UNIVERSITY OF CALIFORNIA
SANTA CRUZ

DATA-DRIVEN APPROACHES AND SYSTEMS FOR THE RELIABILITY OF MOBILE NETWORK SERVICES

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Xiaofeng Shi

June 2022

The Dissertation of Xiaofeng Shi
is approved:

Professor Chen Qian, Chair

Professor Roberto Manduchi

Jia Wang, Ph.D.

Peter Biehl
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Data-driven Approaches And Systems for The Reliability of Mobile Network Services

by

Xiaofeng Shi

A fundamental goal of mobile network service providers is to guarantee the networking services' reliability, which includes two major goals: reliable network access and security of the communication channels. As today's mobile network is getting larger and more complex, traditional protocols and systems for satisfying the above goals, which are mainly based on rules and experiences, are no longer scalable or effective. On the other hand, together with the growth of the mobile network scale is the growth of the mobile service data. The rich context and knowledge behind the large-scale network service data can provide us with new opportunities to understand the network states and support the design of reliable network systems. Therefore, this thesis research mainly focuses on exploring and designing data-driven approaches to enhance mobile network service reachability and security. Specifically, this paper focuses on two challenging problems faced by the mobile service providers regarding network service reliability.

The first problem is how to automatically and efficiently recognize the root cause of a network accessibility problem experienced by the end-device users in cellular network services. Specifically, we design and implement an automatic troubleshooting system named NeTExp . NeTExp learns to identify the root causes of a user-side service problem through deep neural networks (DNNs) that are capable of extracting

the complex spatial-temporal features of the massive cellular network log data. It also uses advanced weakly-supervised learning methods for training the models and thus overcomes the data limitation challenges in practice. The system is trained and validated using an extensive period of network and customer care data from a major US cellular service provider.

The second problem is how to enable low-cost and fast peer-to-peer authentication among a large mobile network ecosystem, such that secure mobile channels can easily be guaranteed for any pair of devices in the network. The critical challenge is that many mobile devices, especially IoT devices, do not have the capability to maintain the certificate revocation status for a vast device universe during the authentication process. To solve this problem, we design a fast on-device authentication prototype called TinyCR. TinyCR utilizes super-efficient data structures to maintain the certificate revocation status on each device locally. It also enables fast synchronization in response to any changes in the certificate universe. Through evaluation, we show TinyCR outperforms other state-of-the-art certificate revocation checking protocols regarding memory cost, checking efficiency, and synchronization cost. Those new features of TinyCR can well enhance the peer-to-peer channel security in a large mobile network.

Acknowledgments

I sincerely thank the tremendous support from my research labs, my thesis reading committee, my research work collaborators, and my family throughout my phd study and the writing of this dissertation.

First of all I am extremely grateful to my faculty advisor, Prof. Chen Qian, for his patient advising and continuous supports throughout my four-year phd study at UCSC. He is not only a great supervisor on my phd academic research, but also a mentor for my lifelong career. Under his supervision, I learned how to become a mature and responsible researcher and how to wisely make plans for my future career.

Then I want to express my special thank to Dr. Jia Wang, who was my project supervisor while I was collaborating with AT& T Labs. I would like to thank her for giving me the opportunities to learn about the large-scale real-world operating networks and the practical research problems concerned by the nationwide network operators. I also want to thank her for the tremendous support and insightful advice while solving those challenging problems.

Next, I want to thank every thesis reading committee member and phd qualifying exam member, Prof. Chen Qian, Prof. Roberto Manduchi, Prof. Yang Liu, and Dr. Jia Wang, for their valuable advice on how to improve my phd thesis work and present my work in the defense. I also want to acknowledge all my lab mates, research and teaching collaborators, and my friends for their great support and assistance in my research projects, teaching activities, and daily lives.

Finally, I would like to thank my parents for their great support while I was

studying and living aboard, and my wife Minmei for her continuous company and help during the whole period of my phd study.

Chapter 1

Introduction

1.1 Motivation.

A fundamental goal of mobile network service providers is to provide reliable networking services to their mobile device users. The reliability includes two major goals: reliable network access and security of the communication channels. For example, if a mobile device experiences any service accessibility issues, the mobile service provider should effectively and timely troubleshoot the problem. In addition, the service provider should provide security mechanisms through which the end-to-end channels between the communicating parties cannot be attacked. Traditional protocols and systems for satisfying the above goads are mainly based on rules, human experiences, and manual efforts. However, today's mobile network is getting more extensive and complex. Hundreds of millions of mobile devices and network devices are connected in a mobile ecosystem with heterogeneous network functions, protocols, and systems. As a result, many traditional designs are no longer scalable or effective in today's giant

mobile networks.

Together with the growth of the mobile network scale is the growth of the mobile service data. The rich context and knowledge behind the large-scale network service data can provide us new opportunities to understand the network states, design the networking systems with automation, and enhance the security of the networks. Meanwhile, how to abstract the massive data for efficient querying is also critical for time-sensitive services that support network reliability, such as real-time troubleshooting and one-shot device authentication.

Therefore, this thesis research explores and designs the methods that use data-driven approaches to improve the robustness and security of state-of-the-practice large-scale mobile networks. We stand at the perspective of the mobile service providers who collect and maintain the network status data and provide their users with the necessary system supports for service reliability. Specifically, this thesis focuses on two key challenges regarding mobile network reliability:

1. How to efficiently troubleshoot service accessibility problems in commercial cellular networks?

2. How to guarantee peer-to-peer authenticity in a scalable way for millions to billions mobile user population?

In order to solve the above two challenges, we use and design data abstraction, analysis, and learning methods upon the huge datasets maintained by the service providers and implement automatic systems for practical networks.

1.2 Contributions and Thesis Outline

1.2.1 Part I. Towards Automatic Reactive Troubleshooting and Issue Root Cause Identification in Cellular Services.

Summary of Part I

An essential task of cellular carriers is providing reliable and high-performance cellular service access for end-device users. In order to guarantee the reliability of access and improve users' experience, the carriers need to put tremendous effort into resolving the service outages or performance degradation issues experienced by customers. In practice, the issues could be attributed to a variety of reasons, such as network outages/maintenance, provisioning errors, mobile phone hardware/software bugs, and external events. Many automated functions have been deployed in the current operating cellular networks to monitor the network status and proactively detect the on-going or potential network failures (such as outages or anomalies). Those systems can effectively detect major network issues that would affect a large population of users in the areas.

Despite the effectiveness of those proactive issue detection system, not all service issues experienced by the individual customers can be properly solved through the proactive systems. There could still be many issues that are case-specific, such as the problems from the specific user equipment, provisioning issues, and some isolated network problems that closely impact the quality of the specific user's experience. In addition, even if the network issue has been known by the provider, the provider also needs to respond to customers about those known issues and resolve their concerns. As a complementary method, upon experiencing those cellular service degradation issues,

one traditional way for the customer to inquire about and resolve an issue is to actively contact the customer care services and report the experienced issues. Then the service provider can respond accordingly regarding known network issues, or reactively investigate the root causes and help customers resolve the problems as timely as they can.

In order to improve the efficiency and accuracy of the reactive troubleshooting, we explore novel automatic troubleshooting methods for cellular services by utilizing the tremendous network log data owned by the service providers. We propose a generic and comprehensive data-driven troubleshooting system called NeTExp (**Ne**twork **T**roubleshooting **Exp**ert) for recognizing the network-related issues in the online reactive troubleshooting phase. NeTExp can automatically answer the following two questions in the customer interaction phase: 1) are there any network anomalies that may have impacted users in particular serving cells; 2) whether the root cause of a service issue reported by the customer is a network problem. Note that some network anomalies may only impact a subset of users in the area.

Our contributions are summarized as follows:

1. We propose a generic framework for automatic service troubleshooting in cellular networks that significantly improves network problem identification and reduces troubleshooting costs.

2. We make the first attempt to jointly model the complex correlation of the network conditions among the neighboring cell sites, as well as their impacts on the user equipment (UE) of the areas using customized machine learning tools.

3. We evaluate the system using massive network log and care data from a large US cellular provider. We also apply the model to study a real network problem in 2020. The results demonstrate the effectiveness of the system.

4. We further explore the data imbalance problem in the customer care data received by the cellular providers. We apply state-of-the-art weakly supervised learning strategies to enhance the system performance. The enhanced model also produces interpretable cell site state profiles and classification workflows to help the human care agents understand the decision-making process.

5. We implement a trial system using the real-time data feeds. The trial system can be deployed online and resolve customers' issues in a short latency.

Chapter 2

Chapter 2 presents the background of cellular networks and the state-of-the-practice *reactive* service issue troubleshooting frameworks used by major cellular providers. Next, this chapter reviews existing works for automatic troubleshooting in cellular networks and explains the major challenges in reactive troubleshooting and the network log datasets that can be used to tackle those challenges.

Chapter 3

In Chapter 3, we present our prototype system design for data-driven automatic troubleshooting in cellular networks. The prototype system extracts feature representations from both the cell site level perspective and user device level perspective and builds a profile of the reported issue. It uses advanced deep learning tools to

model the features and learns to classify the root causes of the issues. We evaluate the prototype system using large-scale network logs and customer care service data from a major US cellular provider. In addition, we apply the system to analyze an actual network problem event in 2020 to show the effectiveness of the prototype model.

Chapter 4

In Chapter 4, we study a practical problem for training the troubleshooting model, namely, the ground truth imbalance problem due to the limitations of the state-of-the-practice troubleshooting process. To solve this problem, we improve the proposed prototype system by introducing a weakly-supervised learning framework that takes only positive class samples and unlabeled samples. In addition, we show the detailed system implementation design of the proposed troubleshooting framework, which enables the system to be deployed in online care contact troubleshooting phases.

1.2.2 Part II. On-device Certificate Revocation Checking for Peer-to-Peer Mobile Network Applications.

Summary of Part II

Recent years have witnessed the rapid growth of Internet of Things (IoT) devices widely deployed in various applications [6]. With the growing trend that IoT services scale from local area domains to wider area domains, there is an increasing demand for secure peer-to-peer communication protocols in a universe with millions of IoT devices. Under this context, many security protocols for IoT should be re-designed. For example, future IoT devices can use any untrusted access point (such as a public 5G

AP) to connect to the Internet or use the short-range wireless media such as Bluetooth and visible lights to communicate with another device directly.

Thus, peer-to-peer *device authentication* becomes a fundamental security problem of novel IoT and the building block of many emerging critical IoT security protocols for communication privacy (such as the TLS-style protocols) as well as IoT data authenticity and integrity (such as the digital signature protocols) [1, 2, 44, 50]. The key method for automatic device authentication is using the Public Key Infrastructure (PKI) based authentication protocols, where each device maintains a certificate that is issued by the certificate authorities (CAs). However, on-device certificate verification remains a challenging problem mainly due to the high latency and bandwidth cost of the revocation-checking step. In particular, when a revocation happens, how soon other parties are aware of the revocation and no longer trust the device becomes a rather critical metric of the security property in the protocol. However, to our knowledge, there is no solution for on-device IoT certificate revocation (CR) checking that satisfies all requirements in real IoT applications.

To fill this gap, this work presents TinyCR [65], the first IoT certificate system for on-device CR checking, which achieves all the five above requirements. Our key innovation is a new compact and fast data structure named Dynamic Asymmetric Set Separator (DASS) to represent the revocation status of all certificates on IoT devices **with zero error**. TinyCR also includes a management program running on a server maintained by the IoT service provider to synchronize the DASS on devices, which can be easily replicated to avoid a single point of failure. We have implemented both the management and on-device programs. Based on our analysis and evaluations, TinyCR

is the ideal solution for CR in the scenarios that demands: 1) Fast or frequent authentication. TinyCR has a clear advantage in latency compared to OCSP and equivalent performance compared to CRLite [41] (the state of the art). 2) Small on-device memory cost. TinyCR costs slightly less memory than CRLite and much less than other CRL solutions when the revocation ratio is low. 3) Low CRL synchronization latency. The faster the devices can realize a revocation made by the CA, the lower is the risk for certificate abuse. To our knowledge, TinyCR is the first on-device CR checking protocol that supports real-time or high-frequency updating in response to the certificate set changes. 4) Low bandwidth cost for synchronization. Experiments show that the bandwidth cost of TinyCR is orders of magnitude lower than that of CRLite for most practical updating scenarios.

Chapter 5

Chapter 5 illustrates the background and the key challenges for the PKI-based authentication methods for an IoT ecosystem. Then the chapter introduces and explains the state-of-the-art works for solving those challenges, as well as their limitations when being deployed in the IoT scenarios. In addition, this chapter introduces a new compact data structure named DASS. DASS can precisely maintain the certificate revocation status using small memory and overhead that can be afforded by the IoT devices.

Chapter 6

Chapter 6 presents a novel peer-to-peer authentication protocol named TinyCR. In this chapter, we first summarize the threat model that characterizes the adversarial

scenarios and the constraints for PKI-based peer-to-peer authentication in IoT. Then we present the overall design of the TinyCR system for certificate revocation checking based on DASS. Next, we show the performance evaluation results for TinyCR and discuss the security and performance properties when TinyCR is deployed in real IoT systems.

1.2.3 Conclusion and Lessons Learned

Chapter 7

Chapter 7 concludes and compares the two research topics included in this thesis. Both problems studied in this thesis are related to handling the large-scale mobile service data and solving a classification problem. However, we adopted two different data abstraction methods for the two problems. For the troubleshooting problem, we applied machine learning based methods, while for the security problem, we designed a set query tool based on hashing and indexing. The discussion of Chapter 7 compares the two tracks of the solutions for big data and illustrates how to select the methods for practical problems.

1.2.4 Chapter 8

Processing heterogeneous data at a trillion-byte scale and applying the results in a practical networking system is not straightforward. In Chapter 8, we summarize the lessons learned from handling the large-scale data for a nationwide network. We hope the experiences can help future studies related to the data-driven approaches with practical big data.

Part I

Towards Automatic Reactive Troubleshooting and Issue Root Cause Identification in Cellular Services.

Chapter 2

Service Issue Troubleshooting in Cellular Networks

In this chapter, we briefly review the current cellular network infrastructures and illustrate the state-of-the-practice *reactive* service troubleshooting framework used by the cellular service providers. Then we present the major challenges in service troubleshooting and the limitations of the existing automatic troubleshooting frameworks for cellular networks. Finally, we show the data sources that can be used to support a data-driven automatic troubleshooting framework.

2.1 Background.

2.1.1 Cellular Networks

Cellular network ecosystems include four major components: user equipment (UE), radio access network (RAN) infrastructures, core network (CN), and the external

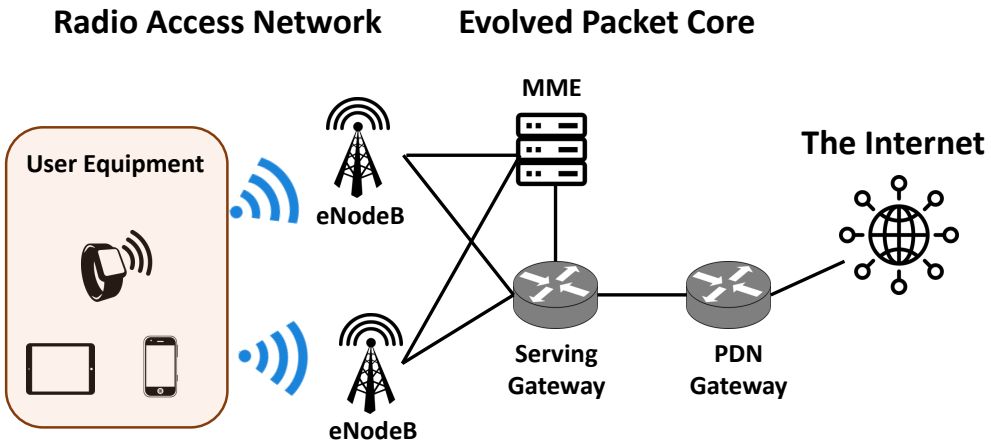


Figure 2.1: The 4G LTE cellular network architecture.

Internet or Public Switched Telephone Network (PSTN) [54]. Fig. 2.1 takes the 4G Long-Term Evolution (LTE) as an example to illustrate the four major components [28]. Recent 5G cellular networks use a similar architecture. Specifically, non-standalone 5G network pairs the 5G RAN with the LTE evolved packet core (EPC) network, while standalone 5G network has its own core network with similar but more sophisticated major components as well as additional virtualized network functions and features to support emerging 5G services such as massive Internet-of-things (IoT), autonomous vehicles, VR/AR, etc.

The user equipment (UE) usually refers to any end-user's device (e.g., a mobile phone, a tablet, a laptop, and various IoT devices) that connects to the base stations (BSes) of the RAN layer to receive cellular services, such as phone call services and Internet access, etc. An important feature of UE is mobility, namely, the device is supposed to have seamless cellular service access as it is mobile, which is supported by the hand-off mechanism.

The UE accesses the cellular service through the radio channel between the UE to the nearby BS radio nodes (such as the LTE eNodeBs via 4G radio, and the gNodeBs via 5G radio) in the RAN. We call the BS node that currently serves the device as the serving node, or the serving cell site. To enable wide coverage of cellular services, the area is covered by multiple cell sites that are geographically distributed at different locations. Each cell site's ratio covers a local area. The radio Reference Signals Received Power (RSRP) and Reference Signal Received Quality (RSRQ) depends on many factors such as the distance from the location to the cell site node, blockages, reflection, and the power of the radio nodes. When a device roams from one location to another and performs handover, it measures the RSRP and/or RSRQ of different radios at the new location. The new serving cell site is then selected based on the measures.

The main function of the core network is switching the mobile data (e.g. voice calls, text messages, and packets). In addition, it is also responsible for logging subscriber profile information and location, performing service authentication, and many other network functions. For example, in LTE evolved packet core (EPC), the Serving Gateway (S-GW) is mainly responsible for routing user's data packets within the core network or performing inter-eNodeB handovers, while the Packet Data Network Gateway (P-GW) is mainly used for routing data packets to the external Internet. The MME (Mobility Management Entity) node is mainly used for tracking and managing user's mobility, idle mode UE paging and tagging, bearer activation/deactivation, user authentication, etc.

In standalone 5G core network, the functionality of the MME is decomposed to the 5G Core Access and Mobility Management Function (AMF), which is responsible for

mobility, authentication, and registration management, and the Session Management Function (SMF), which is mainly responsible for session management, policy control functions, charging functions. In the data plane, the User Plane Function (UPF) is mainly used for switching the traffic between the device and the data network, such as the Internet, and IMS (IP Multimedia Subsystem). In addition, the Unified Data Management(UDM) function is used for the storage of subscriber's profile information, authentication information, encryption keys, etc.

2.1.2 The Troubleshooting Workflow in Cellular Services

In practical cellular networks, although tremendous automatic systems have been deployed for the detection, forecasting, and troubleshooting of the major network issues, it is unavoidable that mobile device users sometimes still experience service accessibility problems that may be attribute to a variety of internal and external issues, such as temporary congestion, mismatch of network state and device state, network maintenance, a misconfigured mobile device, an impaired SIM card, a device software bug, iCloud server issue, bad weather, etc. For example, in rare cases some users may experience accessibility issues to phone call services, Internet, Short Message Service (SMS), etc. Some other impacted users may be able access the services while the performance of the services are significantly degraded compared with their former experiences.

Upon experiencing a case-specific service degradation problem, customers may contact the customer care of the service provider to report and resolve their issues. In some cases if the issue has already been known and is being investigated and resolved by the service providers, the providers would inform the customer about the issue and the

expected resolving time. In other cases if the root cause issue is not known yet, then the service providers would *reactively* investigate the root causes and help customers resolve the problems as timely as they could. One key metric to measure the effectiveness of customer support is the resolution time for customer reported issues. To reduce the resolution time, it is critical to (i) minimize the time spent on inspecting the problem and identifying the root cause during the live conversation between the customers and the agents, (ii) minimize the number of customer tickets that need to be sent to tier-2 support teams, and (iii) minimize number of tier-2 teams that a ticket is routed through before it is resolved. For example, if we can quickly determine that a reported issue is related to a known root cause, then there is no need to create a ticket for further investigation. If we can determine a reported issue is not related to any known event and is likely to be network related (instead of device related), then the ticket will be routed directly to the network support team for resolution. It is important to note that these decisions need to be made at *per user device level*.

According to our study on the reactive troubleshooting and resolution process of a major US cellular provider, the process often consists of two phases: the *customer interaction* phase and the *ticket resolution* phase. A summary of the whole process workflow is illustrated in Fig. 2.2.

During the customer interaction phase, the customer either speaks to an agent by calling the toll-free number or chatting with an agent via online chatting tools. The agent will first collect and verify the information reported by the customer and then go through a sequence of designated steps to troubleshoot and resolve the service issue. These troubleshooting steps may involve checking customer account status, veri-

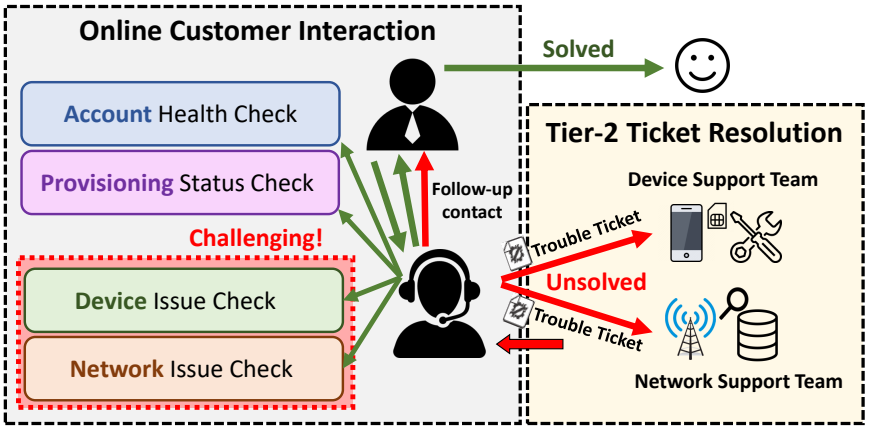


Figure 2.2: A summary of the reactive troubleshooting process for cellular service problems.

fying provisioning status, determining if the customer is impacted by any known issues (e.g., network outages, maintenance activities, device hardware/software bugs, external events), examining device configuration setting and performing other device-specific diagnoses. The goals of this phase are to identify the root cause of the service issue, perform corresponding resolution actions, and verify the effectiveness of resolution actions as much as possible within a short time duration (e.g., a few minutes) while the customer is actively engaged with the agent on call or on chat.

While the majority of customer-reported service issues can be successfully resolved at the end of the customer interaction phase, some service issues may need in-depth investigation before a root cause can be identified and corresponding resolution actions are performed. Note that these remaining services issues can be either network- or device-related. The agents will create customer trouble tickets and dispatch them to the Tier-2 support teams for in-depth investigation. During the customer ticket resolution phase, investigation often requires gathering and analyzing measurement data over a certain time period from the service region of the impacted mobile devices at both

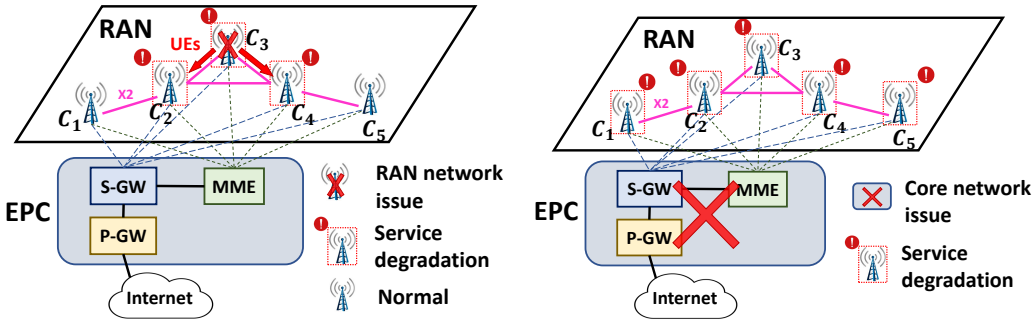
cell site level and individual mobile device level. If no network- or device-related issue is found as the root cause of the reported service issues, the tickets will be returned to the customer care team and customers may be directed to local stores for further assistance. Depending on the complexity of the issues, the ticket resolution phase usually takes hours to days.

2.2 Problem Statement.

2.2.1 Key Challenges in Cellular Service Troubleshooting

While some of the above mentioned tasks (e.g., checking account and provisioning status) can be executed by software in an automated fashion, troubleshooting network- or device-related issues in both the customer interaction phase and the ticket resolution phase are largely manual due to the following challenges.

First, the process of troubleshooting a service issue at per UE (user equipment) level is inherently complex. There are a variety of causes of service degradation including different types of network issues and device issues, many of which produce similar symptoms (such as Internet connection failures, voice call drops, slow data rates, etc). Therefore, diagnosing based on the UE-side symptom itself is insufficient to identify the root causes. It is particularly challenging to discover the non-outage-related service issues that are caused by non-fatal or partial network-side or device-side issues. Each of these non-outage-related service issues likely impacts only a very small number of mobile devices at a given time and a given location. Some of these service issues can be intermittent or chronic. In addition, even the presence of an outage on a cell site does



(A) A RAN issue (B) A core network issue

Figure 2.3: Example scenarios of network issues.

not necessarily mean that all customers who experience service issues in the local area are indeed directly caused by the outage. It is possible that some of these customers are impacted by impaired firmware or external events. Therefore, precisely determining the root cause of each individual service issue often requires applying advanced domain knowledge in analyzing a massive volume of network data.

Second, it may not always straightforward to discover the network problems on the cell level and estimate the scale of the impacted users and areas. Fig. 2.3 illustrates two example network issue scenarios in LTE networks. In Fig. 2.3 (A), the cell site C_3 is experiencing service degradation due to a radio access network (RAN) problem with its infrastructure or software. Consequently, a good portion of UEs that were originally served by C_3 are handed over to its neighboring cell sites C_2 and C_4, which also causes congestion on C_2 and C_4 and impacts the experience of the customers who are served by them at different levels. In the second scenario (B), a network issue happens with the core network of LTE. The problem may influence the service performance of a large number of cells. The impact on the service for different UEs could be various. The

examples in Fig. 2.3 show that the impact of a network problem on a cell site or a core gateway may not only influence the corresponding cells but also propagate to further cells, which make it challenging to correlate user tickets with some known network issues. In addition, since different types of network issues present diverse anomaly and propagation patterns, it requires a decent understanding of the event patterns and their correlation among the neighboring cell sites to figure out the impact of a network problem regarding the user-level quality of experience (QoE). Moreover, the diverse physical factors of different regions, such as the geographic features, the distance between the neighboring cell sites, the density of the cell sites, and the population and mobility of the users, further complicate the problem.

Third, only a small portion of customers report their service issues. Most customers never contact care support upon experiencing a service issue. Depending on type and severity of service issues, some customers wait for a period of time before they contact customer care. The information provided by customers regarding their service issues can be ambiguous or inaccurate. As a result, some of these issues need extensive investigation before they are resolved. In extreme cases, some issues are not resolved due to insufficient information. The corresponding customer trouble tickets may be routed among multiple tier-2 support teams and finally returned as "No Trouble Found". This will result in not only longer resolution time for the mis-routed tickets and unhappy customers, but also preventing agents from working on tickets that they should focus on.

2.2.2 Design Goal: Learning-based Troubleshooting

In this thesis study, we design a machine learning based troubleshooting tool that aims at helping customer care agents effectively distinguish if a customer reported service degradation is likely caused by a network related issue or by a device related issue in the customer interaction stage, and helping tier-2 support teams to identify the cell site(s) that likely contribute to the service degradation if the service degradation is network related in the ticket resolution stage. Our machine learning based approach would greatly reduce the manual investigation involved in the troubleshooting process, and hence reduce the resolution time. In addition, the automatic troubleshooting tool can extract interpretable insights about the network status and the decision-making process, which can assist the care agent in manual inspection and issue resolution.

2.2.3 Related Works

Automatic troubleshooting for customer-level service issues in cellular networks.

Along with the tremendous growth of the market for heterogeneous access networks and end devices in the recent decade, manual troubleshooting strategies for the issues experienced by customers have become less scalable to the growing device population. Thus, automatic troubleshooting methods for customer-level service issues have attracted more attention by the service providers and network research community. Currently, there are two major branches towards automatic troubleshooting and network problem detection: the network data based methods and the natural language processing

(NLP) based methods.

The network data based methods use the network log and/or the trouble ticket data to detect the network anomalies [13, 16, 30, 32–34, 39, 43, 58, 64]. With this perspective, the cellular carrier can predict which customers will call in care, and take the necessary actions to resolve the problems and improve the network quality in advance. For example, IBM research [16] proposes using a random forest model to predict whether a user will contact the care. Sheoran, et al. [64] solve a similar problem based on the UE-level network log data. CableMon [30] is another learning-based system that can proactively detect network faults in the cable broadband network. Iyer, et al. [32] design a system that is specialized on detecting the RAN issues in the cellular network. Jin, et al. [33] propose a network issue detection method by analyzing the distribution and patterns of care contact logs. NEVERMIND [34] is a system that can predict future customer tickets in the digital subscriber line (DSL). All the above existing works target on training a proactive model that can forecast the trend of the emerging network problems or detect anomalies on the network level. However, they cannot directly resolve the user-specific issues in the customer interaction phase. ICCA [52] is a framework that can perform reactive troubleshooting for individual cases. However, it only relies on the UE-level network logs, while the correlation of cell-to-cell and cell-to-UE network states is ignored.

