensions e.g., what are the major problems with devices belonging to manufacturer X? NetSieve leverages *Trait Charts* as a mechanism to support large-scale trend analysis across multiple dimensions by grouping information via predicates such as device type, device model, and data center. Subsequently, the data grouped across concepts can then be rendered as bipartite bubble plots where the left and right partition represent a different ontological class. The size of each bubble is proportional to the number of tickets in which the associated concept appears. Figure 4.12(b) shows our implementation of *Trait Charts*.

There are two requirements in rendering such a chart:

- **TA.R1**: It should be easy to "glean" the big picture and understand the major problems.
- **TA.R2**: The approach should be applicable to cases where there are large number of distinct problems *i.e.*, the visualization should remain uncluttered irrespective of the scale of data.

IMPLEMENTATION: NetSieve implements a flexible front-end to support ticket summarization and trend analysis. and its front-end interface has been built using a number of web technologies [65–69]. NetSieve is able to generate a *Concept Tree* in 1.8 sec and *Trait Charts* in 3.9 sec on an average in our experiments. It offers a visual interface for predicate selection (device type, device model etc.) and dynamically updates the charts based on the collection of tickets retrieved by the *Query Engine* of NetSieve. The radius of the bubbles in the charts is computed as a function of the number of tickets in which the associated concept appears. NetSieve satisfied **TA.R1** by using the left partition to represent the primary concept (Problems, Actions, Activities) and the right partition to represent the entity involved. Using such charts, we were able to uncover major problems in device generations (Section 5.7). NetSieve achieves **TA.R2** by ensuring that when a user moves the mouse cursor over any node, detailed information about the bubble is rendered through a non-intrusive popup. NetSieve is able to generate *Trait Charts* in 3.9 seconds on average in our experiments.



8405727
Actions
Activities
Problems
rma
error
failure
defective
memory module
hardware
crash
files
supervisor engine
Pointing at a node will highlight only neighboring nodes and hence contributes towards usability
Hierachical visualization provide a quick-glance at the problems that appear inside a ticket
(a)
Start Date
MM/DD/YYYY
End Date
MM/DD/YYYY
Device Type
-- Select a device type --
Property
-- Select a device type first --
Device Model
-- Select a device model --
Concept
-- Select a concept --
Visual Interface provides the ability to guide NetSieve in combining data across the desired dimensions.
Collocating action/condition with entities will enable a bird-eye view of Problems, Actions and Activities.
rma
reboot
replace
hard disk
memory module
power supply unit
chassis
ssl card
cable
network
supervisor
traffic
fan
packet
data
switch box
server
kernel
cpu
Indicate action/condition
Indicate entity
(b)

Figure 4.12. Trend Analysis using Trait Charts

Table 4.7
Evaluating NetSieve accuracy using different datasets. High F-scores are favorable.

Source	**Dataset**		**Precision**		**Recall**		**F-Score**		**Accuracy %**	
	Devices	# Tickets	Problems	Actions	Problems	Actions	Problems	Actions	Problems	Actions
Domain Expert	LB-1	122	1	1	0.982	0.966	0.991	0.982	97.7	96.6
	LB-2	62	1	1	1	1	1	1	100.0	100.0
	LB-3	31	1	1	1	0.958	1	0.978	100.0	95.7
	LB-4	36	1	1	1	1	1	1	100.0	100.0
	FW	35	1	1	0.971	0.942	0.985	0.970	97.1	94.3
	AR	410	1	1	0.964	0.951	0.981	0.974	96.4	95.1
Vendor	LB	78	1	1	0.973	0.986	0.986	0.993	97.3	98.7
	CR	77	1	1	1	0.896	1	0.945	100.0	89.6
CR: Core Routers; **LB[1-4]**:Types of Load balancers; **FW**:Firewalls; **AR**: Access Routers										

4.3 System Evaluation

For evaluation, we use two standard metrics from information retrieval: (1) *Accuracy Percentage* [70] computed as $\frac{TP+TN}{TP+TN+FP+FN}$ and (2) *F-Score* [71] computed as $\frac{2TP}{2TP+FP+FN}$, where TP, TN, FP, FN are true positives, true negatives, false positives and false negatives, respectively. F-Scores consider both precision and recall, and its value of 1 indicates a perfect classifier. Precision is defined as the ratio of TP and (TP+FP), and recall is defined as the ratio of TP and (TP+FN).

4.3.1 Evaluating NetSieve Accuracy

To test NetSieve's accuracy, we randomly divided our two year ticket dataset into training and test data. The test data consists of 155 tickets on device replacements and repairs from two network vendors and 696 tickets labeled by a domain expert while the training data comprises the rest of the tickets. We use the training data to build NetSieve's knowledge base. The domain expert read the original ticket to extract the ground truth in terms of Problems and Actions, which was then compared with corresponding phrases in the inference output.

Table 4.7 shows the overall results by NetSieve on the test dataset. On the expert labeled data, we observe the precision of NetSieve to be 1, minimum recall value to be 0.964 for Problems and 0.942 for Actions, F-score of 0.981-1 for Problems and 0.970-1 for Actions, and accuracy percentage to be 96.4%-100% for Problems and 94.3%-100% for Actions. These results indicate that NetSieve provides useful inference with reasonable accuracy in analyzing network tickets.

Next, we validate the inference output against data from two network device vendors that record the ground truth based on the diagnosis of faulty devices or components sent back to the vendor. Each vendor-summary reported the root cause of the failure (similar to NetSieve's Problems) and what was done to fix the problem (similar to NetSieve's Actions). We obtained vendor data corresponding to total 155 tickets on load balancers and core routers from our dataset. Since the vocabulary in the vendor data comprised new, vendor-specific words and synonyms not present in our knowledge base (e.g., port interface mentioned as ’PME’), we asked a domain-expert to validate if NetSieve’s inference summary covers the root cause and the resolution described in the vendor data. For instance, if the vendor data denoted that a router failed due to a faulty supervisor engine (SUP), the expert checked if NetSieve captures “failed device” under *Problems* and “replaced SUP” under *Actions*.

The accuracy of NetSieve on the vendor data is observed to be 97.3%-100% for Problems and 89.6%-100% for Actions. One reason for relatively lower accuracy for Actions on this dataset is due to a small number of false negatives: the corrective action applied at the vendor site may differ from our ticket set as the vendor has the expert knowledge to fix problems specific to their devices.

Overall, NetSieve has reasonable accuracy of 96.4%-100% for *Problems* and 89.6%-100% for *Actions*, measured based on the labeled test dataset. We observed similar results for *Activities* and thus omit them.

4.3.2 Evaluating Usability of NetSieve

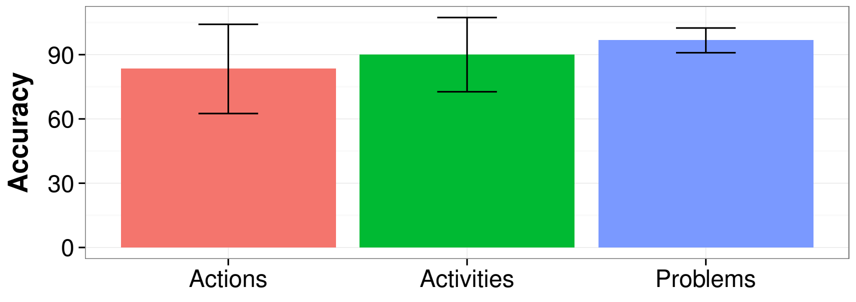
We conducted a user study involving five operators to evaluate the usability of NetSieve for automated problem inference compared to the traditional method of manually analyzing tickets. Each operator was shown 20 tickets selected at random from our dataset and asked to analyze the type of problems observed, activities and actions. Then, the operator was shown NetSieve inference summary of each ticket and asked to validate it against their manually labeled ground truth. We measured the accuracy, speed and user preference in the survey.

Figure 4.13 shows the accuracy and time to manually read the ticket versus the NetSieve inference summary across the operators. We observed average accuracy of 83%-100% and a significant decrease in time taken to manually read the ticket from 95P of 480s to 95P of 22s for NetSieve.

Table 4.8
Top-3 Problems/Activities/Actions and Failing Components as obtained through NetSieve's Trend Analysis

Device	**Problems**	**Activities**	**Actions**	**Failing Components**
AR	memory error, packet errors, illegal frames	verify cable, reseat/clean cable, upgrade OS	replace with spare, rma, reboot	SUP engine, cables, memory modules
AGG	device failure, packet errors, defective N/W card	verify cable, upgrade N/W card, swap cable	replace with spare, reboot, rma	cables, N/W card, SUP engine
CR	circuit failure, N/W card failure, packet errors	verify cable, verify N/W card, upgrade fiber	replace with spare, reboot, rma	cables, N/W, memory modules
ER	circuit failure, N/W card failure, packet errors	verify cable, verify N/W card, upgrade fiber	replace with spare, rma, reboot	N/W card, chassis, cables
LB	PSU failure, device rebooted, config error	verify PSU, verify config, verify cable	replace with spare, reboot, rma	PSU, HDD, memory modules
ToR	connection failure, ARP conflict, SUP engine failure	verify cable, power cycle blade, verify PSU	reboot, replace with spare, rma	cables, OS, SUP engine
FW	connection failure, reboot loop, data errors	verify config, verify connections, verify PSU	reboot, replace with spare, rma	cables, PSU, N/W card
VPN	connection failure, config error, defective memory	verify config, verify N/W card, deploy N/W card	reboot, replace with spare, rma	OS, SUP engine, memory modules

AR: Access Routers, **AGG**: Aggregation Switch; **[E/C]R**: Edge/Core Router; **LB**: Load Balancer; **ToR**: Top-of-Rack Switch; **FW**: Firewall



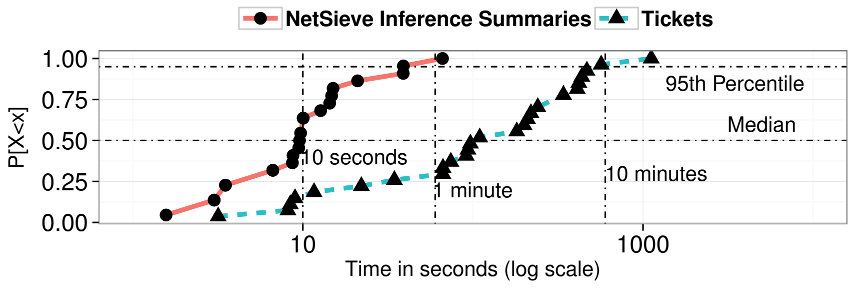


Figure 4.13. Accuracy obtained from the user survey (top). CDF of time to read tickets and inference summaries (bottom).

4.4 Summary

Network trouble tickets contain valuable information for network management, yet they are extremely difficult to analyze due to their free-form text. In this chapter, we present a practical approach towards automatically analyzing the natural language text to do problem inference. We presented NetSieve that automatically analyzes ticket text to infer the problems observed, troubleshooting steps, and the resolution actions. Our results are encouraging: NetSieve achieves a reasonable accuracy of 83%-100%. As we will show in Chapter 5, NetSieve has been applied to answer several key questions for network management.

5 DATA-DRIVEN APPROACHES TO DERIVING ACTIONABLE INSIGHTS

The growing demand for always-on and low-latency cloud services is driving the creation of globally distributed datacenters. A major factor affecting service availability is reliability of the network, inside both datacenters and wide-area links connecting them. While several research efforts focus on building scale-out datacenter networks, little has been reported on real network failures and how they impact geo-distributed services.

In this chapter, we detail one of the first attempts to characterize *intra-datacenter* and *inter-datacenter* network failures from a service perspective. We describe a large-scale study analyzing and correlating failure events over three years across multiple datacenters and thousands of network elements such as Access routers, Aggregation switches, Top-of-Rack switches, and long- haul links. Our study using real-world data focuses on understanding the failure modes and correlation of network failure logs and how they impact cloud services. Specifically, we aim to answer the the following questions:

Q1 **Network Stamp Availability of a Service**: What are the failure characteristics of the network stamp (set of all network elements rooted at a pair of Layer-3 Access routers) of a service inside a datacenter as well as across geo-distributed datacenters? How effective are redundancy mechanisms in handling intra- and inter-datacenter network failures? How many independent network stamps are needed to meet a network redundancy effectiveness service-level-agreement (SLA) of a cloud service?

Q2 **Causes of Network Failures**: What are the main root causes of network failures?

Q3 **Failure Modeling**: How to model failures of network components? Are failures recurrent? Are they bursty? Are device-level repairs effective?

Q4 **Capacity vs. Availability**: For commodity Layer-2 switches, how do their port capacity in terms of connected devices affect their availability in operation?

This chapter analyzes the failure characteristics of network stamp of a service comprising Access routers, Aggregation switches, Top-of-Rack and inter-datacenter links, based on three years' (2010-13) worth of network event logs collected in a cloud provider network across thousands of devices spanning dozens of datacenters. Our data covers a wide range of network data sources, including syslog and SNMP alerts, network trouble tickets, maintenance tracking and revision control system, and traffic carried by links.

Our Contribution: Our study reveals many key findings that can provide useful guidelines to improve network reliability for geo-distributed services. Our major findings are as follows:

1. Network failures cause significant impact to cloud services, dominated by connectivity loss problems (70%) and service errors (43%) due to Intra-Datacenter and Inter-Datacenter network problems, respectively.
2. The number of independent network stamps for a desired network redundancy effectiveness of 3 9's is three and for 4 9's is four.
3. Network redundancy is least effective at the Access router-Aggregation switch layer and is most effective at the Inter-datacenter level.
4. Network device failures are not memoryless and exhibit the "few bad apples" effect.
5. Layer-2 Aggregation switches exhibit high availability when up to half of their port capacity is utilized in terms of Top-of-Rack switch count. However, the availability quickly decreases as the Top-of-Rack switch count increases.
6. Fiber length in Inter-Datacenter links is not correlated with the number of failures, and links with high utilization exhibit 2x-3x higher downtime than expected.

7. Top-of-Rack switches exhibit an increase in probability of a successive device failure after repair, while this probability decreases for Aggregation switches and Access routers.
8. Middlebox failures are prominent and can greatly degrade service performance and availability.
9. Most failures are grey dominated by connectivity errors and link flaps that exhibit intermittent connectivity; fail-stop device failures occur, but they are not as often.
10. Hardware faults, misconfigurations and overload problems are present, but they are not in majority in contrast to common perception [19].
11. There are a variety of misconfigurations such as incorrect rules, VLAN misallocation and mismatched keys.
12. Middlebox redundancy is ineffective in about 33% cases for load balancers and firewalls due to configuration bugs, faulty failovers and software version mismatch.
13. A family of middleboxes having faulty PSUs and another exhibiting the few bad apples effect, could have been detected as early problem indicators.

Note that there are other resiliency mechanisms deployed at compute and storage layers of a service which are complementary to the network layer studied in this paper. While these mechanisms may be able to mask the impact of some of the network failures, they require service operators to carefully balance the trade-off between consistency and availability particularly under network partitions, due to the famous CAP dilemma [72]. We plan to study the overall impact of redundancy mechanisms at the storage, compute, and network layers in future work.

Limitations: A real-world empirical study such as ours raises some potential issues such as (a) representativeness, (e.g., do results hold for other deployments?), (b) soundness of methodology, (e.g., did we miss any events?), and (c) human bias. (e.g., did an operator misclassify an error?).

While we studied only our dataset, we believe it is fairly representative of a larger class of datacenters based on the scale (10k+ core network devices, 100k+ network links, 100k+ servers and 2k+ middleboxes across 10+ datacenters), diversity (multiple network device types and platforms), measurement duration (over two years), and validation with operators having high experience with network devices and some having previously worked at other sites. Further, the network devices we studied are from multiple leading network vendors having a significant share of the network device market.

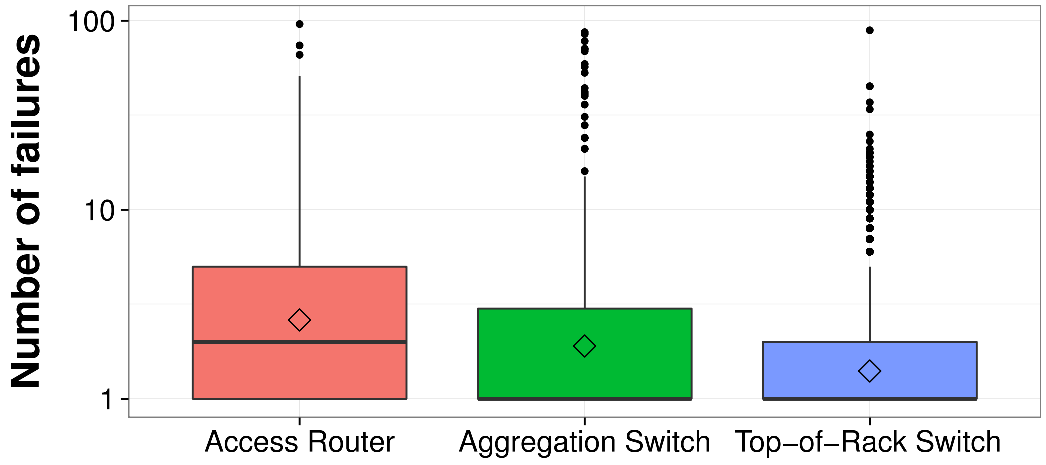
Our network logs are collected from low-level data sources which are subject to failures of monitoring servers, and incomplete or inconsistent event logging. For each failure, our methodology examined *all* the available monitoring sources, including SNMP/syslog alerts, traffic and maintenance data, and validated with both high-severity incident reports and trouble tickets deemed *actionable* by network operators. Thus, the number of false positives and false negatives should be very low.

Finally, a large network such as the one we studied is typically managed by a diverse team comprising operators, engineers and third-party contractors. Thus, any human bias (e.g., due to technical skill, experience, device familiarity) may cause artifacts such as a failure recurrence, longer-than- expected recovery, or a misdiagnosis. Thus, our results should be interpreted with these potential issues in mind.

Chapter Organization: The rest of this chapter is organized as follows. Section 5.1 characterizes failures within a large cloud provide spanning three years and comprising 100k+ servers, 10k+ Layer-2 and Layer-3 devices across 10+ datacenters. In Section 5.2, we model the failure data from Section 5.1. Section 5.4 discusses the effectiveness of network redundancy in masking network failures. Section 5.5 and Section 5.6 make an attempt at answering important operational questions. Section 5.7 describes how NetSieve was used in the real-world on tens of thousands of network trouble tickets to understand the causes behind network failures. Section **??** describes how our results were leveraged by different teams inside a large cloud provider. Finally, we present the summary of our findings in Section 5.9.

5.1 Failure Characterization

In this section we characterize reliability of both the network core and middleboxes in our dataset: firewalls, IDPS, VPNs each spanning two generations (FW[1-2], IDPS[1-2], and VPN[1-2], respectively), and load balancers spanning three of them (LB[1-3]). For a middlebox type, its different generations are from the same network vendor, and they are ordered by their release date (1 is oldest). Where applicable, we first provide an inter-type comparison and then compare devices across different generations within a type.



Type	Mean	Median	Q75	Q95	StdDev	COV
AR	5.2	2	5	19.9	9.9	1.8
AGG	3.7	1	3	12.0	9.2	2.4
ToR	1.8	1	3	6.0	2.6	1.4

Figure 5.1. Comparing the number of failures per device per year across ARs, AGGs and ToRs. In the box-plot, the horizontal bolded line is the median; the box boundaries are inter-quartile ranges; the dots are outliers. The y-axis is on log scale.

Table 5.1
Annual number of failures per middlebox and annualized failure rate (AFR) for devices that were present for at least a month in production.