Another type of approach focuses on understanding customers' feedback or free-text ticket logs using NLP models [56, 69, 72], and diagnosis based on the described symptoms of devices. For example, NetSieve [56] is proposed to diagnose the problem by understanding the network trouble tickets. LOTUS [72] is a system for identifying

which customers are impacted by a local network issue based on the care contact log texts. However, these methods are mainly log-based instead of chatting-based, namely, the problem could only be resolved after a ticket log is generated, rather than during the conversation of the care contacts. This type of methods is orthogonal and complementary to our proposed method.

ML-based cellular network systems.

The state of the practice cellular network is getting ever larger and more complex. As a result, there is an increasing demand for automation in cellular network system designs. In recent years, DNNs have become popular for learning cellular network data because of their high capacity of representing spatial-temporal features. For example, DMM [63] uses a recurrent neural network (RNN) model to learn the traveling trace of the UEs based on their cellular data logs. Microscope [86] adopts a 3D Deformable Convolutional Neural Network for mobile service traffic decomposition and network slicing. DeepLoc [66] utilizes a multilayer perceptron (MLP) for device localization based on the RSS of cellular signals.

2.3 Datasets.

Table 2.1 lists the data sources that are widely used or generated during the troubleshooting phases of the state-of-the-practice framework described in section 2.1.2. The data mainly includes historical customer care contact log and ticket details, and cell/UE-level network status such as cell site Key Performance Indicators (KPIs) and user session states. The cell-level KPIs used in this paper include the average number of

Dataset	Short Description
Care Contact Log	Logs for the *customer interaction* phase. The log data is manually filled by the customer care agents. The data mainly include the care contact time, issue description and the recommended resolution provided by the agents, etc.
Trouble Tickets	The ticket data handled by the *Tier-2 team* in offline. It usually includes the resolutions provided by experts for the hard cases that cannot be resolved during the online phase.
Cell-level Network Log	Real-time KPIs of the cell sites. The data is automatically collected at the eNodeB or gNodeB. The data includes the timestamp of the measures and the performance counter values.
UE-level Network Log	Cellular session log for each UE. The data is automatically collected at the core network gateways. It mainly includes user ID, timestamp of the session, duration, the serving cell sites, and session status.

Table 2.1: Summary of datasets.

Radio Resource Control (RRC) connections (which reflects the temporary user population), and the average utilization ratio of the Control Channel Elements (which reflects the congestion status). We design the data-driven automatic troubleshooting system by learning from the above data. In this work, the datasets are obtained from a large cellular service provider in the US. All datasets are kept anonymous when being used for privacy reasons.

Chapter 3

NeTExp: A Data-Driven Method for Automatic Troubleshooting Individual Customer Issues in Cellular Networks

3.1 System Overview.

We design a machine learning-based troubleshooting framework NeTExp as shown in Fig 3.1. NeTExp includes two major modules: (i) a *proactive cell-level* network state prediction model and (ii) a *reactive UE-level* troubleshooting inference model. The proactive cell site level model predicts the likelihood of a cell site to have network issues that impact customers in the local area (i.e., the covered cells). The UE-level model

infers whether a customer-reported service issue is network-related.

During the training phase, the cell-level prediction model is trained using historical usages, user mobility, performance metrics at the cell site level, and customer care contact and ticket data. The UE-level inference model is trained using the output of the cell-level prediction model, the historical UE level usages, user mobility, performance metrics, and the customer care contact and ticket data. During the inference phase, the cell-level prediction model proactively predicts the cell sites that are having customer impacting issues and quantifies the severity of the problem based on the real-time usages and cell site level performance metrics data. Upon receiving a customer contact reporting a service issue, the UE level inference model will take the cell site level prediction on current customer impacting network issues and current UE level usage, mobility, performance metrics to infer whether the customer reported service issues is caused by network-related issues or by device-related issues.

Different from prior works, the cell-level model fully considers the interaction among neighboring cell sites, and the UE-level model is the first reactive network issue diagnosis method that is based on the perspectives from both the UE side and serving cell site side, and how their states match each other. This will not only help customer care agents to create a trouble ticket and dispatch to the corresponding support team for resolution, but also provide network support team enriched information for them to prioritize and focus on the right cell site for investigation and resolution.

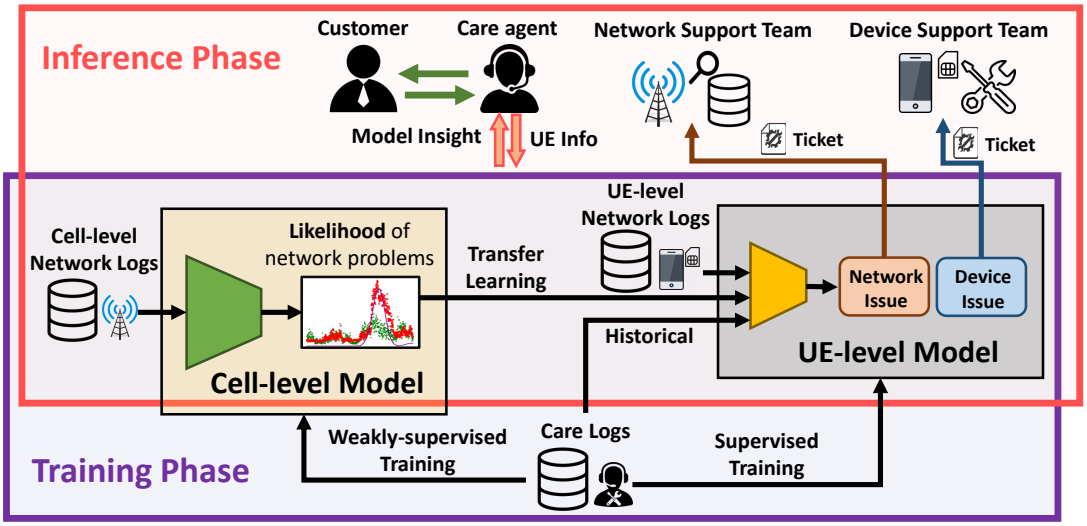


Figure 3.1: Overview of NeTExp .

3.2 Case Illustration

Before explaining the models, we first use a real-world example to illustrate the observations of the network log data when identifying a network problem. In the top chart of Fig. 3.2, we show the (scaled) average Radio Resource Control (RRC) connections (one of the cell level KPIs for learning) of two neighboring cell sites A and B in a local area. We can see the cell site A experienced a network issue from day 3 to day 4 (no RRC connection could be established with A). In addition, the problem also caused network congestion on B, as there was a notable increase of the RRC connections on B during around the same time. Thus, we can infer from the KPI patterns of the two cell sites that many users were handed-off from A to B when the issue occurred on A. As a result, the users who were originally served by B were also influenced by the network outage on A. We call such correlation among the neighboring cell site states as the **cell-to-cell** correlation. Note that other types of network issues may present

different symptoms on the KPIs of the cell sites in the area. However, manually reading and understanding the interaction of the anomalies of different cell sites from the raw KPI data can be difficult.

The bottom chart of Fig. 3.2 presents the example UE-level network logs of a customer who was served by B. The "CCL TS" represents the time when this customer contacted the care. The "Time Limit" and "Cellular Data" are two example state codes that describe the status of the data sessions in the UE-level network log data. Specifically, "Time Limit" indicates whether the UE can maintain the session longer than a time threshold, and "Cellular Data" indicates whether there is any down/up-link traffic generated during the session. On the y-axis, the "N" in the middle represents a normal service state, while the "Err 1" and "Err 2" represent the abnormal states of the two UE-level state metrics. This example clearly shows that shortly after the network problems happened on the cell site A, this device which connected with B also experienced service issues, i.e., it could not keep a stable data session or produce any download/upload traffic. Those symptoms motivated the customer to contact the care afterward. We call the correlation between the cell site states and the UE-side symptoms as the **cell-to-UE** correlation. In practice, the cell-to-UE correlation patterns of different users or different time could be highly diverse as a result of their different locations, usage patterns, mobility patterns, and device types, etc. For example, a mobile device may connect to tens different cell sites in one week. And the delays to contact the customer care after the issue could be various because of different time zones, issue occurrence time, problem severity, and users' habits. Therefore, it is rather time-consuming to track which cell site to inspect from the massive network logs.

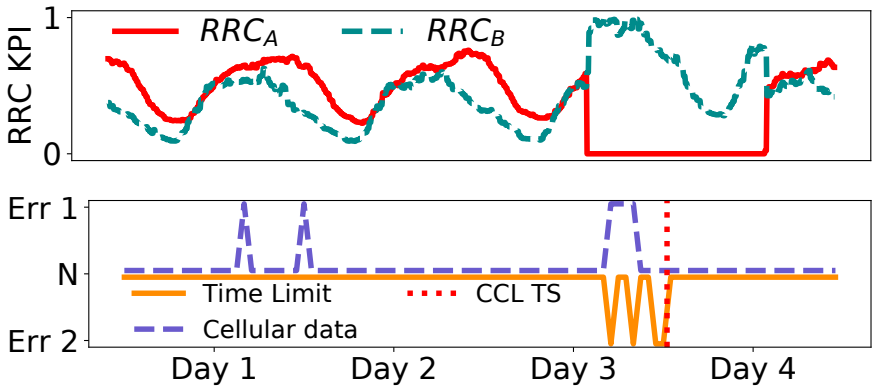


Figure 3.2: Top: the cell level RRC KPIs. Bottom: The UE-level service status of a customer in the area.

From the example in Fig. 3.2, we notice the cell-to-cell and cell-to-UE correlation patterns are important to identify a network problem, which is also the major insight for troubleshooting agents to manually inspect the network. Thus, the automatic troubleshooting model should be capable of learning those patterns.

3.3 The Cell-level Model Design.

3.3.1 Feature Modeling

Learning the correlation and interaction between the neighboring cell sites is important for cellular data analysis [61, 62, 78, 86], which is also challenging as it depends on many real-world factors, such as the local distribution of UEs and cell sites, the mobility of the customers, geographic features, and channel protocols. To solve this challenge, we design a graph model to represent the interaction between cell sites and propose using the graph convolutional neural network (GCN) [26, 59, 77, 79] to jointly learn the cell site node features and their correlation.

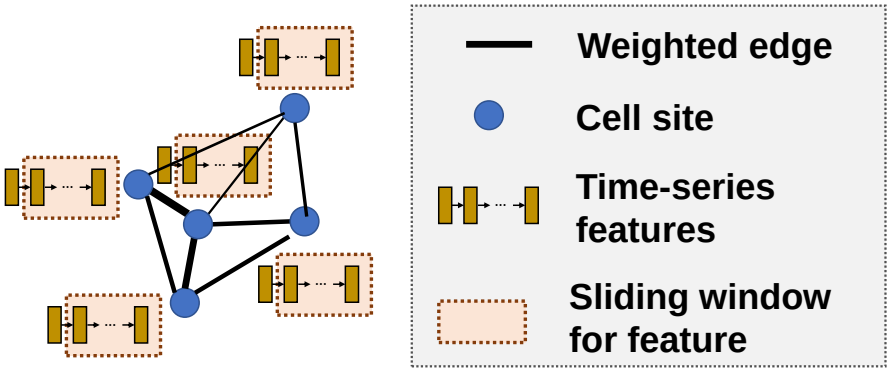


Figure 3.3: The graph modeling of the cell sites.

Specifically, the graph model is shown in Fig.3.3. In the graph G, each node represents a cell site and each edge represents the proximity (weight) between the two neighboring cell sites. The proximity can be defined in multiple ways and is discussed later. On each cell site vertex in the graph, the network condition is represented with a time-series feature acquired by sliding a feature extraction window through the streaming cell-level network log data.

Assume the pair-wise proximity among the k cell sites can be quantified by a 2-D adjacent matrix $A^{k\times k}$ (where each entry $a_{i,j}$ represents the proximity weight from node i to node j), let $G =< V^{k\times m\times w}, A^{k\times k} >$ represent the graph, where V is the $k\times m$ time-series features of the k vertices (i.e., m feature channels for each node, and each channel has time-window length w). Through a GCN layer, the feature on each cell site is recomputed by aggregating the features of itself and the other cell sites in the graph. For example, a typical GCN aggregation rule is defined as:

$$H^{(l+1)} = \sigma((I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})H^{(l)}W^{(l)}), \tag{3.1}$$

where $H^{(l)}$ is the node-wise feature input to the layer l ($H^{(0)} = V$), $W^{(l)}$ is a train-

able weight matrix that decides how the adjacency matrix $A^{k\times k}$ participates in the aggregation of the features, σ is a non-linear activation function, $I_k - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the normalized graph Laplacian, I_k is an identity matrix, D is the diagonal degree matrix with $D_{ii} = \sum_j A_{i,j}$.

In addition, the temporal feature (i.e., $H^{(l)}$) of the time-series network log data for the cell sites can be encoded by the 1D-CNN layers:

$$h_c^{(l+1)} = \sigma_t(h_c^{(l)} \circ W_t^{(l)}), \tag{3.2}$$

where $h_c^{(l)}$ is the time-series feature input of one cell site c in the graph ($h_c^{(0)}$ is the raw input feature), $W_t^{(l)}$ is the 1-D temporal CNN kernel, $\circ$ is the 1-D convolution operation, and σ_t is the activation function. Through the two types of the convolution operations, the model is cable of extracting features with the complicated spatial-temporal context. The detailed DNN architecture is explained in section 3.3.3.

A natural way to quantify the proximity in the adjacent matrix $A^{k\times k}$ is using the geographic distance of the cell sites [86]. However, the absolute distance is not representative enough since the base station selection of mobile devices depends on not only distance but many other factors, such as the geographic features, the density of the cell sites, mobility, and the UE and sector channel capabilities. Therefore, we propose a new metric for adjacency matrix quantification: **the average number of jointly served UEs by the two cell sites in unit time**. This metric is mainly inspired by the key observation that abnormal state transition among the cell sites is mainly caused by the hand-offs when a network problem happens to one cell site. Thus, this metric can be a good estimation of how much traffic will be handed off to a neighboring cell

sites when network problems happen on one cell site, and is a high-level product of all other unique physical factors in the local area. Most importantly, the metric values are easy to obtain by grouping the historical UE-level network log data with time intervals, (anonymous) user IDs, and cell site IDs.

Specifically, to quantify a_{c_1,c_2}, we group the vast historical UE-level network session logs (including all customers in the market regardless of whether they ever contacted the care) by the anonymous user IDs and record the accessed cell sites c_i by each UE in unit time. Then we measure the frequency of each (c_1, c_2) tuple that appears in the grouped logs for all UEs. Since the long-distance mobility of the UEs in unit time also impacts the measurement, we prune the graph by a distance threshold (e.g., 30,000 meters) and then keep the top $k-1$ neighbors with the largest a_{c,c_x} values as the "one-hop" neighbors of c. Note that the graph weight measurement is only performed once using a large historical interval of data (e.g., one month).

3.3.2 The Alternative Learning Target

In order to match the final goal for automatic troubleshooting for each specific user's service problem, it is ideal to use the precise resolution results from the human experts as the end-to-end learning target. However, since we cannot precisely know all customers who are impacted by a network problem (including the majority who do not contact the care upon experiencing an issue) and the cost for decently inspecting and labeling all reported issues is expensive, we can only obtain limited ground truth troubleshooting results for a small subset of the users whose tickets were resolved by the expert human agents. The lack of large-scale ground truth data makes it difficult

to train a DNN that learns from the high-dimensional network logs with tremendous spatial and temporal context.

To solve this challenge, we adopt the ideas from weakly-supervised learning and transfer learning [53, 82, 91]. Specifically, NeTExp uses an alternative learning target to pre-train the cell-level model (the heavy part of the overall NeTExp system): **how likely the cell is experiencing a UE-impacting network event** at each timestamp t. Although the model with the alternative target cannot directly answer whether a reported issue is a network-side issue, it is expected to provide the high-level representation of the network performance status for the cell sites, which is an essential insight for case-specific troubleshooting according to experienced ticket troubleshooting operators.

We take the following steps to build the transferred learning target: for each customer, we first retrieve a 7-day historical window of the UE network log records right before the care contact time, and obtain a set of cell sites that are accessed by the users and their accessing patterns. Then for each UE, we only keep the cell sites that are frequently or regularly accessed by the UE in recent time, and consider those cell sites as the "reference" cell sites for this UE. Then we aggregate the total number of customer contacts within a unit time interval by each reference cell site. The aggregation results provide an idea of how many service issues are reported for each reference cell site in each unit time interval. Thus, the intensive gathering of service issues for a reference cell site usually implies network issues in the corresponding cells. Similarly, we measure the aggregation numbers of the service issues that are diagnosed as network issues through the customer interaction phase and the ticket resolution phase using the ground truth

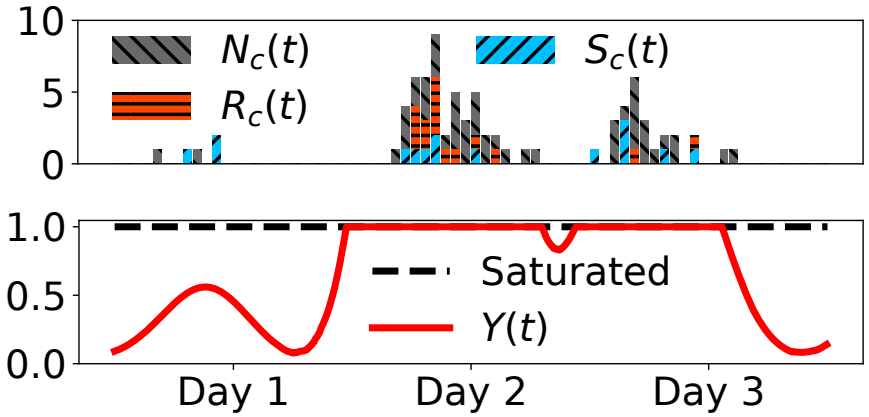


Figure 3.4: Top: the distribution of the reported service issues. Bottom: The transferred learning target.

troubleshooting tickets of the two phases, which provides extra dimensions about the scale of the influenced users in the area.

Thus, the learning target uses three vectors for each cell site c: $N_c(t)$, the number of total service issues over time; $R_c(t)$, the number of network issues identified during customer interaction; and $S_c(t)$, the number of network issues detected through ticket resolution. If a network issue happened on a cell site (or on its neighbors), a significant increase of N_c, R_c and S_c can usually be observed shortly after the issue occurrence time. An example of such case is shown in the top chart of Fig. 3.4 (in day 2 and day 3 compared with day 1). Based on this observation, the new learning target, i.e., the likelihood of network issues for the cell site c, can be quantified using N_c, R_c, and S_c. Specifically, we use the 1-D Gaussian Probability Density function $G(t, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{t^2}{2\sigma^2}\right)$ and compute the convolution of density kernel and the measurement vectors over the time dimension: $G(t, \sigma) \circ N_c(t)$, $G(t, \sigma) \circ R_c(t)$ and $G(t, \sigma) \circ S_c(t)$. Then the overall transferred learning target is defined as a weighted sum of the three density

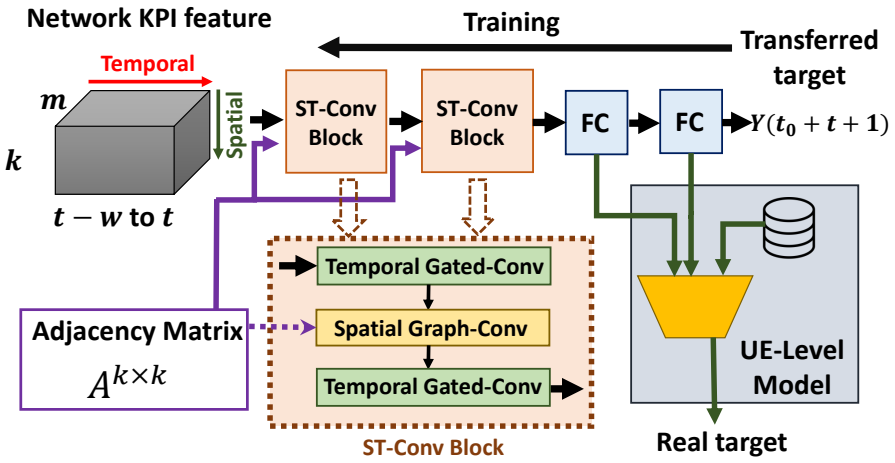


Figure 3.5: Model design of the cell-Level model.

vectors:

$$Y(t) = \alpha G(t,\sigma) \circ N_c(t) + \beta G(t,\sigma) \circ R_c(t) + \gamma G(t,\sigma) \circ S_c(t) \tag{3.3}$$

We then normalize $Y(t)$ and let $Y(t)$ saturate at 1 to make the likelihood values are in the range $[0, 1]$ and minimize the differences caused by the UE population size. An example of the normalized $Y(t)$ is shown in the bottom chart of Fig. 3.4. α, β, and γ are decided empirically and should be adjusted based on the effectiveness of the practical troubleshooting phases (N_c, R_c, and S_c) in the wild. Specifically, we look into the known network problems in the history and check the z-scores of N_c, R_c, and S_c during the network issue periods. A larger z-score indicates the corresponding measurement is more important. For example, for our studied cellular provider, we use $\beta \geq \gamma > \alpha$, since the network issue tickets (from both online and offline phases) are always more accurate network issue indicators than the total number of care calls.

3.3.3 Model Design and Training

In Fig.3.5, we show the overall design of the cell-level model to encode the graph-based cell-level features. The neural network is inspired from the STGCN [83] architecture.

The whole cell-level model is used as feature extractor to learn the cell-level features for each local area. A local area refers to the cells that are covered by the k neighboring cell sites. In the input feature matrix of height k, the first $m \times w$ feature slice refers to the features of the cell site that directly carries the target UE, while the rest $k-1$ slices are the features of its nearest neighbors ordered by the edge weights. Once trained, the whole model parameters are consistent for different areas in a large market.

In the input layer, the m cell-level time-series network KPIs are used as the input features. For the real-number KPI features, we first smooth the data with moving average to denoise the data. Since the traffic loads and capabilities of the cell sites are highly diverse, the KPI data is normalized before being fed for learning. One evident feature for the cell-level network KPI data is that the pattern of the KPI time series repeats every 24 hours because of the similar daily traffic patterns. Therefore, we normalize the KPI data by: $\hat{s}_t = \frac{s_t}{\bar{s}_{(t \ mod \ T)}}$, where s_t is the observed KPI at timestamp t of the global clock, and $\bar{s}_i$ represents the expectation of the KPI of the ith timestamps of a day based on the historical data, T is the number of total timestamps in a day. Thus, the normalized KPI $\hat{s}_t$ represents that at a particular timestamp $(t \ mod \ T)$ of the day, how the observed KPI compares with the expectation of the KPI for the same time

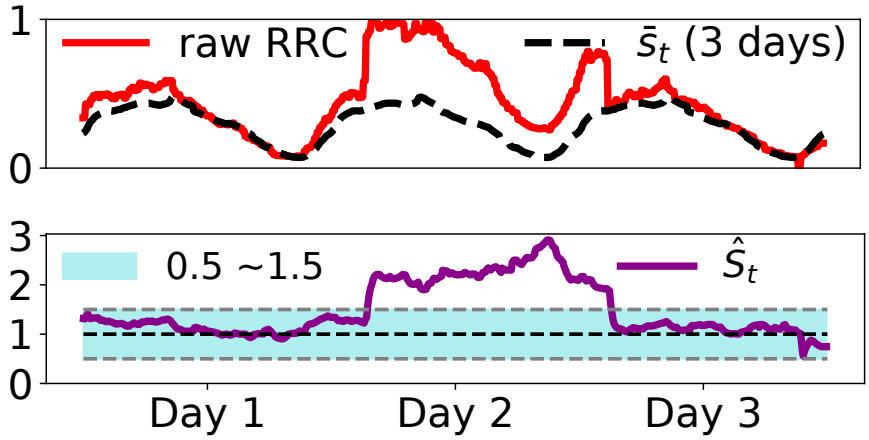


Figure 3.6: Top: The raw and the expected RRC KPIs (scaled). Bottom: The normalized RRC features.

of the day. We find this normalization method is effective for the KPIs that reflect or are related to traffic loads. For example, in Fig. 3.6, we show the raw and normalized average Radio Resource Control (RRC) connections for the same cell site in the case study of Fig. 3.4. From Fig. 3.6, we can find the normalization makes the abnormal network KPIs (in day 2) highly distinguishable. The abnormal RRC KPI states in day 2 of Fig. 3.6 can explain the increase of issue reports in day 2 and day 3 of Fig. 3.4.Thus, this normalization method can also work as an effective outlier detection module to highlight the abnormal patterns for a given timestamp of the day.

Next to the input layer are the two ST-Conv blocks (as shown in Fig.3.5) that can encode the spatial-temporal features. In each ST-Conv block, the feature of each cell site is fed into a 1-D temporal CNN layer (Eq.3.2) with the gated linear units (GLU) [15] as the activation. Then the processed features of all cell sites in the local graph are aggregated using a spatial GCN layer (Eq.3.1). The GCN block is then followed by another temporal CNN layer for each cell site of the graph to generate

the feature representation of the network conditions $H_l^{k\times g\times w_l}$, where g is the number of kernels in the last 1-D CNN layer and w_l is the resampled window size. After the two ST-Conv blocks, the model flattens the feature matrix over the time channel and use a fully connected (FC) layer with kernel size h to compute the $(k \times h)$-D feature representation $H_{FC}^{k\times h}$ of the network conditions on the k cell sites for the sampled timestamp. Thus, $H_{FC}^{k\times h}$ can be used as the extracted feature for the network conditions of the local area at a given time.

In the output layer, the model uses a regression loss function to learn the target Y^k of the k cell sites in the local area. The mean-square-error (MSE) loss is used for training:

$$L(H_o^k(t), Y^k(t+1)) = \frac{1}{k}\sum_i (h_o^i(t) - y^i(t+1))^2 + \lambda L_2, \tag{3.4}$$

where $H_o^k(t)$ is the output of the model with the input time window that ends at time t, $Y^k(t+1)$ is the transferred learning ground truth of the sampled k cell sites at $t+1$, $h_o^i(t)$ and $y^i(t+1)$ are the ith entry of $H_o^k(t)$ and $Y^k(t+1)$, and λL_2 is the L2 regularization term of the trainable parameters. The model is trained with Adam optimizer [36].

In our implementation, the four Temporal 1-D Gated-Conv layers of the two ST-Conv Blocks have 32, 16, 8, 4 CNN kernels respectively. The size of each kernel is 4, namely, the perceptive field length of the first CNN layer is 20 minutes. The number of neurons h in the feature embedding layer is set as 8. Through validation, we find larger model size provides limited accuracy improvement but more memory cost and overhead. Since our model is executed on CPU servers rather than GPU servers (due to data access restrictions), we do not choose to use a larger model configurations. The trade-off of the input data sizes, i.e., k and w, and model efficiency is discussed in section

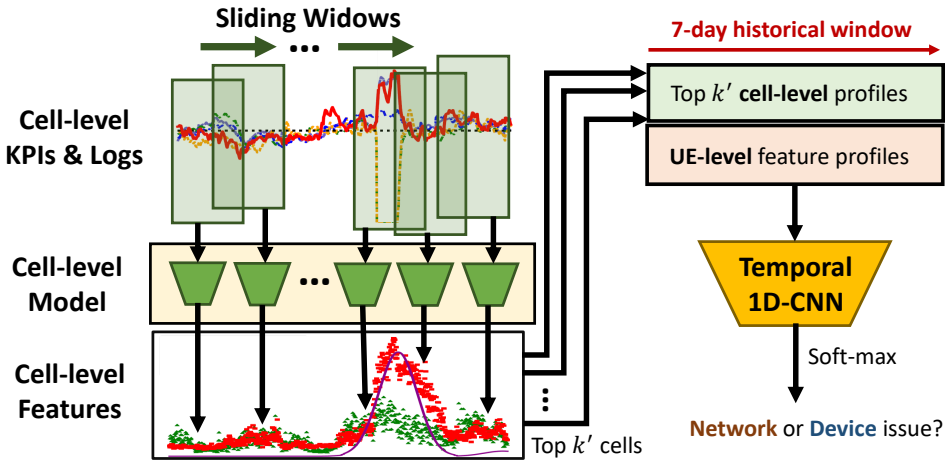


Figure 3.7: Design of the UE-Level model.

3.5.3.

After training using the transferred target, NeTExp freezes the parameters of the neural network. Then it feeds the learned $H_{FC}^{k\times h}$ and H_o^k to the UE-level model as the high-level feature representation of the cell site performance status.

3.4 The UE-level Model Design.

The UE-level model is the direct interface for the care agents to learn whether a reported problem is a network-side issue or a device-side issue. Besides the patterns of the UE network logs [52], another important feature is the temporal correlation of the UE-level service errors and the cell-level anomaly states. Thus, we design the UE-level model such that it learns from the features in both aspects.

Based on the historical data session logs for each individual UE, we can create the UE-level feature profiles for learning. Specifically, for each customer who contacts the care, we retrieve the data session logs (time/duration of the session, the accessing

cell sites, and the session status) for the target UEs. Then we can create a session usage pattern feature matrix $U_d^{n\times w'}$ for each UE d, where n represents the n feature channels (e.g. the different session metrics in the log data) and w' is the historical feature time window size for UE-level trouble inspection. Some other statistical features (such as how many times the UE is handed-over between multiple cell sites in unit time, and the session usage breakdowns on each reference cell site) that can be obtained from the UE-level log data are also included in $U_d^{n\times t}$. We also measure how many times the UE is handed-over between multiple cell sites in one single data session. If the UE is frequently handed-over between cell sites even when the UE is not traveling at high speed, it may also imply some software or network problems with the UE. Thus, we also use it as one feature channel in $U_d^{n\times t}$.

In addition, based on which cell site the data session is connected with, we use a detailed break-down of session usage features $B_d^{(k^{'}\times n)\times w'}$ for the top $k^{'}$ cell sites that are most frequently accessed by each device d. In our implementation we use $k^{'} = 5$. According to our measurement over a large market in 30 days, the top 5 cell sites totally contribute to 86% of the cellular sessions and 91% of the usage time on average for each customer. Thus, if a user suffers from a network problem, the cell sites who are responsible are most likely among these top 5 cell sites.

For effective troubleshooting, NeTExp also correlates the UE-level profile features with the network status of the top $k^{'}$ reference cell sites. To achieve this goal, NeTExp creates a cell-level profile for the reference cell-sites by using the learned features from the cell-level model, namely, $H_{FC}^{k\times h}$ and H_o^k. To obtain the learned cell-level profile features, NeTExp uses a sliding window that scans the cell site KPI traces and feeds

the feature windows to the cell-level model for feature learning (as shown in Fig. 3.7). For each UE, NeTExp looks back a 1-week historical time window and construct the corresponding feature profiles. The extracted UE-level and cell-level features are concatenated over the *time* dimension for temporal correlation learning. The UE-level model is designed as shown in Fig. 3.7. In the left side of Fig. 3.7, NeTExp applies the pre-trained cell-level model (Fig. 3.5) and use a sliding window to extract the cell-level profile features over the one week history. The stride of the sliding window is 1 hour.

The final decision-making model is a CNN classifier that contains several 1-D temporal CNN layers (Eq.3.2), followed by two fully connected layers with a softmax layer at the end. The learning target is whether the UE's problem is caused by a network-side issue or a device-side issue. Then the UE-level model is trained as a binary classifier using the case-specific manual resolution results from the troubleshooting log data. Since the UE-level model has a much smaller parameter size and only contains a few 1-D CNN layers, it is much easier to train than the cell-level model. Thus, the model can be properly trained using the limited ticket resolution data.

3.5 Evaluation

3.5.1 Datasets

In our experiments, we use nation wide datasets (i.e., network logs and the care contact logs/tickets, see Table 2.1 for the details) collected over an extensive period from a major US cellular service provider. In particular, we use the care contact log data (logs for the customer interaction phase) and the trouble ticket data (logs for the ticket

resolution phase) as the learning ground truth. Note that we focus on the *service issues* reported to the customer care in our study. The rest user inquiries are not considered. After filtering the care contact log data by the problem type, we obtain *hundreds of thousands* of care contact logs and *tens of thousands* of customer trouble tickets that were sent to tier-2 support teams for resolution *. Because care contact logs contain information manually inputted by agents in the format of free text when they engage with customers, some of these information can be ambiguous. We only use the care contact logs that can be verified and all trouble tickets from the ticket resolution phase for evaluation, while all log data is used for the training under the weakly-supervised learning framework.

We use the cell-level and UE-level network log data as the model inputs. Specifically, we use the average number of Radio Resource Control (RRC) connections, and the average utilization ratio of the Control Channel Element (CCE) as the cell-level KPI features. These two KPIs reflect the number of the connected UEs and the traffic loads (as a ratio of capacity) in real-time for the cell sites. In our dataset, the cell-level KPIs are measured in every 5-minute interval. In addition, the UE-level log contains the status of every data session with the UEs. Each log record represents one cellular session, ending with a termination code that indicates why the session is closed. There are 12 distinct codes in our dataset, each of which can be used as a categorical feature that describes the session status. We also extract other statistical features from the UE-level log data for the model design, including cell site usage frequency/duration of

*In this paper, we only show the scales rather than the exact sizes of the datasets to hide sensitive information about the service provider.

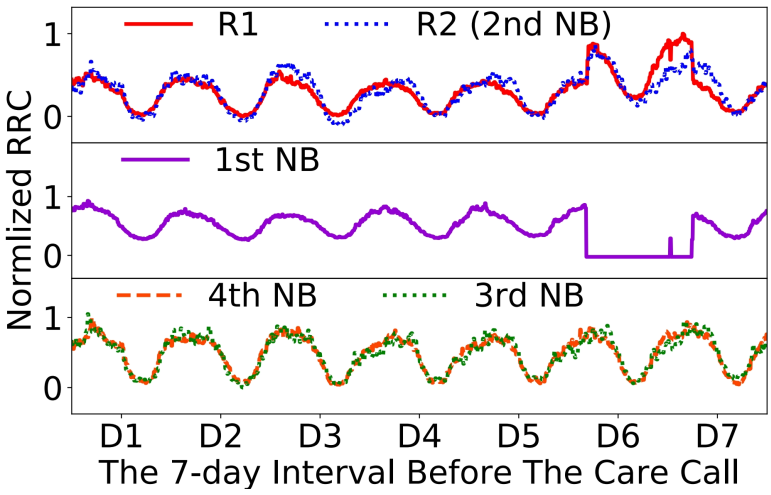


Figure 3.8: Normalized RRC data examples.

each UE (for finding the reference cell sites), the number of commonly served UEs of two neighboring cell sites (for quantifying the adjacency matrix), and the session usage pattern of the UE (for building the UE-level profiles). In total, the magnitude of the UE-level log records for the customers who reported a service issue is *hundreds of millions*.

Note that as a *generic* model for troubleshooting, NeTExp can be easily extended to learn from other cell-level and UE-level features that are available to the cellular providers.

3.5.2 An Example Illustration

In Fig. 3.8, 3.9, and 3.10, we use a real care call example to illustrate the behaviors of NeTExp for handling a specific case and how it can help the customer service. In this example, a customer contacted the customer care and complained that the service performance is degraded with his/her phone. The root cause was not identified

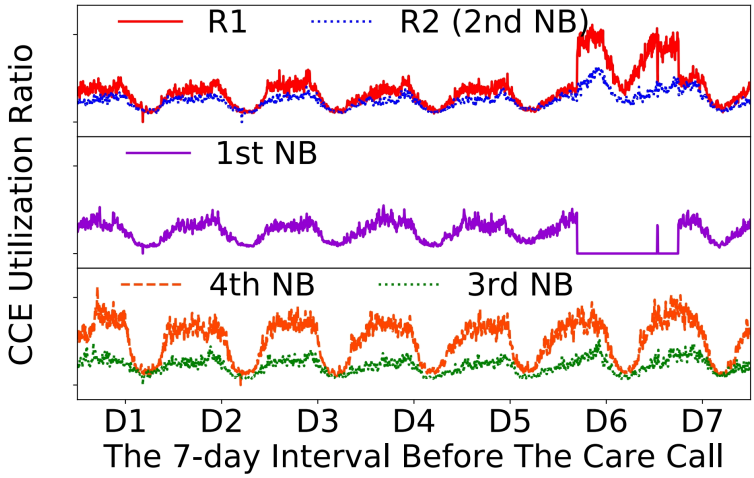


Figure 3.9: CCE Utilization Ratio data examples.

after a long period interaction between the customer and the agent. Then an offline troubleshooting ticket was submitted.

We then use NeTExp to analyze the root cause of this case. In Fig. 3.8 and Fig. 3.9, we illustrate the RRC KPI and CCE Utilization KPI time series of the relevant cell sites for the past 7 days before the care contact. The raw RRC data series are normalized to $[0, 1]$ using min-max normalization for visualization. The "D1" to "D7" on the x axis represent the first to the seventh day of the historical window used for issue inspection. In the top panels of Fig. 3.8 and Fig. 3.9, "R1" and "R2" represents the top two cell sites that are most frequently accessed by the user in the past week. From these panels along, we could observe a noticeable increase of connected users (the RRC KPI) and traffic load (the CCE KPI) on R1 during day 6 and 7 (i.e., around 0-48 hours before the care contact) compared with the earlier days. There are similar but less noticeable changes on R2. However, those abnormal patterns themselves cannot necessarily indicate an anomaly on this cell site, as it could also be observed in many

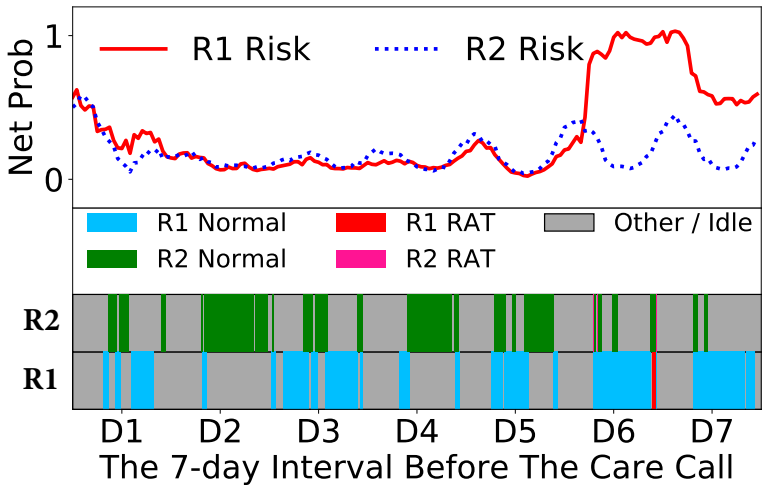


Figure 3.10: Cell-level model outputs (top) and UE-level profile examples (bottom).

other scenarios such as gathering of people (e.g. concerts, sport matches) in the cells.

Next, the cell-level model also investigates neighboring cell sites. In Fig. 3.8 and Fig. 3.9, we also show the KPIs of the top 4 cell sites in R1's neighborhood, i.e., "1st NB" to "4th NB", which are ranked by the graph weights (proximity) to R1 (R2 is the "2nd NB"). We can find cell site "1st NB", the closest (by graph weights) cell site, had an long outage during day 6 and 7, while the other further neighbors looks normal. Clearly the KPI patterns on R1 were affected by the outages in the neighborhood, although no outage could be identified on R1. Taking those KPI patterns observed in the surroundings as the input, the cell-level model learns the risk of network issues for the users associated with the corresponding cell sites. The learned network issue probabilities for R1 and R2 are shown in the top panel of Fig. 3.10. By using the graph-based model, the model infers that the users on R1 had an increased risk of network problem in day 6 to day 7. However, it is still not sufficient to conclude the issue

experienced by this customer is indeed impacted by the outage in the neighborhood. The symptoms on the device side should also be investigated.

To understand what really happened with the customer, we also show the session states (i.e., the UE-level profiles) in the past 7 days for this user's device in the bottom panel of Fig. 3.10. In this panel, the rectangles represent the intervals of the cellular sessions of the UE that were carried by each of the two reference cell sites R1 and R2. Specifically, "Other / Idle" means the device was carried by other cell sites or the device was idle, "R1/R2 Normal" means the sessions with R1/R2 were closed normally, "R1/R2 RAT" means the radio access technology (RAT) was changed. The simultaneous session occupations with R1 and R2 (e.g., the overlapped session intervals in day 6) represent that the device was handed off from one to another cell site. Note that the session states themselves cannot directly indicate whether the performance is degraded. For example, the "Normal" state only shows the session is terminated following a normal procedure, whereas the performance of the session is unknown. However, by correlating the top and bottom panel of Fig. 3.10, we can discover that: (1) After the occurrence of the outages (day 6 and 7), the total session length with R2 was significantly reduced and the device was mostly carried by R1. The changes might be due to the maintenance and network setting changes for resolving the nearby outages. (2) R1 was significantly impacted by the outages in the neighborhood according to the cell level model predictions. From the raw KPI data in Fig. 3.8 and Fig. 3.9, we can now infer similar as this studied user case, many other devices nearby were also moved to R1 from their original carriers, and thus cause the significant increase of RRC connections and CCE utilization ratio of R1. (3) Along with the carrier changes, the RAT was

degraded due to the congestion and resource limitations on R1.

Thus, the strong temporal correlation between the cell-level states and UE-level states indicates the root cause of the reported issue was indeed a network problem, which could also be learned by NeTExp using the proposed feature modeling methods. In this example, the root network failures were on the neighboring cell sites rather than the major cell sites that served the user. However, the impact of outages propagated, which affected a larger population in neighboring cells. In fact, the propagation of the network failure impact is usually triggered by the fault tolerance mechanisms in current cellular networks. By handing over the customers from the cell site with failures to its neighboring towers, it can dramatically reduce the customer impact of eNodeB/gNodeB failures, although it may cause some congestion on the neighboring cell sites. Due to the complexity of the problem, the root issue was hard to be identified over the live care call. Note that there are a variety of detailed cell-level and UE-level symptoms when customers experience a network issue. The case illustrated above is just one common type of issue. The model's task is learning all those issue patterns through training.

3.5.3 Evaluation for The Cell-level Model

For the cell-level model, we split the cell-level network log data into two parts: we used the first consecutive 20 days of data for training and the rest consecutive 10 days of data for validation. The mean-absolute-error (MAE) and the model time costs are used as the metrics. Then we analyze the model from both the spatial and temporal dimensions as follows.

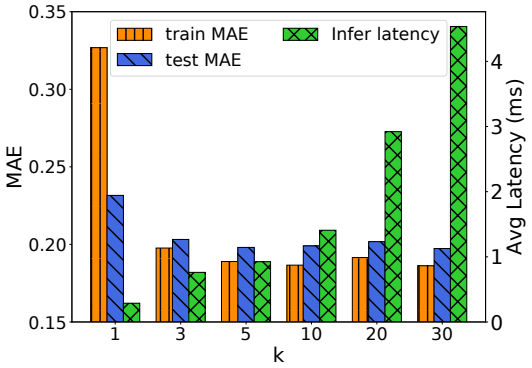


Figure 3.11: The MAE and cost with different ks.

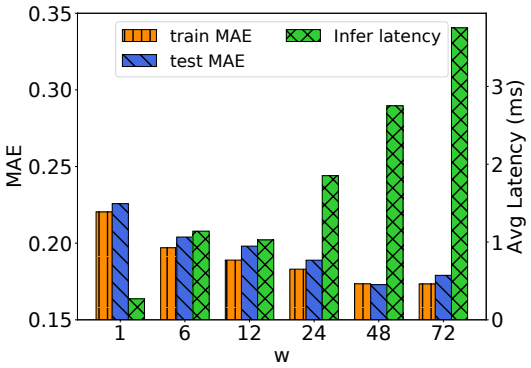


Figure 3.12: The MAE and cost with different ws.

Analysis on the spatial dimension

In the spatial dimension, we want to find out how many neighboring cell sites $(k-1)$ should be considered when predicting the likelihood of network issues in the cells centered by a target cell site. We fix the historical window size w of the KPI data as 12 hours. Then we train the cell-level model with different graph sizes, ranging from $k = 1$ to $k = 30$, where k is the total number of cell sites considered for prediction in the area. For comparison, each model is trained with 10,000 batches and each batch contains 256 samples of the data window samples. The model is trained and executed on a 64-core CPU cluster. The whole training process on the CPU server takes around 2-10 hours for different ks.

When training is finished, we use the model to generate a time-series trace of the predicted likelihood for each cell site using the sliding window method introduced in Section 3.4. The window sliding stride is 5-minute.

We randomly select 10,000 cell sites, and measure the mean absolute error

(MAE) and the average inference delay (the inference cost for a single cell site at a single timestamp) of the cell-level models. The comparison of different ks are shown in Fig.3.11. We find the model yields high training errors when $k = 1$, showing it fails to learn the target well when only the target cell site is considered. For the $k > 1$ scenarios, we find that our graph-based models have much smaller training and testing errors. The results suggest that the interaction of neighboring cell sites is an important feature when analyzing network issues on the cell level.

In addition, we also notice from Fig. 3.11 that the model does not improve if more than 5 neighboring cell sites are included in consideration, while the inference latency grows linearly with k. The observation suggests that the transition of the abnormal network states indeed exists but only among the nearest neighboring cell sites that have the largest jointly served population, as the fault tolerance mechanism of cellular networks can dilute the impact of a single network fault on the distant cells. Thus, we use $k = 5$ as a default cell-level model setting for our system.

Analysis on the temporal dimension

We show how the historical window size w impacts the cell-level model. In our evaluation, we set $k = 5$ and changes the window size w from 1 hour to 96 hours. We use the similar training and evaluation strategy as discussed above. The comparison results are shown in Fig. 3.12.

From Fig. 3.12, we find that the model accuracy improves when longer time windows are used. The training and validation errors significantly reduce when $w = 48$ hours. As w grows larger than 48 hours, the errors only reduce marginally. Hence, the

historical data beyond 48 hours is less important in inferring network issues. Fig. 3.12 also shows the cost of the model grows significantly with the window size w. Hence, we choose $w = 48$ hours instead of a larger w as an optimal setting for the model.

3.5.4 Evaluation of The UE-level Model

Overall classification performance

We evaluate the root cause diagnosing performance of NeTExp using the real historical care contact data introduced above. Specifically, the system is evaluated by 5-fold cross validation and is compared with 5 other baseline diagnosis models. The training of NeTExp for each fold takes around 20 minutes. A brief introduction of those models is as follows:

ICCA (auto) [52]: ICCA is a state-of-the-art diagnosis system for distinguishing between user and network faults in real time. It extracts the most discriminative *UE-level* event sequential patterns using PrefixSpan [27] and information gain by creating a model-based search tree [20] on historical data. A Gradient Boosting Decision Tree (GBDT) [21] model is then applied for classification. In addition, ICCA [52] also uses manual expert features which are not available to us. Thus, we only implement and compare the *automatic* feature extraction and learning modules of ICCA based on network log data.

Fisher Score + KNN: We compute the fisher scores [24] of the Cell-level and UE-level features obtained by NeTExp for the 7-day historical window with respect to the root cause categories and select the top n discriminative features for classification. We search the optimal n in range $[5, 1000]$ and use $n = 150$, which gives optimal

performance through validations. Then a k-nearest neighbor (KNN) classifier is used for case-specific diagnosis using the selected top n features.

Sparse Representation-based Classification (SRC) [76]: SRC creates an annotated feature library with cell-level and UE-level feature profiles of the training instances. At the online inference stage, for each customer's case, the model reconstructs input feature profile of the customer using a sparse liner combination of the library profiles by solving an optimization problem with L1-normalization. Then the decision is made by selecting the root cause category that minimizes the reconstruction error using the training samples from that category only and the learned sample coefficients.

CNN with only Cell-level or UE-level features: We apply the same CNN classification model introduced in Section 3.4 while using the extracted features from only cell-level observations or UE-level observations, in order to illustrate the importance of the extracted features from both sides for troubleshooting.

The overall 5-fold validation results (Accuracy, F1-score, RoC-AUC) are shown in Fig. 3.13. We find NeTExp outperforms the other baseline methods for different classification metrics. In addition, as a simpler and distance-based model, Fisher Score + KNN with the features learned by NeTExp also yields good classification results. This shows the feature engineering methods in NeTExp provide discriminative features for root cause classification. Meanwhile, we find the UE-level sequential patterns learned by ICCA are under-representative for describing the user's historical events that correlate with the issue root causes. For example, the sequential pattern feature ignores the timestamps of cell-level and UE-level abnormal events and the temporal correlation among them.

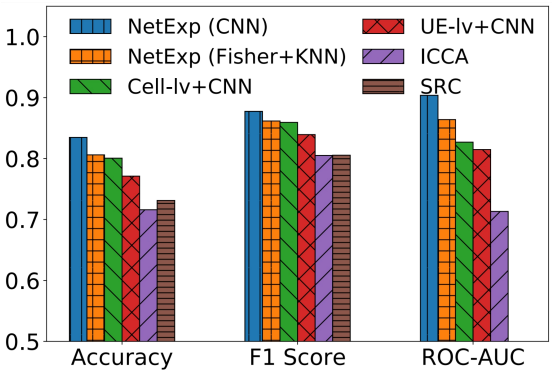


Figure 3.13: Root cause classification.

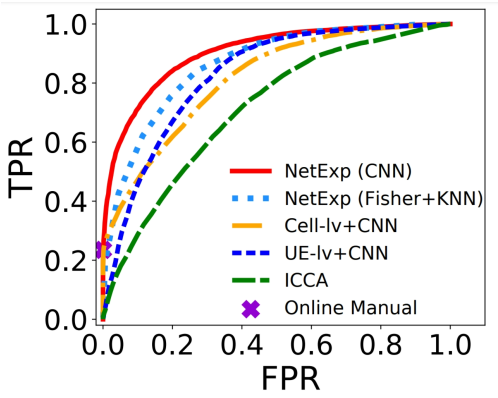


Figure 3.14: Overall RoC curve.

In Fig. 3.14, we present the ROC curve of the compared methods, where network-side issue is denoted as "positive" and device-side issue is denoted as "negative". The "cross" mark in the figure shows the ratio of network problems that could be identified during the online troubleshooting phase. Since many non-outage network issues are difficult to be timely recognized, the recall of the manual network identification is low (less than 25%). For the rest 75% cases, the issues have to be forwarded to offline inspection. From the results, we can clearly find NeTExp significantly improves the accuracy of the network identification without introducing a large fraction of "false positives". For example, NeTExp can improve the recall to 75% with only around 10% "false positives" included. Note that the network issues identified in our ground truth data are only a subset of the all real network-issue-related cases. This is due to the limitations of obtaining complete ground truth via existing troubleshooting procedures in practice. Thus, those "false positives" may not be real negatives (i.e., non-network-related issues). In Section 3.6, we will show a case study for this type of

tickets.

In Fig. 3.15 and 3.16 we show the breakdowns of the performance for different groups of troubleshooting samples. Specifically, we divide the dataset into two subsets based on whether the ticket (if it is network-related) was eventually resolved in the customer interaction stage (the "easier" dataset) or in the ticket resolution stage (the "harder" dataset). Hence, the majority of the network issues included by the first dataset are the network problems that caused severe service degradation (such as outages), which are already detected proactively by the service carrier before the care contacts; while the network-related cases in the second dataset are much harder to be recognized and require in-depth inspection, which usually are related to intermittent or chronic experience symptoms. The results shows that the features learned from only the cell-level data can work well for identifying the "easier" network issue cases, while they fail to work well for the "harder" cases. On the other hand, the information included by the UE-level features can better describe the symptoms on individual user device regardless types of the root causes. But the symptom of individual service experience itself is insufficient for locating the problem as different problems may produce similar symptoms on the UE-side. Therefore, NeTExp , which correlates both the cell-level and UE-level observations chronologically, provides best network issue detection performance.

In Fig. 3.17 and Fig. 3.18, we shown the accuracy and the ROC-AUC scores of the compared models for different categories of service problems. Cellular data problems (30%) and voice call problems (37%) are the most common reasons that motivate the customer to call in the care. The "other" category represents all other less com-

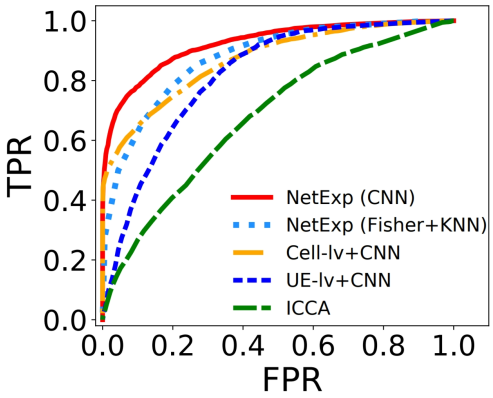


Figure 3.15: RoC - easy cases.

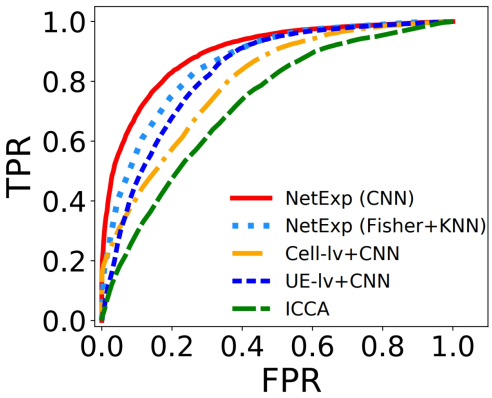


Figure 3.16: RoC - hard cases.

mon types, many of which are experience-specific. The results show that NeTExp is a comprehensive model that always outperforms the baseline methods for different issue categories.

3.5.5 Summary of The Evaluation Results

Through evaluation, we find the cell-level model achieves the best performance when learning the likelihood of network issues by jointly incorporating the KPI features of the neighboring cell sites. In addition, the UE-level model of NeTExp can more accurately identify the root causes of the individual tickets for different service-issue types and different network-issue root causes by correlating the cell-level and UE-level network profiles.

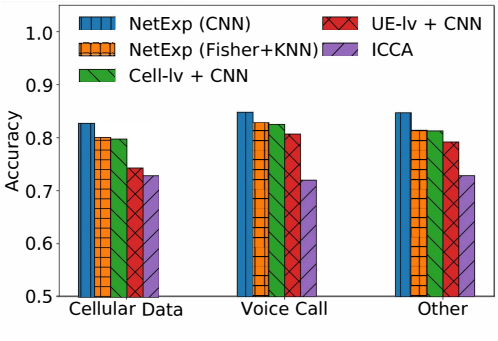


Figure 3.17: Accuracy for different issue types.

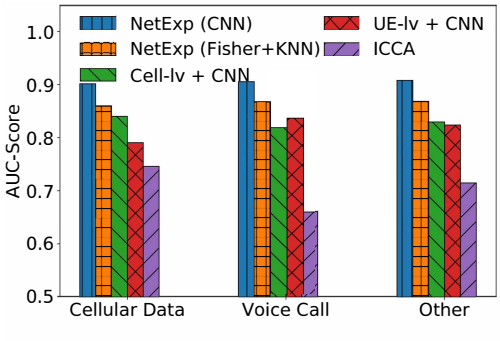


Figure 3.18: ROC-AUC for different issue types.

3.6 Case Study

We perform a case study to show how NeTExp can help troubleshoot service degradation issues in practice for a cellular provider. Specifically, we apply our system to analyze an external incident that caused network issues in the U.S. in 2020. During the event period, some areas that were directly affected by the incident experienced network outages, while other areas were indirectly affected due to network changes and maintenance works after the incident.

Fig. 3.19 shows the distribution of the network-related care contacts in a heatmap and the distribution of the cell sites that are considered to experience network problems (the small blue dots) at different levels on Google Map. Specifically, the intensity of the heatmap reflects the aggregated number of care contacts that are considered to be network-related in the unit area based on the **ground truth** troubleshooting log data.

In Fig. 3.20, we show the heatmap and the impacted cell-sites on Google Map

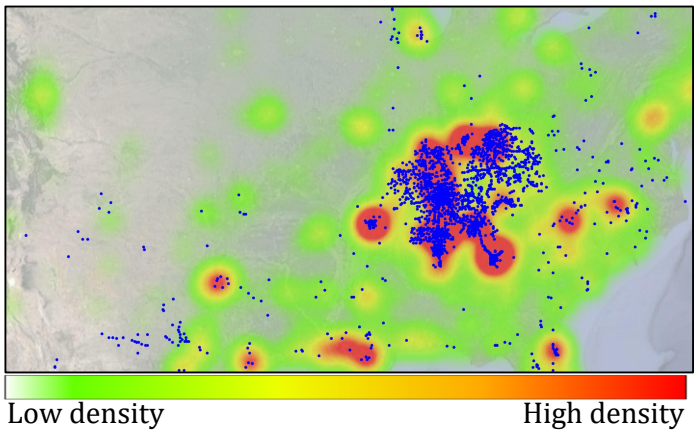


Figure 3.19: Manual: heat map and issue positions.