Type	**Mean**	**Stddev**	**25P**	**50P**	**75P**	**95P**	**COV**	**AFR% (July 2010-11)**	**AFR% (July 2011-12)**
FW	2.1	2	1	2	3	5	0.9	19	13
IDPS	3.5	2.9	1	1	6	8.3	0.8	23	18
LB	1.5	0.9	1	1	2	3	0.6	31	19
VPN	3.4	2.2	2	3	5	7	0.7	7	12

5.1.1 Annual Failure Rate

We first analyze the number of failures per device across three types of network core elements: Access routers, Aggregation switches and Top-of-Rack switches. Figure 5.1 shows a box-plot of the number of failures per device across the three device types. Notice that the mean number of failures per device is highest for ARs. ToRs exhibit the lowest mean number of failures due to their large population, which is consistent with our previous findings [20]. Though AGGs experience fewer failures per device with a median of one failure (table below Figure 5.1), the device population exhibits a high variability with a COV[1] of 2.4. This is also evident from the presence of outliers in Figure 5.1 (the dots present at the top of each box) indicating the presence of a “few bad” devices that log numerous failures.

Next, we analyze the number of failures per device across middleboxes. Figure 5.2 shows the CDF of the number of failures per device per year for devices with at least a month in operation. Most devices experience few failures per year with a median value of at most three (Table 5.1), and they occur with low variability (COV ¡ 1) as

[1]COV [73] shows the extent of variability to mean of the population and is defined as the ratio of the standard deviation σ to the mean μ. Distributions with COV < 1 exhibit low-variance and vice versa.

expected in a service provider network carrying customer traffic. Across middlebox types, IDPS devices show relatively highest annual failures per device with a 95th percentile value of 8.3 while LBs exhibit lowest with a 95th percentile value of 3. The distribution for firewalls have a relatively long tail of up to 24 annual failures per device compared to other types.

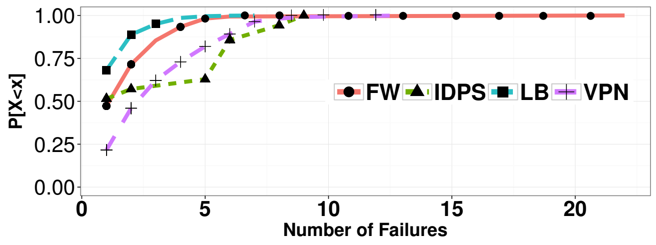


Figure 5.2. Failures per device per year for middleboxes.

We next compute the annualized failure rate (AFR) by dividing the number of devices of a given type that observe at least one failure by the population of that type.

Inter-type: Table 5.1 shows the AFR for different device types during July 2010-11 and July 2011-12. We observe that LBs are the least reliable with about a 1 in 3 chance of failure during July 2010-11 which improved to about a 1 in 5 chance in the next year. The reason for this improvement is due to on-boarding of the more reliable LB-3 platform in the deployment and the end-of-life retirement of some of the older LB-1 devices. FWs and IDPSes also exhibit improvement in AFR over the two years. VPNs show relatively the lowest AFR amongst all middleboxes.

Intra-type: Figure 5.3 shows the AFR and Figure 5.4 shows the fraction of failures and downtime, respectively, across different generations of each middlebox type. We find that during the first year, FW-2 exhibits relatively low failure rate of 14% whereas FW-1 exhibits 2x higher failure rate at 32%. Through trouble tickets, we found that the reliability of FW-2 started improving when the vendor released an end-of-life notice for FW-1 and provided faster bug fixes and better technical support to handle FW-2 failures. Figure 5.4 shows that FW-2 contributed majority to the total failures due to its dominant population while FW-1 devices decreased due to being decommissioned. This observation combined with the relatively low AFR of FW-2 (Figure 5.3) indicates that a few devices contributed the most, due to limited device familiarity of operators with the new platform and unexpected hardware problems. Further, the number of failures contributed by FW-2 is roughly proportional to its downtime while they are relatively higher for FW-1. This indicates that most FW-1 failures seem to be relatively short-lived and resolved through reboot or rule reconfiguration.

In the case of load balancers, LB-1 exhibits a large AFR and contributes significantly (Figure 5.4) toward the total number of failures and downtime. However, note that in comparison to other LBs, the fraction of failures contributed is higher than the downtime indicating that most problems are short-lived. We validate this observation by using the time-to-repair (TTR) plots (Figure 5.5(c)) where a short TTR indicates transient problems. We find that a majority of these problems were due to link flapping i.e., interface continually goes up and down due to a bad cable, faulty line card, etc. In most cases, the network engineer adjusts/reseats the cable/network card or the problem gets self-corrected. LB-2, on the other hand, exhibited an increased AFR indicating that this generation experienced new types of problems such as unexpected reboots. However, these problems seem to have been resolved in its successor, LB-3, which exhibits relatively the highest reliability. Similar to the case of firewalls, note that the significant improvement in LB-3 reliability occurred due to the vendor releasing an end-of-life notice for older generations, as we observed through tickets.

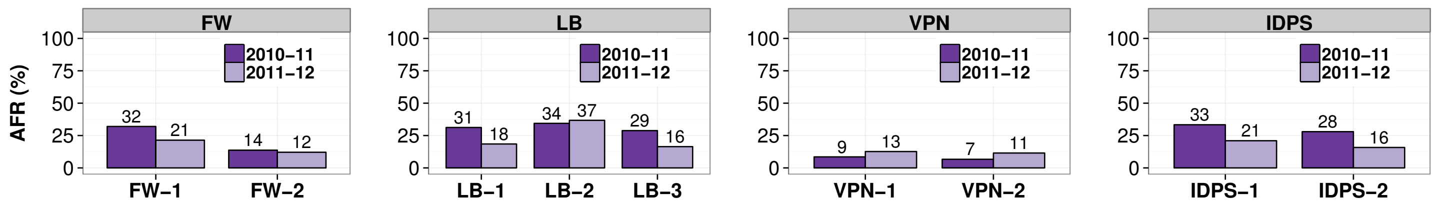


Figure 5.3. Comparing annualized failure rate across middleboxes over the two years measurement period.

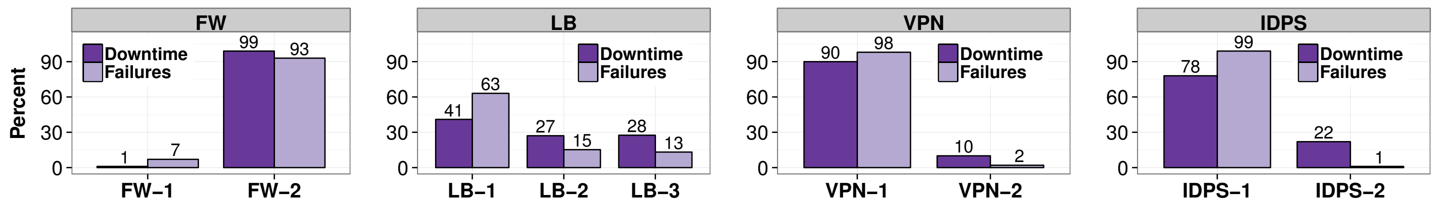


Figure 5.4. Comparing fraction of failures and downtime across middleboxes.

Table 5.2
Comparing TTR across ARs, AGGs and ToRs.

Type	Mean (hrs)	Median (mins)	Q75 (hrs)	Q95 (hrs)	StdDev (days)
AR	12.4	21.5	2	37.2	2.5
AGG	2.1	4.8	0.4	5.2	0.6
ToR	2.9	7.1	0.3	5.2	0.8

Across the four middlebox types, VPN devices exhibit relatively higher reliability with failure rates between 7%-9% which increased slightly year over year to 11%-13%. The increase in AFR over time is due to (a) the older generation VPN-1 which was being decommissioned exhibiting an increasing number of connectivity errors, and (b) the newer generation VPN-2 exhibiting the infant mortality effect e.g., memory errors. Further, the failures contributed is roughly proportional to the downtime caused by both VPN-1 and VPN-2 suggesting that each failure roughly causes the same downtime on average. We find a common root cause of VPN failures to be VLAN misconfigurations. Similar to other middleboxes, we also observed an end-of-life announcement for VPN-1 recently.

5.1.2 Quantifying Time-to-Repair and Time-to-Failure

Time-to-repair: We define the time to repair (TTR) for a device as the time between a down notification for a device and when it is next reported as being back online. There are two main types of failures: *short-lived transient faults* where an engineer may not always intervene to resolve them and *long-term failures* where the device or its component is usually replaced or decommissioned. Note that for long-term failures, the failure durations may be skewed (up to days) by when the trouble ticket got closed e.g., till a spare is on-site.

We observe that most failures are short-lived which occur when the device unexpectedly reloads and then quickly comes back into an operational state. Consider the time-to-repair distribution for these device types shown in Table 5.2: both AGGs and ToRs have a small median time to repair of ≈5-7 minutes. We confirmed this observation from the trouble tickets associated with these events (see *Section* 5.7.1). ARs had the highest mean time to repair of ≈12 hours and a median of 21.5 minutes. This is a surprising result considering that ARs are positioned higher up in the network hierarchy and one expects that their problems get repaired relatively faster. We observed dominant problems related to network modules and switching fabric errors where the module either reloads unexpectedly or exhibits CRC packet errors. Besides line card failures, AGGs also exhibited soft-parity errors that cause the device to transition into an unexpected state. Soft-parity errors occur when the energy level within the chip changes, most often due to radiation (e.g., gamma rays, electro-magnetic interference from a neighboring device, electro-static discharge due to improper handling of the device). When referenced by the CPU, these errors cause the system to crash.

Figure 5.5(c) shows the TTR results for middleboxes. The 95th percentile values show that the distribution has a long tail for both LBs and FWs. Using problems inferred from trouble tickets, we found several incidents for device replacements and ambiguous customer complaints which resulted in an increased TTR. For instance, in several cases, after receiving vague complaints from many customers about occasional slow file-transfer rates between two datacenters, determining the rootcause was challenging as the problem could not be easily reproduced. The rootcause, determined after three weeks, was a software bug that prevented some FWs from recognizing large flows. Across device types, middleboxes exhibit typically a low median TTR, indicating short-lived failures that get resolved within a few minutes to at most a few hours in most cases. The short-lived failures for LBs are due to interface flaps and software problems (e.g, OS watchdog timer triggering a reboot) and the long-lived failures to delays in equipment replacement (due to spare unavailability), faulty spares and problem misdiagnosis. VPNs exhibited mostly link flapping and connectivity errors which get resolved quickly (e.g., protocol convergence, port disabled) resulting in a small TTR.

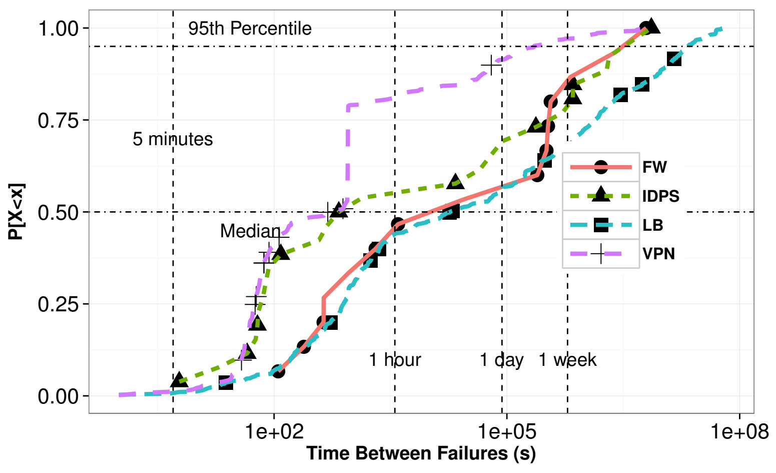
Time-to-Failure: We define the time to failure (TTF) of a device as the time elapsed between two consecutive failures. As this metric requires that a device fail at least twice, we exclude devices having a single failure during our measurement period.

Inter-type: Figure 5.5(a) shows the TTF across middleboxes. We observe that the lines of FWs and LBs nearly overlap exhibiting a high similarity with a median TTF of 7.5 hours and 5.2 hours, respectively. Many FW failures are caused due to connectivity errors, hardware faults and overload conditions (Figure 5.18). Similarly, LB failures are mostly due to short-term transient faults, hardware and parity errors e.g., due to incompatible protocol versions between devices. For IDPS, the median is about 20 minutes while the 95th percentile is around 40 days. This indicates that even amongst devices that fail multiple times, there are two types: (a) robust ones that fail at most once in one or two months and (b) failure prone ones that experience numerous failures (the few bad apples effect), mostly within a few hours. In comparison, VPN devices show relatively the smallest median TTF of about 10 minutes due to connectivity errors (Figure 5.18) and 95th percentile of about three days.

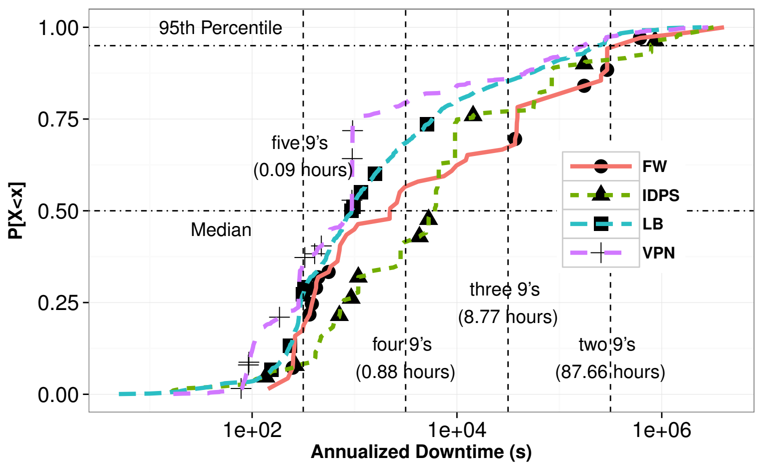
In terms of availability, Figure 5.5(b) shows that FWs exhibit a median of four 9's and a 95th percentile of two 9's. VPNs perform slightly better with their 95th percentile above two 9's. While the median value for LBs is above four 9's, their 95th percentile is two 9's due to all three LB[1-3] generations exhibiting high AFR over the two year measurement period as observed in Figure 5.3.

5.2 Failure Analysis and Modeling

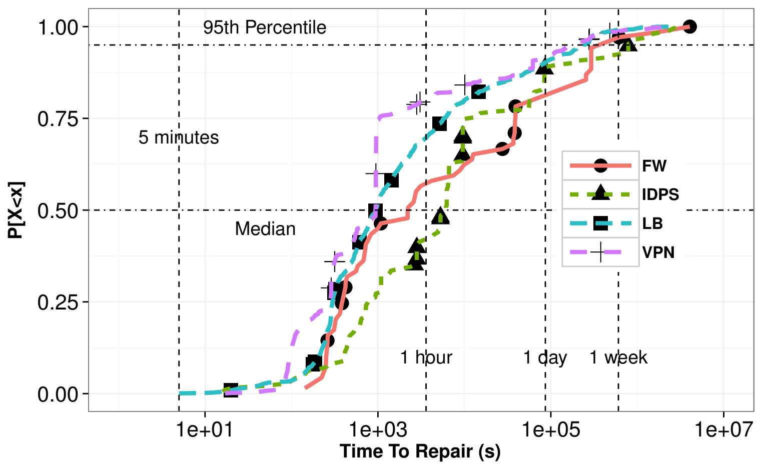
In this section we aim to answer the following key questions: (1) Are repairs effective in fixing problems? (2) Are failures transient or independent? If not, what



(a)



(b)



(c)

Figure 5.5. Comparing (a) Time to Failure, (b) Annualized downtime, and (c) Time to Repair across middleboxes.

Table 5.3
The (conditional) probability of device failures. The last column indicates if the repairs were effective.

Type	**Pr**[1^{st}]	**Pr**[2^{nd}—1^{st}]	**Pr**[3^{rd}— 2^{nd}]	**Eff?**
AR	1 in 4.0	1 in 5.2	1 in 5.8	✓
AGG	1 in 3.7	1 in 6.1	1 in 8.6	✓
ToR	1 in 17.1	1 in 7.7	1 in 5.4	—

are their properties? (3) How soon does a device fail after experiencing a failure? and (4) Does the length of fiber links affect the probability of link failure?

5.2.1 Effectiveness of Repairs in Fixing Failures

We begin by quantifying the probability that a device will fail multiple times in its lifetime based on the observed failure rates. For each device platform, we analyze the conditional probability that a device will fail if it previously experienced one or more failures. Table 5.3 shows the conditional probabilities of successive device failures split by the three device types. An increase in the probability with every subsequent failure indicates that actions taken to "fix" failures are not effective and vice versa. We make the following observations from Table 5.3:

• **Access Routers**: ARs in general exhibit a decreasing trend to fail indicating that the repairs at each failure level are effective. These devices exhibited a 1 in 4 chance of a first failure during the observation period. After a device has failed once, its failure probability decreases by a factor of $\approx$1.5, and the probability continues to decrease with subsequent failures indicating that it is favorable to consider repairing devices of this type when they fail. However, note that for some device generations, repairs may not be as effective e.g., the probability of failure likely increases with each subsequent repair for old devices close to their end-of-life as expected.

• **Aggregation Switches**: AGGs exhibit a decreasing failure trend similar to ARs. However, some old generations of AGGs showed an increased probability of successive failures.

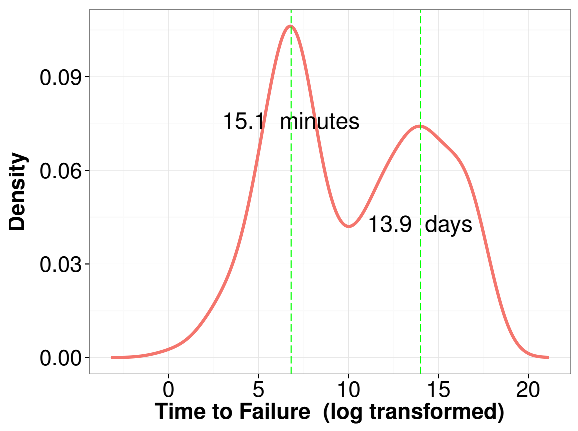
• **Top-of-Rack Switches**: The most surprising result relates to ToRs — they exhibit the worst behavior of increase in failure probability after every subsequent failure. This increase indicates that repairs carried out as a response to failures are not quite effective in mitigating the failure root cause. One likely reason is that ToRs have a low priority for repair and thus, a quick-fix solution (e.g., reboot) typically applied may delay finding the true root cause of the problem.

5.2.2 Modeling Failures

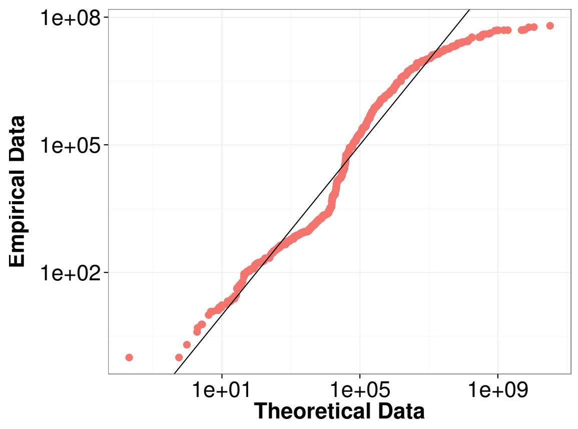
Building statistical models of failures is important as researchers can use them to understand their failure modes, analyze a large number of network design solutions [23, 74, 75], and derive implications to improve fault detection and troubleshooting operations. Although this section discusses the specific case of LBs, note that our analysis can be extended to other types of network devices.