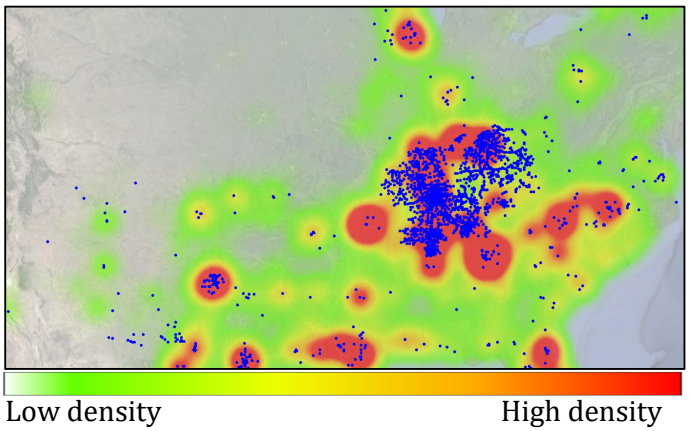


Figure 3.20: NeTExp : heat map and issue positions.

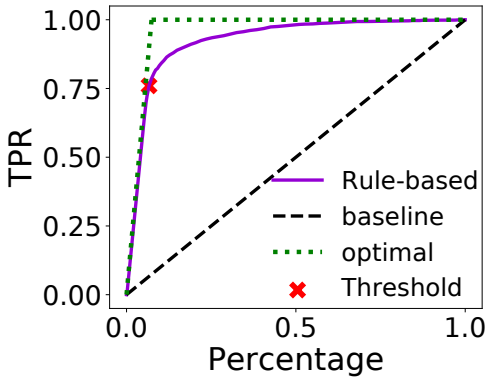


Figure 3.21: Recall of cell-level issue detection.

based on the inference results of the cell-level model and UE-level model of **NeTExp** . Specifically, we use NeTExp to troubleshoot all reported service issues of the market in the same time period and create the heatmap based on the model decisions. And we also mark the positions (with the blue dots) of the cell sites that are considered to experience a network issue during those days based on the predictions of the cell-level model. Namely, for each cell-site c, we consider it as a impacted cell-site if $\max_t H_o^c(t) > \hat{h}$.

The parameter $\hat{h}$ used in Fig. 3.20 is determined by our analysis on the recall of the detection, as shown in Fig. 3.21. In Fig. 3.21, by tuning $\hat{h}$, we show the recall of the cell-level issue detection as a function of the percentage of the detected cell sites over the whole market (i.e., $|C_d|/|C|$, where $|C_d|$ is the number of the cell sites that are considered to have network problems, and $|C|$ is the number of all cell sites). The optimal curve is the green dotted line, which corresponds to the "optimal" detection result based on the ground truth positions shown in Fig. 3.19. We then select the $\hat{h}$ at the "Threshold" point in Fig. 3.21 as our model policy threshold for the detection. At the "Threshold" point, the cell-model can achieve around 75% recall with near to

0 false positive rate. By comparing the results in Fig.3.20 and the groundtruth data in Fig.3.19, we find the model automatically learns the impact of the disaster event at different areas with different root causes, and infers how the impact propagated in a wide area crossing multiple states.

From the manual and the model troubleshooting results (Fig.3.20 and Fig.3.19), we find the model tends to recognize more tickets as network-issue related. The extra tickets that are recognized as network problems become the "false positive" samples in the classification results. However, we find some "false positives" (FP) are not false but due to incomplete/incorrect groundtruth data.

To better understand the decisions of NeTExp , we show three classification examples from different categories (true positive (TP), FP, and false negative (FN)). In the top charts of Figs. 3.22, 3.23, and 3.24, we visualize a selected representative UE-level feature for the three cases, as well as the 7-day time-series of the learned cell-level model feature $H_o^c(t)$ of their top-1 reference cell site. The bottom charts show the number of care contacts (aggregated for every 3 hours) for the top-5 reference cell sites in the area. The "Disaster ts" and the "CC ts" represent the time when the external incident occurred and the time when the customer contacted care.

For customer A, the model infers a network issue, which is a TP case according to the ground truth. The UE-level KPI shows the device is in abnormal states (the device cannot maintain a stable session) for a period right before the care contact time. The temporal consistency of the cell-level and UE-level features clearly demonstrates how the model makes the correct decision. However, from the troubleshooting logs, we find the customer's issue was not immediately resolved during the customer interaction

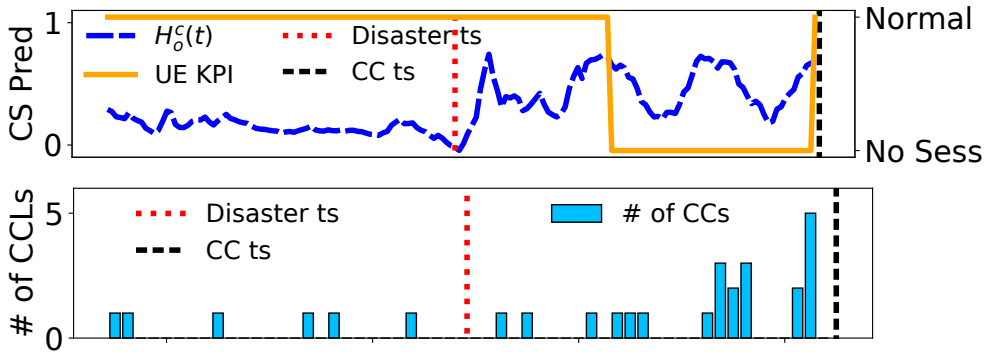


Figure 3.22: Example case A.

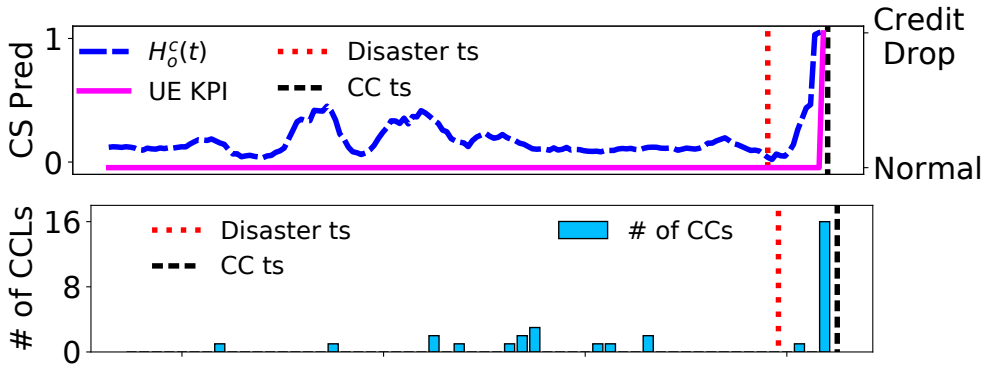


Figure 3.23: Example case B.

phase and was eventually resolved by the ticket resolution with a longer delay.

Customer B's case shows a "false positive" example. From the trends of the cell-level model predictions and the growth of the care contacts in the local cells, and their close correlation with the UE-level "Credit Drop" failure patterns, we believe the root cause is indeed a network problem. However, we find the customer's problem was not resolved during the customer interaction phase, neither was there an offline troubleshooting ticket generated. Thus, this case study shows a mislabeling case in the ground truth data. In fact, we find this case is not unique in our dataset, as the raw

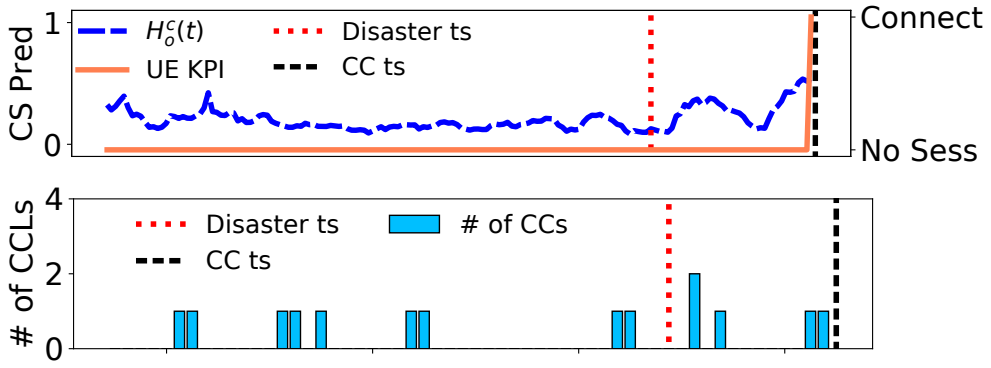


Figure 3.24: Example case C.

care contact log data could be incomplete and noisy due to the difficulty to troubleshoot some cases in the online phase. Still, NeTExp learns to make correct decisions despite the noise in the training/validation data.

Customer C's case shows a "false negative" decision. The model fails to detect a network issue, although we can manually observe a subtle growing trend of the learned cell-level feature and the number of daily care contacts. We infer the model fail to make a correct prediction because of the inconsistency between the UE-level features and the cell-level features. Specifically, the UE was not able to build data sessions for many days until the past hour before the customer contacted care, which does not match the expected timeline of the disaster impact. After inspecting the care contact logs, we learn that the customer just activated a new device, which explains why the UE-level network log shows no effective data session in the earlier days until the hour before the care contact. However, NeTExp does not learn this context from the data, and thus makes an incorrect decision.

From the third case, we find there could be some corner cases when creating

the UE-level profiles for the customers. Those corner cases are not fully considered by NeTExp , since some real-world context is missing from the data source itself. Due to this limitation, at this stage, we think a reasonable deployment for NeTExp is using it as a reference knowledge base that provides the insights from the data-driven perspective for the care agents in the customer interaction stage, rather than a full replacement of human care agents.

3.7 Discussion

3.7.1 Model Updating for Unseen Scenarios

NeTExp models need to be updated as cellular network configurations and application scenarios change. We summarize the possible network changes and discuss how NeTExp need to be updated or retrained.

Permanent changes of traffic pattern. The traffic pattern observed for a local cell may change permanently because of the following reasons: the upgrading of network infrastructures, popularity of the trending network applications, or the changes of user behaviors affected by major events such as COVID-19 pandemic. Thus, it is necessary to keep monitoring the local network KPI patterns of recent days and updating the rolling average of the KPIs at each timestamp of the day for effective data normalization as introduced in Section 3.3.3.

Graph changes. Since the cell-level model is trained as a generalized feature extractor that can fit for different sub-graphs rather a specialized model for a fixed graph, the model weights does not need to retrained for minor graph changes (such as

adding or removing a cell site node), although the input graph adjacent matrix should be updated for inference. However, if there are major changes regarding the global graph feature distribution, for example, the cellular provide decides to shut down all 3G towers and activate more 5G towers in nationwide, both the cell-level and UE-level models should be retrained using the new graphs and network measurement data.

New KPI counters. Our current model is trained using the cell-level KPI counters and the UE-level state records that are available to us, while new other counters or performance metrics might be available in the future. Those new measurements can be easily added as the additional feature channels for learning using the proposed model design, whereas the model need to be retrained. The DNN models in NeTExp can adaptively extract good features from the available data if properly trained.

3.7.2 Fine-grained Root Cause Identification

NeTExp solves a practical problem raised by a real cellular provider: how to automatically identify the root cause of a service issue experienced by a customer. In this work, the root cause classification is conducted at a coarse level (i.e., whether is issue is from network side or device side). This binary classification result which can fit into the state-of-the-practice troubleshooting workflow (as illustrated in section 2.1.2) is a fundamental need for the cellular providers to improve the troubleshooting efficiency. On the other hand, as a highly representative DNN model, NeTExp can learn the high level feature mapping from the raw network observations and generate insightful feature profiles, which provides the potential for fine-grained root cause identification, for example, recognizing whether the issue is on a particular cellular tower or in the

core network. Unfortunately, due to the shortage of fine-grained troubleshooting ground truth data. it is rather challenging to train and validate the ML model.

3.7.3 The "False Positive" Instances

According to our evaluation results, the historical troubleshooting tickets that were not labeled as "network related" but are identified as "network related" by NeTExp become the "false positive" instances using the automatic troubleshooting models. In the next chapter, we will discuss those "false positive" instances into more details.

3.8 Conclusion

We present the NeTExp system prototype for automatic and reactive service issue troubleshooting in cellular networks. NeTExp is a generic and comprehensive data-driven approach that considers both the cell site network conditions and UE network logs, as well as their correlations for troubleshooting. The system can be easily extended to incorporate different network KPI data depending on the accessible supporting cellular infrastructures, software, and databases. Our evaluation with an extensive period of real-world data from a major US cellular provider and a case study based on a real event demonstrates the effectiveness of the system.

Chapter 4

NeTExp with PU-learning and Trial System Implementation

From the evaluation results of the NeTExp prototype, we find some tickets which were not labeled as "network-related" are recognized as "network-related" by the machine learning models. We call those tickets "false positive" instances. The case studies show that not all those "false positive" samples are genuinely false. In fact, with the current manual troubleshooting workflow, not all network-related cases can be successfully recognized or correctly labeled in the ticket data.

This chapter will investigate how to understand the "false positive" samples and improve NeTExp using the limited ground truth data properly. We also provide an enhanced NeTExp system design that fully considers practical concerns in the real world.

4.1 NeTExp with PU-learning

4.1.1 Problem Description.

The supervision for training NeTExp mainly comes from the online and offline manual troubleshooting tickets in the history. However, the troubleshooting ticket ground truth could be noisy due to the following possible reasons:

First, the resolution provided by the care agents cannot always be verified. The whole troubleshooting process requires the intensive engagement of the customers. When the care agent proposes a possible solution in a remote troubleshooting scenario, the customer needs to validate whether the solution can solve the problem. However, the customer may not always be able to effectively validate the resolution or provide a confident validation result to the care agent. For example, if the customer reports the network speed is slow, the problem may be mitigated after the care agent performs some configuration changes, whereas the customer may experience the issue again sometime later when the network becomes congested. Unfortunately, many customers may not choose to follow up with the case when the issue occurs again. Therefore, in the troubleshooting ticket data, the care agents only provide what resolution is used, while whether the resolution can completely solve the issue is unknown.

Second, the troubleshooting process may not be finished due to some unexpected reasons. An online troubleshooting process may take tens of minutes. However, not all customers can stay on the line until the problem is solved. Sometimes the care call line is accidentally dropped. This scenario is particularly common if the caller is experiencing some service issue. In those scenarios, the care agent cannot collect sufficient

information to troubleshoot the problem effectively.

Third, not all troubleshooting results in the tickets can be trusted. Due to the limited information that is collected in the online phase, the care agents may choose an incorrect direction to troubleshoot the issue. For example, as the case shown in section 3.5.2, the care agents may not be aware of the propagation of some network issues and thus ignore the potential network problem.

Since the model-based troubleshooting framework is trained and evaluated based on the noisy ground truth troubleshooting data, we post a new concern: how responsible the model is if it is used in practice. To solve this concern, we propose two major questions: (1) how should we train the model with partially trusted data? (2) how should we verify the effectiveness of the model given the limited ground truth data? In the following sections, we will introduce a weakly-supervised learning framework to solve the above problems.

4.1.2 Learning from Positive and Unlabeled Examples.

Manual troubleshooting tickets.

To figure out how responsible the ML model could be in practical scenarios, we first need to understand how responsible the training dataset is. The labels we used for the training dataset are from the manual troubleshooting tickets that are generated in the online customer interaction phases and the offline ticket resolution phases (as shown in Fig. 2.2). Since the offline ticket resolution phase is carried out by the experienced tier-2 troubleshooting experts who can access the network log data, perform data analysis, probe the possible treatments, and correlate the case with other reported

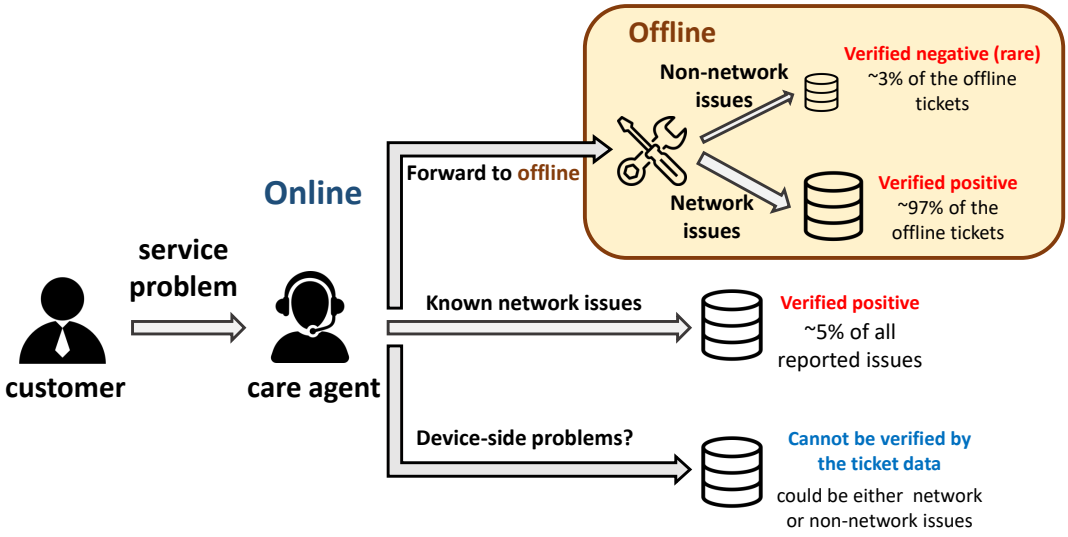


Figure 4.1: A summary of the different types of tickets in the ticket datasets.

issues nearby in the offline, we think this troubleshooting phase can provide accurate issue identification labels. On the other hand, due to the time constraints and the information constraints in the live phone call, we do not think all resolution results in the online care log data are accurate. Specifically, we can only trust the care log tickets where the agents specify that a known network problem (e.g., network outages, scheduled maintenance) can be identified. For those issues, the care agents can find the associated cell site with evident network failure using the current outage detection software. However, for the rest tickets, since the detailed troubleshooting process and customer's responses are not logged (for user privacy reasons), we cannot verify the correctness of the manual troubleshooting results based on the data.

Hence, a summary of the troubleshooting log data types based on the confidence of the proposed resolution in the tickets is shown in Fig. 4.1. From the figure, we can find that only the offline troubleshooting tickets and the online network-issue-related

tickets can be trusted. In addition, based on our observation of the offline troubleshooting tickets that were collected for one month, we find that over 97% of the offline tickets were recognized as network issue related. This observation further demonstrates that the current manual troubleshooting framework can yield high **precision** for network issue detection, while the **recall** is unknown. Namely, based on the ticket data, we can think of the detected network issue cases as the correctly labeled tickets, while we cannot know how many network issue cases are not detected in the manual troubleshooting process.

Learning from positive and unlabeled data.

According to the nature of the manual troubleshooting ticket data, we can formulate the automatic network issue detection task as follows.

Let P be the universe of tickets where the root cause is a network side problem, and N be the universe of tickets where the root cause is not a network problem. After being classified by the manual troubleshooting process, P can be divided into two parts: $P_{labeled}$, which is the set of network issue related cases whose root cause can be identified through manual troubleshooting, and $P_{unlabeled}$, which is the remaining set, i.e., the network issue is not successfully found. In addition, we use N to represent the non-network-related tickets. The goal is training a ML model f that can distinguish between P and N. However, the training data can only give $P_{labeled}$. We call the rest of the unlabeled tickets in the training dataset U, where $U = P_{unlabeled} + N$.

The above learning problem is a PU-learning task, i.e., learning from positive and unlabeled data [42]. Unlike binary classification problems, only partial supervision

can be obtained from the ground truth data in PU-learning. Thus, to solve the PU-learning problem, we usually need to train a binary classifier using weakly-supervised learning strategies, which is more challenging than supervised learning. There are three major types of methods for solving the PU-learning problem [7]: two-step techniques, biased learning, and class prior incorporation. Different techniques hold different assumptions regarding the training data distribution.

Two-step techniques [22, 45] assumes that all positive data samples (P) are similar to the labeled positive samples ($P_{labeled}$), while negative samples (N) hold a very different distribution. Thus, the key idea of two-step learning is first to find some trusted negative samples from the unlabeled data (U) based on the dissimilarity to the positive samples, and then to use semi-supervised learning methods [92] to train the classifier using the labeled positive data and trusted negative data.

Biased learning is based on the SCAR (selected completely at random) assumption, namely, the labeled positive samples are selected completely at random for the positive data sample universe, i.e., $P_{labeled}$ and $P_{unlabeled}$ have identical distribution. In biased learning, all the unlabeled data U is considered as the negative class, while $P_{unlabeled}$ is considered as the noise in U. Then a classifier is trained based on this assumption, while the different weights are given for different training penalty sets [14, 60, 88]. However, the SCAR assumption may not always be realistic [35]. For example, in the troubleshooting ticket data, the positive class samples mainly come from the network outage cases that could be easily identified during the online phase and the complicated network issue cases that are resolved through offline troubleshooting. The experience and knowledge of the online care agents may affect which cases should be

forwarded to ticket resolution in practice.

Class prior incorporation [18] assumes the positive class prior, i.e., $|P|/(|P| + |N|)$, is known. Based on the class prior, a probabilistic classifier is trained using $P_{labeled}$ and the positive class and U as the negative class. Then the model is adjusted based on the output sample class probabilities such that the learned positive class frequency is similar to the class prior. The class prior can be decided by experiences or validation.

In our system design, we use a combination of the above PU-learning approaches to solve our problem. We also incorporate side-channel knowledge from historical data to help train the model in a weakly-supervised way.

4.1.3 Enhanced NeTExp Model Design with PU-learning.

A teacher-student model framework.

In actual application scenarios, NeTExp should provide root cause classification result during the customer interaction phase, which means the knowledge about the network performance and UE-level symptoms should be obtained from the historical data before the users contact the customer care. However, we also notice that the "future" network log data that is generated after the care contact can also include rich information to support the classification. In particular, by comparing the network performance or states before and after the care contact on both the cell-level and UE-level, we could infer (1) when the network problem (if any) begins and ends, (2) whether the UE-side performance is improved or the symptom disappears after the care agents handle the issue, and (3) whether the correlation between the UE-level performance states and the cell-level network states has been changed after manual troubleshooting.

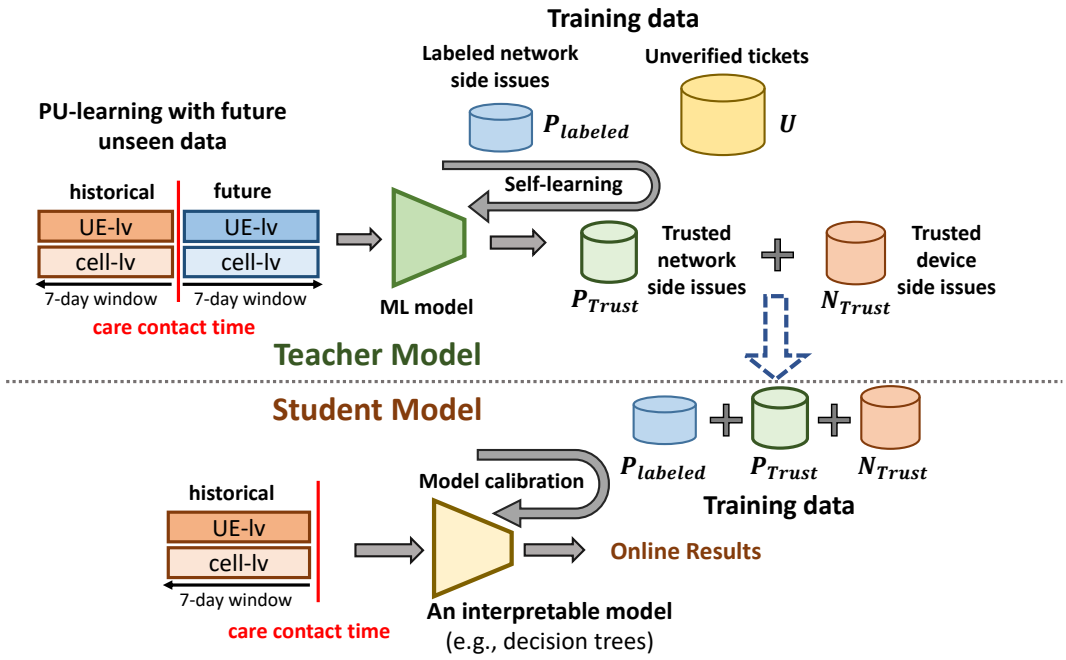


Figure 4.2: The teacher-student PU-learning framework.

Thus, the comparison between the data observations before and after the care contact can provide an idea of whether the troubleshooting actions executed by the care agents effectively solve the issue. If the UE-level performance gets improved immediately after the care contact, the issue is more likely a device-side or configuration problem. On the contrary, if the UE-level performance cannot get improved immediately after the care contact but is correlated with the changes of cell-level network status, the issue is more likely a network side problem. In summary, although we cannot verify the effectiveness of most troubleshooting actions based on the ticket data, we could utilize the historical and future network log data to infer the effectiveness. The "future" network log data that cannot be obtained during the real-time troubleshooting phases can be used as side-channel supervision for training the PU-learning models.

Based on the above insight, we propose a teacher-student model training framework to solve the PU-learning problem. The framework overview is shown in Fig. 4.2. The overall teacher-student model design follows the two-step PU-learning framework, where the teacher model is responsible for augmenting the training data set using advanced self-training strategies, while the student model is used as a classifier that can answer user's queries in real-time.

Specifically, the teacher model is a PU-learning classifier that is trained based on historical ticket data in offline. The purpose of the teacher model is to pre-classify the tickets and generate a trusted and automatically-labeled dataset that includes both the positive and negative instances. This dataset is then used as the training dataset for training the student model, which is used for real-time troubleshooting in practice. To obtain such a dataset, we use two primary techniques in the teacher model design: (1) advanced self-training strategies and (2) feature profiling with side-channel supervision.

The student model is a binary classifier. It can also be considered as the second classifier that is trained using the trusted data samples generated by the teacher model in the two-step PU-learning framework. Since the student model is used for real-scenario troubleshooting, it can only use the historical network data for learning. In addition, to be responsible for the real customers, the decision-making process of the student model should be interpretable by human care agents. Thus, the human care agents can understand why the model thinks the problem is a network-side issue or a device-side issue, explain the root causes to the customers, and perform further investigations regarding the key observations if necessary.

Teacher model feature profiles.

We use a similar CNN model design as illustrated in Fig. 3.4 for the teacher model. However, we redesign the feature profiling method such that the model can learn with the side-channel supervision from the future data. Specifically, the new feature profile includes the cell-level network state features that are learned by the cell-level model and the raw KPIs of the top 5 reference cell sites. Let t_{care} represent the timestamp of the care contact and w be the length of the historical or future feature window size, the cell-level feature covers the duration between t_{start} to t_{end}, where $t_{start} = t_{care} - w$ and $t_{end} = t_{care} + w$. In addition, the feature profile also includes the UE-level states from the UE-level network log data for the same period. Each UE-level state is a feature vector of length $t_{end} - t_{start}$, and each entry of the vector represents whether the UE is in a session at the corresponding timestamp with a particular termination code. The cell-level and UE-level feature profiles are concatenated along the time channel so that the temporal correlation of the two-side features are preserved. Using this feature profiling method, the model can learn to automatically compare the data before and after the care contact and use this knowledge as additional potential high-level features through training.

In addition, we also augment the CNN feature map before the last fully-connected layer with a collection of manual features based on the raw KPI data. The manual features include the statistics and high-level observations from both the cell-level data and the UE-level data.

In the cell-level, for every KPI, we think the KPI values on each cell site as a

time-series signal, and compute the following features based the KPI data for each each site:

- The statistical features, such as average, standard deviation, maximum, and minimal values.
- The signal shape features, such as shape factor, impulse factor, crest factor [4].
- The signal-to-noise ratio (SNR).

Each of the above features is computed for the overall 7-day historical and future time windows before and after the care contact time, and every 24-hour interval in the two large time windows. Meanwhile, in the UE-level, we measure the occupation pattern of the each type of the cellular sessions (classified based on the session termination codes) and compute the following features:

- The occupation ratio, i.e., the total time while the UE is with the corresponding session divided by the window size.
- The longest occupation time, i.e., the longest period of time while the UE is with the corresponding session in the time window.
- The number of intervals in which the user is not with the corresponding session.

In addition, we also measure the number of handoffs in the sessions and compute the same time-series features for the handoff pattern. Similar to the cell-level manual features, the UE-level features are also computed for the overall 7-day windows and every 24-hour interval.

Those manual features are found closely related to the network side anomalies and device-side symptoms and can be used to explain which key observations are used for making a decision in a human-readable way. For the teacher model, we extract the manual features based on both the historical and future raw features, in order to profiling the differences of the KPI statistics in the two contrasting time windows.

Self-training for the teacher model.

We use a self-paced training strategy [12] to train the teacher model in a weakly-supervised manner. Specifically, given the labeled positive tickets $P_{labeled}$ and the unlabeled tickets U, the goal of the teacher model learning is to generate a trusted positive ticket set $P_{labeled} + P_{trust}$ and a trusted negative ticket set N_{trust} through PU-learning, where $P_{trust} \subseteq U$, and $N_{trust} \subseteq U$. There could be some instances that are hard to be classified with a high confidence using the given conditions, for example, the instances that are near to the decision boundary. We call those instances as $U_{untrust}$, i.e., $U_{untrust} = U - P_{trust} - N_{trust}$. The key objective of learning is to find the subsets P_{trust} and N_{trust} from U, such that instances in P_{trust} are similar to the instances in $P_{labeled}$, while instances in N_{trust} have a completely different distribution. In the whole learning process, the P_{trust} and N_{trust} are initialized as empty and grow incrementally until the model gets converged.

Let $f(x)$ be the probabilistic output of the binary classifier f for instance x, where $0 \leq f(x) \leq 1$. If $f(x)$ is closer to 1, x is more likely from a positive class; otherwise, x is more likely to be negative. Let $L(y_{pred}, y_{true})$ be the loss function, such as the cross-entropy loss. To formally characterize the above PU-learning goal, we can

use the following loss components:

- $E_{PL} = \frac{1}{|P_{labeled}|} \sum_i L(f(x_i^{PL}), 1)$, where $x_i^{PL} \in P_{labeled}$. E_{PL} can represent the cross-entropy loss for classifying a labeled positive instance to the positive class.

- $E_{PPL} = \frac{1}{|P_{labeled}|} \sum_i L(f(x_i^{PL}), 0)$, where $x_i^{PL} \in P_{labeled}$. E_{PPL} can represent the penalty for classifying a labeled positive instance to the negative class.

- $E_{PT} = \frac{1}{|P_{trust}|} \sum_i L(f(x_i^{PT}), 1)$, where $x_i^{PT} \in P_{trust}$. This is the cross-entropy loss for the artificial P_{trust} set.

- $E_{NT} = \frac{1}{|N_{trust}|} \sum_i L(f(x_i^{NT}), 0)$, where $x_i^{NT} \in N_{trust}$. This is the cross-entropy loss for the artificial N_{trust} set.

- $E_{UU} = \frac{1}{|U_{untrust}|} \sum_i L(f(x_i^{UU}), 0)$, where $x_i^{UU} \in U_{untrust}$. This is the cross-entropy loss for the untrusted samples in the unlabeled data. Here we assume those untrusted samples are a noisy negative dataset, i.e., the similar assumption in the Biased Learning methods.

The training process includes three stages: (1) the warm-up stage, (2) the PU-loss pretraining stage, (3) the self-paced learning stage. Initially, $P_{trust} = \varnothing$, $N_{trust} = \varnothing$, $U_{untrust} = U$.

In the warm-up stage, we only consider $P_{labeled}$ as the whole positive data and all U as the negative class. The model is trained as a standard binary classifier using the cross-entropy loss:

$$L_{warmup} = E_{PL} + E_{UU}, \tag{4.1}$$

The goal of the warm-up stage is to initialize the model weights so that the model can

find the unlabeled samples that are most similar to the labeled class (i.e., those "false positives") and the samples that are most dissimilar to the labeled class (i.e., those "true negatives"). After a few rounds of training in the warm-up stage, the model switches to the PU-loss pretraining stage.

In the PU-loss pretraining stage, the model is trained using $P_{labeled}$ as the positive class and U as the negative class. The optimization goal is to minimize the unbiased PU-loss [37, 80, 81]:

$$L_{warmup} = \pi_p E_{PL} + E_{UU} - \pi_p E_{PPL}, \tag{4.2}$$

where π_p is the prior probability of the positive class. Different from traditional binary cross entropy loss, in the unbiased PU-loss, $\pi_p E_{PL}$ is the direct estimation of the positive class loss, while $\pi_n E_N = E_{UU} - \pi_p E_{PPL}$ is the indirect estimation of the negative class loss using the labeled positive data and the class prior, where $\pi_n = 1 - \pi_p$. This biased reweighting method prevents the model from too aggressively thinking all unlabeled data samples are negative in the training process, and produces a reasonable PU-classifier. However, the reweighting method estimates the loss at the statistic level based on the SCAR assumption, while, as explained before, the SCAR assumption may not be realistic for the troubleshooting problem. Therefore, we still need to consider individual sample-level errors in the next self-paced learning stage.

The key idea of the self-paced learning stage is that in each training round, we select a subset of the unlabeled samples as the "trusted" negative samples based on the current model output probabilities and add those samples to the trusted negative set N_{trust}. Similarly, we also augment P_{trust} based on the sample probabilities. Then we

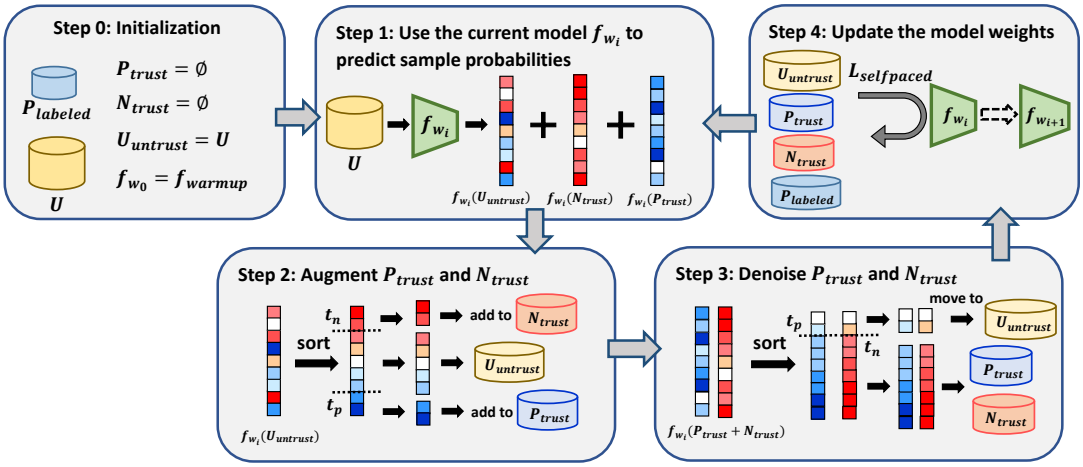


Figure 4.3: The self-paced learning steps.

update the model parameters again based on the new N_{trust} and P_{trust}, $U_{untrust}$, and P_{label}. The process is repeated until the model get converged.

Fig. 4.3 shows the key steps of one round in the self-paced learning stage. In step 1, for the ith round, we use the current model to predict the probability $f_{w_i}(x)$ of each sample x in U being classified as the positive class. In the first round, the current model f_{w_0} is the obtained model after the warm-up stage f_{warmup}.

In step 2, we select the top $r_p \times |U|$ samples with largest predicted probabilities as the positive sample candidates, and the top $r_n \times |U|$ samples with the smallest probabilities as the negative sample candidates, where r_p and r_n are the learning pace for trusted sample augmentation. In addition, we also use two probability thresholds t_p and t_n to decide whether the sample candidates could be trusted. For a selected positive sample candidate x_p, if $f_{w_i}(x_p) > t_p$, we move x_p from $U_{untrust}$ to the trusted positive set P_{trust}. Similarly, for a selected negative sample candidate x_n, if $f_{w_i}(x_n) < t_n$, we move x_n to the trusted negative set N_{trust}. The goal of step 2 is to augment the set

of the trusted samples based on the current model and enhance the learning by using those samples and the labeled samples as the target to compute the loss.

In step 3, we also want to eliminate the noise in the trusted data. As the learning proceeds, some samples that are trusted in the first few rounds may become untrusted in the later rounds. Therefore, we move those samples from P_{trust} or $U_{untrust}$ back to $U_{untrust}$ if the current probability $f_{w_i}(x) \leq t_p$ or $f_{w_i}(x) \geq t_n$. This "in-and-out" self-correction mechanism [12] makes sure that the model does not get over-fitted to the noisy samples that are added to the trusted sets in early learning stages.

In step 4, the model parameters are updated using the current trusted/untrusted sets and the following loss function:

$$L_{selfpaced} = E_{PT} + E_{NT} + \pi_p E_{PL} + E_{UU} - \pi_p E_{PPL}, \tag{4.3}$$

where $E_{PT} + E_{NT}$ is the cross-entropy loss for the trusted samples, and $\pi_p E_{PL} + E_{UU} - \pi_p E_{PPL}$ is the PU-loss of the rest data.

The model get converged when (1) the changing rate of P_{trust} and N_{trsut} is smaller than a small value δ, where the changing rate is defined as $1 - \frac{|Xi \cap X_{i-1}|}{|Xi \cup X_{i-1}|}$, $|X_i|$ and $|X_{i-1}|$ are the current and the previous trusted set size; (2) the L_{self_paced} value get converged. After training, we can obtain four final data sets: P_{labled} (unchanged), P_{trust}, N_{trust}, and $U_{untrust}$. P_{trust} and N_{trust} are the augmented labeled data sets through PU-learning with the historical and future data observations. Those data sets can also provide meaningful supervision for the online troubleshooting phase model where limited observation of the massive data is available.

The student model design.

The student model is the inference model used in the customer interaction stage. The function of the student model is to predict the root cause of a reported service issue in short latency. Therefore, only historical network status data can be used in the student model for online inference. To incorporate the knowledge from the future data with additional supervision information, we train the student model using the raw labeled data P_{labled} as well as the augmented ground truth data P_{trust} and N_{trust} that are obtained by PU-learning with the teacher model.

In addition, as a model that intends to resolve real customers' concerns, the student model should provide insightful and interpretable troubleshooting logic to the customers and care agents. Motivated by previous research in model interpretability for networking systems [47], we choose to use decision-tree-based models rather than the deep neural networks for the student model.

Specifically, we use the manual statistical features (introduced above) of the cell-level and UE-level network measurement data as the feature profiles for the student model. The purpose of using those features instead of the raw time-series data is that the care agents can understand how those features contribute to the model's decision-making process. Note that for the student model, only the historical 7-day window features can be used.

To decide which machine learning model should be used, we tested the performance of the following interpretable models: decision trees, random forest, XGBoost [11], logistic regression, and fisher score [24]. The models are trained using

$P_{labled} + P_{trust}$ as the positive class and N_{trust} as the negative class. Among the tested methods, XGBoost provides the best classification result and good interpretability. Detailed comparison and model interpretation results are shown in section 4.3.

4.2 Trial System Design.

4.2.1 Overview

In this section, we mainly illustrate how to implement a trial troubleshooting system for real deployment. Different from the system prototype, there are a few key practical challenges and concerns that need to be addressed for real system implementation:

- Data synchronization and delay: different data sources used for troubleshooting are fed with different granularity and have different delays. In the trial system, different data sources should be properly synchronized. In addition, the feature profile generating delay should be evaluated.

- Model efficiency: we also need to consider the model efficiency at different configurations such that the care agent can timely utilize the model output results and perform troubleshooting during the care calls.

- Model updating: whenever a newly trained model is available, the model should be able to be easily replaced in the whole system pipeline.

- Distributed troubleshooting agents: since the volume of care contacts could be large in the peak hours, the system should be able to work in a distributed way

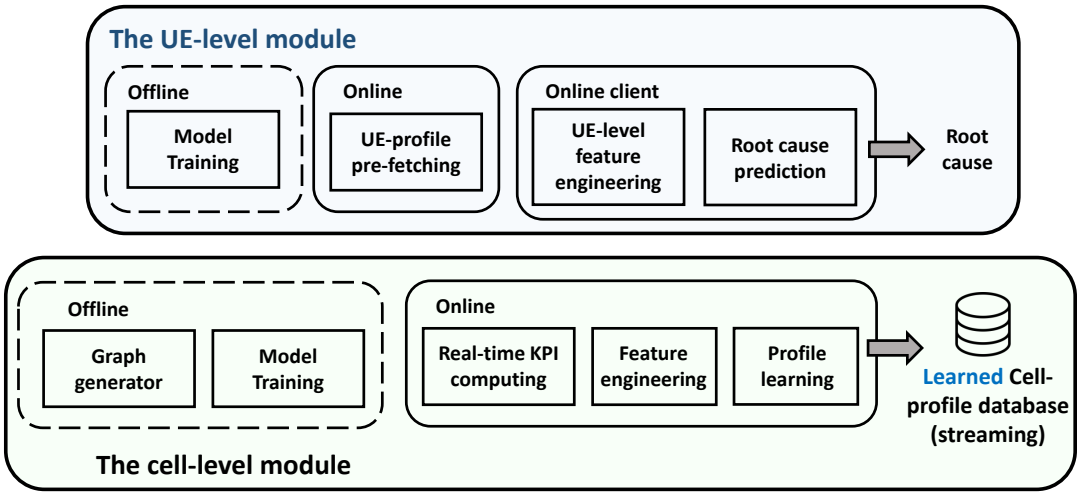


Figure 4.4: The NeTExp trial system design overview.

such that it is scalable to the market size.

In Fig. 4.4, we show the design overview of the NeTExp trial system. Similar to the NeTExp prototype, the real system design also includes the two major modules: the cell-level module and the UE-level module. The cell-level module can operate independently from the UE-level module. Namely, the cell-level model generates the cell-level feature profiles that describe the running status of each cell site in real-time, regardless of whether there is any care call coming in. In addition, the cell-level model archives the extracted cell-level feature profiles in a streaming database for lookups requested by the UE-level model. Then for the UE-level module, the inference functions are triggered when the care agent inputs the user's information (phone number, care contact time, etc.). The UE-level module looks up the most frequently accessed cell sites for the customer based on that information and then retrieves the corresponding cell-level feature profiles from the profile database created by the cell-level module. The

UE-level module also queries the UE-level network log database and creates the UE-level feature profiles. Finally, the feature profiles are sent to the UE-level ML model for troubleshooting inference. From this design overview, we can find the UE-level modules can work in parallel with the cell-level module, which greatly reduces the end-to-end responding delay for a case query in the online troubleshooting. In addition, multiple UE-level modules can work in parallel using the same cell-level feature profile database so that the system can handle multiple queries simultaneously or in a distributed manner.

The detailed system implementation of each module is shown in the following sections.

4.2.2 The Cell-level Module.

Graph generator.

One important input to the cell-level module is the adjacency graph of the cell sites. To obtain this graph, we implement a graph generator that runs in the offline phase. Specifically, the graph generator reads all UE-level session logs over a long period (e.g., one month) and analyzes the handover patterns of the users in a global market. Specifically, for each UE, if the UE builds sessions with cell site A and B simultaneously within a unit time (e.g., 1 hour), we add the proximity counter for the edge (A, B) by one. After measuring the graph edge weights using the whole database, we can obtain a vast and dense graph representation for all cell sites in the market. Then the graph weights are pruned by only selecting the top $k-1$ cell site nodes in the neighborhood. The pruning can effectively reduce the input graph size and eliminate the noisy measures

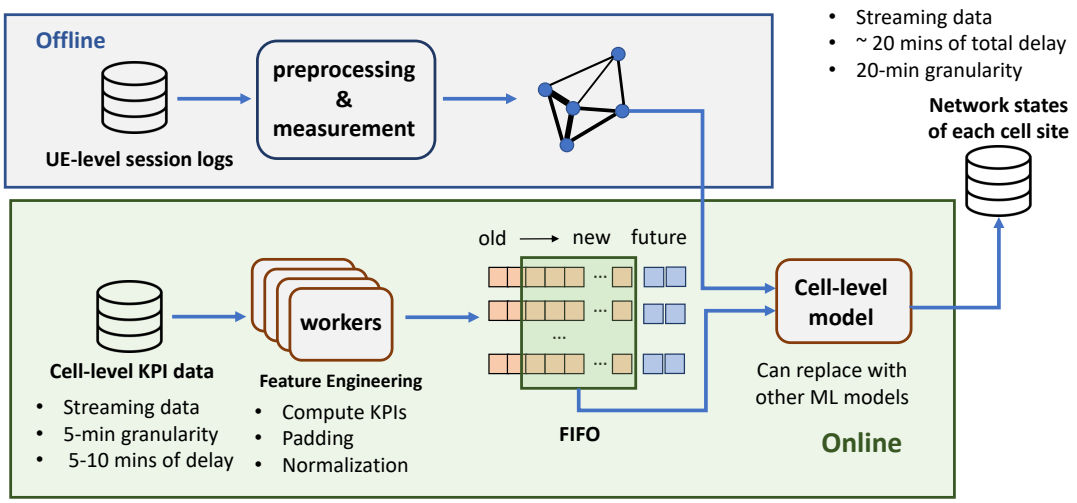


Figure 4.5: The cell-level module system design.

that are caused by the user's long-distance mobility. The k is set as 5, according to our analysis in section 3.5.3.

The graph generator runs in the offline (as shown in Fig. 4.5). For a large market with hundreds of millions of users and tens of thousands of cell sites, the graph measurement could be very time-consuming (around three days to finish). Fortunately, the graph weights do not change significantly over time for a stable network, and the overall graph only needs to be computed once assuming the network is not updated. If the network is updated in a local area, such as adding or removing a cell site, we only need to recompute the graph for the small area where the configuration change is deployed.

The online feature generation pipeline.

In the online phase, the cell-level module reads the streaming measurement data, compute the KPIs, engineer the features, and feeds the features to the model for

learning. Specifically, some of the available raw KPI counters are fed at 5-minute granularity while others are fed at 1-hour granularity. The counters fed at 5-min granularity have less than 10 minutes of delay, while the other counters have around 30 minutes of delay. For the counters that measure the average KPI in every hour and have around 30 minutes of delay, the total delay for being aware of a network issue is around one and half hour in worst case. To minimize the possible delay of the available streaming data for NeTExp , we synchronize all the data feeds to the counters with 5-min granularity. Specifically, for the data counters with larger granularity and longer delay, we interpolate the data to finer granularity and pad the data using the latest counter values for the delayed time slots. Although the data filled by interpolation and padding may not be accurate if that network KPI changes significantly, we can still use the other KPIs with finer granularity to learn the network status as timely as possible.

The online feature engineering component runs in real-time, namely, it processes the data every 5 minutes when a new counter value is available. For each KPI, we use a thread (i.e., the "worker") to compute the KPI value based on the counters, pad and interpolate the value if necessary, and normalize the current value data based on the historical KPI statistics for the same cell site at the exact timestamp of the day (the detailed normalization method is shown in section 3.3.3). Then the normalized KPI values are inserted into a FIFO (first-in-first-out) queue with fixed length (48 hours according to our evaluation in section 3.5.3). When the queue is updated (i.e., new counter values are inputted), we can scan over all the cell sites in the market and generate the feature profiles of each cell site and its nearest neighbors using the feature vectors of those corresponding cell sites in the queue. Next, the feature profiles of each

cell site are fed to the machine learning model for inference. The model output represents the current estimated network issue risk of each cell site. We save the outputs in a new cell-level network state database using the cell site ID as the key for future usage in the UE-level module. When a new machine learning model is available, for example, if the model parameters are fine-tuned based on recent data, we can replace the old model with the new model without changing other parts of the pipeline.

For a large market with tens of thousands of cell sites, the cell-level ML model inference delay for all cell sites could be as long as a few minutes (detailed evaluation results are shown in 3.5.3). Therefore, we degrade the inference frequency to once every 20 minutes (i.e., after four counter inputs) in our CPU servers for a global market. The degradation will cause a longer delay for tracking a network problem on the cell level. Compared with the UE-level data feed delay (which is one hour), this delay is acceptable. For more effective network problem detection on the cell level, we can also use distributed cell level modules for a large market, where each module only monitors and learns the network status of a small number of cell sites locally.

4.2.3 The UE-level Module.

The system design of the UE-level module is shown in Fig. 4.6. The UE-level module mainly includes two components: the data retrieving phase and the decision-making client phase.

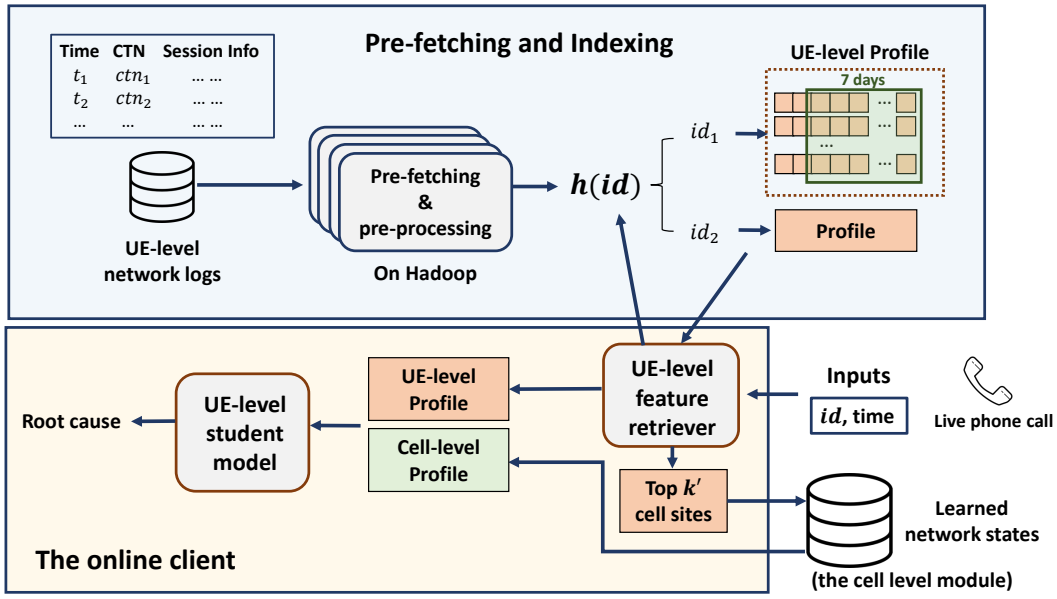


Figure 4.6: The UE-level module system design.

Fast UE-level data retrieval.

When the care agent inputs the UE's information to the NeTExp system, the first step of the UE-level module is retrieving the corresponding network log data for the UE from the backend database. However, the raw UE-level log database is vast for a large market. For example, the data block size for a market with around 3 million users over a 7-day historical window is hundreds of Gigabytes. In addition, for efficient session record logging, the raw database streams the real-time session records in a chronological way. Namely, the records are ordered by the timestamps instead of being indexed by user IDs. Hence, it is costly to load the whole database to memory and search for the required records during the online query phase.

To solve this problem, we implement a pre-fetching and indexing component for fast UE-level data retrieval. Specifically, we use independent threads to preprocess

the raw database, group the database records by the UE IDs, parse the raw data, and save the preprocessed data blocks to disk using the hash of the UE IDs as the indexes. The preprocessing module is implemented on Hadoop with Apache Pig. This module is triggered each hour whenever the new streaming UE-level raw data block is available (the raw data streaming granularity is 1 hour). Moreover, the generated new database indexed by the UE IDs can be shared by multiple troubleshooting clients in the online phase. By using the hash of the UE IDs as the key, the inference client only needs to read a small data block to memory for a given ID, which significantly reduces the I/O and data searching time cost. In fact, based on our evaluation, this part of I/O and data searching cost is the dominating latency in the end-to-end online inference latency, even with the proposed preprocessing module. Still, the preprocessing module successfully reduces the I/O and data searching cost to an acceptable level for practical online inference (detailed analysis is shown in Section 4.3).

Online client for root cause classification.

Finally, we implement the online client for root cause classification in the customer interaction phase. Specifically, the client first retrieves the corresponding UE-level historical network log data from the pre-fetched database. Then it computes the statistic of the user sessions associated with each of the cell sites the user accessed and finds the top k' cell sites that are most frequently used by the user. Through analysis, we find the top $k' = 5$ cell sites can cover 91% of usages for the customers on average and would be sufficient to locate the network problem that greatly impacts the user's experience. Based on the top k' cell sites, the troubleshooting client retrieves the

learned network status of the corresponding cell sites from the network status database created by the cell-level module. Finally, the client computes the cell-level and UE-level feature profiles for the UE and feeds the feature to the machine learning model for inference. The machine learning model is an XGBoost model that can also output the key features and value thresholds in each step of the decision trees for the final decision. The explanation of those key features can enable the human care agents understand why the problem is more likely a network-side or device-side problem over the other one.

In addition, the XGBoost model can be easily replaced by other fine-tuned models or interpretable models. For example, if the current manual troubleshooting framework is improved and more supervision knowledge could be obtained from the manual tickets, we can retrain the teacher and student model with advanced ground truth knowledge and new data using PU-learning or standard supervised machine learning.

4.3 Evaluation.

4.3.1 Performance of NeTExp with PU-learning.

The teacher model training process.

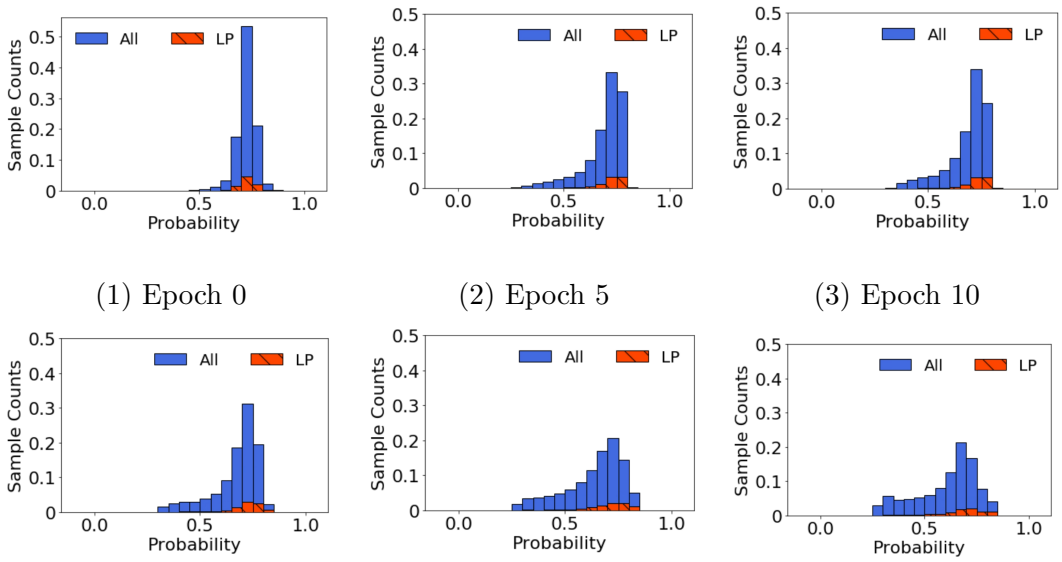
We collect the care contact log and ticket resolution log data for one month in 2021 to evaluate the system. Around 10% of tickets can be verified to be network-issue cases, while the root cause of the rest tickets cannot be verified. We use 70% of data for training and 30% of data for validation. The KPI feature data sources are similar

to the datasets illustrated in section 3.5.1. Since all current manual troubleshooting tickets are obtained from the existing troubleshooting framework, there is no confident labeling method to label a uniformly random set of the reported service issues and create a trustful benchmark for the PU-learning evaluation. To measure how well the model performs, we compute the following metrics:

- The recall of the labeled network issue samples (LP-recall): $\frac{|TP|}{|P_{labeled}|}$, where TP is the true positive data that can be verified by the known labels, i.e., the tickets that are labeled as network issues and are successfully recognized by the model.

- The labeled positive rates in the predicted positive class (LPP-rate) and predicted negative class (LPN-rate), i.e., LPP-rate = $\frac{|TP|}{|P_{predicted}|}$ and LPN-rate = $\frac{|FN|}{|N_{predicted}|}$, where $P_{predicted}$ and $N_{predicted}$ are the data that are classified as positive and negative respectively, $|FN|$ is the false negative data that can be verified by the known labels, i.e., the tickets that are labeled as network issues and are mistakenly recognized by the model. For an effective PU-learning model, LPP-rate > LPN-rate.