Before modeling the failure distribution, we need to first characterize their statistical properties. As Figures 5.5(a) and 5.5(c) show, the range of TTF and TTR durations can be several orders of magnitude across devices. To reduce this high variance, we use the Box-Cox transformation [76], a family of parametric power transformation techniques that transform data such that it approximately follows the normal distribution. Using this technique, we found that the logarithmic power transform yielded the best normal fit for our failure data which we verified using a Q-Q plot (not shown).

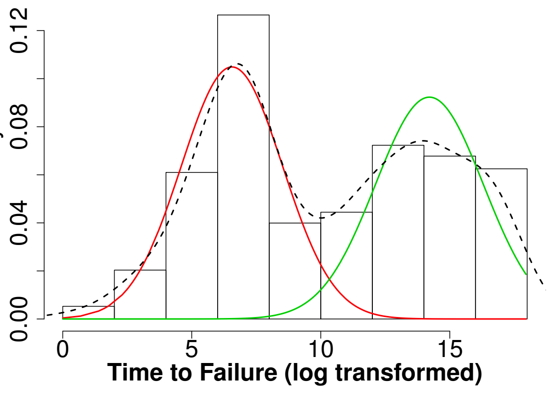
Figures 5.6(a) and 5.7(a) show the kernel density plot of the log-transformed TTF and TTR respectively. We observe that the distributions are right skewed and are clearly not unimodal (observe the peaks). To find these peaks automatically, we use a local peak finding algorithm where a peak is defined as the locally highest value



(a)

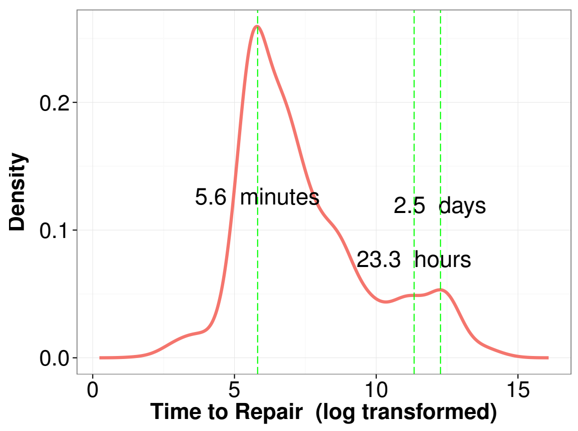


(b)

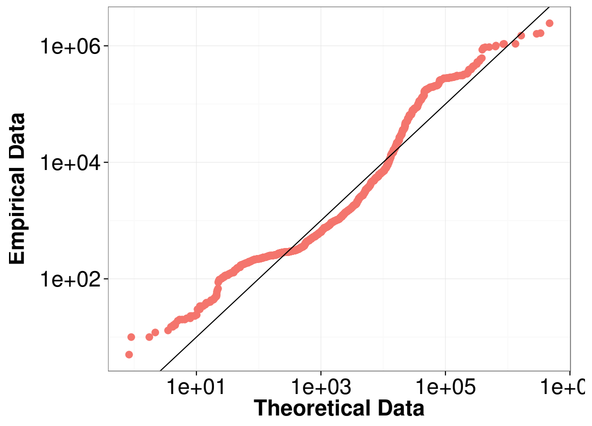


(c)

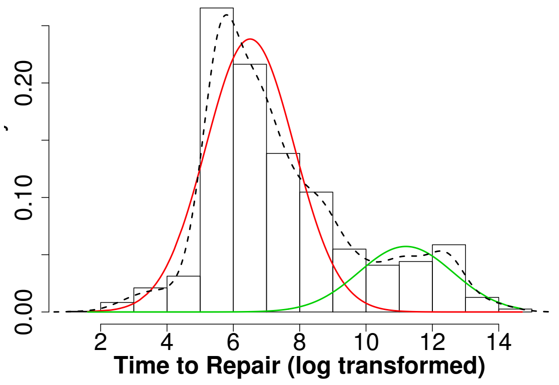
Figure 5.6. Modeling Time to Failure [a-c] for load balancers. Figure (a) shows the kernel density plot of the log transformed TTF. Figure (b) shows fit of a single log-normal distribution. Figure (c) shows fit of a mixture of log-normal distributions.



(a)



(b)



(c)

Figure 5.7. Modeling Time to Repair [d-f] for load balancers. Figure (d) shows the kernel density plot of the log transformed TTR. Figure (e) shows fit of a single log-normal distribution. Figure (f) shows fit of a mixture of log-normal distributions.

in a time series, before a change in signal direction has occurred. Observe that the main peaks occur at 15 minutes for TTF (Figure 5.6(a)) and 5.6 minutes for TTR (Figure 5.7(a)). Further, there are secondary peaks, which are relatively small, at 14 days for TTF and 24 hours and 2.5 days for TTR.

These findings indicate that there are two or three qualitatively different *types* of failures and repairs in effect, respectively. For TTF, the peak at 15 minutes indicates that connection errors and interface flaps dominate middlebox failures, and the secondary peak at 14 days indicates problems due to both hardware faults (e.g., line card, memory errors) and software bugs. For TTR, most repairs take a short time (< 6 minutes), but once the repair window exceeds a day (the 24 hours peak), it will likely take longer e.g., hardware replacement.

Failure Modeling: Based on prior work [74, 77], we first tried to fit existing heavy-tailed distributions to our failure data. However, they did not capture the multi-mode properties of our dataset. For instance, in the case of lognormal distribution, the Q-Q plots in Figures 5.6(b) and 5.7(b) show that the data does not follow the absolute line and hence are not memoryless[2].

Intuitively, the presence of multiple peaks indicates that a two or three-component mixture of right-skewed distributions might be able to provide a good approximation of the process generating the data. We next present the mixture model framework [78] we used. Suppose the real-valued variables $\mathbf{X}_1, ..., \mathbf{X}_n$ are a random sample of time periods from a finite mixture of $m(> 1)$ arbitrary distributions, called *components*. The density of each $\mathbf{X}_i$ may then be written as:

$$h_\theta(\mathbf{x}_i) = \sum_{j=1}^{m} \lambda_j \phi_j(\mathbf{x}_i), \mathbf{x}_i \in \mathbf{R}^r \tag{5.1}$$

where $\boldsymbol{\theta} = (\boldsymbol{\lambda}, \boldsymbol{\phi}) = (\lambda_1, ...\lambda_m, \phi_1, ..., \phi_m)$ denotes the model parameter and $\sum_{j=1}^{m} \lambda_m = 1$. We further assume that ϕ_j are drawn from some family $\mathcal{F}$ of univariate density functions on $\mathbf{R}$. We consider a univariate lognormal family $\mathcal{F} = \{\phi(\cdot|\mu_{ln}, \sigma_{ln}^2\}$, in

[2]Memoryless property indicates that failure inter-arrival times are exponentially distributed

Table 5.4
Parameters for the two-component lognormal mixture distribution for load balancers

Type	λ	μ_{ln_1}	σ_{ln_1}	μ_{ln_2}	σ_{ln_2}
TTF	0.516284	6.557005	1.963986	14.210978	2.090600
TTR	0.428609	6.045474	0.690533	8.524983	2.541691

which μ_{ln} and σ_{ln} denote the mean and standard deviation in log scale. Thus, the model parameter reduces to $\boldsymbol{\theta} = (\boldsymbol{\lambda}, (\mu_{ln_1}, \sigma^2_{ln_1}), ..., (\mu_{ln_m}, \sigma^2_{ln_m}))$. By substituting these parameters, Equation (5.1) can be written as:

$$h_\theta(\mathbf{x}_i) = \sum_{j=1}^{m} \lambda_j \frac{1}{\sigma_{ln_j} \mathbf{x}_i \sqrt{2\pi}} e^{-\frac{(ln(\mathbf{x}_i) - \mu_{ln_j})^2}{2\sigma^2_{ln_j}}}, \mathbf{x}_i \in \mathbf{R}^r \tag{5.2}$$

We obtained a reasonable fit by choosing a two-component ($m = 2$) lognormal mixture, where our model from Equation (5.1) becomes: $\lambda f(\mu_{ln_1}, \sigma_{ln_1}) + (1 - \lambda) f(\mu_{ln_2}, \sigma_{ln_2})$. To obtain the model parameter λ, we use the well-known Expectation Maximization(EM) [79] algorithm. Table 5.4 gives the values of the parameters for the two-component lognormal mixture distribution. We use the two-sample Kolmogorov-Smirnov (KS) test [80] as a goodness-of-fit test to compare data generated from the two-component lognormal mixture distribution with our failure data. We obtain a p-value of 0.111 indicating that we cannot reject the *null hypothesis* that the samples are drawn from the same distribution.

Figures 5.6(c) and 5.7(c) show how this model fits our data; the dotted-line is the kernel density curve of our data and the solid lines are the individual mixture components. We observe that our model closely approximates the real-world data of load balancer failures in our study.

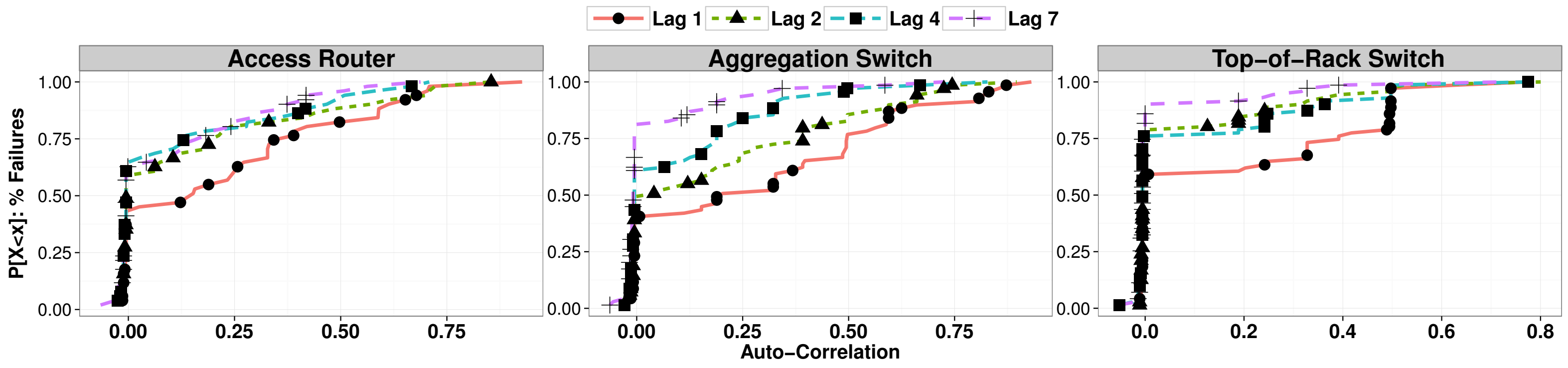


Figure 5.8. Correlations of device failures over a week period for different device types at different time lags (curves towards the top-left show low correlation).

5.2.3 Burstiness of Failures

To understand how quickly devices fail after experiencing a failure, we compute the auto-correlation function for the number of failures observed per device on a daily-basis. For each device, we construct a binary time series as tagging 1 on a day if it exhibited a failure on that day and 0 if the device was functioning normally.

Figure 5.8 shows the CDF of the auto-correlation values for different device types at different lag levels (shift in the time series). ToRs exhibit a short term stable behavior i.e. they do not exhibit any statistically significant correlation with respect to next day failures indicating that fixes deployed are at least temporarily effective. However, over the long term, as described in the previous section, their long-term reliability trends show an increased probability of failure.

ARs and AGGs indicate that for the devices that do fail multiple times (graph omitted), 20%-30% of this population is likely to fail the next day or within a week of getting fixed. We observed that this happens when either the deployed fix is ineffective (e.g., a “reboot” was performed as a quick-fix) or when the root cause was mis-diagnosed (e.g., the supervisor engine was faulty, but the cable was replaced).

5.2.4 Correlation between Fiber Length and Failures

Intuitively, longer the link, higher is the probability of at least one of its components (e.g., segments) to fail. We use the Pearson Product-Moment [81] to analyze if there is a correlation between the number of failures observed on a fiber and #segments per circuit;. Table 5.5 shows the results. Surprisingly, we find that the fiber length has no statistically significant correlation with the number of failures observed. We attribute this result to the fact that these components exhibit high reliability [82] and fail independently (likely due to issues such as construction, rodent bites and under-water fiber cuts). Hence, the overall reliability even for very-long links is not affected.

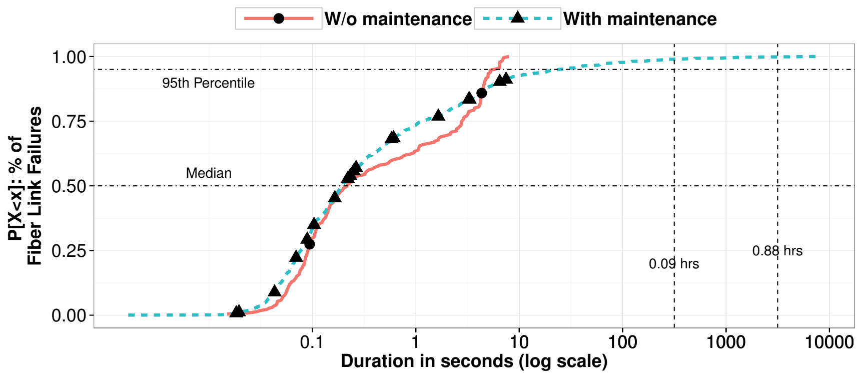


Figure 5.9. Link out per km of long-haul links

Table 5.5
Pearson correlation

Attribute Pair	Pearson Correlation
Distance vs. Segments	0.854
#Failures vs. Distance	0.238
#Failures vs. Segments	0.285

To understand how many seconds a link is down per kilometer, we define *link out per km* as the ratio between the duration that a link is down to the length (in kms) of the fiber involved in the failure event. Figure 5.9 shows the distribution of this metric for *all* fiber link failures and the ones not due to maintenance. We observe a median link out time of 0.4 seconds/km and a 95P value of 5.9 seconds/km for the latter and ≈26.7 seconds/km for the former with a long tail for maintenance events. Computing the Pearson correlation between fiber length and link out per km yielded -0.13 for all events with maintenance and -0.16 for impactful events indicating that they are not statistically correlated.

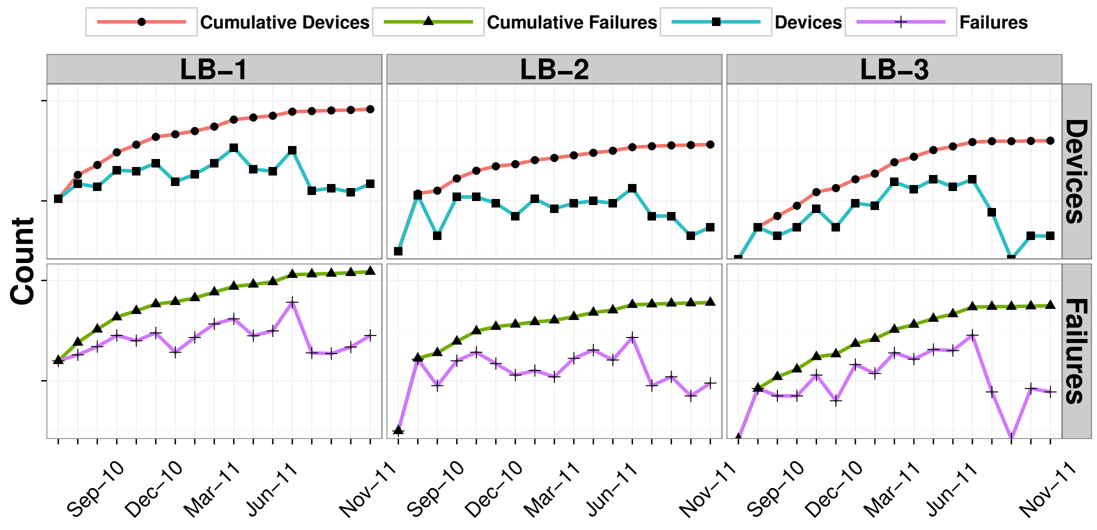


Figure 5.10. Failure trends across different generations of load balancers. The y-axes are unlabeled for confidentiality.

5.3 Failure Trend Analysis

In this section we analyze failure trends across different device generations of middleboxes. We use trend analysis as it is a useful tool to uncover outliers i.e., set of devices which are more failure-prone than the rest of the population. We present detailed case studies on LB[1-3].

Figure 5.10 shows the timeline for the number of failure events and the number of devices that failed; the Y-axis is on log scale. Each subplot shows two curves: a cumulative curve that sums up the number of failures (or devices that failed) across time and the other showing the actual number of failures (or devices that failed). For LB-1, the number of failure events far exceeds the number of devices that failed indicating that a few bad devices are responsible for a majority of their failures.

Validation. We validate this inference by computing two statistical metrics from the failure distribution (Figure 5.11). (1) skewness [83], an indicator used in distribution analysis as a sign of asymmetry where positive values indicate a right-skewed distribution and vice versa, and (2) COV. The observed skewness values are 2.2 (LB-

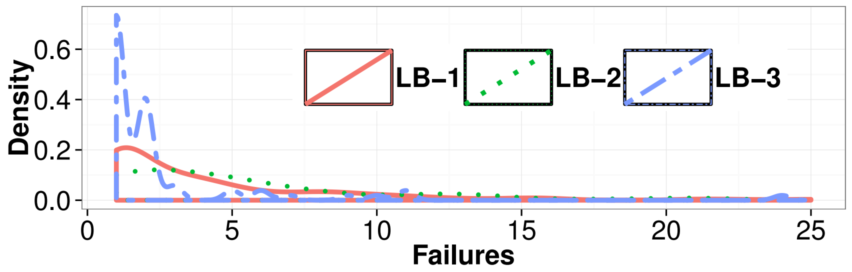


Figure 5.11. Distribution of load balancer failures.

1), 1.5 (LB-2) and 3.9 (LB-3), indicating that the distributions for LB-1 and LB-3 are right-skewed (tail on the right). Though the COV across all LBs is 0.6 (Table 5.1), the COV for LB-1 is 5.8 whereas it is 1.3 for LB-3 indicating that even though both have a right-skewed distribution, the "few bad devices" effect is higher in LB-1. Further, in Figure 5.10, the slope of the cumulative curve indicates the rate of increase of the observed value. Sudden increase in the slope indicates major problems during that period of observation. A key root cause of this behavior is connectivity problems as observed in Figure 5.19.

Combining failure rates with trouble tickets (approach described in detail in Chapter 4), trend analysis revealed two key issues:

1. **Early problem symptoms:** Inference on the problems observed in trouble tickets for LB-1 devices revealed a recurring PSU problem. Aggregating problem statistics across LB-1 tickets indicated that 46% of the tickets called for a device replacement due to a narrow focus of operators in mitigating a problem (compared to its diagnosis) and limited tools at their disposal in checking vendor-specific problems. In retrospect, a simple approach of aggregating root causes across a device generation can identify the problem patterns. In this case, failing PSUs could have been used as early indicators of problems with the product line. A *Cost of Ownership* analysis (*Section* 7) can then help decide whether to continue using LB-1 or gracefully upgrade it to a newer, more stable generation.
2. **The few bad apples affect:** LB-1 suffers from the *few bad apples* effect i.e., the rate of increase of the slope of the cumulative count of failures curve is higher

than the rate of increase of the slope of the cumulative count of devices curve. This indicates that only a few devices are contributing to a majority of failures and is validated by high COV value for LB-1 of 5.8. There were several reasons why this happened. First, we observed a re-occurring fault using LB-1 trouble tickets with the regulated PSU, despite being redundant, that frequently caused power fluctuations leading to damaged parts. Even if the damaged parts are replaced, unless the fault is localized to the power supply unit, the problem likely will repeat itself. Second, a memory bit corruption (e.g., SDRAM ECC error) kept being masked temporarily by the quick-fix solution of reboots as observed in Figure 5.20.

These observations can be leveraged to (a) prioritize replacement of frequently failing devices and scheduling of repair actions based on the top-k observed problems, and (b) aid repair vs. replace analysis by combining early problem symptoms with a cost of ownership metric. Finally, frequently occurring hardware problems can aid in detecting faults at the vendor.

5.4 Redundancy Analysis

5.4.1 Estimating Traffic Loss

Quantifying the impact of a failure is difficult as it requires attributing discrete “outage” levels to annotations used by network operators such as *severe, mild*, or *some impact*. To circumvent this problem, we correlate the network event logs with link-level traffic measurements to estimate the impact of a failure event. However, it is still difficult to precisely quantify how much data was *actually* lost during a failure because of several complications: (i) traffic rerouted via alternate routes in datacenters, (ii) temporal variations in traffic patterns, and (iii) grey failures.

Building upon our prior work [20], we *estimate* the failure impact in terms of lost network traffic that would have been routed on a failed link in the absence of the failure. Specifically, we first compute the median number of bytes on links connected

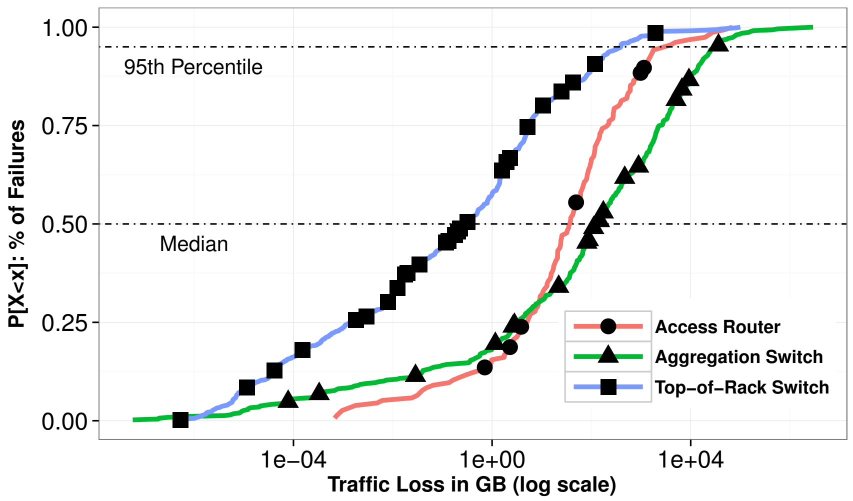


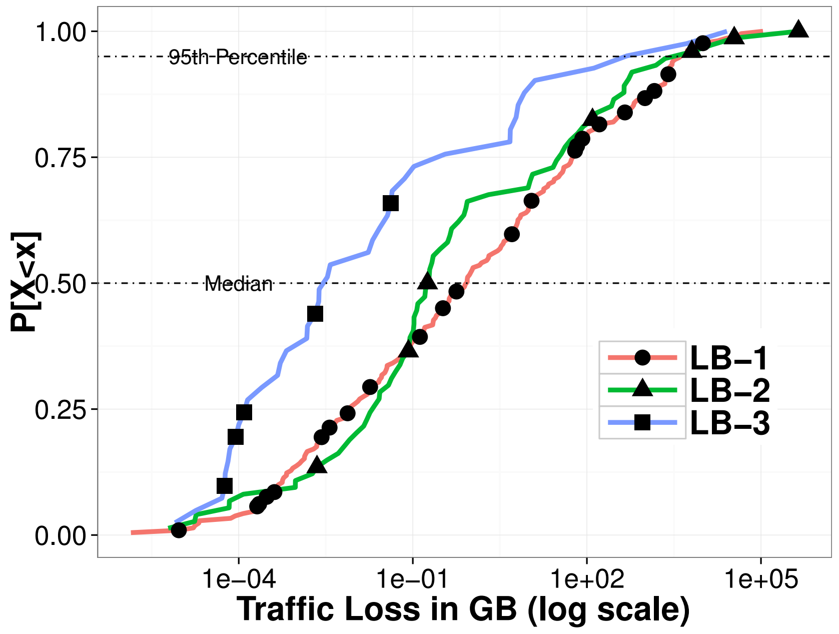
Figure 5.12. Distribution of the estimated traffic lost per day across ARs, AGGs, and ToRs. The x-axis is on log scale.

to the failed device in the time period preceding the failure, Med_b, and the median number of bytes during the failure, Med_d. Then the estimated median traffic loss per day for a failed device can be defined as:

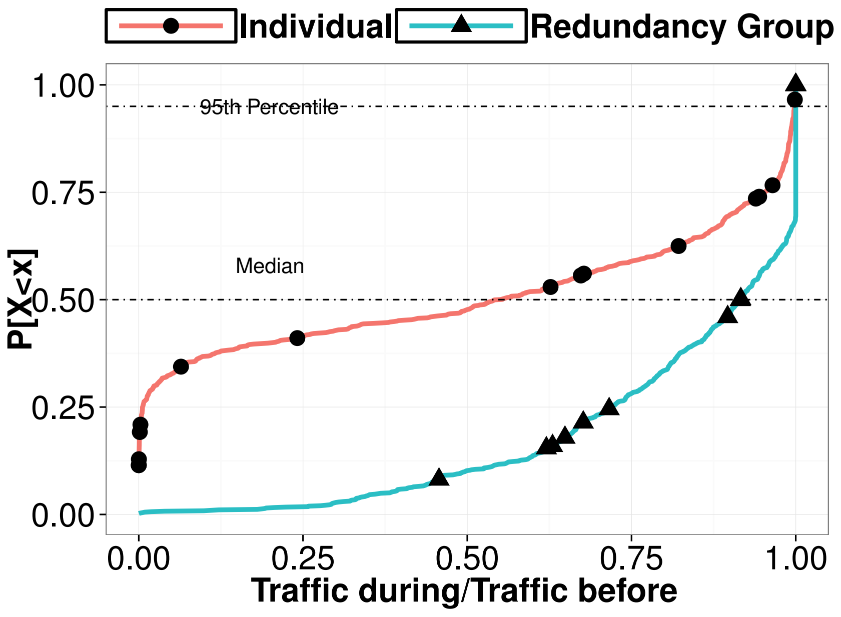
$$\sum_{\forall\ events\ \in\ day} (Med_b - Med_d) \times failure\_duration \tag{5.3}$$

where $failure\_duration$ denotes how long the device/link failure lasted.

Figure 5.12 shows the CDF of estimated traffic loss per day across ARs, AGGs and ToRs computed using Equation 5.3. We observe that the median number of bytes lost during failures is highest for AGGs (130 GB/day), but it is significantly less for ARs (38 GB/day) due to the “few bad apples effect” exhibited by AGGs as observed in *Section* 5.1.1. In comparison, the median traffic loss per day for ToRs is less than 1GB/day likely due to (a) relatively small bandwidth capacity compared to AGGs and ARs, (b) low downtime values (from *Section* 5.1.1), and (c) low traffic utilization at the ToR layer in the network hierarchy [20].



(a)



(b)

Figure 5.13. (a) Estimated traffic loss during failure events for LBs, and (b) Normalized bytes (quartiles) during failure events per device and across redundancy group compared across LBs.

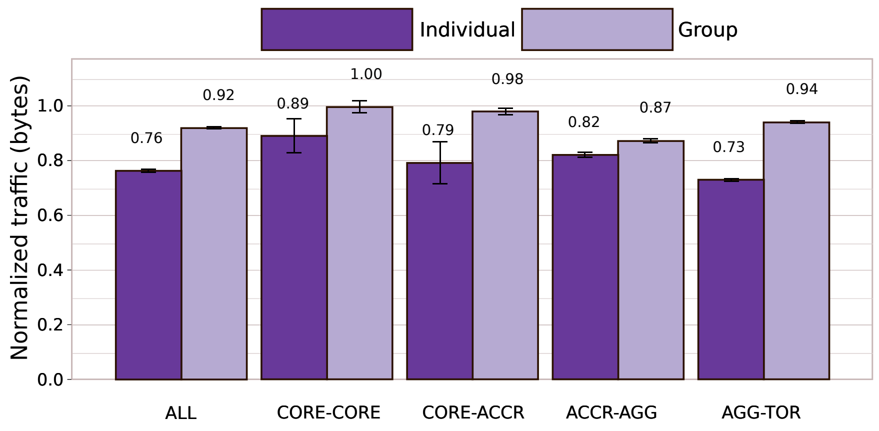
Figure 5.13(a) shows the estimated traffic loss across the three LB generations. We observe the estimated median number of bytes lost during failures is about 1 GB for LB-1 (exhibiting a relatively high failure rate as seen in *Section* 5.1.1 and *Section* 5.3) while it is close to 5 MB for newer LB-3 generation devices, which have a low failure rate. Notice that LB-2, which also exhibited high failure rates compared to LB-3, shows a median traffic loss close to LB-1.

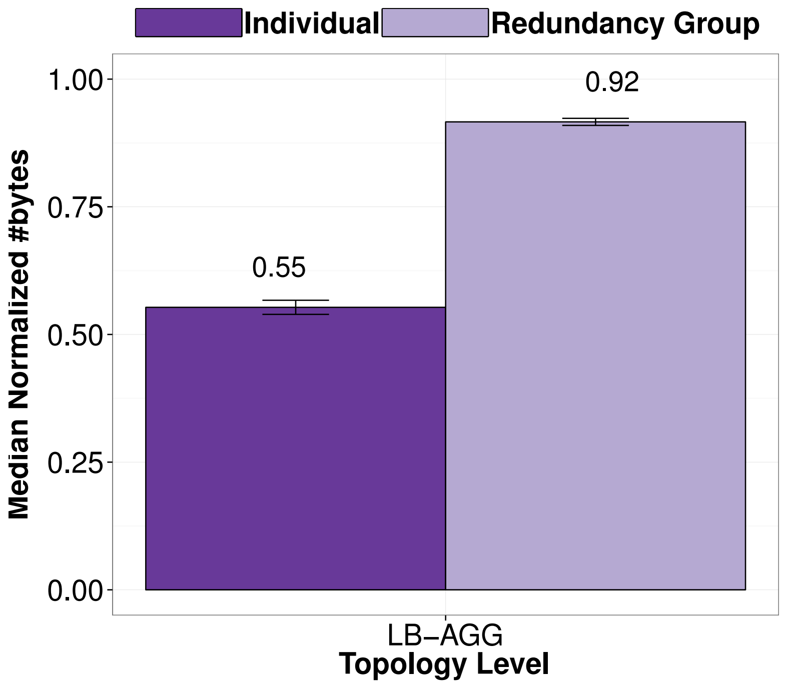
5.4.2 Analyzing Network Redundancy Effectiveness

We next analyze the effectiveness of network redundancy in mitigating the failure impact as it is a *de-facto* resiliency mechanism to handle faults in datacenters. Within a redundancy group, one device is typically designated as active (primary) and the rest as standbys (backups). Other configurations are possible such as active-active pairs where both the devices carry traffic simultaneously. We observed several large redundancy groups comprising up to tens of Aggregation switches connected to a pair of Access routers in the network datacenter hierarchy.

To estimate the effectiveness of network redundancy, we first compute the ratio of median traffic (bytes) entering a device across all links during a failure and the median traffic entering the device before the failure, and then compute this ratio across all devices in the redundancy group where the failure occurred. Network redundancy is considered 100% effective if this ratio is close to one across a redundancy group. We refer to this ratio as *normalized traffic*.

Figure 5.14(a) shows the normalized traffic (bytes) at each layer of the datacenter network hierarchy. Overall, the median traffic carried at the redundancy group level is 92% compared with 76% at the individual level, an improvement of 21% in the overall median traffic as a result of network redundancy. While this redundancy is more effective at the AGG-ToR level (an improvement of 28.7%) and CORE-ACCR level (an improvement of 24%), it is relatively less effective at the ACCR-AGG level (an improvement of 6.09%). The maximum gain is observed at the CORE-CORE





(b)

Figure 5.14. (a) Normalized traffic (bytes) for failure events for individual devices and their redundancy groups and (b) Normalized traffic (bytes) during failure events for a device as well as within a redundancy group (inter-datacenter) layer where failures are completely masked due to redundancy. One reason is that network layers close to the root carry significant service traffic and connect application servers inside datacenters to users across the Internet. Hence, these layers are monitored closely by network operators to enable fast failure detection and troubleshooting in order to minimize service downtime.

Evaluating LB redundancy effectiveness. To estimate the effectiveness of LB redundancy, we first compute the ratio of median traffic on a device during a failure and its median traffic before the failure, and then compute this ratio across all devices in the redundancy group where the failure occurred. Network redundancy is considered 100% effective if this ratio is close to one across a redundancy group. We refer to this ratio as *normalized traffic* [20].

Figures 5.14(b) and 5.13(b) show the distribution of normalized traffic for individual LBs and their redundancy groups. We observe that the redundancy groups are effective at moving the ratio close to one with 40% of the events experiencing no failure impact at the redundancy group level. The median traffic carried at the redundancy group level is 92% compared with 55% at the individual level, a relative improvement of 67.2% in median traffic as a result of middlebox redundancy.

Evaluating redundancy effectiveness for other middleboxes. To analyze redundancy effectiveness for FWs, IDPSes, and VPNs, we measure the fraction of overlapping failure events where both the redundant pairs were down simultaneously. In particular, after one of the pair fails, we check if the other device also undergoes a failure before the former becomes operational. Our analysis of redundancy effectiveness across FWs, IDPSes and VPNs revealed that both the redundant pairs failed in 33% and 1% of the failure events for FWs and IDPS, respectively. In comparison, VPN redundancy exhibits 100% effectiveness in mitigating failures.

5.5 Network Stamp Availability

To provide a network redundancy effectiveness service-level-agreement (SLA) of a geo-distributed service, a key requirement is to analyze the network redundancy effectiveness of a network stamp hosting the service and then compute the number of network stamps needed to meet the service SLA. Recall that in Section 5.4.2 we already computed the overall effectiveness of redundancy r. We can leverage this finding to determine the minimum number n_{min} of independent network stamps needed to meet a desired SLA of a service. In particular, we solve for the minimum integer n that satisfies the following equation:

$$1 - (1 - r)^n \geq SLA_{desired}$$
$$\Rightarrow log(1 - SLA_{desired}) \geq nlog(1 - r)$$

As $log(1 - r) < 0 \; \forall \; r \in (0, 1)$, dividing by $log(1 - r)$ on both sides we get:

$$n \geq \frac{log(1 - SLA_{desired})}{log(1 - r)} \tag{5.4}$$
$$\Rightarrow n_{min} = \left\lceil \frac{log(1 - SLA_{desired})}{log(1 - r)} \right\rceil \tag{5.5}$$

where $\lceil\rceil$ is the ceiling function. For instance, to provide 99.9% network redundancy effectiveness, solving the above equation yields $n = 3$. Similarly, for 99.99% network redundancy effectiveness we get $n = 4$. Note that as expected, setting n to higher values would likely yield diminishing returns in improving service reliability.

5.6 Capacity vs. Availability

In network design, a conventional approach is to adopt a multi-layer architecture that breaks up the network into small, more manageable Layer-2 domains and then use a spanning tree to provide redundancy and network load sharing. The size of these domains is kept small to reduce the delays in spanning-tree convergence (which

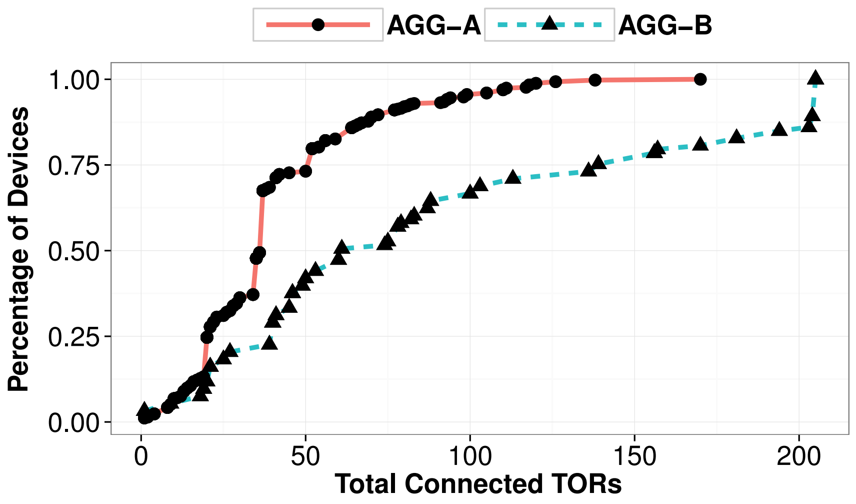


Figure 5.15. Distribution of the number of ToRs connected to two AGG platforms.

is sensitive to the network diameter). Due to the Layer-2 topology, the network can scale at this layer simply by adding more switches. However, the drawback is that scaling Layer-2 domains increases the *fault domain size* e.g., a broadcast storm caused by a malfunctioning device or human error can cause failure of the entire sub-tree. Further, to avoid loops, all links cannot be in a forwarding state at all times because broadcast packets risk saturating the VLAN, thereby adversely affecting the network performance.

This raises a fundamental question of *scale-up* vs. *scale-out* for operators to deliver services in a cost-effective manner: *Do we deploy high-density, expensive Aggregation switches that can provide connectivity to hundreds of ToRs or leverage small port-count, low-cost commodity switches (perhaps with lower reliability), but deploy them in large numbers?* Specifically, we aim to analyze how does the availability of Layer-2 AGGs depends on the number of ToRs connected to it.

Figure 5.15 shows the distribution of the number of ToRs connected to two platforms of AGGs, AGG-A and AGG-B, in our dataset. Observe that a small percentage (5%-25%) of devices serve more than 100 ToRs. To understand the tradeoff between capacity and availability, we divide each platform of Aggregation switches into four

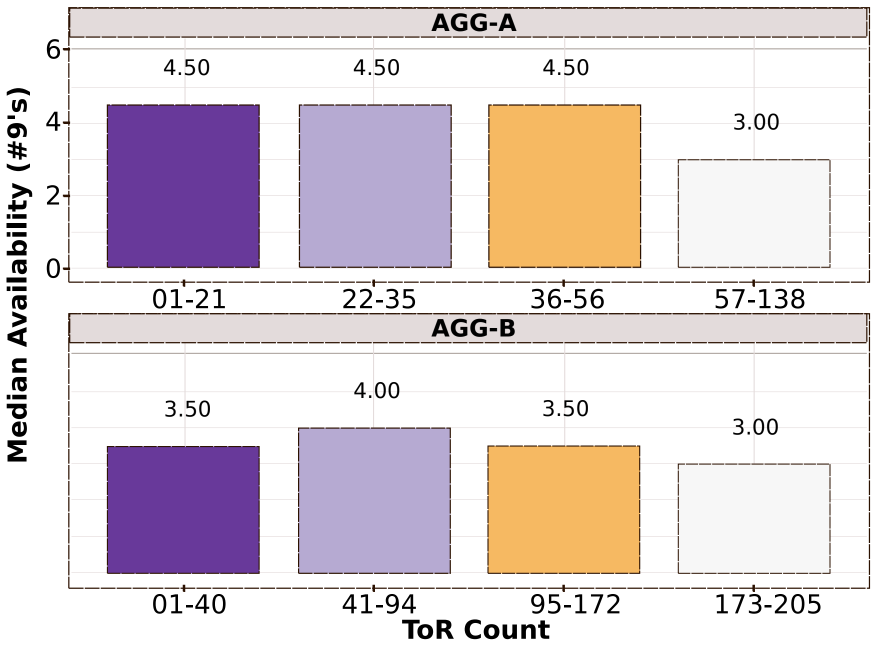


Figure 5.16. Median availability based on the number of ToRs connected to the two AGG platforms.

categories based on quartiles on the ToR count (i.e., the first quartile forms the first category and so on). In each category, we compute the availability of the population based on the number of logged failure events.