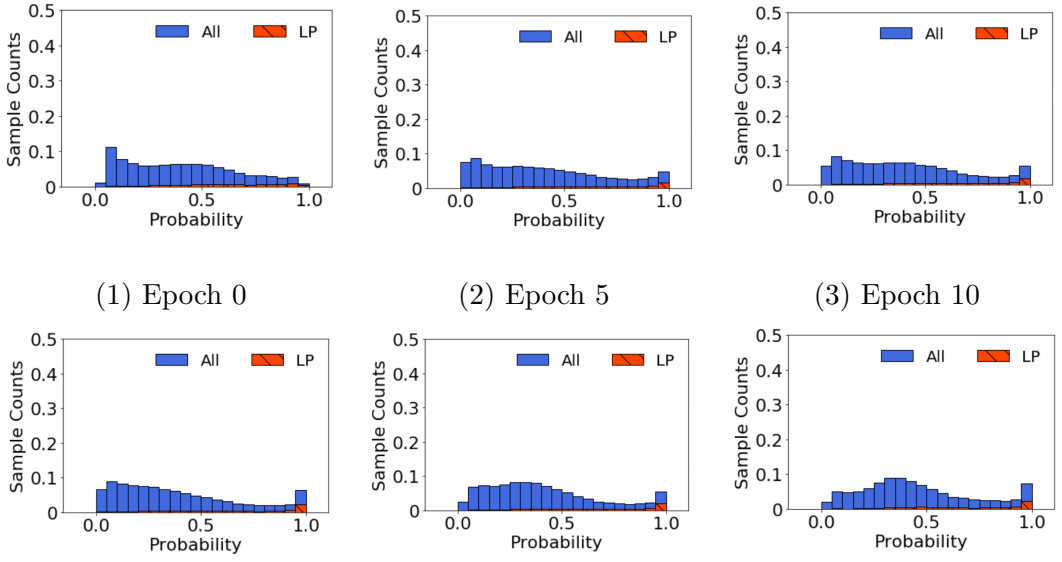
In Fig. 4.7, Fig. 4.8, and Fig. 4.9, we show the learned probability histograms during the warm-up stage, the PU-loss pretraining stage, and the self-paced learning stage of the teacher model. In the histograms, the X-axis is the probabilistic output of the teacher model for the input instances, and the Y-axis is the count of how many samples are classified to the specific probability range. In addition, we separate the labeled network issue cases and the unlabeled cases with different colors in the figure. The "All" bars in the figures represent the distribution of the predicted probability of



(1) Epoch 0 (2) Epoch 5 (3) Epoch 10

(4) Epoch 15 (5) Epoch 20 (6) Epoch 25

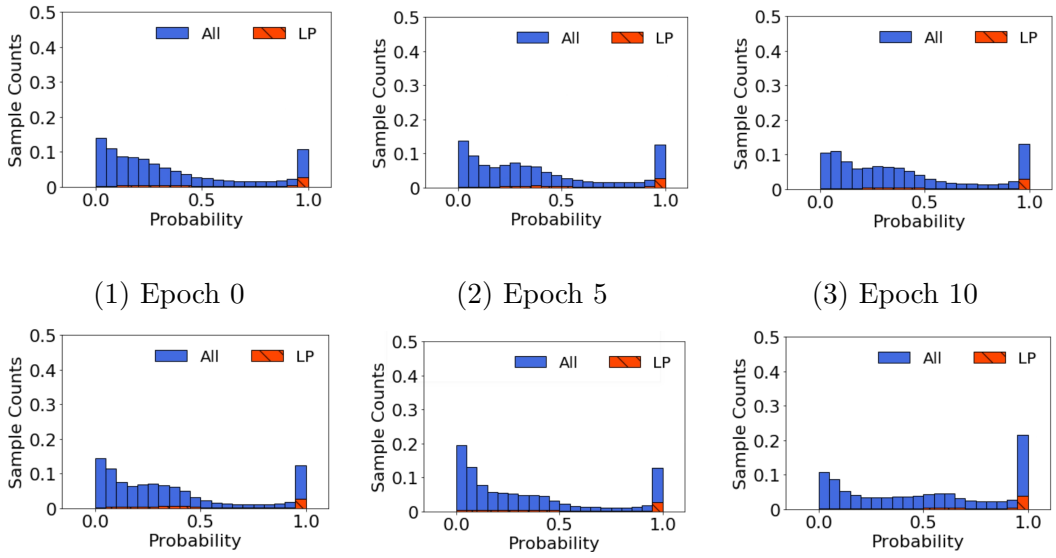
Figure 4.7: The training process of the warm-up stage.



(1) Epoch 0 (2) Epoch 5 (3) Epoch 10

(4) Epoch 15 (5) Epoch 20 (6) Epoch 25

Figure 4.8: The training process of the PU-loss pretraining stage.



(1) Epoch 0 (2) Epoch 5 (3) Epoch 10

(4) Epoch 15 (5) Epoch 20 (6) Epoch 25

Figure 4.9: The training process of the self-paced learning stage.

network-side issue for all received service issue reports, while the "LP" bars represent the learned distribution of the labeled network issue reports only. The histograms are normalized by dividing the bar heights by the total number of all received service issue reports. In the ideal case, if the model can perfectly classify every case, we could obtain a histogram where all labeled network issue cases are concentrated to the right side of the probability distribution (i.e., $f(x) > 0.5$), showing those cases are more likely to be network issues; while a good proportion of the unlabeled cases are distributed in the left side of the distribution (i.e., $f(x) < 0.5$), since it is evident that a large number of the reported problems are indeed device-side problems according to the experiences of the network operators, although those tickets cannot be verified by the ground truth tickets. However, since troubleshooting the individual service issue is inherently challenging and limited ground truth data is available, the real model we can obtain cannot perfectly

classify all cases.

From Fig. 4.7, we can find that the warm-up training stage with the cross-entropy loss function can learn to push the labeled positive samples towards the positive class side in the probability distribution histograms. However, since the "negative" class in the training stage includes many "real positive" samples, which are considered as noises in the "negative" class, the warm-up stage with the standard binary classification method cannot correctly figure out a clear boundary between the real positive and negative classes. In fact, many samples in the unlabeled data are still network-issue-related samples that are not detected by the ground truth ticket data. Since those instances have similar feature patterns as the labeled network issue samples, the standard binary classification loss function that ignores this fact would generate incorrect gradient directions for adjusting the decision boundary.

In Fig. 4.8, we notice that training with the PU-learning loss function can make the probability distribution of the training samples spread out over the range space $[0, 1]$. The pre-training stage with PU-loss does not aggressively think all unlabeled data are negative and can classify those unlabeled samples based on their similarity to the positive classes. As the learning proceeds, the model can push the label positive samples towards the positive class side and push the samples that are not similar to the positive class towards the negative class side. However, it is still hard to decide a proper decision boundary based on the results of the warm-up learning stage, as most data samples are distributed to the "ambiguous" zones based on their probabilities.

The self-paced learning stage, as shown in Fig. 4.9, can make the decision boundary clear by pushing most samples towards the "0" or "1" sides of the X-axis.

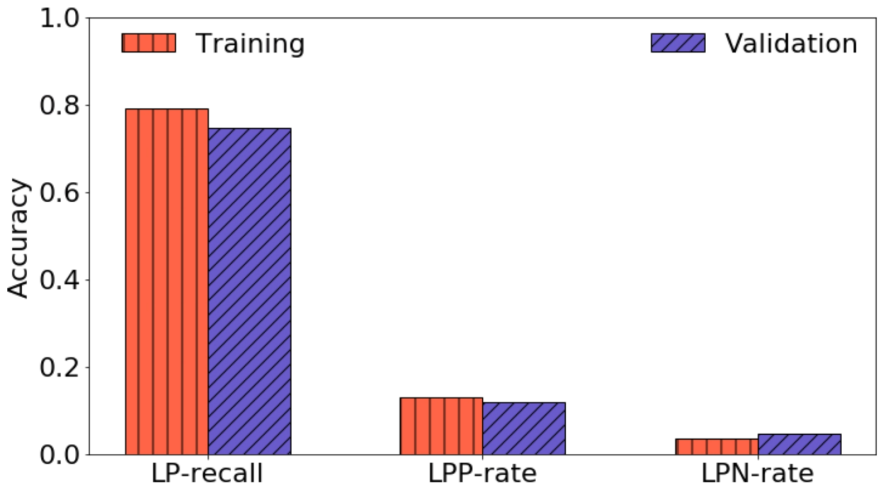


Figure 4.10: The LP-recall, LPP-rate, and LPN-rate of the teacher model.

The key mechanism is that the model gradually considers more samples with high classification confidence as the trusted positive and negative data. Thus, the model can become more and more confident for most samples while avoiding overfitting by eliminating the samples with less-confidence from the trusted sets in each training round.

In Fig. 4.10, we show the LP-recall, LPP-rate, and LPN-rate of the teacher model for the training and validation data respectively, where we use 0.5 as the probabilistic decision boundary for the positive and negative class. The results show that the model successfully recognizes 75% of the labeled network issues (LP-recall) for the validation dataset. In addition, the LPP-rate is more than twice the LPN-rate, showing that the model is effective for the PU-learning objective.

The student model performance.

The student model is trained to mimic the behavior of the teacher model using only the historical observations for the service issue cases. Specifically, we use the

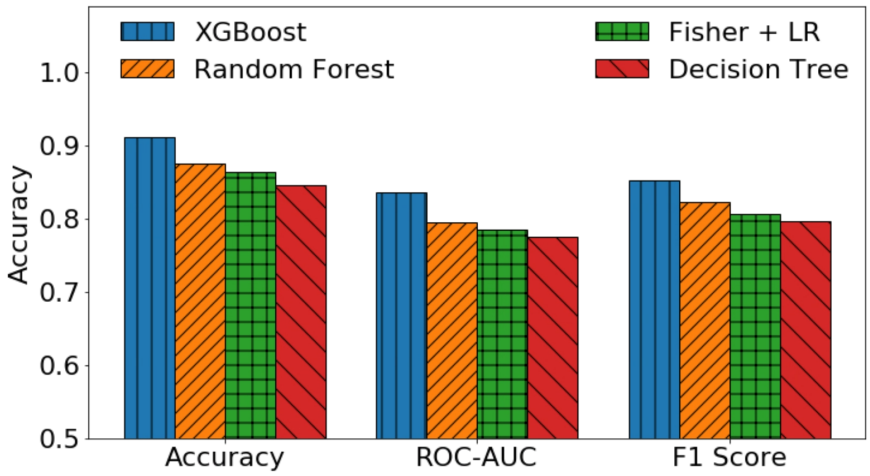


Figure 4.11: The ROC-AUC, Accuracy, and F1-Score of the student model.

N_{trust} as the negative class and $P_{trust} + P_{labeled}$ as the positive class when training and evaluating the student model. The performance (RoC-AUC, Accuracy, and F1-score) of the different interpretable student models on the validation dataset are shown in Fig 4.11. From the results, we find the XGBoost outperforms the other machine learning models because of its powerful capability of representing complex features.

In addition, the detailed accuracy breakdowns for $P_{labeled}$, P_{trust}, and N_{trust} of the models on the validation dataset are shown in Fig. 4.12. For the $P_{labeled}$ set, the XGBoost and Random Forest model can achieve more than 80% of recall, which is even higher than the LP-recall of the teacher model, which takes more knowledge for its input. The key reason is that the student model can obtain more supervision knowledge from the training data (namely, the confident labels), which is provided by the teacher models. Besides, XGBoost can also achieve more than 80% accuracy on the other two groups with artificial labels, i.e., P_{trust} and N_{trust}.

In Fig. 4.13 we show the top 18 features ranked by the information gains in

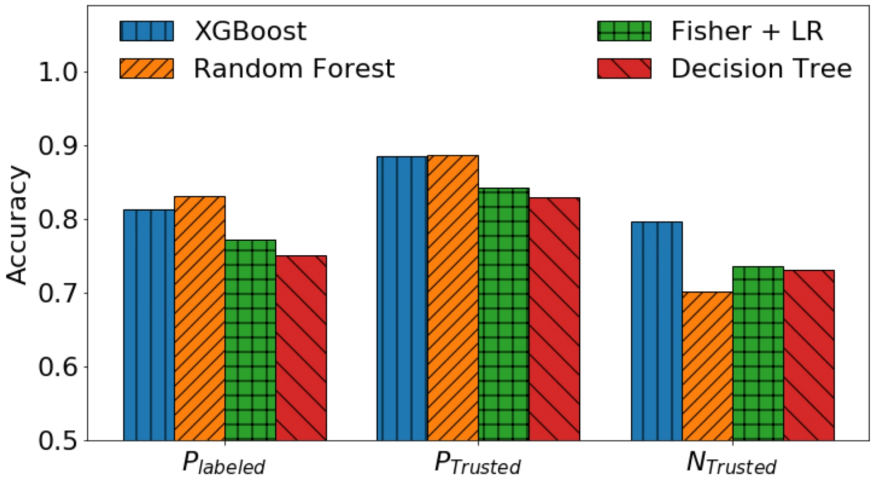


Figure 4.12: The accuracy scores for N_{trust}, P_{trust}, and $P_{labeled}$.

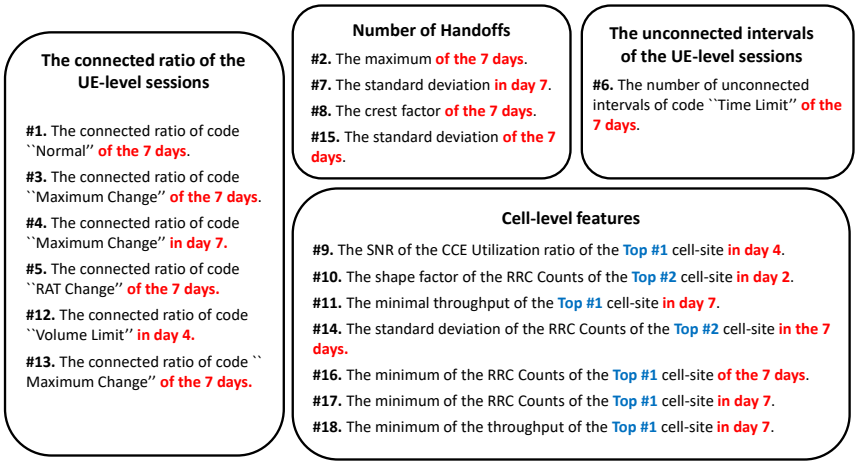


Figure 4.13: The top 18 features with maximum information gains in the XGBoost model.

XGBoost trees. We highlight the time period and the cell site for which the feature is computed. For example, "day 7" means the last day of the 7-day historical window, i.e., the closet 24-hour interval before the care contact time. The "top # 1" cell site means the cell site that is most frequently accessed by the user in the past week. For the UE-level session codes, "Maximum Change" is related to the frequency of handoffs, "RAT Change" reflects the change of radio access technologies, "Volume Limit" is related to the volume of the data used in the session, and "Time Limit" is related to the duration length of the session. The feature names are grouped based on the categories. From the results, we can find: (1) Those automatically learned features are generally consistent with the experiences of the expert human network operators. Specifically, the overall connected ratio of the normal sessions and handoff behaviors can characterize the key symptoms on the UE-side, and the abnormal cell site KPIs can characterize the network states on the network-side. (2) The features on the most frequently access cell sites in the nearest 24-hour interval before the care contact (i.e., day 7) can most properly characterize the issues, which are also thought as the more important features by the model. (3) The XGBoost model can automatically decide the splitting thresholds on those features, while those thresholds are hard to be decided precisely by human experts.

In summary, with the tree-based student models, the care agents can understand how a decision is made in the automatic troubleshooting process. Therefore, they can properly evaluate the correctness of the model's decision and perform appropriate future steps if necessary.

	Log query	UE-feature	Cell-feature	ML-model
Average	11.45	0.073	0.061	0.408
90^{th} percentile	15.29	0.13	0.080	0.431
95^{th} percentile	17.46	0.15	0.086	0.456

Table 4.1: The average, 90^{th} and 95^{th} percentiles of the time costs (in seconds) of NeTExp components.

4.3.2 System Efficiency.

To understand the feasibility of deploying NeTExp into the existing online customer service framework, we evaluate the end-to-end responding time cost of NeTExp to individual queries. According to Fig. 3.11 and 3.12, the cell-level module of NeTExp takes millisecond-level delay for cell site state learning on CPU servers, and thus can support real-time (i.e., in 5-minute granularity) state feature representation for a market with tens of thousands cell sites. Since the cell-level module can work in parallel regardless whether there is any care contact, we only take account of the following delays for online customer query responding. **(1) Log query delay**: the cost to read the 7-day historical UE-level logs from database on disk. **(2) UE-feature delay**: The cost to parse the UE-level logs, extract the top k relevant cell site IDs, and create the UE-level feature profiles. **(3) cell-feature delay**: The cost to read the cell-level model features for the corresponding cell sites and time windows. **(4) ML-model delay**: The cost for the execution of the UE-level XGBoost model for inference.

We simulate the online query process using 1000 historical care contacts from a market with more than 2 million users and measure the delays of each components

for this market. The average, 90^{th} and 95^{th} percentiles of the time costs of different components are shown in Table 4.1.

From the results, we notice the longest delay is the UE-level network log query delay. The reason is that the UE-level network log data for a million-scale user population market is rather massive and cannot by loaded entirely to memory. Thus, the record searching and disk I/O cost for reading the corresponding log data for one particular user dominates the end-to-end online process. This part of cost can be effectively optimized by using faster storage, finer-grained key-value pointers for data localization, or a memory cache that prefetches the profiles of the users who are likely to contact the care based on learning and forecasting [64]. One practical solution to further reduce this cost is dividing the large market data into smaller data blocks by user locations and use finer-grained key-value pointers to localize the corresponding user profile blocks. Another possible optimization method is adding a memory cache that prefetches the profiles of the users who are likely to contact the care based on learning [64].

In addition to the data I/O cost, the feature engineering and model execution costs are small enough (less than 1 second in total) to be neglected in a live phone call conversation. Taking account of the large I/O cost, the average end-to-end delay is still less than 20 seconds, while a customer care contact usually takes a few minutes or longer. Thus, the inference results generated by NeTExp can well assist the care agent at early stages of the care calls.

4.4 Conclusion.

In this chapter, we study how to address the label insufficiency problem of the ground truth data in the cellular network troubleshooting task. Specifically, we applied a teacher-student model design to address the problem under a PU-learning framework. The evaluation results show that the PU-learning approaches can indeed improve the model accuracy for many challenging reported issues that cannot simply be characterized by the rules. In addition, we use interpretable machine learning models in the decision-making client phase, so that the troubleshooting clients can provide the route map of how the conclusion is made based on the available human-readable features. We implement a trial system of the enhanced NeTExp prototype design and show that the system can well support fast root cause identification during the online customer interaction phase.

Part II

On-device Certificate Revocation Checking for Peer-to-Peer Mobile Network Applications.

Chapter 5

Authentication and Certificate Revocation Checking in Mobile Networks.

5.1 PKI-based Authentication in IoT Ecosystems.

With the rapid growth of IoT service market in recent decades, there is an increasing demand for secure peer-to-peer communication protocols in a universe with millions of IoT devices. Therefore, peer-to-peer *device authentication* becomes a fundamental security problem of novel IoT and the building block of many emerging critical IoT security protocols for communication privacy (such as the TLS-style protocols) as well as IoT data authenticity and integrity (such as the digital signature protocols) [1, 2, 44, 50]. The state-of-the-art solution of device authentication is to use *device*

certificates based on the Public Key Infrastructure (PKI) [1,2]: each device is assigned a device certificate by a Certification Authority (CA). For example, the IoT architecture of Symantec enterprise security (now Broadcom) allows assigning certificates to millions of devices [1] and verifying the device certificates by the device management servers [2]. Note that the "CAs" for IoT devices refers to not only general public SSL certificate authorities, but private certificate issuers in a service managed by the service provider.

Therefore, *on-device certificate verification* [70,74], i.e., allowing one IoT device to verify the certificate of another device, becomes an emerging and vital component for IoT security. For example, many emerging and future IoT applications require secure peer-to-peer communication directly or via the global mobile network, such as autonomous robotic systems, vehicular communication, wearable healthcare systems, smart industrial control, and IoT-based post-disaster management. A device should use its own data and power to verify the certificate of another communicating device, to further build a secure channel using protocols such as DTLS [57]. In addition, an IoT device may need to process the sensing data collected from other devices, which carries the digital signatures from the sensing sources [44]. Verifying the public-key certificates is essential in validating digital signatures to ensure data authenticity and integrity. For example, in a smart city, an IoT sensor has to authenticate the mobile devices of the authorized users so that it can only provide the sensing data to the users who have the authority. Meanwhile, user devices also need to verify the signatures of sensing data to ensure the data are not tempered by an attacker.

However, on-device certificate verification remains a challenging problem mainly due to the high latency and bandwidth cost of the revocation-checking step. Verifying

a digital certificate takes three main steps: 1) check its validity period; 2) validate the CA's digital signature using CA's public key; 3) verify the certificate revocation (CR) status. Step 1 is simple. Step 2, although involving public key cryptography, takes bounded time and memory that can be afforded by most IoT devices. Step 3 is considered an expensive process even for a desktop machine [41, 67, 87]. Some issued digital certificates may have been revoked by the CAs [38,87,89] due to a number of reasons: 1) the device is stolen; 2) the private key of a device could be compromised by attackers; 3) the CA may find that a certificate is a mis-issuance; 4) a device may unsubscribe from an IoT service while its certificate is still within its valid time period; 5) the database of an IoT service provider or device manufacturer might be hacked and the private key information could be leaked. Upon being notified with these situations, the CA should immediately labeling these certificates as "revoked".

For SSL certificates, a certificate revocation list (CRL) [29] containing all revoked certificates is prepared by the CA and sent to end-users for revocation checking [67, 87, 89]. The CRL introduces substantial overhead even if it runs on a desktop machine, because the CRL size is proportional to the number of revoked certificates, which can be in millions [67, 87, 89]. For IoT, the overhead problem is more severe, because: 1) the number of IoT devices could be more than that of web servers; 2) the memory, CPU, and network resource of an IoT device is much more limited than those of a desktop. In addition, unlike web servers, IoT devices are small in size, have better mobility and are usually maintained by individual users, which also means the devices are much easier to be hacked or stolen. Hence, revocations for IoT certificates happen more frequently and have to be properly and timely handled. When a revocation hap-

pens, how soon other parties are aware of the revocation and no longer trust the device becomes a rather critical metric of the security property in the protocol. The main requirements of practical on-device IoT CR checking are summarized as follows:

1. **Accuracy**: A device should determine a certificate revocation status without error.

2. **Efficiency**: The protocol should cost small memory, computation, and network resource on IoT devices.

3. **Low latency**: Two types of latencies are essential, namely the synchronization latency and query latency (defined in Sec. 6.1.1). Both latencies should be low.

4. **User privacy:** The protocol should not leak the identities of the accessing devices, locations, and/or communication pattern/frequency of users.

5. **Compatibility:** The protocol is required to be compatible with current certificate standards and existing certificates.

5.2 Related works.

5.2.1 Certificate Revocation Checking Approaches

Existing approaches for checking CR status are mainly based on either remote or local queries [17, 23, 40, 41, 49, 67]. A typical remote querying protocol is the Online Certificate Status Protocol (OCSP) [17, 49]. In OCSP, an authorized OCSP server returns the signed revocation status at the request of the client. However, the remote queries also reveal the clients' footprints of their SSL sessions to the OCSP server. In

addition to the privacy leakage threat, OCSP also suffers from the long network latency problem, which makes it unfeasible for time-sensitive applications. To mitigate the above problems, recent OCSP-based methods, such as OCSP Stapling [17, 25], Revocation in the Middle (RITM) [68] and Certificate Revocation Guard (CRG) [31], choose to offload the CR checking to the certificate provider or a middle-box intercepting TLS traffic. However, those approaches would increase the overhead on the certificate provider side, and thus is not scalable for device-to-device authenticate scenarios.

On-device CR checking preserves user privacy by allowing them to check CR status locally through a compact data structure model, such as CRLSets [40], OneCRL [23], CRLite [41] and CRV [67], which is periodically synchronized from the CAs or device management servers. These methods are also known as the push-based models. Since the raw CRLs can be very large, there is a clear trade-off among the on-device memory cost, the overhead to query or update the on-device data structure, and the checking accuracy. For example, CRLSets [40] and OneCRL [23] trade checking accuracy for efficiency by maintaining a subset of the CRLs. CRLite [41] and Let's Revoke [67] can provide 100% checking accuracy with a rather compact data structure, while updating the data model is expensive.

A comparison of the existing CR checking methods and our work TinyCR is shown in Table 5.1, where the results of Let's Revoke [67], CRL [29] and OCSP [49] are from the original papers or a measurement paper [46]. Note that all those compared methods are required to provide zero error assuming the on-device data models are properly synchronized. Compared with the state-of-the-arts, TinyCR achieves the best trade-offs for the IoT device authentication scenarios: (1) it costs close-to-optimal on-

device memory and computational overhead for checking the CR status, (2) it uses magnitudes-less time and bandwidth for CRL state synchronization with the CAs, (3) as a push-based model, it protects users' privacy, (4) it does not require any field changes in the certificates and it is back-compatible for all X.509 certificates.

5.2.2 Efficient Data Structures for Membership Queries

On-device certificate revocation checking requires implementing compact data structures on IoT devices for maintaining the global CRL states and performing efficient membership query. To fulfill the low-memory-cost needs, standard membership query methods, such as bloom filters [8], cuckoo filters [19] and their variants [48, 75, 84], use probabilistic key-value store models that trade efficiency for accuracy, namely, they can allow a small fraction of false positives at a controlled rate. Clearly, those methods are not suitable for adversarial scenarios, since those false positives can be easily abused by the adversaries to compromise the system. Other methods that can provide zero error for querying, such as Filter cascades [41], Othello hashing [85], SetSep [90], and Coloring Embedder [71], are also not ideal, either due to the high memory or computational costs for storing and updating the CRL states.

5.3 A Novel Dynamic Asymmetric Set Separator.

CR checking can be modeled as a binary set query problem.

Definition 1 (Binary set query problem)**.** *Let U be a finite set of keys that can be divided into two disjoint subsets P and N, and $U = P \cup N$. The binary set query*

Method	Memory cost	Query time	Δ-msg size per update	Δ-msg size per day	Sync. latency	Push model	CA compat.
CRL [29, 46]	$\sim$ 38 MB	$\gg$ 250 ms	-	-	$\gg$ 250 ms	✗	✓
OCSP [46, 49]	$\sim$ 1 KB/req.	$\leq$ 250 ms	-	-	-	✗	✓
Othello [85]	29.1 MB	**$<$ 1 μs**	**0$\sim$100 B**	**0$\sim$20 KB**	**$<$ 1 ms**	✓	✓
CRLite [41]	**1.7 MB**	**$<$ 1 μs**	-	0.53 MB	1 day$*$	✓	✓
Let's Rev. [67]	**1.3 MB**	$\sim$ 10 ms	-	62.6 KB	1 day	✓	✗
TinyCR (ours)	**1.7 MB**	**$<$ 1 μs**	**0$\sim$108 B**	**2.8$\sim$21.6 KB**	**$<$ 1 ms**	✓	✓

Table 5.1: Comparison of certificate revocation verification protocols with 100 million certificates, assuming 1% revocation rate and 0.02% new revocations per day.

problem is that given $k \in U$, determine if $k \in P$ or $k \in N$.

All certificates that are checked for CR status are both time-valid and signature-valid, otherwise they will be rejected in expiration and signature checks. The IDM server knows all time- and signature-valid certificates (U) and they can be classified into to two finite sets: one for the legitimate certificates ('negatives' N) and the other for the revoked ones ('positives' P). Hence the CR checking result can be either 0 (not revoked) or 1 (revoked).

In this work, we design a data structure named Dynamic Asymmetric Set Separator (DASS) to solve the binary set query problem. We design DASS using an *innovative combination of existing algorithmic tools*. We first briefly introduce these tools.

5.3.1 Preliminaries

Filter tools and Cuckoo Filter.

A filter data structure is used for approximate membership queries [8,19]. For a given set S of keys, a filter F answers each query of key k and returns $F.\texttt{Query}(k) = 1$ if $k \in S$. However, filters introduce a small number of false positives at a controlled probability. Although those filters cannot meet the requirements of zero-error CR checking, we can use the filter tools to preprocess the queries and significantly reduce the certificate space that should be further investigated with extra overhead.

To satisfy the practical requirements of CR checking, the filer tool that is used for on-device certificates preprocessing should have the following properties: (1) low

memory cost, (2) easy to lookup, and most importantly (3) easy to update (adding and removing keys). We find Cuckoo Filter and its variants [19, 75] can well meet the above demands as it achieves optimal memory-accuracy tradeoffs and it is easy to be queried and updated. Cuckoo Filter is inspired by Cuckoo Hashing Table [51], in which a key can be stored in two candidate buckets of a hash table, whose positions are calculated with two hash functions. We take a (2, 4)-Cuckoo Filter as an example to illustrate the algorithm. As shown in Fig. 5.1, the Cuckoo Filter maintains a cuckoo hashing table with two hash functions $h_1(x)$ and $h_2(x)$. Each bucket of the table has four slots.

Insert(k): To insert a key k into the Cuckoo Filter, the operation can be accomplished by inserting the fingerprint of k, i.e., $fp(k)$, into either one of the two candidate buckets of cuckoo hashing table. Specifically, the two candidate positions, i.e., $h_1(k)$ and $h_2(k)$, can be calculated using a single uniform hash function $h(x)$ by:

$$\begin{aligned} h_1(k) &= h(k) \bmod m, \\ h_2(k) &= h_1(k) \oplus (h(fp(k)) \bmod m), \end{aligned} \tag{5.1}$$

where $\oplus$ is the bit-wise *xor* operation, m is the size of buckets. Since it is easy to show $h_1(k) = h_2(k) \oplus (h(fp(k)) \bmod m)$, the cuckoo filter can find the alternate bucket position of k by simply calculating the *xor* of one bucket position and the hash of the fingerprint, i.e.,

$$h_j(k) = h_i(k) \oplus (h(fp(k)) \bmod m), \{i, j\} = \{1, 2\}. \tag{5.2}$$

If either of the two candidate buckets contains an empty entry, then the fingerprint $fp(k)$ is safely inserted to the empty entry. Otherwise, the insertion algorithm chooses a random entry of the two buckets and reallocate the stored fingerprint FP' into its

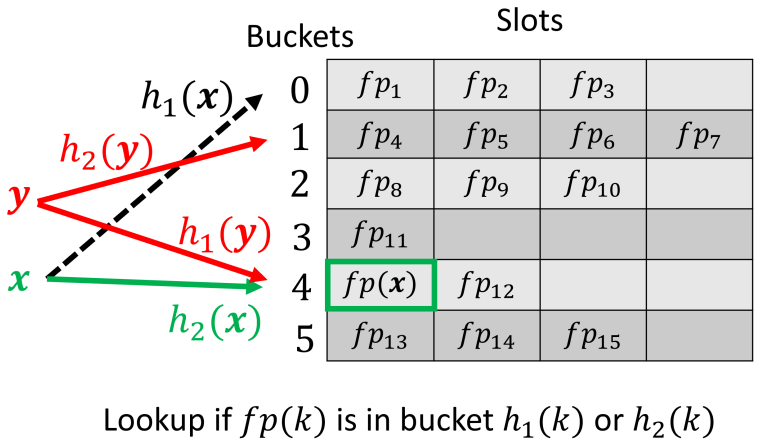


Figure 5.1: A (2,4)-Cuckoo Filter example.

alternate buckets in the hashing table, then insert $fp(k)$ to that entry. When reallocating FP', if the alternate bucket of FP' is also full, the algorithm will repeat randomly kicking off another fingerprint from the table and reallocate the other fingerprint until an empty entry is found, or until the maximal number kicking-off operations is reached, which implies the filter is too full to insert the new key k.