Figure 5.16 shows the median availability of AGGs based on the number of ToRs they are connected to. Notice that the category-1 devices for AGG-A exhibit 4.5 9's availability. By analyzing the trouble tickets of AGG-A devices in this category, we observed that the time to troubleshoot their problems was the lowest likely due to small number of ToRs resulting in high availability. For AGG-B, the availability in this category was relatively less of 3.5 9's due to higher time to debug platform-specific problems and relatively a wider range of up to 40 connected ToRs compared to AGG-A. f For both platforms, category-2 exhibited about the same availability as category-1 with a slight increase for AGG-B value of 4 9's. We also observed similar results for category-3 where AGG-A exhibited 4.5 9's availability while AGG-B exhibited 3.5 9's availability. We observed from the tickets of AGG-B devices in this category (having a ToR count range of 95-172) that the AGGs were being provisioned with new ToRs which increased the likelihood of a failure during troubleshooting e.g., due to operator mistakes.

In category-4, we observe the lowest availability of 3 9's across both platforms. Note that many of these devices were long standing in production whose ToR provisioning was close to port capacity. Using the tickets associated with this category, we found that maintenance becomes harder and more error-prone for high ToR counts as also observed for category-3 AGG-B devices. Further, due to these devices operating close to capacity, it was not feasible to find alternate available paths to re-route the entire traffic load in case of a failure. This resulted in additional downtime where the operators had to provision a temporary set of replacement devices and then configure them to route the service traffic.

5.7 Causes of Network Failures

In this section we analyze the root causes of intra- and inter-datacenter network failures.

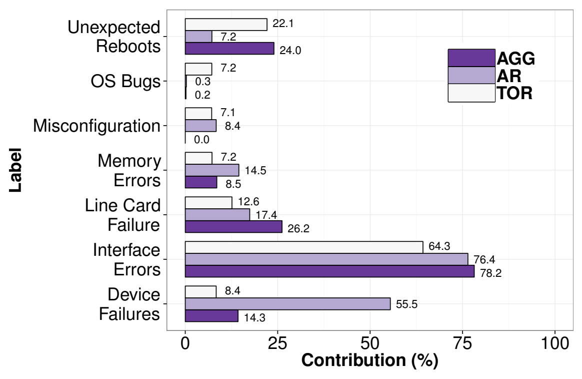
5.7.1 Intra-Datacenter Failures

To determine the failure root causes, we leverage the information recorded by operators in trouble tickets attached to the network events. Specifically, we leverage NetSieve [24], an automated problem inference system that analyzes the free-form text in a trouble ticket to generate its synopsis: (1) the problems observed e.g., link down, misconfigured rule, switch in 'freeze' state, (2) the troubleshooting performed e.g., check cable, track configuration changes, verify BGP routes, and (3) the actions taken for resolution e.g., replaced the supervisor engine, reboot the device, clean the fiber.

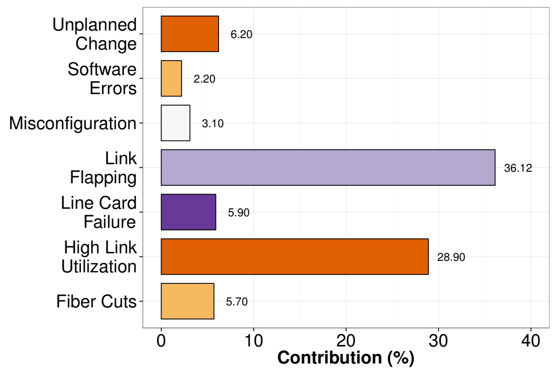
Figure 5.17(a) shows the histogram of the top-k problems observed from trouble tickets associated with intra-datacenter failures. Observe that there is a broad range of problems such as hardware faults (e.g., device failures, memory errors), OS bugs, and misconfigurations (e.g., ARP conflict). Interface-level errors, network card problems, and unexpected reloads are prominent amongst all the three device types. Interface errors usually last for about 5-7 minutes as observed in Table 5.2. During these periods, we observe that the service would be available, but its users may experience high latency or packet drops. For instance, due to interface errors TCP may likely timeout and re-transmit in the slow-start phase thus degrading the service performance. Our discussion with operators revealed there are multiple reasons for interface errors such as faulty cable installation, faulty optical transceivers, and protocol convergence delays. In addition to interface errors and hardware problems, ToR failures were also due to OS-related problems and misconfigurations. For instance, in some cases a specially crafted IPv6 packet (e.g., Type-0 Routing Header [84] packets for source routing) was found to crash the device. In others, certain types of IPv4 packets (e.g., ICMP echo-requests) destined to a physical or virtual interface on the device caused a memory leak.

For middleboxes, Table 5.6 shows the classification of the problems into five categories based on the ticket data and their example causes. Note that there are two potential issues to be considered in interpreting these categories. First, a problem observed in one category can be due to issues in another one. For instance, parity errors in VPNs indicate failed sanity checks which may be caused due to misconfigured pre-shared keys. Another example is that a malformed message could be due to incompatible protocol versions between devices. Second, in some cases, there may be multiple problems pertaining to the same failure. In these cases, we take a conservative approach and attribute each observed problem to its respective category.

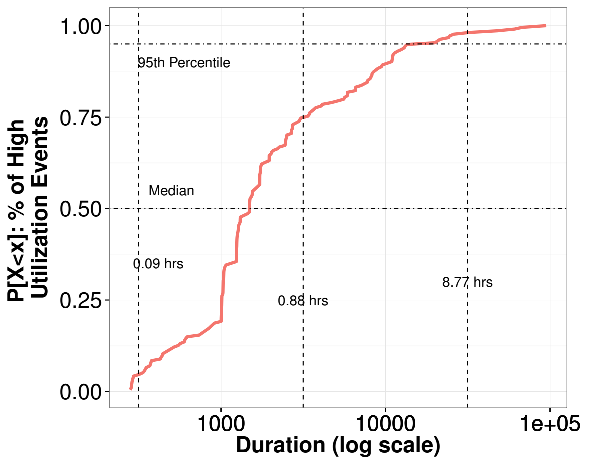
Inter-type: Figure 5.18 shows the breakdown of problems observed across the four types of middleboxes. We find that connectivity errors dominate failures across all middleboxes as also validated from the results in Section 5.1.2. Connectivity errors are mainly interface/link flaps which can be due to many factors such as protocol convergence, line card errors, cpu overload, bit corruption and cable problems. As



(a)



(b)



(c)

Figure 5.17. (a) Intra-Datacenter root cause analysis: Interface errors dominate across all device types. (b) Inter-Datacenter root cause analysis: Link flapping and high link utilization are the major problems. (c) Distribution of the duration of the high link utilization events.

Table 5.6
Classification and examples of problem types observed across middleboxes.

Problem Classification	**Example causes or explanation**
Connectivity errors	Device/neighbor unreachable, link flaps, ARP conflicts, address not found, message/frame parity errors, checksum failed, malformed message, port errors
Hardware	faulty PSU/ASIC, memory error, defective chassis/disk/fan, failed SUP engine/line card
Misconfiguration	bad rule, VLAN misallocation, configuration conflict/out-of-sync/corrupted/missing, mismatched crypto keys, expired security association, incorrect clock causing certificate expiry, incorrect root password
Software	reboot/reload, firmware bug, OS errors
Overload	High utilization exceeding a specified load threshold

a result, these errors can significantly degrade service performance and availability due to TCP timeouts, convergence delays, and retransmissions e.g., due to malformed packets or parity errors. The second main root cause is hardware problems except for VPNs where it accounts for only 1% of the problems. One implication of high percentage of hardware problems is long repair times as the faulty component or the device gets replaced. Another one is that this observation is in direct contrast to the industry view of equating hardware boxes with high reliability.

VPN problems are dominated by parity errors which may be attributed to faulty cables (e.g., copper, SFP fiber optics) and their common root causes are power surge, bad connectors, or a short circuit. For these problems, we observed mainly two repair actions of either cleaning the fiber or replacing them. In comparison, misconfigurations and overload problems are relatively in minority, contrary to the common view of being a dominant contributor [19].

Table 5.6 (row 3) gives the common categories of misconfigurations across device types. While bad rules as a common root cause (≈70% cases) are expected, there are a wide variety of other problems such as VLAN misallocation, corrupted or missing configuration files, mismatched cryptographic keys, and expired certificates.

Across middleboxes, misconfigurations contribute lowest to IDPS problems due to the fact that centralized management in a datacenter continuously monitors and immediately enforces policy changes (e.g., security access) upon violation. Software problems contribute the minimum percentage to problems indicating that the software running on these devices exhibits reasonable stability. Finally, software problems are fixed relatively quickly e.g., via code update or power cycling the device.

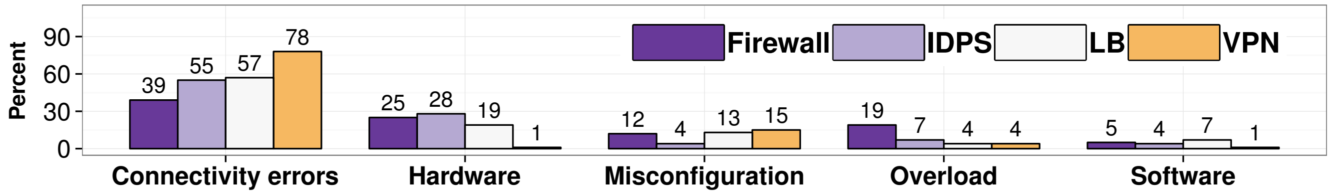


Figure 5.18. Problem types observed across middleboxes.

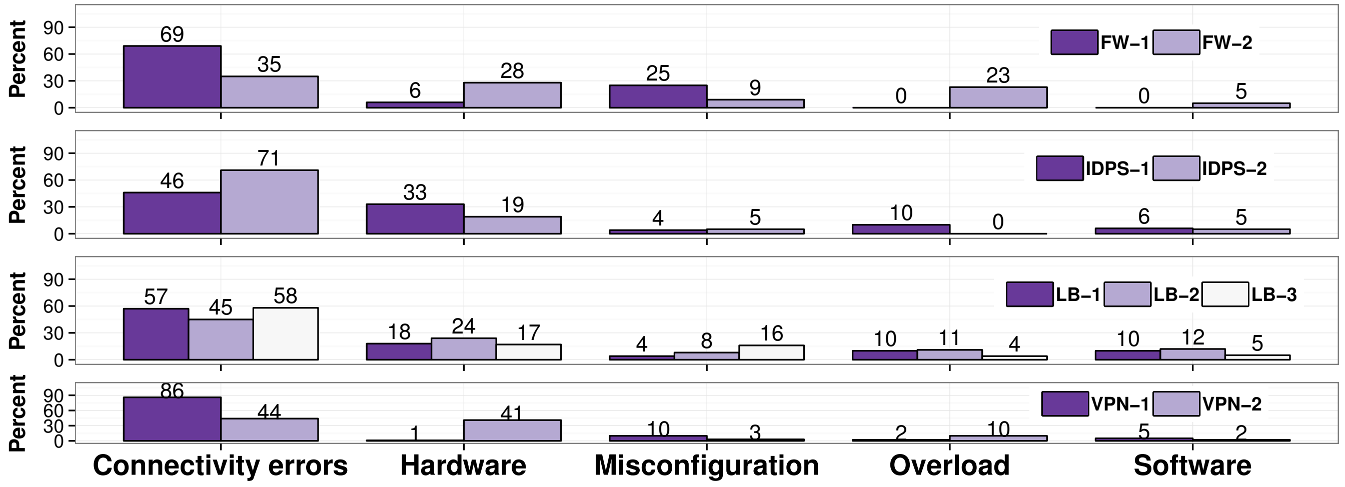


Figure 5.19. Comparison of problem types across different generations of middleboxes.

Intra-type: Figure 5.19 shows the problems observed split by different generations for each of the four types of middleboxes. For Firewalls, FW-1 exhibits lower hardware and software problems compared to FW-2. However, the fraction of misconfigurations and connectivity errors is higher, due to better configuration tools and more reliable port connectors and cables for the new platform FW-2. Surprisingly, even though FW-2 has a higher capacity compared to FW-1, it experiences a high fraction of overload induced failures which are not observed in FW-1. Unlike firewalls, for IDPS devices, the port errors increased with the newer device IDPS-2, while contribution of the other problem types reduced. For load balancers, LB-3 exhibits fewer problems due to hardware and software improvements compared to its predecessors, LB-1 and LB-2. In sharp contrast to trends observed across FWs, IDPSes and LBs, VPN-2 exhibits a higher percentage of hardware faults and overload problems compared to VPN-1 due to the instability of the newer platform.

Overall, these trends provide a fine-grained analysis of what aspects of a new platform are improving over its predecessor and vice versa, to guide researchers and vendors in developing better designs and tools to improve middleboxes.

Figure 5.20 shows the breakdown of actions taken to mitigate middlebox failures. The nine categories show the resolutions applied to faulty components at a fine granularity. For instance, the cable category implies that the cable was either cleaned or replaced. The software category denotes that either a patch was applied or the code was upgraded to the latest stable version. Similarly, the categories of different hardware components such as chassis, disk, fan, and PSU denote that they were replaced using on-site spare or sent back to the vendor for repairs, or warranty purposes.

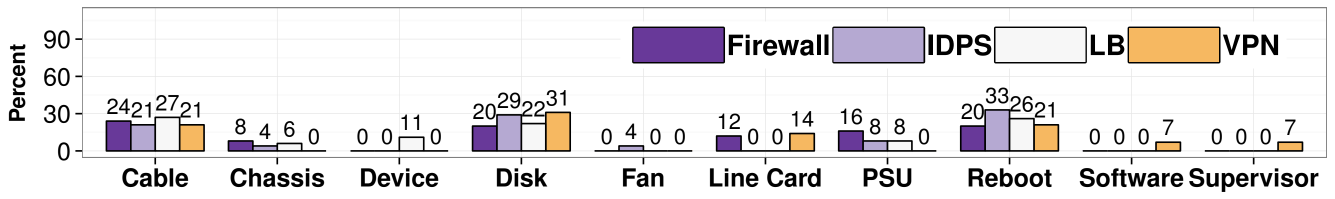


Figure 5.20. Resolution actions observed across middleboxes.

We observe that across middleboxes, the most prevalent actions taken are replacing the cable and disks, and reboot to fix software-related issues. The former can be attributed to the relatively low-cost and high failure rates of these commodity components as operators tend to replace them quickly after observing a problem. In comparison, the decisions to replace specialized components such as chassis, supervisor engines, or even the device, requires a detailed investigation due to their high cost and long turnaround time from their vendor. The reason for a high percentage of reboots is that it is a quick-fix solution in many cases and hence run as a popular first attempt to mitigate a problem to reduce downtime.

5.7.2 Inter-Datacenter Failures

We analyze the network trouble tickets associated with Inter-Datacenter link failures and found that link flapping (e.g., due to BGP, OSPF protocol issues and convergence) dominate the problem root causes (36%) as seen in Figure 5.17(b). Due to optical protection configured in some areas of the network, a physical layer problem might end up triggering an optical re-route — a technique to reduce the bandwidth loss by shifting existing lightpaths to new wavelengths, but without changing their route. However, it incurs a control overhead (of about 10 seconds), and more importantly, it risks a service disruption in the rerouted lightpaths [85]. Depending on the protocol timers, such an event is observed as a "link flap", yet the true underlying cause could be an optical re-route, possibly in response to a fiber cut (e.g., due to construction, bullet hole, vandalism, shark attack on under-sea cables). Therefore, it may not be possible in some cases to attribute the exact root cause. The second major root cause is high link utilization (29%). However, note that high utilization does not necessarily imply a physical circuit failure or take-downs, but it may be an indicator of packet errors; we analyze them in detail next. Software errors, misconfigurations, and unnotified maintenance were observed, but they did not constitute a significant fraction of root causes.

Next, we analyze the properties of links with high utilization. By examining trouble tickets associated with high utilization links, we observe that: (1) the percentage of error packets exceed the error threshold specified by operators, and (2) the traffic utilization of the link is above the specified utilization threshold. When there is insufficient capacity during a congestion period, LSP[3] re-routing occurs during which the traffic is switched to an alternate LSP set up after the failure. The new route is selected at the LSP head-end (router that requested the LSP establishment) and may reuse intermediate nodes in the original route. This alternate route maybe computed either on demand or pre-computed, and stored for use when a failure is reported. There is, however, a risk that the alternate route may become out of date due to other changes in the network. This can be mitigated to some extent by periodic recalculation of idle alternate routes. In practice, about 10-25 minutes are allowed for LSP re-routing. However, we observe in Figure 5.17(c) that the average incident duration is about 1.27 hours while the median is 0.41 hours with the 95P value >3 hours, which is about 2x-3x higher than the expected value. Further, 80% of the high utilization events took more than 15 minutes to resolve and 50% of the events took more than half an hour.

Longer downtimes can be attributed to scenarios where LSP re-routing had no alternate routes to compute on demand which caused higher switching times. Long-lived congestion is expected when there is a surge in traffic demand that is relatively long-lived, and the bottleneck is not the transit capacity but the capacity at a source or sink for the traffic. This is believed to happen when there is a significant outage with lack of sufficient redundant capacity. Although it was not possible to pin point the exact root cause behind the longer downtimes in our dataset, one potential cause from discussion with operators was attributed to congestion. Rather than the entire circuit going down under congestion, the links logged errors continuously (a significant

[3]In MPLS wide-area networks, data transmission occurs on label-switched paths (LSPs). LSP is a path through the network from an ingress to an egress router established through the distribution of labels (using label distribution protocol (LDP) or piggybacked on routing protocols like BGP) that define hop-by-hop forwarding.

fraction of packets were being discarded) for long periods of time which inflated the event duration. Note that these events logged as "failure events" do not imply physical circuit breaks but they risk performance degradation due to congestion. The packet discards indicate that during the congestion period, LSP rerouting did not have an alternate path with sufficient capacity to forward the excess traffic. There are many reasons why congestion could arise:

- **Product-related**: During product migration (cloud service changing datacenters) or launch (new software being released), there is a high volume of traffic.
- **Traffic Shift**: Workloads arising due to unexpected service outages or bulk data transfer e.g., web crawl documents, periodic data backups.
- **Port-name Cleanup**: As part of improving network meta-data consistency and integrity, operators perform port-name cleanup on a regular basis. If these changes are not reflected higher up in the topology, then routes may not be available on demand.

5.8 Real-World Impact

NetSieve has been deployed in the cloud provider we studied to enable operators to understand global problem trends instead of making decisions based on isolated incidents. Further, NetSieve complements existing tools (e.g., inventory db) by correlating device replacements with their failures and problem root causes. A simple alternative to mining tickets using NetSieve is to ask operators for explicit feedback e.g., to build Table 4.8, but it will likely be biased by anecdotal or recent data.