Query(k): To lookup whether a key k is a member, we only need to visit the two candidate buckets of the cuckoo filter using Eq. 5.1. If either of the buckets contains $fp(k)$, then we conclude that k is in the set; otherwise it is not.

Delete(k): Similarly, the deletion of a key k from the membership set can be accomplished by simply removing one copy of $fp(k)$ from the found bucket entry.

Set query tools and Othello Hashing.

A set query tool [9, 10, 71, 85, 90] can do exactly what we demand for binary set queries of CR checking. It returns 1 if $k \in P$ and 0 if $k \in N$ for any $k \in P \cup N$.

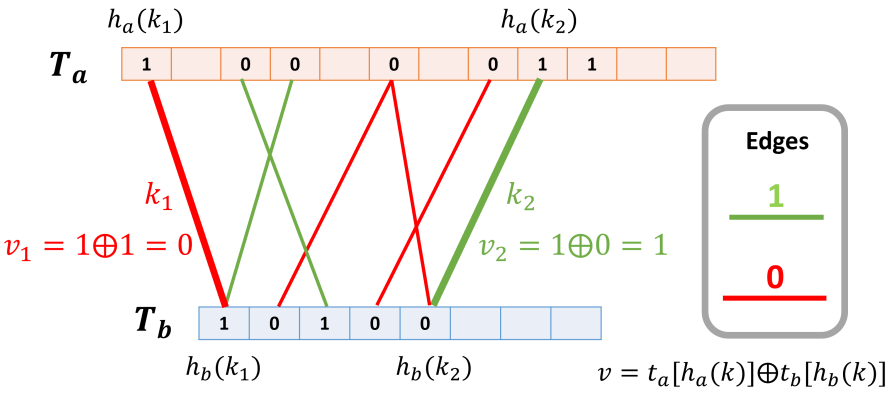


Figure 5.2: The Othello data structure for binary set query.

However, the space cost of set query data structures is proportional to $|U| = |P| + |N|$, i.e., they still introduce non-trivial memory cost for CR checking because $|N|$ is usually extremely large. On the other hand, although set query tools are too expensive for on-device CR checking over the entire certificate space, we could use those tools to only precisely identify the CR status of a small certificate subset that should be carefully investigated. Similar to the filter tools, we also concerns about the memory cost, query and updating overhead of those candidate set query models for the CR checking task. We find Othello hashing [85] is an ideal choice as it uses minimized memory cost for balanced key sets and is easy to be queried and updated with low overhead.

The structure of a one-bit Othello for binary classification is illustrated as Fig. 5.2, in which each bucket of the hashing tables contains a one-bit slot. Othello maintains two hashing tables, with each bucket of the hashing tables containing L bits, where $L = \lceil \log_2 n \rceil$ and n is the number of distinct sets. For example, $L = 1$ for binary set query (i.e., $n = 2$). Suppose the lengths of the two hashing tables T_a and T_b are m_a and m_b, and the corresponding uniform hash functions are $h_a(x)$ and $h_b(x)$. Othello is

built by finding an undirected acyclic $G = (V_a, V_b, E)$, where E is the edge set, V_a, V_b are the vertex sets with each node $v_a^i \in V_a$ $(0 \leq i < m_a)$ and $v_b^j \in V_b$ $(0 \leq j < m_b)$ representing the ith and jth bucket of T_a and T_b. Initially, $E = \varnothing$. For any key-value pair (k, v) with $k \in U$ and $v \in \{0, 1\}$, v can be stored in graph G by inserting a new edge $\left(v_a^i, v_b^j\right)$ in E, where $i = h_a(k)$ and $j = h_b(k)$ (as shown by the red or the green edges in Fig. 5.2). The **query function** $f : U \to V$ for the key-value mapping is defined as: $\boldsymbol{Query}(k) = t_a[i] \oplus t_b[j]$, where $t_a[i]$ and $t_b[j]$ represent the entry in the ith and jth bucket of T_a and T_b respectively. If the graph G remains acyclic after inserting all keys in U, then it can be proved that there exists a solution to fill the buckets in T_a and T_b with either "1" or "0", such that for any $k \in U$ and its corresponding value $v \in \{0, 1\}$, $f(k) = v$. However, when a circle is found while building the graph G, the graph should be rebuilt by using different hash functions $h_a(x)$ and $h_b(x)$. Given all key-value pairs, Othello first finds the two valid hash functions $h_a(x)$ and $h_b(x)$ that do not create any circle in the graph, and then uses the depth-first-search (DFS) order of the resulting acyclic to insert all keys.

$\boldsymbol{Construct}(P, N)$: Let P and N are the two sets used to construct an Othello table. Suppose list $(e_1, e_2, ..., e_m)$ be the edge set E sorted in its DFS order. Then for any edge e in the sorted list, we find the corresponding key k which is represented by e, i.e., the indexes i, j of the two vertices are $h_a(k)$ and $h_b(k)$ respectively. Let v be the mapping value of k, namely, $v = 1$ if $k \in P$ (as shown by the green edge in Fig. 5.2), and $v = 0$ if $k \in N$ (as shown by the red edge in Fig. 5.2). Then v can be inserted to the table by the following steps. If both $t_a[i]$ and $t_b[j]$ are empty, we set $t_a[i] = 0$ and $t_b[j] = v$. Otherwise, one bucket of $t_a[i]$ and $t_b[j]$ must be empty since G is acylic and

e is visited according to the DFS order of E. In this case, we set the empty bucket to be the "xor" result of the value in the other non-empty bucket and v.

It can be proved that if $m_a \geq 1.33n$ and $m_b \geq n$ (n is the number of all keys), the memory is sufficient for finding the appropriate hash function pairs that avoid cycles for the inserting all keys with small rebuilding probability [85]. In addition, with this memory setting, Othello can also support value flipping (change the value of a key k from "0" to "1" or from "1" to "0") ***Flip***(k), deletion ***Delete***(k) and, insertion ***Insert***(k, v) functions using $O(1)$ time [85].

Insert(k, v): Let $G = (V_a, V_b, E)$ be the maintained graph in Othello and t_a, t_b are the hash table arrays. Inserting a key-value pair (k, v) into Othello is equivalent to adding an edge e in G, where $e = (V_a(h_a(k)), V_b(h_b(k)))$, and h_a and h_b are the selected hash functions that map the key k to the graph vertices. If the resulting graph $G = (V_a, V_b, E + \{e\})$ creates a cycle, showing the table is too full to insert the key, then the Othello hash table should be rebuilt by selecting a new pair of hash functions h_a and h_b. Otherwise, the insertion is successful and we need to assign v as the value of this edge e. If the value ("0" or "1") of e equals to $t_a[h_a(k)] \oplus t_b[h_b(k)]$, then the insertion is done. Otherwise, we need to flip the value of e by tweaking the values of vertices stored in t_a and t_b, namely conduct the value flipping operation (see below).

Flip(k): Let T be the tree that contains the edge e whose value should be flipped. Assume T is separated into two sub-trees T_1 and T_2 by e. The method to change the value flag of e is to flip all values stored in the vertices of either T_1 or T_2 (whichever is smaller).

Delete(k): Deletion of a stored key k from the othello table can be accom-

plished by removing the corresponding edge $e = (V_a(h_a(k)), V_b(h_b(k)))$ from G. After deletion, the actual hashing tables T_a and T_b are not changed. Thus, the ***Delete***(k) function is only a logical deletion process: it will not change the inference behavior of Othello. Instead, it only remove redundant edges to provide space for future new keys.

5.3.2 DASS Design for Optimized Memory Cost.

DASS data structure

The idea of DASS for efficient and accurate CR checking is simple: we concatenate a filter tool (a Cuckoo filter) with a set query tool (a Othello hash table). Practical measurements show that the revoked certificates only contribute to 1% of all certificates [41, 73], hence $|N| \gg |P|$. DASS is particularly designed based on this fact.

Recall P is the set of revoked certificates and N is the set of legitimate certificates, and $|N| \gg |P|$. We construct DASS as shown in Fig. 5.3. DASS has two levels. The first level is a Cuckoo filter [19] with P being the membership set. Specifically, the Cuckoo filter F inserts the fingerprints of all certificates in P (Step 1). In Step 2, we test set N against the filter F. Most certificates of N will be tested 'negative' and they are all true negatives (set TN). However a few certificates of N are tested 'positive' due to the fundamental limitation of a filter, and they are false positives (set FP). In Step 3, we construct an Othello data structure O for binary set classification and use FP as set 0 and P as set 1. Note both FP and P are very small sets compared to N, hence DASS saves the majority memory cost.

The query of DASS about a certificate k is executed as shown in Fig. 5.4. In Step 1, k is tested by the filter F. If $F.\texttt{Query}(k) = 0$, we must have $k \in N$ and k is

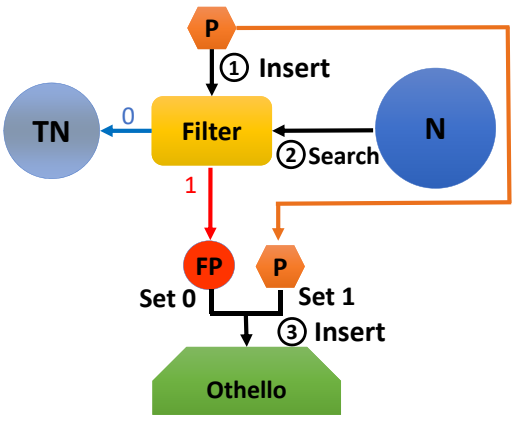


Figure 5.3: DASS Construction

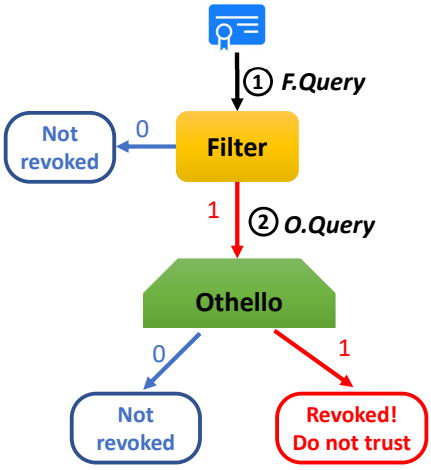


Figure 5.4: DASS Query

legitimate. If $F.\texttt{Query}(k) = 1$ then k is either revoked or false positive. Then it is tested by Othello O. If $O.\texttt{Query}(k) = 0$, k is legitimate. If $O.\texttt{Query}(k) = 1$, k is revoked.

Memory cost analysis of DASS

Despite its simplicity, DASS is rather memory-efficient to memorize the binary values of keys, especially when the sizes of the negative key set and positive key set are highly imbalanced (namely, set ratio $r = |N|/|P|$ is large). Here we show how to optimize DASS so that the total memory cost is minimized for the given two key sets P and N.

In DASS, there exists a trade-off between the sizes of the filter F and that of the Othello O. The false positives will be fewer if F uses more space, and hence O needs less space. Let ε be the false positive rate of the F in the first layer, then the expectation

of the number of false positives of F is $\varepsilon |N|$. Since Othello costs 2.33 bits per key, O needs $2.33 (\varepsilon |N| + |P|)$. Meanwhile, let the memory cost of the first filter layer F be M_f, such that the expected false positive rate of F is no greater than ε. According to the recent implementation of Cuckoo filters [75],F produces a false positive result when the fingerprint of a negative key collides with at least one stored fingerprint in the two candidate buckets, with each bucket containing b entries. Therefore, the upper bound of the probability of a false positive fingerprint collision is $1 - (1 - 1/2^f)^{2b} \approx 2b/2^f$, where f is the number of bits of the fingerprint. Hence, $\varepsilon \geq 2b/2^f$, and we get

$$f \geq \lceil \log_2(2b/\varepsilon) \rceil = \lceil \log_2(1/\varepsilon) + \log_2(b) + 1 \rceil . \tag{5.3}$$

Then, the amortized space for each positive key stored in the filter is f/α, where α is the load factor of the Cuckoo hashing table. Thus, if we use the (2,4)-Cuckoo hashing table in F, and the expected load factor rate of 0.95 to initialize the Cuckoo Filter (which is a common setting for the filters to guarantee the success rate of insertion and efficiency of query), the amortized space for each positive key in F is $(\log_2(1/\varepsilon) + 3)/0.95$ [19, 75]. In addition, the Cuckoo Filter implemented with the semi-sorting trick [19] can further save one bit per fingerprint. Hence, the total cost of F with semi-sorting implementation is $|P| (\log_2(1/\varepsilon) + 2)/0.95$.

Let $r = |N|/|P|$. In total, DASS uses M bits where

$$M = ((\log_2 (1/\varepsilon) + 2) /0.95 + 2.33\varepsilon r + 2.33) |P| \tag{5.4}$$

Since $|P|$ and r are constant, we can minimize the total memory cost by letting $\frac{\partial M}{\partial \varepsilon} = 0$. Hence, M is minimized when $\varepsilon \approx \frac{0.652}{r}$ and $M_{\min} = (1.05 \log_2 r +$

6.604) $|P|$. The result further instructs us to set the fingerprints of the Cuckoo filter to be $\lceil 3.6 + \log_2 r \rceil$ bits according to Eq. 5.3.

Compared to the memory cost of Othello, $2.33(|N| + |P|) = \Theta(r|P|)$, DASS significantly reduces the memory cost to $\Theta(|P| \log r)$. The optimal filter cascade used in CRLite [41] costs $|P|(1.44 \log_2 r + 4.2)$ bits, which is similar to DASS. But CRLite is a static structure, while DASS can support *in-place* incremental updates, as will be shown in the next section.

Chapter 6

TinyCR: A On-Device Certificate Revocation Checking Protocol for IoT

6.1 Overview of TinyCR.

6.1.1 System Model

Secure communication in an IoT network requires that devices can easily and automatically authenticate each other, which is nowadays achieved by digital certificates based on the Public Key Infrastructure (PKI). The most expensive step of the PKI-based authentication is verifying the revocation status of a certificate. TinyCR enables on-device efficient certificate revocation (CR) checking with 100% query accuracy through the novel binary set query data structure DASS. Fig. 6.1 illustrates the

system model of TinyCR. The CR checking protocol is designed on top of the current IoT/Mobile device management system (MDM) [2], where an IoT device management (IDM) server requests the certificates from CAs for users and delivers the certificates (usually through a patch file for installation) to the end devices when the devices are registered to the service. Note that the CA could be the world-wide ceritificate issuers or the PKI service that is managed by the service provider (such as Symantec Managed PKI Service [2]). The CA issues new certificates at the request of IDM and actively sends updated CRLs to the IDM server when a new revoke happens (Operation 1). The IDM server constructs and updates DASS based on the global certificate database and the newest CRLs (Operation 2). For each device, the IDM server will install DASS on it (Operation 3) when the device is enrolled to the service. The DASS installation process could be conducted together with the certificate installation on the device. Whenever the CRLs have changed, the IDM server would send update messages if necessary in a pushing way (Operation 4). In the system, each device can check the CR status of *any certificate* completely based on its local DASS copy and perform real-time updates to DASS (Operation 5). The model fits or can be easily extended to most IoT management systems.

We define two types of latencies in the whole process: 1) **synchronization latency** is defined as from the time of the CA revoking a certificate to that of a device being able to find this revocation event from its local state; 2) **query latency** is defined as the time used to get the CR status on a device. While query latency can be effectively optimized using existing on-device checking methods [23, 40, 41, 67], synchronization latency is usually ignored. However, synchronization latency is also crucial

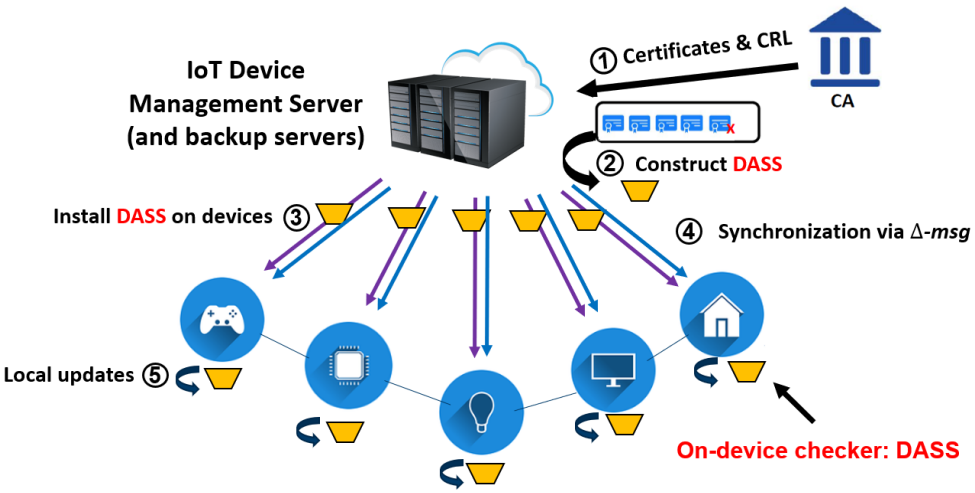


Figure 6.1: System model of TinyCR

for the security of the whole system. For example, an attacker can easily utilize the synchronization latency window to perform attacks with the revoked certificate. Hence, reducing the synchronization latency can effectively minimize or eliminate such security risk. Therefore, in this work, TinyCR aims to minimize both two latencies. There is another type: revoking latency, defined as from the time a certificate being hacked to that of the CA revoking the certificate, which is out of scope of this work.

6.1.2 Threat Model

Since the certificates issued by the CA might be revoked, an attacker can effectively abuse the revoked certificates. The security vulnerability in this process is apparent: the revoked certificate are still valid if a device only verifies the expiration dates and CA signatures (called time- and signature-valid). Hence the on-device maintenance of all revoked certificates is necessary. We are mainly concerned about the attacker who can obtain a set of time- and signature- valid but revoked certificates and

the corresponding private keys, such that the attacker can masquerade as legitimate users in the IoT to perform Man-in-the-Middle (MITM) attacks during TLS setups or tamper with the sensing data.

We summarize the threat model and assumptions in this paper:

1. The IDM server and the CAs are trusted and they communicate via a secure channel with integrity. Each device also maintains a channel from/to the IDM server with integrity.

2. The attacker can acquire a set of time- and signature-valid certificates. But this behavior could be detected by the CA and those certificates are revoked.

3. The attacker can obtain all information of the shared DASS, but is not able to tamper it.

4. The size of the certificate universe in IoT is large. Note that the current number of web server certificates is on a scale of 100 million [41, 67]. It is a reasonable estimate that the future IoT devices should be much more than the number of web servers.

5. The number of revoked certificates is smaller than that of legitimate ones in an IoT network by at least an order of magnitude. Otherwise, the CA who issues many revoked certificates will not be trusted. This assumption is validated by measurements [73].

6. IoT devices have limited memory and computing resources, while the IDM server and attackers can be powerful. The IDM server knows all time- and signature-valid certificates.

7. We do not consider deny-of-service attacks.

6.2 TinyCR Design.

We present the detailed design considerations of TinyCR. The TinyCR system contains two programs: the *tracker* running on the IDM server and the *verifier* running on the devices. The tracker is responsible for receiving new certificates and revocations from the CAs, constructing DASS, and sending the DASS update messages to devices. The verifier is the compact DASS data structure running on the IoT devices to support CR checking. This section discusses how the tracker and verifier should execute and communicate.

6.2.1 Updates of Cuckoo Filter and Othello

As introduced in 5.3.1, Cuckoo filter supports key addition and deletion by calling $F.\texttt{Insert}(k)$ and $F.\texttt{Delete}(k)$ respectively. Othello supports key addition, deletion, and value flipping. Adding a key k to set 1 is by calling $O.\texttt{Insert}(k, 1)$, indicating the value of k is 1. Adding a key k to set 0 is by calling $O.\texttt{Insert}(k, 0)$. Deletion and value flipping is by $O.\texttt{Delete}(k)$ and $O.\texttt{Flip}(k)$. All these functions cost constant time on average [19, 85].

However, it is important to note that insertion and deletion of keys in Cuckoo Filters would impact the distribution of the potential false positive keys in the whole key space. Specifically, inserting a new fingerprint into the Cuckoo hash table would create a set of new potential false positive keys that match the fingerprint stored in the corresponding bucket. Similarly, deleting a fingerprint from the table would eliminate a fraction of potential false positive keys. Thus, updating DASS is non-trivial, since this

issue may cause incorrect inference results if not properly addressed, leading to security holes or accessibility issues of the IoT service. For simplicity of the design description, we temporarily ignore this issue in Sec. 6.2.2. We then look back and explain the solution to address this issue in Sec. 6.2.3.

6.2.2 Updating DASS on the Tracker

On-device DASS needs to be updated when 1) a new certificate is issued by CAs, 2) a certificate is revoked by CAs, 3) a certificate is expired, or 4) in rare cases CA un-revokes a revoked certificate. All these situations can be addressed by the following three update functions on the tracker.

- ***Insertion***: adding a certificate to N or P (very rare cases).
- ***Value Flipping***: moving a certificate from P to N (very rare cases) or from N to P.
- ***Deletion***: removing a certificate from P or N.

For each update, the tracker will compute the *delta message*, including only the bit positions that need to change for on-device DASS. Using the delta message instead of the complete DASS significantly saves bandwidth cost.

Insertion

When a device joins the network with a new certificate, this information should be immediately reflected in DASS. Otherwise other devices may reject this certificate if DASS returns 1. In rare cases, the CA may also revoke a certificate before it is actually

installed on any device. Let k be the new certificate. If k is added to the positive set P, according to the design, k should first be inserted to the filter F and then inserted to the Othello O in the second layer with its corresponding value $O.\texttt{Query}(k') == 1$. On the contrary, if k is inserted to the negative set N, we can check whether F tests it as positive. If $F.\texttt{Query}(k) == 0$, then the original DASS classifies k correctly and no updating is required. Otherwise, k' is a false positive and should be inserted to O with $O.\texttt{query}(k) == 0$. Both $F.\texttt{Insert}(k)$ and $O.\texttt{Insert}(k, v)$ take $O(1)$ time to complete in average.

Value Flipping

When a valid certificate is revoked by the CA if, for example, the device is compromised by an attacker, the revocation status of this key should be updated from 0 to 1 in DASS. In another case, the CA may also want to un-revoke a revoked certificate, implying the revocation status should be updated from 1 to 0. In both cases, all devices in the network should be noticed with the updating information to avoid abuse of the revoked certificates or mistakenly rejecting a legitimate one. Suppose a key k is moved from N to P. The tracker first checks whether k is considered as a (false) positive key by the filter, then inserts k to the filter F. If k is a false positive, k has already been stored in the second layer O. In this case, the tracker needs to execute $O.\texttt{Flip}(k)$ to change the stored value of k. Otherwise, the tracker inserts k to O with corresponding value 1 by $O.\texttt{Insert}(k, 1)$.

In another value flipping case, k is moved from P to N. In such case, k should have been already inserted in both F and O. Therefore, to update the DASS, the first

layer filter F first removes k's fingerprint from its cuckoo hashing table and then check whether k would be recognized as a false positive key after removal. If k is not a false positive, it should be deleted from O. Otherwise, O flips the value of k using $O.\texttt{Flip}(k)$.

Deletion

Certificates may expire. Although the removal of these certificates from DASS is not necessary – the expired certificates are rejected in early steps – it helps to maintain the DASS compact. DASS has to be rebuilt when it is too full to insert new certificates, which would cost considerable computation resources and network bandwidth. Hence, removing expired certificates can avoid unnecessary rebuilds. Let k be the key that should be removed from either P or N. If $k \in P$, both of the two layers need to remove k by calling their delete functions. Otherwise if $k \in N$, we need check whether k is a false positive for the first layer F. If it is, then the second layer O needs to delete it. Otherwise, neither F nor O store the information k, thus no operation is required.

6.2.3 Handling Inconsistency of Updating

The above updating algorithms assume the false positive universe of the first layer filter for the given certificate set remains stable. However, after inserting or deleting a key from the filter, the assumption may no longer hold, because the fingerprint added to or removed from the cuckoo filter would change the distribution of the potential false positive keys. If a former TN (true negative) key becomes a FP (false positive) key after an insertion, the key should be recorded in the second layer O, such that the key can be correctly queried. Similarly, if a former FP key becomes a TN key after

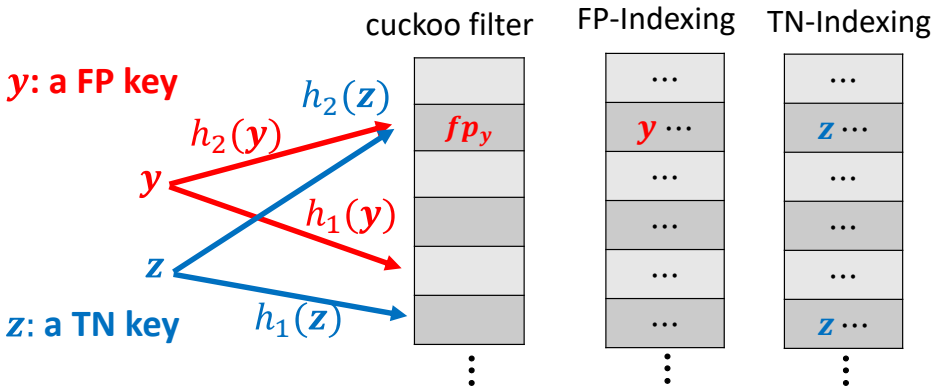


Figure 6.2: TN-indexing table and FP-indexing table

a deletion, then the key should be removed from O. Although the correction process is simple, finding these impacted keys from the negative key set is expensive. A naive solution is thoroughly checking the negative key set with the updated cuckoo filter to find the influenced keys. However, this solution is time-consuming as the negative key set is big, causing $O(|N|)$ rather than $O(1)$ updating cost in the worst case.

In TinyCR tracker, we propose to solve the problem by using two additional indexing tables that have similar number of buckets as the cuckoo filter to index the sets of the potentially influenced keys for every fingerprint in the cuckoo filter. Specifically, at the construction time of DASS, when we iterate through the entire negative set N to find the FP sets by querying F, we insert the TN keys into the "TN-indexing" hash table and FP keys into the "FP-indexing" hash table at the exact two bucket positions that are queried in F to lookup the fingerprint (as shown in Fig. 6.2). Therefore, when a fingerprint is inserted into a particular bucket in F at the updating time, only the TN keys stored at the same bucket positions of the TN-indexing table would be potentially influenced by the insertion. Hence, only these TN keys need to be queried with F again

to check whether they become FP keys after the insertion. Then those new FP keys are inserted to the O in the second Othello layer. Similarly, when a fingerprint is deleted, only the FP keys at the corresponding buckets in the FP-indexing table need to be checked again. Then the keys that become TN keys after the deletion are removed from O.

Since $|N| = r|P|$ and the number of buckets in F is $O(|P|)$, the amortized length of each bucket in FP-indexing and NP-indexing is $O(r)$. Thus, the updating cost decrease from $O(|N|)$ to $O(r)$ in worst case with this indexing strategy. Meanwhile, the total size of the indexing tables is $O(|N|)$. **Since these tables are maintained by the server and not related to the devices, the cost is affordable.** By properly handling the inconsistency issues, the tracker is able to create a perfect DASS that yields ***zero*** query error.