Currently, our NetSieve prototype supports SQL-like queries on the inference output. For instance, "SELECT TOP 5 Problems FROM InferenceDB WHERE DeviceType = 'Load Balancer' would output the top-5 failure problems observed across load balancers. Next, we present how NetSieve has been used across different teams to improve network management.

Network Architecture: This team used NetSieve to compare device reliability across platforms and vendors. In one instance, NetSieve showed that a new generation of feature-rich, high capacity AR is half as reliable as its predecessor. In another instance, it showed that software bugs dominated failures in one type of load balancers. Based on grouping tickets having Problem inference of Switch Card Control Processor (SCCP) watchdog timeout for LB-2, NetSieve showed that hundreds of devices exhibited reboot loops due to a recurring software bug and were RMAed in 88% of the tickets.

Capacity Planning: This team applied NetSieve to analyze cases when network redundancy is ineffective in masking failures. Specifically, for each network failure, we evaluate if the redundancy failover was not successful [20] and then select the corresponding tickets. These tickets are then input to NetSieve to do problem inference which output the following to be the dominant problems:

1. **Faulty Cables**: The main reason was that the cable connected to the backup was faulty. Thus, when the primary failed, it resulted in a high packet error rate.
2. **Software Mismatch**: When the primary and backup had mismatched OS versions, protocol incompatibility caused an unsuccessful failover.
3. **Misconfiguration**: Because operators usually configure one device and then copy the configuration onto the other, a typo in one script introduced the same bug in the other and resulted in an unsuccessful failover.
4. **Faulty Failovers**: The primary device failed when the backup was facing an unrelated problem such as software upgrade, down for scheduled repairs, or while deploying a new device into production.

Incident Management and Operations: The incident management team used NetSieve to prioritize checking for the top-k problems and failing components while troubleshooting devices. The operation team uses NetSieve to determine if past repairs were effective and decide whether to repair or replace the device. Table 4.8

shows NetSieve's inference output across different device types.We observe that while Problems show a high diversity across device types such as packet errors, line card failures and defective memory, verifying cable connectivity is a common troubleshooting activity, except for Firewalls and VPNs where operators first verify device configuration. For Actions related to Firewalls, VPNs and ToRs, the devices are first rebooted as a *quick fix* even though the failing components are likely to be bad cable or OS bugs. In many cases, we observe that a failed device is RMAed (sent back to the vendor) which implies that the network is operating at reduced or no redundancy until the replacement arrives.

Complementary to the above scenarios, NetSieve inference can be applied in several other ways: (1) prioritize troubleshooting steps based on frequently observed problems on a given device, its platform, its datacenter, or the hosted application, (b) identify the top-k failing components in a device platform and resolving them with their vendors, and (c) decide whether to repair, replace or even retire a particular device or platform by computing a total cost-of-ownership (TCO) metric.

5.9 Summary

This chapter presents one of the first large-scale study of cloud network failures at both intra-datacenter and inter-datacenter layers. We find that network failures cause significant impact to cloud services, dominated by connectivity loss problems and service errors. The main takeaways from our study are: (1) A service hosted on a single, large network stamp may risk low reliability because network redundancy is least effective at the AR-AGG layer. To improve reliability, build a small number of independent network stamps — three for 99.9% and four for 99.99% network redundancy effectiveness; (2) Network redundancy is most effective at the Inter-datacenter level. However, long-haul links exhibit 2x-3x higher downtime than expected under high utilization; (3) Scale-out switches with low to medium port density may deliver relatively higher availability in comparison to their expensive, higher capacity counterparts; (4) Network device failures are not memoryless and exhibit the "few bad apples" effect; techniques such as regression analysis and trend analysis can be leveraged to identify and troubleshoot the most failure-prone devices; (5) Interface errors, hardware failures and unexpected reboots dominate the problem root causes; and (6) Top-of-Rack switches exhibit an increase in probability of a successive device failure after repair motivating the need to develop automated correlation and diagnosis techniques. We hope that our work sheds light on answering several key questions to improve network reliability for geo-distributed services.

6 RELATED WORK

This dissertation contributes one of the first large-scale study of failures of network stamps from a service perspective, and the first study characterizing failures on inter-datacenter links in a cloud service provider. Several of our analyses have not been previously examined such as computing the number of independent network domains to meet a particular SLAs, checking whether network failures are memoryless or bursty, analyzing how does Aggregation switch port capacity impact their reliability, finding correlation of fiber length with failure rate, and studying high-utilization of inter-datacenter links.

Our work mainly falls into research relating to failures in datacenters [18, 32, 86–89], fault localization [90–93], and root cause analysis [94–98]. In this chapter, we cover related work in each of these areas in detail.

6.1 Failures in Datacenters

Datacenter Failures: Failures in datacenters, in general, have received significant attention in the recent years [18, 32, 86–89]. Our event processing methodology in *Section* 3.2 draws some similarity with the analysis carried out by Turner et al. [18] and Gill et al. [20]. Turner et al. [18] used router configurations, syslog and email records to analyze network failures in the CENIC network. Gill et al. [20] study intra-datacenter network failures in a large cloud provider, but they do not analyze reliability of network stamps from a service viewpoint or characterize failures of inter-datacenter links. Sherry et al. [19] conduct an operator survey of middlebox failures across several enterprise networks.

Inter-Datacenter Links: Much of the earlier work in analyzing inter-datacenter links focused on characterizing traffic flows [99, 100], providing bandwidth on demand [101] or optimizing traffic flows [101, 102]. Laoutaris et al. [99] present a system for bulk data transfer over wide area that employs a network of storage nodes and uses a store-and-forward algorithm to schedule data transfers. Chen et al. [100] characterize inter-datacenter traffic using anonymized NetFlow datasets collected at the border routers of Yahoo! data centers. They find that peak traffic volumes between datacenters are dominated by non-interactive, bulk data transfers.

Mahimkar et al. [101] present a globally reconfigurable photonic network between datacenters that improves operational flexibility by providing a bandwidth-on-demand service. Li et al. [102] present a scheduling scheme that considers both bandwidth utilization and ISP friendliness to reduce the inter-domain traffic. Unlike most work in this area, we focus primarily on analyzing long-haul link failures and studying properties of inter-datacenter links with high utilization. Perhaps, the closest work to ours is by Kandula et al. [103], who analyze when and where congestion happens inside datacenters but they do not consider inter-datacenter traffic. In an extended abstract [104], we performed a preliminary analysis of the causes behind intra- and inter-datacenter network failures.

There has also been a large body of work on improving performance in Content Distribution Networks (CDNs). In these systems, different clients are redirected to different servers in an attempt to provide load balancing and to improve client performance. Several research efforts focused on evaluating CDN performance [105–109] and optimizing load balancing [110, 111]. Our work is largely complementary to these efforts in two ways. First, we find high utilization to plague inter-datacenter links. To mask service outages and to avoid overload at any datacenter site, one approach is to assign a URI to the cloud service whose DNS resolution maps to a set of globally distributed monitoring servers e.g., in a content distribution network. These servers can then dynamically re-route requests to the edge datacenter hosting the service selected either based on latency [110] or load balancing approaches that depend on apriori information about the underlying network [111]. Second, our work on failure characterization of inter-datacenter links can help in deciding what and how services should be deployed across multiple sites (including datacenters and CDNs), what caching and load-balancing strategies [100, 112] should be employed, and how to manage the traffic in the wide-area network backbone connecting the data centers to optimize performance and minimize operational costs [100, 112].

Hardware Failures: There have been several prior studies on hardware failures (e.g., [11–16] and the references therein), but they do not consider network failures in datacenters. Our findings on conditional failure probability relate to the findings in recent studies on DRAM errors [15] and disk-subsystem failures [11]. By contrast, we found diverse trends across different device types in our dataset which could be leveraged in a data-driven approach to decide whether to repair or replace a device. Nightingale et al. [16] study desktop failures running Windows and find that PC failures are not memoryless, consistent with our observation for the three types of network devices.

6.2 Fault Localization

Fault-localization is performed after a failure is observed. While it is a well-studied topic in research, it is typically domain-dependent. Depending on the level of intrusiveness required, there are several different types of fault localization and detection systems proposed in the literature. Sophisticated commercial tools, such as Ionix's SMARTS [113], IBM Tivoli [114], Microsoft Operations Manager [115] provide powerful generic frameworks for correlating alarms from standardized alerts generated by individual servers through SNMP [25]. Techniques such as FUSE [91], and Pinpoint [116] keep track of requests that flow through the system by instrumenting middleware on end hosts. The components with failed requests are then correlated to diagnose the source of failure. SCORE [93] and Shrink [90] build a cross-layer model which includes propagation of failures from optical links to the IP layer. Spotlight [92] constructs multi-tier dependency graphs and uses novel graph compression algorithms that simplify the former into probabilistic bipartite graphs over which it runs a weighted greedy set cover algorithm for fast diagnosis. Our work is complementary to these efforts — existing fault localization systems can benefit from our techniques in two ways. First, our statistical techniques can be used for narrowing down to events that have caused service-level impact. This has the potential advantage of reducing the number of events that need further examination by the fault localization algorithms. Second, in addition to performing fault localization, the knowledge base constructed by NetSieve can be used to estimate the likely reason for a failure.

6.3 Middlebox Failures

Broadly, the research community has pursued two key directions to build better middleboxes:

Middlebox Architectures: Research in this direction proposes techniques for explicit control of the middleboxes e.g., middlebox communication (MIDCOM) [117–120] and IETF Next Steps in Signaling (NSIS) [121]. The motivation behind these efforts draws from the fact that while operators increasingly require support for new applications (e.g., teleconferencing, cloud and mobile applications), they are unable to use third-party software/hardware specific to the application and are tied to their middlebox vendor to make the necessary upgrade [19, 117]. There has been some recent work [122, 123] in separating policy from reachability and centralized management of networks. Joseph et al. [124] proposed a policy-aware switching layer for data centers comprising inter-connected policy-aware switches that enable forwarding of different types of traffic through a specified sequence of middleboxes. dFence proposes adding DoS mitigation middleboxes on-demand on the data path to servers under attack [75].

Middlebox Processing Analysis: Research in this direction largely focused on either characterizing NAT properties such as mapping type and filtering rules [125, 126] or quantifying the end-to-end performance degradation induced by middleboxes [23]. Our work falls into the latter category of middlebox reliability analysis. We do not argue the benefits or shortcomings of current middlebox architectures but instead focus on analyzing their performance.

Failures in datacenters have received significant attention in the recent years [18, 32, 86–89]. Wang et al. measure the impact of middlebox policies on performance, energy and security in cellular networks [127]. However, they did not study middlebox failures. Our work is complementary to these efforts in that we focus on characterizing the device reliability across middleboxes and understanding the causes and impact of their failures.

Markopoulou *et al.* [88] studied failure in the Sprint backbone using passive optical taps and high-speed packet capture hardware. Their results indicated that 20% of all failures is due to planned maintenance and that maintenance is usually scheduled during periods of low network usage to minimize the impact on performance. As operators likely have a good understanding of problems under scheduled maintenance, we apply a planned maintenance filter on network events to analyze device-level reliability due to unexpected outages.

Our work draws some similarity with the analysis carried out by Turner et al. [18] and Gill et al. [20]. Turner et al. [18] used router configurations, syslog and email records to analyze network failures in the CENIC network that serves educational and research institutions in California. Gill et al. [20] study network failures in datacenters. Similar to us, they observe that LBs exhibit high annualized failure rates.

While we share the broader goal of studying network failures with these efforts, this work differs in three important ways driven by our goal of understanding the causes and impact of middlebox failures. First, we focus on several new questions to characterize middlebox reliability e.g., What are the key problems observed and their resolution actions? How prevalent are misconfigurations and what are their different types? Does reliability improve with new generations? and How effective is middlebox redundancy? that have not been addressed in the past. Second, our methodology filters maintenance events to identify "unexpected/unplanned" outages, filters events for devices with redundant failures, and performs hypothesis testing to validate traffic based impact filtering. Finally, our results provide useful guidelines to improve middlebox reliability.

6.4 Automated Problem Inference

Network Troubleshooting: There has been a significant work in analyzing structured logs to learn statistical signatures [94, 95], run-time states [96] or leveraging router syslogs to infer problems [98]. Xu et al. [128] mine machine logs to detect runtime problems by leveraging the source code that generated the logs. NetSieve complements these approaches to automate problem inference from unstructured text. Expert systems [129, 130], on the other hand, diagnose network faults based on a set of pre-programmed rules. However, they lack generality as they only diagnose faults in their ruleset and the ruleset may become outdated as the system evolves. In comparison, the incremental learning phase in NetSieve updates the knowledge base to improve the inference accuracy.

Mining Network Logs: Prior efforts have focused on automating mining of network failures from syslogs [97, 98] or network logs [131]. However, these studies do not analyze free-form text in trouble tickets. Kandula et al. [132] mine rules in edge networks based on traffic data. Brauckhoff et al. [133] use association rule mining techniques to extract anomalies in backbone networks. NetSieve is complementary to these efforts in that their mining methodologies can benefit from our domain-specific knowledge base. TroubleMiner [30] selects keywords manually from the first two sentences in tickets and then performs clustering to group them.

Analyzing Bug Reports: There is a large body of work in software engineering analyzing [134, 135], summarizing [136] and clustering [137, 138] bug reports. Betternburg et al. [136] rely on features found in bug reports such as stack traces, source code, patches and enumerations and, hence their approach is not directly applicable to network tickets. Others [137, 138] use standard NLP techniques for the task of clustering duplicate bug reports, but they suffer from the same limitations as keyword based approaches. In comparison, NetSieve aims to infer "meaning" from the free-form content by building a knowledge base and an ontology model to do problem inference.

Natural Language Processing: N-gram extraction techniques [39–41, 43] focus on extracting all possible n-grams thereby incurring a high computation cost for large datasets (Section 4.2.3). NetSieve addresses this challenge by trading completeness for scalability and uses the dictionary built by its WLZW algorithm. Wu et al. [139] detect frequently occurring text fragments that have a high correlation with labels in large text corpora to detect issues in customer feedback. Other efforts [140, 141] achieve summarization via paragraph and sentence extraction. Much research in this area deals with properly-written regular text and is not directly applicable to our domain. In contrast, NetSieve focuses on free-form text in trouble tickets to do problem inference.

7 CONCLUSION

The growing demand for always-on and low-latency cloud services is driving the creation of globally distributed datacenters. A major factor affecting service availability is reliability of the network, both inside the datacenters and wide-area links connecting them. This dissertation addresses the challenge of improving the dependability of the underlying network infrastructure along two directions: (1) characterizing intra-datacenter and inter-datacenter network failures from a services perspective and (2) showing how operational knowledge can be used towards improving network management.

In the following, we discuss several implications of our work that can form basis for subsequent research to improve network reliability for geo-distributed services.

7.1 Implications for Networking Research

DNS-redirection Based Network Load Balancing: To mask service outages (Chapter 1) and to avoid overload at any datacenter site, one approach is to assign a URI to the cloud service whose DNS resolution maps to a set of globally distributed monitoring servers e.g., in a content distribution network. These servers dynamically re-route requests to the edge datacenter hosting the service that will provide the lowest latency based on a client's location. The monitoring servers track each datacenter hosting site via heartbeat signals and can further execute a specified set of mini-transactions [142] on the service to continuously check its performance and availability.

Link Bundling and Wavelength Provisioning: To reduce high utilization events (*Section* 5.7.2) on inter-datacenter links and to avoid the undesirable latency changes due to rerouting (e.g., caused by a fiber cut) at the optical layer, several techniques can be explored:

- Link Bundling: Link bundling, widely used in the context of MPLS [143] traffic engineering, combines multiple links into a single logical channel to increase aggregate bandwidth and fault tolerance. For fiber links, it can significantly reduce LSP fragmentation which arise from large flows being unable to fit on individual circuits. In particular, bundling allows creating many smaller, parallel LSPs between core routers, bringing the bandwidth to a packable number such as 2Gbps per LSP. The resulting large number of LSPs can then be re-routed occasionally and fit across a fragmented 10G optical mesh.
- Automatic Bandwidth: MPLS auto-bandwidth feature measures the traffic flows through the LSP, adjusting the bandwidth based on measured traffic and defined parameters on a per-LSP basis. While it allows the network to react faster to sudden traffic spikes without manual intervention, there are two key challenges of (a) managing significant routing churn by small-sized LSPs, and (b) tuning parameters to automatically create/delete LSPs.
- Mirror Physical Topology: Wavelength provisioning and link bundling can be planned so that bundles are created, mirroring the underlying physical topology. In this way, the reliability of an optical network is better modeled by the IP router topology. When there is an optical failure, the whole link bundle will go down, and should therefore be unavailable from a network and capacity planning point of view. This helps reduce complexity in understanding and mitigating failures because the IP topology closely mirrors the optical topology.

Techniques to Improve Network Redundancy: Our analysis of redundancy effectiveness (*Section* 5.4) revealed the following problems to cause unsuccessful failovers:

- Misconfigurations and software version mismatch between primary and secondary redundant pairs.

- Faulty failovers when the backup exhibited a problem unrelated to primary failure (e.g. due to software errors, protocol bugs etc.)
- Faulty cables where the cable connected to the backup showed a high error rate on failover

Solving these broad range of problems requires exploring several research directions. Techniques from software engineering such as static analysis which found success in detecting BGP misconfigurations [144] can be explored for checking configurations of Layer-2 and Layer-3 network devices. Similarly, proactive fault injection [145] techniques can be leveraged to randomly shoot down network elements and testing service resilience in masking them.

Commoditize Middleboxes: Chapter 1 and *Section* 5.1.1 indicated that hardware middleboxes contribute significantly to network failures, and that their failovers are not 100% effective (*Section* 5.4). Therefore, a natural alternative is to commoditize them [146–148] as they can enable a *scale-out* design and provide n:1 redundancy. While there has been promising work on software routers [149], achieving high performance, scalability and fault tolerance simultaneously poses several challenges. First, achieving high performance and programmability are often competing goals. For instance, the communication overhead introduced by software LBs may impact performance of latency-sensitive applications. Second, well-known software issues such as code bugs, frequent updates, and misconfigurations, may cause a commodity middlebox to be less reliable than hardware boxes. Finally, running a large number of commodity servers or VMs as middleboxes risks high operational costs such as power and cooling. Efforts such as [150, 151] to reduce network energy footprint can be explored to address them. In some cases, strict application requirements for hardware accelerators such as hardware-based deep packet inspection (DPI) for IDPS and deep-packet modification for WAN optimizers, may become a limiting factor.

Feasibility of Making Middlebox Processing as a Cloud Service: A recent proposal is to offload middlebox processing to the cloud [19]. Similar to CDNs, the idea is to use DNS-based redirection to route requests across multiple cloud PoPs and deploy middlebox instances in the cloud to process them based on defined policies. While this approach shifts the burden of owning and managing middleboxes from enterprises to the cloud, our study indicates that it risks making the problem worse as middlebox failures are prevalent in datacenters (*Section* 5.1.1) and redundancy is not 100% effective (*Section* 5.4) . Several other challenges also need to addressed before pushing middleboxes as a cloud hosted service: (a) how to guarantee correct middlebox traversal and coordinate state during churn? (b) how to manage the state and complexity for multiple tenants sharing the same set of middlebox instances? (c) how to address grey failures (*Section* 5.7) that can degrade service performance and availability? and (d) how to elastically scale-out and scale-in the processing capabilities (e.g., for stateful services) as demand changes? For instance, the stateful session nature of many middleboxes (e.g., load balancers) can limit how quickly instances can be (de)instantiated or risk a traffic spike as disconnected customers attempt to simultaneously rejoin when an instance is shutdown. To deliver traffic reliably in the presence of grey failures, multi-path protocols such as MP-TCP [152] can be explored.

Software-Defined Middlebox Networking: Software-defined networking (SDN) is an emerging trend to address the inefficiency in resource usage and complexity of managing network devices. The basic idea is to put more intelligence on centralized management servers and reduce code and policy settings on individual network devices. While this approach is appealing, there are several challenges that need to be addressed. First, applying it to middleboxes requires careful management and control of their state to ensure correct behavior [153]. Second, building an open platform in which hardware and software may evolve independently, is still an important research question [148]. Finally, while network routers alone determine the processing to apply to packets, the middlebox approach makes it complicated as the physical location of the middlebox (typically at choke points) combined with ad-hoc routing determines the processing applied by the middlebox to its traffic flows.

7.2 Implications for Network Operations

Repair vs. Replace: *Section* 5.2.3 indicated that when devices fail multiple times, they do so within one week of getting fixed. Therefore, finding and replacing the "*few bad apples*" will proactively help avoid serious problems. In addition, the analysis in *Section* 5.2.2 indicates that failures are not memoryless and while repairs were effective for ARs and AGGs, the probability of successive failure for ToRs increased. Thus, while a naive approach is to *immediately* replace a failed ToR, in practice this decision should be driven by two key factors: (1) Computing a Cost of Ownership (COO) [154] for devices to include their capital, operational, and repair and maintenance costs; *Section* 5.1.1 provides empirical results on some of these metrics, and (2) adopting a data-driven approach to compute the conditional probability $P_{N+1|N} = P((N+1)^{th} failure|N^{th} failure)$ for a device type/platform and then comparing it with both a threshold δ based on the network device platform's annualized failure rate and $P_{N|N-1}$. The intuition behind the latter is that if $P_{N+1|N} > \delta * P_{N|N-1}$, the probability of the device experiencing a subsequent failure is higher and thus it becomes a candidate for replacement.

Middlebox Verification: Section 5.7 indicated that connection errors and interface problems dominate middlebox failures, and that there is a wide range of misconfigurations observed on middleboxes. As a result, a key requirement for network operations is to automate management and debugging of network problems e.g., using SDN [155, 156]. To improve network debugging, one approach is to do static and run-time verification of desired properties e.g., to check configurations, verify reachability, detect routing loops, guarantee isolation [144, 155, 157]. However, there are several challenges that need to be addressed. First, current tools lack support for common middlebox features such as tunneling and load balancing. Second, the task of parsing policy specifications and configurations is non-trivial and error-prone, as vendors use varying semantics which in-turn may change with device generations. Third, checking for correctness of policies involves designing a set of constraints that can be tested against the device configuration. However, defining a complete constraint set is difficult and evaluating it, computationally expensive [158]. Finally, the large-scale of data center networks makes it challenging to apply formal verification techniques due to the well-known state space explosion problem.

Software-Assisted Improvement of Hardware Reliability: In Section 5.7, we revealed the various hardware failures experienced by middleboxes. It is expected that components inside these devices will fail for many reasons including design defects, aging or wear-out, infant mortality due to insufficient burn-in, and so on [159]. Such scenarios require mechanisms for detection, diagnosis, recovery, and possibly reconfiguration around the failed components to ensure reliable and continuous operation. One idea is to achieve this through a symbiotic hardware-software solution. Similar to trusted party modules [160], a fault-free co-processor can be delegated the role of diagnosis while a software system monitors for error symptoms. In the event of a fault, the co-processor can trigger a checkpointing-based recovery to enable system recovery from the previous checkpoint. Such a software-level monitoring system can also be interfaced with machine learning systems for fault prediction. This enables a proactive strategy as opposed to a reactive one.

Detect Early Faulty Product-Line: In *Section* 5.3, we uncovered using trend analysis that an LB generation had lower reliability compared to other generations. This raises a serious dilemma: keep replacing faulty devices in existing deployment or upgrade to a new product line? While capital and operating costs dominate the cost vs. benefit analysis, answering this question is challenging for two additional reasons:

- Device familiarity: Upgrading devices with their high-speed, feature-rich counterparts will require additional training for network engineers (and has the undesirable consequence of increasing TTR due to unfamiliarity).
- Active/Standby Compatibility: Where 1:1 redundancy is dominant, both devices should be simultaneously upgraded to avoid unsuccessful redundancy failovers.

By computing a Cost of Ownership (COO) [154] metric, decisions such as whether to buy more spares for a faulty product line or to gracefully replace a product line can be made. The primary cost elements of the COO model are: (i) initial investment in purchasing the product, (ii) depreciation, (iii) cost of operation that includes space, personnel, and power expenses, (iv) cost of maintenance operations including downtime, repair cost, failure rates, and cost of spares, and (v) enhancement costs that include hardware/software upgrades and personnel training. Our reliability analysis attempts to provide some of these metrics as described in *Section* 5.1.1 and *Section* 5.1.2.

Make the Cost-Metric of Middlebox Hardware Reliability Aware: To compare network hardware costs across platforms and vendors, the conventional metric is to use \$/Gb/s as the unit-cost basis e.g., \$ per 10 Gbps port. For instance, using this metric, we find that hardware LBs are about an order of magnitude more expensive than commodity Layer 3 switches. As *Section* 5.1.1 shows, middleboxes experience a significant fraction of failures, and incur substantial operating costs for repair and upgrade [19]. Thus, to balance the trade-off between improving service availability while cutting costs down, the cost metric should comprise both capital and operational expenses, e.g., \$/Gbps/99.9% availability, so that administrators can compute the total cost of ownership (TCO) and determine the network costs to provide a target SLA (e.g., 99.9%), to hosted services. Note that the accuracy of this metric needs to be continuously validated and improved by leveraging data from middlebox deployments in production.

Overall, we hope this dissertation sheds light on answering several key questions to improve network reliability for geo-distributed services.

7.3 Future Work

To deliver cloud services in a secure, reliable, and in a cost-effective manner, we need a) reliable network infrastructures for hosting services, b) automated network management systems that encourage preventive maintenance and c) secure architectures robust to various attacks. There are a number of exciting future research directions that can be explored to enable such cloud services. In this section, we describe some of these directions.

Service-Aware Datacenter Networks: Much of this dissertation on datacenter reliability has been largely agnostic of the application services running within mainly due to lack of access to application logs. Due to this limitation, we designed metrics that rely purely on network traffic to determine if a failure had any service level impact [161, 162]. However, we believe that much greater performance can be extracted from the data plane when both the applications and the network cooperate with each other. For instance, "hints" from an application about its intended communication pattern can be used to smoothen out the burstiness of traffic and ultimately reduce congestion and packet losses. Considering application services in the design of datacenter networks and protocols introduces new research questions. For example, how does the effectiveness of network-level redundancy change when application-level redundancy is taken into account? How do you build a specialized datacenter with custom servers and network switches that are optimized for a given class of applications? How do you facilitate sharing of hints for optimizing network performance, such as application communication patterns and behavior, between a cloud tenant and cloud provider while ensuring that both privacy and security are preserved?

Towards the Next-Generation of Network Management Tools: Monitoring the operations of tens of thousands of commodity servers is a daunting task. Further, analyzing performance logs and fault traces from such a large number of machines is challenging to say the least. Expert developers possess vast amounts of knowledge and insight, which they synthesize to diagnose complex problems, so attempting to replace them immediately with automation is something we learnt is very difficult and detrimental in a practical setting. Instead, a logical first step is to create techniques that use systems knowledge, machine learning, statistics and visualization to *augment* developer's diagnosis efforts — for example, automatically localize the root cause of a new problem from the myriad components in the system to just a few relevant potential candidate subsets. Much work must be done to identify promising approaches, but we believe ones with any hope of success will have to address three key challenges:

1. Many recent cloud outages, were exacerbated because developers implemented locally optimal fixes that had adverse global effects. Therefore, such diagnostic tools must incorporate knowledge of system dependencies, so as to be aware of how a specific action will effect the rest of the system.
2. One must have access to a large knowledge base of previous diagnostic actions (e.g., output of NetSieve [24]) and network-related information (e.g., through software-defined networking (SDN)) which must then be integrated with learning systems capable of suggesting fixes to on-going problems.
3. One must incorporate models to evaluate the *risk* of various actions, so as to be able to choose between multiple possible fixes.

We have pursued these goals in-part through NetWiser [163], a Microsoft Research project, which is a real-time incident monitoring system for datacenter operators. NetWiser enabled several key network operations such as prioritization of network alerts, proactive replacement of devices with high-risk-of-failure, proactive verification of top-k devices with high likelihood of simultaneous failures and improving the effectiveness of network redundancy.

Augmenting Human Intelligence With Smart Tools: Understanding natural language has been the holy grail of AI since its inception. Full understanding of natural language implies human-level capacity to process language and draw appropriate conclusions. This task is not only very complex but also quite unbounded in terms of scope. The crucial question is, what level of understanding and complexity of drawn conclusions should we require from a machine? While working towards answering this question is a great challenge, we believe the problem dual is much more interesting — how can we make effective use of information technology in augmenting human intelligence? Consider the following example. While understanding natural language is hard, our working hypothesis is that the very same task becomes feasible if we clearly restrict the scope of what must be understood. To do this, we can focus on a specific domain model — for instance, think of NetSieve [24], described in this dissertation, where we limited ourselves to the networking domain — and specify which conclusions a system must draw depending on the high-level task (e.g. what are the most frequently used repair actions to fix problems?).

One important direction for the coming years could be to make computers understand natural language to a limited but well-defined degree, required by a certain usecase, application or domain. The granularity and structural complexity of what a system needs to understand is clearly determined by the usecase, which for instance was modeled as an ontology in NetSieve.

One could take this understanding to the next level and explore other related areas such as question answering, information extraction and diagnostic systems. A concrete example is enabling intelligence augmentation in call center management. One technique is that you rely on software entities or *software agents* that are typically long- lived, continuously running, fairly simple, and that can help you in semi-automating tasks such as helping customers with their computer troubleshooting problems — a software agent can eavesdrop the conversation a customer is having with a call center specialist and suggest potential solutions on the specialist's monitor, for instance. These systems must be developed in a principled way so that they are portable to other domains or usecases. When trying to transfer such systems to practice, we face several questions: How much does the system cost in terms of both human and machine cycles? How much time do we devote to customizing it for a specific purpose? Are debugging tools available? While the focus so far in traditional computer science has been on developing generic and domain-independent solutions, such as parsing and disambiguating news articles, we have no clear answers to the previous questions; consequently, the transfer of research into practice has been somewhat limited. An important goal for anyone wanting to explore this direction is to lay a special emphasis on the usability viewpoint and develop systems that nonexperts can easily adapt to their domains.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Chris Talbot. Dropbox Outage Represents First Major Cloud Outage of 2013. `http://goo.gl/rszmb`, January 2013.

[2] Darrell Etherington. Dropbox Currently Experiencing Widespread Service Outage. `http://goo.gl/rszmb`, May 2013.

[3] Smith G. and Isaacson B. Websites Scramble as Hurricane Sandy Floods Data Centers. `http://goo.gl/zOXDb`, October 31 2012.

[4] Spice Works. Hurricane Sandy - AC2 Transatlantic Cable Cut. `http://goo.gl/dywVO`, October 2012.

[5] Amazon. Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. `http://goo.gl/yUlTJ`, May 2011.

[6] Data Center Knowledge. Data Center Global Expansion Trend. `http://goo.gl/SOvtA`, November 2012.

[7] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proceedings of ACM SIGCOMM CCR*, 2009.

[8] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters. In *Proceedings of ACM SIGCOMM*, 2011.

[9] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Datacenter Network. *Journal Proceedings of ACM SIGCOMM CCR*, 2009.

[10] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated Data Placement for Geo-distributed Cloud Services. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2010.

[11] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are Disks the Dominant Contributor for Storage Failures?: A Comprehensive Study of Storage Subsystem Failure Characteristics. *Journal Proceedings of ACM Transactions on Storage*, 2008.

[12] B. Schroeder and G.A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You. In *Proceedings of FAST*, 2007.

[13] E. Pinheiro, W.D. Weber, and L.A. Barroso. Failure Trends in a Large Disk Drive Population. In *Proceedings of FAST*, 2007.

[14] L.N. Bairavasundaram, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, G.R. Goodson, and B. Schroeder. An Analysis of Data Corruption in the Storage Stack. *Journal Proceedings of ACM Transactions on Storage*, 4(3), 2008.

[15] B. Schroeder, E. Pinheiro, and W.D. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *Proceedings of ACM SIGMETRICS*, 2009.

[16] E.B. Nightingale, J.R. Douceur, and V. Orgovan. Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs. In *Proceedings of the ACM Conference on Computer Systems*, 2011.

[17] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L.N. Bairavasundaram, and S. Pasupathy. An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In *Proceedings of ACM SOSP*, 2011.

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

[19] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing As a Cloud Service. In *Proceedings of ACM SIGCOMM*, 2012.

[20] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of ACM SIGCOMM*, 2011.

[21] J.W. Lockwood. An open platform for development of network processing modules in reprogrammable hardware. *Journal Proceedings of IEC DesignCon*, 2001.

[22] D. Johnson. NOC Internal Integrated Trouble Ticket System. `http://goo.gl/eMZxX`, January 1992.

[23] M. Allman. On the performance of middleboxes. In *Proceedings of ACM SIGCOMM Internet Measurements Conference*, 2003.

[24] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2013.

[25] J. Case, M. Fedor, M. Schoffstall, and J. Davin. Simple Network Management Protocol. `http://goo.gl/az3Fv`, May 1990.

[26] M. McCloghrie, K. ad Rose. Management Information Base for Network Management of TCP/IP-based internets. RFC 1213.

[27] H.B. Mann and D.R. Whitney. On a Test of Whether One of Two Random Variables Is Stochastically Larger Than the Other. *Proceedings of the Annals of Mathematical Statistics*, 1947.

[28] A. Hart. Mann-Whitney Test Is Not Just a Test of Medians: Differences In Spread Can Be Important. *Journal Proceedings of BMJ*, 2001.

[29] D. Ford, F. Labelle, F.I. Popovici, M. Stokely, V.A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[30] A. Medem, M.I. Akodjenou, and R. Teixeira. Troubleminer: Mining Network Trouble Tickets. In *Proceedings of IEEE International Workshop Symposium on Integrated Network Management*, 2009.

[31] C.D. Manning, H. Schütze, and MITCogNet. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[32] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Backbone Failures. In *Proceedings of FTC*, 1999.

[33] Y. Huang, N. Feamster, A. Lakhina, and J.J. Xu. Diagnosing Network Disruptions With Network-Wide Analysis. *Journal Proceedings of ACM SIGMETRICS Performance Evaluation Review*, 2007.

[34] M. Roughan, T. Griffin, Z.M. Mao, A. Greenberg, and B. Freeman. IP Forwarding Anomalies and Improving Their Detection Using Multiple Data Sources. In *Proceedings of ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice meet Malfunctioning Reality*, 2004.

[35] C. Melchiors and L.M.R. Tarouco. Troubleshooting Network Faults Using Past Experience. In *Proceedings of IEEE Network Operations and Management Symposium*, 2000.

[36] M. Ziefle. Effects of Display Resolution on Visual Performance. *Journal Proceedings of The Journal of the Human Factors and Ergonomics Society*, 1998.

[37] R. Fielding. Representational State Transfer (REST). *Jounal Proceedings of Architectural Styles and the Design of Network-based Software Architectures*, 2000.

[38] M. Benson. Collocations and General-Purpose Dictionaries. *International Journal of Lexicography*, 1990.

[39] F. Smadja. Retrieving Collocations From Text: Xtract. *MIT Press Computational Linguistics*, 1993.

[40] K.W. Church and P. Hanks. Word Association Norms, Mutual Information, and Lexicography. *Journal of Computational Linguistics*, 1990.

[41] M. Nagao and S. Mori. A New Method of N-Gram Statistics for Large Number of N and Automatic Extraction of Words and Phrases From Large Text Data of Japanese. In *Proceedings of ACL Computational Linguistics*, 1994.

[42] Le Zhang. N-Gram Extraction Tools. `http://goo.gl/7gGF1`, April 2012.

[43] M. Yamamoto and K.W. Church. Using Suffix Arrays to Compute Term Frequency and Document Frequency for All Substrings in a Corpus. *Proceedings of Computational Linguistics*, 2001.

[44] J. Zhang, J. Gao, and M. Zhou. Extraction of Chinese Compound Words: An Experimental Study on A Very Large Corpus. In *Proceedings of ACL Workshop on Chinese Language Processing*, 2000.

[45] T. Welch. Technique for High-Performance Data Compression. *Journal Proceedings of Computer*, 1984.

[46] A.V. Aho and M.J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Journal Proceedings of ACM Communications*, 1975.

[47] S. Behnel, R. Bradshaw, D. Seljebotn, G. Ewing, et al. Cython: C-extensions for Python. `http://cython.org`, 2008.

[48] E. Ukkonen. Online Construction of Suffix Trees. *Journal Proceedings of Springer Algorithmica*, 1995.

[49] U. Manber and G. Myers. Suffix Arrays: A New Method for Online String Searches. In *Proceedings of ACM SIAM Journal on Computing*, 1990.

[50] K. Heafield. KenLM: Faster and Smaller Language Model Queries. In *Proceedings of Workshop on Statistical Machine Translation*, 2011.

[51] Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. Streaming for Large Scale NLP: Language Modeling. In *Proceedings of the Association for Computational Linguistics*, 2009.

[52] J.S. Justeson and S.M. Katz. Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text. *Journal Proceedings of Natural language engineering*, 1995.

[53] M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. Building a Large Annotated Corpus of English: The Penn Treebank. *Journal Proceedings of MIT Press Computational Linguistics*, 1993.

[54] E. Loper and S. Bird. NLTK: The Natural Language Toolkit. In *Proceedings of the Association for Computational Linguistics Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, 2002.

[55] Dragon S. Sharp NLP. `http://goo.gl/Im4BK`, December 2006.

[56] K. Toutanova and C.D. Manning. Enriching the Knowledge Sources Used in a Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the ACL SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 2000.

[57] A. McCallum and W. Li. Early Results for Named Entity Recognition With Conditional Random Fields, Feature Induction and Web-Enhanced Lexicons. In *Proceedings of the Association for Computational Linguistics Conference on Natural Language Learning*, 2003.

[58] A. Kittur, E.H. Chi, and B. Suh. Crowdsourcing User Studies With Mechanical Turk. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems*, 2008.

[59] Thomas R Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *Journal Proceedings of Human Computer Studies*, 1995.

[60] N.F. Noy, D.L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, 2001.

[61] S. Goryachev, M. Sordo, Q.T. Zeng, and L. Ngo. Implementation and Evaluation of Four Different Methods of Negation Detection. Technical report, DSG, 2006.

[62] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML). *Journal Proceedings of WWW*, 1997.

[63] J.J. Garrett et al. Ajax: A new approach to web applications. `http://goo.gl/UCl1BJ`, 2005.

[64] M. Zur Muehlen, J.V. Nickerson, and K.D. Swenson. Developing web services choreography standards – the case of rest vs. soap. *Journal Proceedings of Elsevier Decision Support Systems*, 2005.

[65] K.A. Evans, A. Kamanna, and J. Mueller. *XML and ASP. NET*. New Riders Publishing, 2002.

[66] I. Hickson and D. Hyatt. HTML5: A Vocabulary and Associated APIs for HTML and XHTML. *W3C Working Draft*, 19, 2010.

[67] B. Bos, T. Çelik, I. Hickson, and H.W. Lie. Cascading Style Sheets, Level 2 Revision 1 CSS 2.1 Specification. *W3C Working Draft*, June 2005.

[68] jQuery: Write Less, Do More. `http://goo.gl/VNTJa`, April 2012.

[69] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *Journal Proceedings of the IEEE Transactions on Visualization and Computer Graphics*, 2011.

[70] I. Mani, D. House, G. Klein, L. Hirschman, T. Firmin, and B. Sundheim. The TIPSTER SUMMAC Text Summarization Evaluation. In *Proceedings of the Association for Computational Linguistics*, 1999.

[71] C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press Cambridge, 2008.

[72] Eric A Brewer. Lessons from Giant-Scale Services. *Journal Proceedings of IEEE Internet Computing*, 2001.

[73] Charles E Brown. Coefficient of Variation. In *Applied Multivariate Statistics in Geohydrology and Related Sciences*. Springer, 1998.

[74] Vincent Liu, Arvind Krishnamurthy, and Thomas Anderson. F10: Fault-Tolerant Engineered Networks. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2013.

[75] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, and Y. Zhang. dFence: Transparent Network-Based Denial of Service Mitigation. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2007.

[76] RM Sakia. The Box-Cox Transformation Technique: A Review. *The Statistician*, pages 169–178, 1992.

[77] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE ToN*, 1994.

[78] Timothy L Bailey and Charles Elkan. Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Bipolymers. In *Proceedings of ISMB*, 1994.

[79] Todd K Moon. The Expectation-Maximization Algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996.

[80] Hubert W Lilliefors. On the Kolmogorov-Smirnov Test for Normality with Mean and Variance Unknown. *Journal Proceedings of JASA*, 1967.

[81] G.E.P. Box, J.S. Hunter, and W.G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery.* Wiley, 2005.

[82] H. Jiang, F. Kéfélian, S. Crane, O. Lopez, M. Lours, J. Millo, D. Holleville, P. Lemonde, C. Chardonnet, A. Amy-Klein, et al. Long-distance Frequency Transfer Over an Urban Fiber Link Using Optical Phase Stabilization. *Journal Proceedings of Optical Society of America*, 2008.

[83] RB Bendel, SS Higgins, JE Teberg, and DA Pyke. Comparison of Skewness Coefficient, Coefficient of Variation, and Gini Coefficient as Inequality Measures within Populations. *Journal Proceedings of Oecologia*, 1989.

[84] S. Deering and R. Hinden. Internet Protocol, Version (IPv6) Specification. RFC 2460.

[85] G. Mohan and C.S.R. Murthy. Lightpath Restoration in WDM Optical Networks. *Journal Proceedings of IEEE Network*, 2000.

[86] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed Diagnosis in Enterprise Networks. In *Proceedings of ACM Sigcomm CCR*, 2009.

[87] V.N. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye. A Study of End-to-End Web Access Failures. In *Proceedings of ACM CoNEXT*, 2006.

[88] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.N. Chuah, Y. Ganjali, and C. Diot. Characterization of Failures in an Operational IP Backbone Network. *Proceedings of IEEE/ACM Transactions on Networking*, 2008.

[89] A. Shaikh, C. Isett, A. Greenberg, M. Roughan, and J. Gottlieb. A Case Study of OSPF Behavior in a Large Enterprise Network. In *Proceedings of ACM SIGCOMM WIM*, 2002.

[90] Srikanth Kandula, Dina Katabi, and Jean-Philippe Vasseur. Shrink: A Tool for Failure Diagnosis in IP Networks. In *Proceedings of ACM SIGCOMM Workshop on Mining Network Data*, 2005.

[91] John Dunagan, Nicholas J.A. Harvey, Michael B Jones, Dejan Kostic, Marvin Theimer, and Alec Wolman. FUSE: Lightweight Guaranteed Distributed Failure Notification. In *Proceedings of USENIX Conference on Operating System Design and Implementation*, 2004.

[92] Dipu John, Pawan Prakash, Ramana Rao Kompella, and Ranveer Chandra. Shedding Light on Enterprise Network Failures Using Spotlight. In *Proceedings of IEEE Conference on Reliable Distributed Systems*, 2010.

[93] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C Snoeren. IP Fault Localization via Risk Modeling. In *Proceedings of USENIX Conference on Networked Systems Design & Implementation*, 2005.

[94] M.K. Aguilera, J.C. Mogul, J.L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance Debugging for Distributed Systems of Black Boxes. In *Proceedings of the ACM SIGOPS Operating Systems Review*, 2003.

[95] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, Indexing, Clustering, and Retrieving System History. In *Proceedings of the ACM SIGOPS Operating Systems Review*, 2005.

[96] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy. SherLog: Error Diagnosis by Connecting Clues from Run-time Logs. In *Proceedings of the ACM SIGARCH Computer Architecture News*, 2010.

[97] K. Yamanishi and Y. Maruyama. Dynamic Syslog Mining for Network Failure Monitoring. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005.

[98] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu. What Happened in My Network: Mining Network Events From Router Syslogs. In *Proceedings of the ACM Internet Measurement Conference*, 2010.

[99] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter Bulk Transfers with NetStitcher. In *Proceedings of SIGCOMM*, 2011.

[100] Y. Chen, S. Jain, V.K. Adhikari, Z.L. Zhang, and K. Xu. A First Look at Inter-data Center Traffic Characteristics via Yahoo! Datasets. In *Proceedings of IEEE INFOCOM*, 2011.

[101] A. Mahimkar, A. Chiu, R. Doverspike, M.D. Feuer, P. Magill, E. Mavrogiorgis, J. Pastor, S.L. Woodward, and J. Yates. Bandwidth On Demand for Inter-Data center Communication. In *Proceedings of ACM HotNets*, 2011.

[102] Y. Li, H. Wang, P. Zhang, J. Dong, and S. Cheng. D4D: Inter-datacenter Bulk Transfers with ISP Friendliness. In *Proceedings of IEEE CLUSTER*, 2012.

[103] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Datacenter Traffic: Measurements & Analysis. In *Proceedings of ACM SIGCOMM*, 2009.

[104] Rahul Potharaju and Navendu Jain. An Empirical Analysis of Intra-and Inter-datacenter Network Failures for Geo-distributed Services. In *Extended Abstract Proceedings of ACM SIGMETRICS*, 2013.

[105] Cheng Huang, Angela Wang, Jin Li, and Keith W Ross. Measuring and Evaluating Large-scale CDNs. In *Proceedings of ACM SIGCOMM Internet Measurements Conference*, 2008.

[106] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman, and Bill Weihl. Globally Distributed Content Delivery. *Journal Proceedings of IEEE Internet Computing*, 2002.

[107] Athena Vakali and George Pallis. Content Delivery Networks: Status and Trends. *Journal Proceedings of IEEE Internet Computing*, 7(6):68–74, 2003.

[108] Gang Peng. CDN: Content Distribution Network. *arXiv preprint cs/0411069*, 2004.

[109] Al-Mukaddim Khan Pathan and Rajkumar Buyya. A Taxonomy and Survey of Content Delivery Networks. Technical report, University of Melbourne, 2007.

[110] Michael J Freedman, Karthik Lakshminarayanan, and David Mazières. OASIS: Anycast for Any Service. In *Proceedings of USENIX NSDI*, 2006.

[111] Rupa Krishnan, Harsha V Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving Beyond End-To-End Path Information to Optimize CDN Performance. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement Conference*, 2009.

[112] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. YouTube Traffic Dynamics and Its Interplay With a Tier-1 ISP: An ISP Perspective. In *Proceedings of ACM SIGCOMM Conference on Internet Measurement*. ACM, 2010.

[113] SMARTS Inc. `http://www.smarts.com`.

[114] IBM Tivoli. `http://tivoli.com`.

[115] Microsoft Operations Manager. `http://www.microsoft.com/mom`.

[116] Mike Y Chen, Anthony Accardi, Emre Kiciman, David A Patterson, Armando Fox, and Eric A Brewer. *Path-Based Failure and Evolution Management*. PhD thesis, University of California, Berkeley, 2004.

[117] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan. Middlebox Communication Architecture and Framework. *RFC 3303*, 2002.

[118] RP Swale, PA Mart, P. Sijben, S. Brim, and M. Shore. Middlebox Communications Protocol Requirements. *RFC 3304*, 2002.

[119] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. *RFC 3234*, 2002.

[120] M. Stiemerling and J. Quittek. Middlebox Communication (MIDCOM) Protocol Semantics. *RFC 4097*, 2008.

[121] R. Hancock, S. Bosch, G. Karagiannis, and J. Loughney. Next Steps in Signaling (NSIS): Framework. RFC 4080, 2005.

[122] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. *Journal Proceedings of ACM Sigcomm CCR*, 2005.

[123] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. *Journal Proceedings of ACM Sigcomm CCR*, 2007.

[124] D.A. Joseph, A. Tavakoli, and I. Stoica. A Policy-Aware Switching Layer for Data Centers. In *Proceedings of ACM SIGCOMM CCR*, 2008.

[125] J.L. Eppinger. TCP Connections for P2P Apps: Software Approach to Solving the NAT Problem. *ISR*, 2005.

[126] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig. NATBLASTER: Establishing TCP Connections Between Hosts Behind NATs. In *Proceedings of ACM SIGCOMM Asia Workshop*, 2005.

[127] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An Untold Story of Middleboxes in Cellular Networks. *Journal Proceedings of ACM SIGCOMM CCR*, 2011.

[128] W. Xu, L. Huang, A. Fox, D. Patterson, and M.I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *Proceedings of the ACM SIGOPS Symposium on Operating systems Principles*, 2009.

[129] S. Brugnoni, G. Bruno, R. Manione, E. Montariolo, E. Paschetta, and L. Sisto. An Expert System for Real-time Fault Diagnosis of the Italian Telecommunications Network. In *Proceedings of the International Symposium on Integrated Network Management*, 1993.

[130] G. Khanna, M.Y. Cheng, P. Varadharajan, S. Bagchi, M.P. Correia, and P.J. Veríssimo. Automated Rule-based Diagnosis through a Distributed Monitor System. *Journal Proceedings of the IEEE Transactions on Dependable and Secure Computing*, 2007.

[131] C. Lim, N. Singh, and S. Yajnik. A Log Mining Approach to Failure Analysis of Enterprise Telephony Systems. In *Proceedings of the IEEE Conference on Dependable Systems and Networks*, 2008.

[132] S. Kandula, R. Chandra, and D. Katabi. What's Going On?: Learning Communication Rules in Edge Networks. *Proceedings of ACM SIGCOMM Computer Communication Review*, 2008.

[133] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian. Anomaly Extraction in Backbone Networks Using Association Rules. In *Proceedings of the ACM Internet Measurement Conference*, 2009.

[134] S. Just, R. Premraj, and T. Zimmermann. Towards the Next Generation of Bug Tracking Systems. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 2008.

[135] P. Hooimeijer and W. Weimer. Modeling Bug Report Quality. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 2007.

[136] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Extracting Structural Information from Bug Reports. In *Proceedings of the ACM International Working Conference on Mining Software Repositories*, 2008.

[137] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim. Duplicate Bug Reports Considered Harmful... Really? In *Proceedings of the IEEE International Conference on Software Maintenance*, 2008.

[138] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of Duplicate Defect Reports Using Natural Language Processing. In *Proceedings of the IEEE International Conference on Software Engineering*, 2007.

[139] F. Wu and D.S. Weld. Open Information Extraction using Wikipedia. In *Proceedings of Computational Linguistics*, 2010.

[140] M. Mitray, A. Singhalz, and C. Buckleyyy. Automatic Text Summarization by Paragraph Extraction. *Journal Proceedings of Compare*, 1997.

[141] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. Summarizing Text Documents: Sentence Selection and Evaluation Metrics. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999.

[142] Keynote Web Performance Testing. `http://goo.gl/khl9Q`.

[143] K. Kompella, L. Berger, and Y. Rekhter. Link Bundling in MPLS Traffic Engineering (TE). RFC 4201, 2005.

[144] N. Feamster and H. Balakrishnan. Detecting BGP configuration Faults with Static Analysis. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2005.

[145] Jon Brodkin. Netflix Attacks Own Network With "Chaos Monkey" and Now You Can Too. `http://goo.gl/XhiKM`, July 2012.

[146] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael Reiter, and Guangyu Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2012.

[147] Hardeep Uppal, Vjekoslav Brajkovic, Dane Brandon, Thomas Anderson, and Arvind Krishnamurthy. ETTM: A Scalable Fault Tolerant Network Manager. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2011.

[148] Adam Greenhalgh, Felipe Huici, Mickael Hoerdt, Panagiotis Papadimitriou, Mark Handley, and Laurent Mathy. Flow Processing and the Rise of Commodity Network Hardware. *Proceedings of ACM SIGCOMM CCR*, 2009.

[149] K. Argyraki, S. Baset, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedevschi, and S. Ratnasamy. Can Software Routers Scale? In *Proceedings of ACM PRESTO*, 2008.

[150] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing Network Energy Consumption via Sleeping and Rate-adaptation. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2008.

[151] A. Greenberg, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In *Proceedings of ACM PRESTO*, 2008.

[152] Michael Scharf and Alan Ford. MP-TCP Application Interface Considerations. *IETF Draft*, 2010.

[153] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward Software-Defined Middlebox Networking. In *Proceedings of ACM HotNets*, 2012.

[154] L.M. Ellram. Total Cost of Ownership: An Analysis Approach for Purchasing. *Journal Proceedings of PDLM*, 1995.

[155] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: Static Checking for Networks. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation*, 2012.

[156] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. Where Is the Debugger for My Software-Defined Network? In *Proceedings of the ACM Workshop on Hot Topics in Software-defined Networks*, 2012.

[157] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, and S.T. King. Debugging the Data Plane with Anteater. *Journal Proceedings of ACM SIGCOMM CCR*, 2011.

[158] A. Feldmann and J. Rexford. IP Network Configuration for Intradomain Traffic Engineering. *Journal Proceedings of IEEE Network*, 2001.

[159] Shekhar Borkar. Designing Reliable Systems From Unreliable Components: The Challenges of Transistor Variability and Degradation. *Journal Proceedings of IEEE Micro*, 2005.

[160] Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of USENIX Security*, 2006.

[161] Rahul Potharaju and Navendu Jain. Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement*, 2013.

[162] Rahul Potharaju and Navendu Jain. When the Network Crumbles: An Empirical Study of Cloud Network Failures and Their Impact on Services. In *Proceedings of the ACM Symposium on Cloud Computing*, page 15, 2013.

[163] Navendu Jain and Rahul Potharaju. Netwiser. `http://goo.gl/3Qaev4`, 2014.

[164] Rahul Potharaju and Navendu Jain. An Empirical Analysis of Intra- and Inter-Datacenter Network Failures for Geo-Distributed Services. In *Extended Abstract Proceedings of ACM SIGMETRICS 2013*, 2013.

[165] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets. In *Microsoft Machine Learning Summit 2013, Italy*, 2013.

[166] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the Jigsaw: Towards Automated Problem Inference from Network Trouble Tickets. In *Poster Proceedings of USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2013.

[167] K. Chen, D.R. Choffnes, R. Potharaju, Y. Chen, F.E. Bustamante, D. Pei, and Y. Zhao. Where the sidewalk ends: Extending the internet as graph using traceroutes from p2p users. In *CoNEXT*. ACM, 2009.

[168] Kai Chen, David Choffnes, Rahul Potharaju, Yan Chen, Fabian Bustamante, Dan Pei, and Yao Zhao. Where the Sidewalk Ends: Extending the Internet AS Graph Using Traceroutes From P2P Users. 2013.

[169] Md Endadul Hoque, Hyojeong Lee, Rahul Potharaju, Charles E Killian, and Cristina Nita-Rotaru. Adversarial Testing of Wireless Routing Implementations. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2013.

[170] Rahul Potharaju, Andrew Newell, Cristina Nita-Rotaru, and Xiangyu Zhang. Plagiarizing Smartphone Applications: Attack Strategies and Defense Techniques. In *Proceedings of the ACM International Symposium on Engineering Secure Software and Systems (ESSoS)*. Springer, 2012.

[171] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using Probabilistic Generative Models for Ranking Risks for Android Apps. In *Proceedings of the ACM Conference on Computer and Communication Security (CCS)*. ACM, 2012.

[172] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android Permissions: A Perspective Combining Risks and Benefits. In *Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2012.

[173] Rahul Potharaju, Endadul Hoque, Cristina Nita-Rotaru, Saswati Sarkar, and Santosh S Venkatesh. Closing the Pandora's Box: Defenses for Thwarting Epidemic Outbreaks in Mobile Adhoc Networks. In *Proceedings of the 9th IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2012.

[174] Christopher Gates, Ninghui Li, Hao Peng, Bhaskar Sarma, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Generating Summary Risk Scores for Mobile Applications. 2013.

[175] Bogdan Carbunar, Rahul Potharaju, Michael Pearce, and Venu Vasudevan. Network Aware Caching for Video on Demand Systems. In *Proceedings of the IEEE Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2012.

[176] Bogdan Carbunar, Rahul Potharaju, Michael Pearce, and Venu Vasudevan. A Framework for Network Aware Caching for Video on Demand Systems. *Journal Proceedings of ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP)*, 2013.

[177] Bogdan Carbunar and Rahul Potharaju. You Unlocked the Mt.Everest Badge on Foursquare! Countering Location Fraud in GeoSocial Networks. In *Proceedings of the IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2012.

[178] Bogdan Carbunar, Radu Sion, Rahul Potharaju, and Moussa Ehsan. The Shy Mayor: Private Badges in Geosocial Networks. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*. Springer, 2012.

[179] Bogdan Carbunar, Radu Sion, Rahul Potharaju, and Moussa Ehsan. Private Badges for Geosocial Networks. *Journal Proceedings of IEEE Transactions on Mobile Computing (IEEE TMC)*, 2013.

[180] Bogdan Carbunar, Radu Sion, Rahul Potharaju, and Moussa Ehsan. The Shy Mayor: Private Badges in GeoSocial Networks. In *Poster Proceedings of IEEE Symposium on Security and Privacy (Oakland S&P)*, 2013.

[181] Rahul Potharaju, Jeff Seibert, Sonia Fahmy, and Cristina Nita-Rotaru. Omnify: Investigating the Visibility and Effectiveness of Copyright Monitors. In *Proceedings of Passive and Active Measurements (PAM)*. Springer, 2011.

VITA

VITA

Rahul Potharaju was born in Hyderabad, India. In 2007, he came to the United States, where he did his MS in computer science at Northwestern University, Evanston, Illinois. After graduating in 2009, he did his PhD in computer science at Purdue University, West Lafayette, Indiana. His research focuses on improving the reliability and security of large-scale services spanning from smartphones to datacenter networks. He is a recipient of the Motorola Engineering Excellence award in 2009 and the Purdue CERIAS Diamond Award for outsanding academic excellence in 2014. Several components of his dissertation either have undergone a tech-transfer or are being used by multiple business groups inside Microsoft; NetWiser, a Microsoft Research project on which he worked, was awarded the *Microsoft Trustworthy Computing Reliability Award* for 2013. He applies techniques from measurements, data analysis, natural language processing and modeling to a wide range of problems such as characterizing network failures in large datacenters [161, 162, 164], automating problem inference on human-written trouble tickets [24, 165, 166], determining an accurate Internet topology mapping to aid Internet emergency response systems [167, 168], adversarial testing of wireless routing implementations [169], understanding the extent of spread of malware and plagiarism in smartphone applications [170–174], designing caching algorithms in video systems [175, 176] and alleviating privacy concerns in geosocial networks [177–180] and peer-to-peer networks [181].