6.2.4 Updates on Devices

Though the TinyCR tracker requires $O(|N| + |P|)$ extra space to maintain the certificates, each on-device verifier requires much less memory and computational resources to support updating. In the verifier, only the cuckoo filter and Othello are stored in memory, costing approximately $(1.05 \log_2 r + 6.604)\,|P|$ bits. The inference of DASS in verifier can be simply accomplished by at most four hashing and memory read operations. In addition, the DASS verifier can also be synchronized with *delta messages*. When an update is necessary, the tracker sends a delta message patch to all devices. The delta message includes the certificate digest and the indexes of the bits that need to be changed in O's hash tables, and is small in size (9 to 150 bytes on

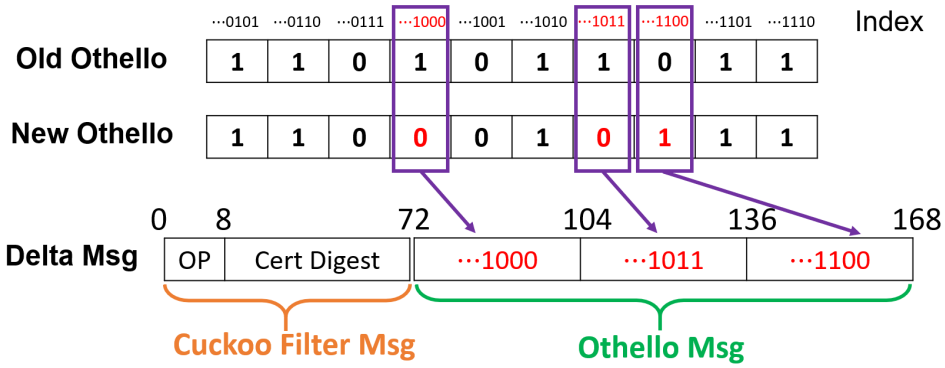


Figure 6.3: Structure of a delta message

average for 100 million certificates). Note that the indexes of the flipped bits in O are tracked as an intermediate result while updating Othello. Thus, there is no extra cost to compute the indexes after the update is done. Our experiment also shows the raw delta message does not scale with the size of the certificate sets. Then the tracker **signs the delta message** and attach the signature to the updating patch data to guarantee the integrity. This updating strategy differs from other CR checking synchronization methods that use static data structures, such as CRLite [41], which needs to rebuild the entire data structure for every update (if correctness of verifier is obligatory at any time) and sends it to all clients. The raw delta message of CRLite is much larger than that of DASS.

In our design, the raw delta-msg is encoded as Fig 6.3. Specifically, the updating instruction for F uses only 9 bytes, including 1 byte for the operation type (insert, delete or do nothing) and 8 bytes for the 64-bits digest of the certificate. Then the F in the verifier DASS can insert or delete the certificate through the corresponding operations of the local Cuckoo filter. Meanwhile, the updating instruction for the O is a

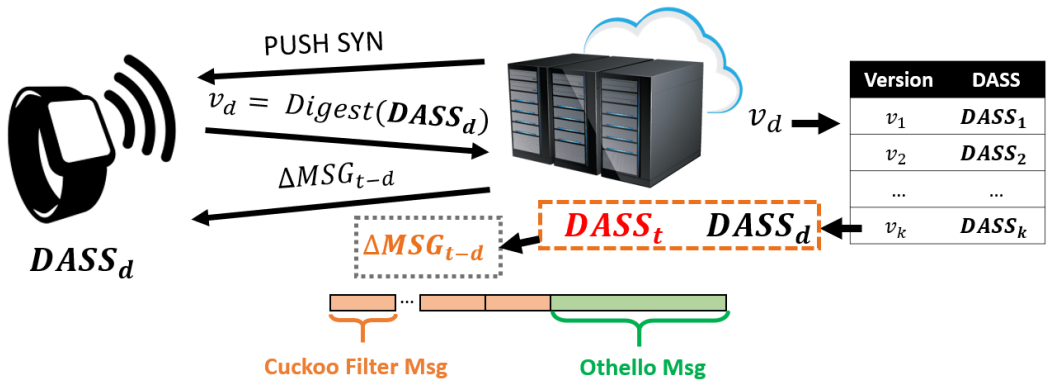


Figure 6.4: Multi-way version control protocol.

list of 32-bit integers representing the bucket positions at which the stored value should be flipped. For every position index pos, if $pos \leq |T_a|$, we flip the entry at bucket pos of T_a; otherwise, we flip the entry at bucket $pos - |T_b|$ of T_b, where T_a and T_b are the two maintained hash tables in Othello [85]. In our evaluation, we will show on average only a small number of buckets in O (if any) need to be flipped.

6.2.5 DASS Version Control

Since TinyCR uses delta messages to update the on-device checker, the new state of DASS relies on the previous state. Thus, DASS correctness may suffer from network failures or packet loss when sending delta messages. To solve this problem, we introduce a DASS version control protocol as an optional design choice (as shown in Fig. 6.4). In Fig. 6.4, the IDM server initiate a *PUSH-SYN* packet when a new tracker $DASS_t$ is generated. Then the device sends back the digest v_d of its local verifier $DASS_d$. Meanwhile, the IDM server maintains a mapping table to keep track of a history of t recent verifier DASS version IDs and the corresponding delta-msg increments. According to our evaluation in Sec. 6.3.4, the average delta-msg increment

size is fewer than 100 bytes. Then the IDM server simply retrieves all the missed delta-message increments and concatenates them to generate the cross-version delta message ΔMSG_{t-d} that denotes the differences between $DASS_d$ and $DASS_t$. In the ΔMSG_{t-d} that skips over multiple versions, we could include multiple Cuckoo Filter Msg fields and one single Othello Msg field using the similar encoding format as shown by Fig. 6.3. If v_d is not maintained by the version table, that means the device has missed a large amount of updates. Then the server directly send the $DASS_t$ instead of the delta message to the device. Optionally, the device returns an ACK when the local DASS updating is accomplished.

If the updating frequency of certificate sets is too high in some scenarios, it is not practical for the IDM server to send a signed delta message after each update and track every DASS version. In such case, we can use the version control design to batch the updates with a bounded time granularity. For example, the IDM server can only send one single aggregated delta message in per-hour, and maintain only 24 delta message increment versions in each day.

6.3 Implementation and Evaluation

6.3.1 System Implementation

We implement the TinyCR tracker on a Google Cloud VM instance with 64 vCPUs and 624 GB memory using C++. The on-device DASS verifier is implemented on a Raspberry Pi 3 with one single 1.4 GHz processor and 1 GB RAM. Note the device used in the experiments is just an example of **a wide spectrum of devices that can**

use TinyCR. TinyCR can be easily deployed on more powerful devices like mobile phones and less powerful IoT devices. In addition to TinyCR, we also implemented the CRLite filter cascades [41] and Othello hashing [85] data structures with similar synchronization settings as the TinyCR protocol for performance comparison. The parameters for CRLite and Othello are set according to the authors' suggestions [41,85]. Both TinyCR and Othello can support dynamic updating of the revocation checking list, while CRLite has to be rebuilt for most updates.

6.3.2 Metrics and Dataset

We evaluate the CR methods by the following metrics:

- On-device memory cost: the overall memory cost of the data structures on a device.
- Update time and synchronization latency: the time for each update on the IDM server.
- Bandwidth: the message cost caused by updates.
- Query cost: the delay to get a CR checking result.

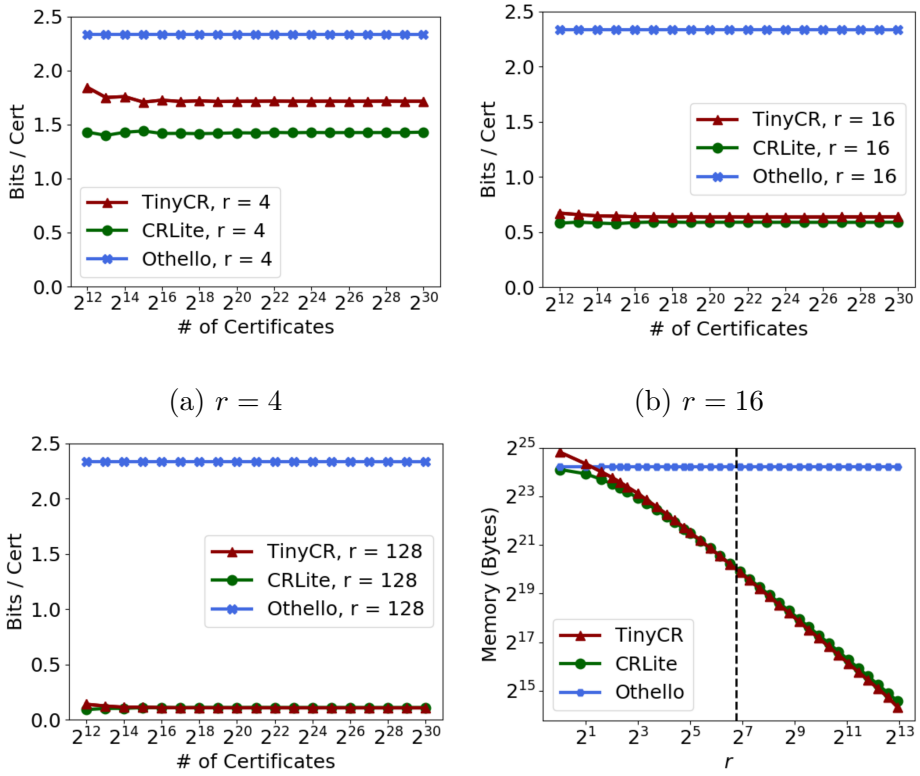
We use both real and synthetic certificate datasets for the evaluation. Since there is no IoT certificate dataset available, we use the Censys web certificate dataset [3, 41] to evaluate how those protocols perform in real-world CR verification scenarios. We downloaded 30 millions items of historical NSS trusted certificates over 3 months from Censys using Google BigQuery. After removing the duplicated certificates, there are totally 28,593,752 items in the dataset. Then we use the CRLs or OCSP to obtain

the revocation status of all downloaded certificates. Among the 28.6 million certificates, 274,926 were revoked, i.e., the ratio between the legitimate and revoked certificates is 103 : 1. To evaluate the scalability, we create synthetic datasets containing up to 1 billion certificates with different revocation ratios.

6.3.3 Memory Cost

We construct the on-device data structures of TinyCR (DASS), CRLite (filter cascade), and Othello respectively using the entire Censys certificate data. We find TinyCR, CRLite, and Othello requires 430 KB, 439 KB, and 8,328 KB memory respectively to maintaining the CR status of the 28.6 million certificates.

Then we conduct experiments on the synthetic dataset to investigate how the memory sizes scale with the sizes and the distribution of the keys. In Fig. 6.5, we show the amortized memory cost (i.e. bits per certificate) with respect to the total size $|N| + |P|$ of the certificates by setting $r = |N|/|P|$ as 4 (Fig. 6.5 a), 16 (Fig. 6.5 b) and 128 (Fig. 6.5(c)) respectively. Meanwhile, in Fig. 6.5(d), we present the total memory cost (in bytes) for storing the revocation status of 2^{26} certificates, by varying the ratio r. The vertical dash line in Fig. 6.5(d) represents the ratio r of the Censys dataset in real-world scenario. Fig. 6.5 shows that the memory cost per certificate of all three data structures keeps stable when r is fixed. For example, the amortized memory sizes for TinyCR, CRLite, and Othello are around 0.108 bits, 0.111 bits, and 2.333 bits per certificate respectively for arbitrarily large key sets when $r = 128$. The amortized memory for Othello is independent with r (it is controlled by a hyper-parameter and is set as 2.33 bits), whereas both TinyCR and CRLite use much less memory as r grows.



(a) $r = 4$ (b) $r = 16$

(c) $r = 128$ (d) $|P| + |N| = 2^{26}$

Figure 6.5: (a) to (c): Amortized memory cost when $r = |N|/|P|$ is 4, 16, 128 respectively. (d): Memory cost for 2^{26} keys with respect to r.

It can also be seen from the graph that both TinyCR and CRLite use less than 1 MB to store the around 64 million certificates when $r = 100$ (which is close to the ratio for real-world CR lists) and use less than 8 MB when $r = 10$, while Othello always requires around 20 MB.

6.3.4 Updating Efficiency

In this section, we evaluate the synchronization overhead of the data structures regarding any change of the global CRL. Specifically, we utilize the Censys certificates and synthetic data sets to simulate the following updating scenarios.

Short-term insertion/value flipping: We use a certificate dataset to initialize the CR checking data structures in a static way, then evaluate the latency of the inserting/value flipping on the initial data structures without reconstructing the data structures (except for filter cascades).

Long-term insertion: We use 100 million certificates to initialize the data structures, then insert another 100 million certificates item by item to them. In the simulation, we assume the revocation ratio of the initial and the inserted certificate sets are consistent.

Long-term value flipping: We use 100 million certificates to initialize TinyCR. Then we randomly sample $|P|$ validate certificates and revoke those certificates, where $|P|$ is the number of revoked certificates in the initial set. We simulate the scenario where the number of revoked certificates is doubled during the usage period before expiration. Note that the revocation of the sampled set is a gradual process, i.e., one certificate is revoked each time. The maintained value of a newly revoked certificate

# of Certs	Method	Mem	Add P	Add N	$P \to N$	$N \to P$
Censys 28.6M	CRLite	458 KB	3.2 s	3.2 s	3.2 s	3.2 s
	Othello	8.3 MB	11.4 μs	9.9 μs	10.1 μs	9.2 μs
	TinyCR	448 KB	349.9 μs	1.6 μs	27.0 μs	345.3 μs
10M	CRLite	172 KB	1.0 s	1.0 s	1.0 s	1.0 s
	Othello	2.9 MB	4.6 μs	5.0 μs	4.6 μs	4.4 μs
	TinyCR	169 KB	280.9 μs	1.2 μs	16.6 μs	289.9 μs
100M	CRLite	1.7 MB	10.1 s	10.1 s	10.1 s	10.1 s
	Othello	29.2 MB	8.5 μs	7.5 μs	7.1 μs	7.0 μs
	TinyCR	1.7 MB	304.9 μs	1.6 μs	21.6 μs	311.5 μs
1B	CRLite	17.2 MB	153.9 s	153.9 s	153.9 s	153.9 s
	Othello	291.7 MB	10.0 μs	10.2 μs	8.2 μs	7.0 μs
	TinyCR	16.9 MB	296.0 μs	2.7 μs	27.3 μs	319.5 μs

Table 6.1: On-device memory cost and average updating latency on the tracker for different set sizes. The revocation ratio for synthetic data is 1%.

should be changed from 0 to 1.

Overhead on the IDM server (tracker).

The tracker on the IDM server is required to react quickly for every update (insertion and value flipping) of the CRLs. In Table 6.1, we show the on-device memory cost and the average computational latency of the tracker to update the data summaries and generate the delta message in short-term updating scenarios. Specifically, we simu-

late the scenarios with the Censys dataset and the synthetic data sets of different sizes to evaluate the scalability of the methods in an IoT network with up to billions of devices. In our synthetic data, we set the certificate revocation ratio to be 1%, which is close to the ratio of the Censys dataset. We discuss the insertion of revoked certificates and legitimate certificates (the more common case) separately in the fourth and fifth columns, as they will cause different updating overhead based on the algorithms. Similarly, we also evaluate the value flipping case where a revoked certificate is moved to the legitimate list, and the case where an legitimate one is moved to the revoked list (the more common case) in sixth and seventh columns respectively.

Table 6.1 shows that the updating time of CRLite significantly increases with the size of the sets. As a static data structure, filter cascade has to be reconstructed using the entire certificate sets for any updates, which would cause tremendous overhead to the server and large bandwidth overhead. Meanwhile, the long latency of updating can also cause memory concurrency issues for the tracker when the updating pace is high. Therefore, in practice, CRLite is only updated in a batching way, for example, the tracker and verifier are recommended to update once every day [41]. Consequently, this strategy would introduce a synchronization latency of one day – a big security vulnerability. On the other hand, the update latency of TinyCR and Othello is significantly lower than CRLite and scales much better with the size of certificate sets. Overall, Othello achieves the highest updating throughput for most cases, at the cost of around 16x more memory than TinyCR and CRLite. We also notice TinyCR is most computational-efficient for inserting legitimate certificates to the CR status list, which is the most common type of updating. Even in its worst case, the corresponding up-

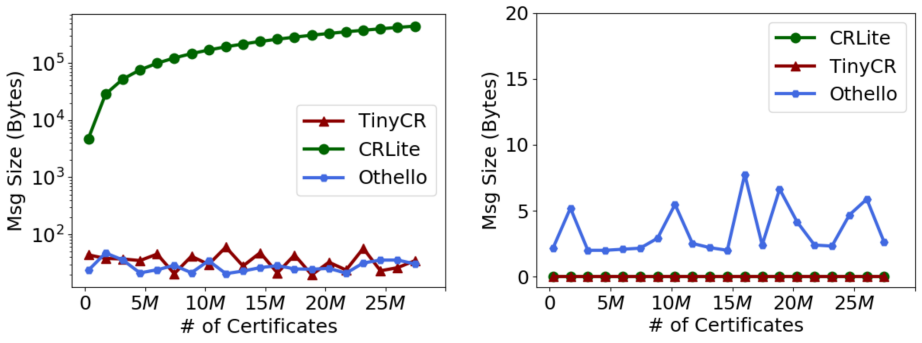
dating latency is smaller than 1 millisecond for up to 1 billion keys, which is usually overwhelmed by the network latency in practice, showing TinyCR can sufficiently support the real-time synchronization with neglectable extra processing overhead. Thus, TinyCR is a more efficient and secure choice for the IoT CR verification task where the certificate universe is large. The theoretical synchronization latency of TinyCR could be just the update time plus network latency in a real-time updating manner.

Due to the connection maintenance and signing cost in practice, real-time updating is not always practical when the updating frequency is too high. The recommended practical deployment settings and analysis are presented in Sec. 6.3.6.

Delta Message Size

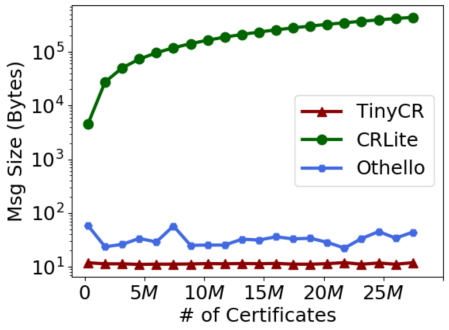
The IDM server of TinyCR requires to send updating messages to all devices, so that the devices can update their own CR status classifier locally. Therefore, the delta message size is a critical metric, as a large message size would significantly increase the network traffic overhead and transmission latency.

In Fig. 6.6, we show the average raw delta message size for each type of updating operations of TinyCR, Othello and CRLite in the short-term updating scenarios using Censys certificate data. Note that in short-term updating scenarios, we conduct limited numbers of updates such that the data structures (except CRLite) are not reconstructed. For inserting legitimate certificates, TinyCR and CRLite usually do not need to be updated as the certificate key is highly likely to be rejected by the first filter layer. For other cases, we notice the delta message sizes of TinyCR and Othello do not scale with the growth of key sizes for all types of the insertion and value flipping

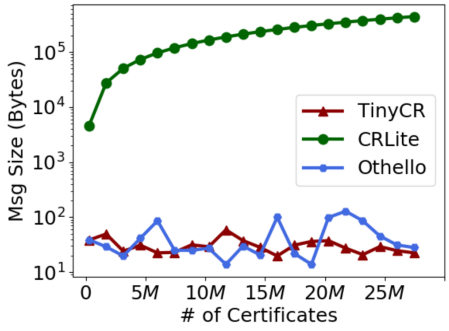


(a) Insert (P)

(b) Insert (N)



(c) Flip ($P \rightarrow N$)



(d) Flip ($N \rightarrow P$)

Figure 6.6: Δ-msg size: (a) Insert a revoked certificate. (b) Insert a legitimate certificate. (c) Unrevoke a revoked certificate. (d) Revoked a certificate.

operations. Specifically, both TinyCR and Othello requires around 0 to 100 bytes of the delta message for all different types of updates (though Othello requires 16x more total memory), whereas CRLite requires to push a significantly larger message to all IoT devices. In addition, for the most common certificate insertion operation shown in Fig. 6.6 (b), TinyCR do not need to send any delta message to devices for most of the insertions (the average delta message size is around 0.1 bytes), whereas Othello has to synchronize a delta message for most of the cases.

In Fig. 6.7, we show the distribution of the raw delta message size (without the signature) in long-term insertion and value flipping scenarios. In these scenarios, when DASS is too full to support the desired update, it has to be reconstructed In the figure, the top of each bar in the figure represents the $90th$, $99th$, $99.9th$ percentile of the delta messages sizes. For the long-term insertion scenario in Fig. 6.7 (a), the result shows more than 90% and 99% of the delta messages are equal to 0 bytes when the ratios of the legitimate and revoked certificate sizes ($|N|/|P|$) are 100 and 1000 respectively. Namely, for most insertions, the verifier DASS do not need to be updated.

In some rare cases, TinyCR can no longer accommodate a space for the new key. Then the DASS need to be reconstructed on the server and then be pushed and reinstalled on the IoT devices. Therefore, a reconstruction of the data structure would cost much higher overhead on both devices' computing resources and network bandwidth. In the experiments, we notice the total times of DASS reconstruction are 44, 31, 28 respectively to insert the 100 million new certificates, when $|N|/|P|$ equals 10, 100 and 1000. On average*, the bandwidth costs of raw delta messages (not including the

*the cost of reconstruction is amortized to every insertion

signatures) for each insertion are only 12.2, 1.25 and 0.13 bytes when $|N|/|P|$ equals 10, 100 and 1000.

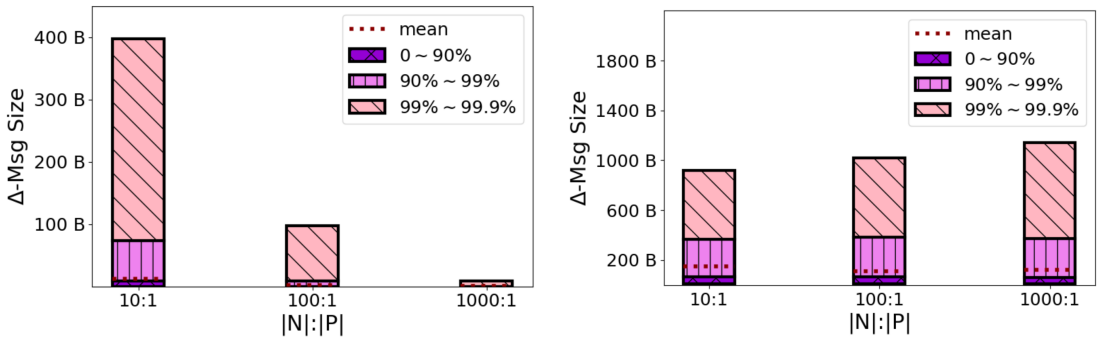
The long-term value flipping result in Fig. 6.7 (b) shows that revoking an existing certificate costs more bandwidth in TinyCR compared with the insertions. Specifically, most revocation events will trigger an updating of the verifier DASS and more than 90% of the updates need a delta message smaller than 65 bytes for all three scenarios with different revocation ratios. In addition, less than 1% revocations will cost more than 385 bytes and less than 0.1% revocations (including reconstruction cases) will cost more than 1 KB for the delta messages. In total, DASS is reconstructed for 64, 31, and 29 times in order to randomly revoke another around 10M, 1M and 0.1M legitimate certificates in the three 100M sets with different initial revocation ratios. The average delta message size[†] for the tree scenarios are 150.58, 108.08 and 119.87 bytes in the three value flipping scenarios.

In summary, TinyCR only needs 0 to 150 bytes on average for any CRL update. Since nearly all current IoT data links (including LANs, LPWANs and Cellular Networks, etc.) can provide larger than 1KBps bandwidth in practice, the TinyCR synchronization process introduces a neglectable extra data transmission cost to the overall network latency.

6.3.5 Query

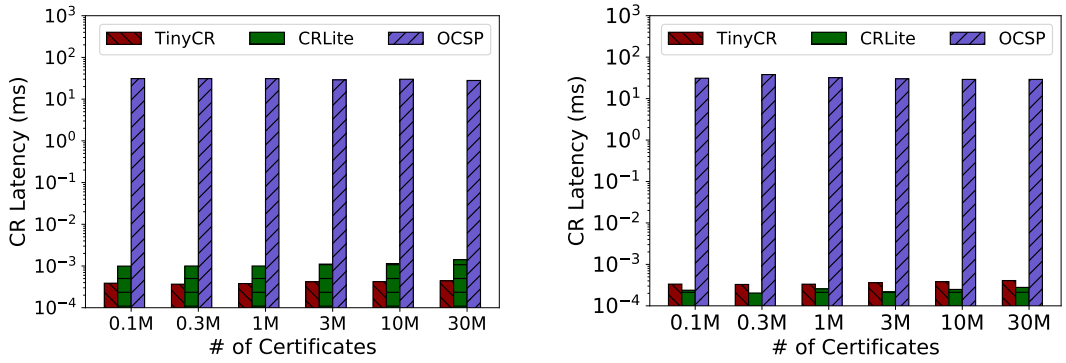
The IoT devices that have installed the DASS verifier would be able to check the CR status of a particular certificate after validating the integrity and expiration

[†]the cost of reconstruction is amortized to every revocation event



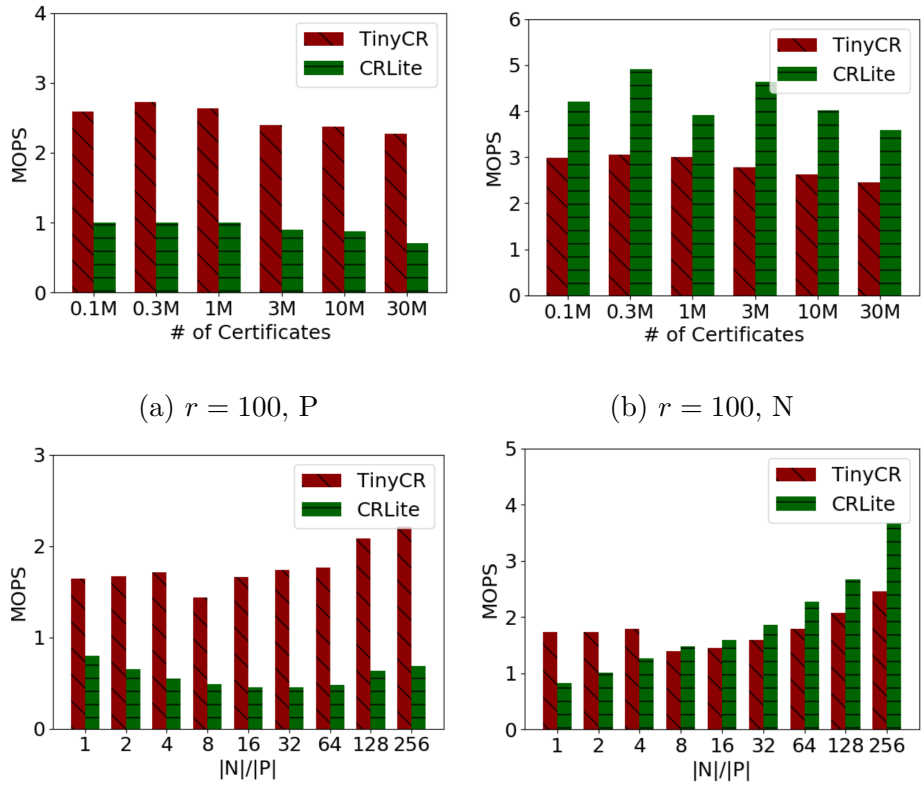
(a) Insert (b) Flip

Figure 6.7: The average, and the 90th, 99th, 99.9th percentiles of the generated *Delta*-Msg sizes for long-term insertion (a) and value flipping (b).



(a) Revoked certificates (b) Legitimate certificates

Figure 6.8: Query latency on Raspberry Pi 3.



(a) $r = 100$, P

(b) $r = 100$, N

(c) $n = 2^{26}$, P

(d) $n = 2^{26}$, N

Figure 6.9: Query throughput on Raspberry Pi 3. (a) Query revoked certificates, $r = 100$. (b) Query legitimate certificates, $r = 100$. (c) Query revoked certificates, $n = 2^{26}$. (d) Query legitimate certificates, $n = 2^{26}$.

date of the certificate. Standard certificate integrity validation requires cryptography computation. Recent works introduce delegated or distributed reference protocols based on the chain of trust [5], which still requires at least millisecond-level latency. Compared with the validation process, the latency for the revocation status checking process using the TinyCR verifiers is neglectable (usually in sub-microseconds).

In Fig. 6.8, we test the average query latency to get a revocation status using the CenSys dataset on the Raspberry Pi 3 testbed and compare the result with CRLite and OCSP. For OCSP, we use a local 8-core CPU server deployed in the local town as the

OCSP server. In addition, on the server side, we use DASS instead of the whole CRL to maintain the CR status. As the on-device CR verifiers, TinyCR and CRLite can verify a CR status in sub-microsecond level, which is a few magnitudes faster than OCSP, as both of them only require $O(1)$ hash operations and memory loads for checking. In particular, the query delay of TinyCR is slightly shorter for the revoked certificates, while the delay of CRLite is slightly shorter for the legitimate certificates. The major query cost for OCSP is the network delays when inquiring the CR status through a remote server. Thus, OCSP is not an ideal method for the scenarios where the device available bandwidth is limited and the latency is sensitive.

In addition, we test the query throughput (measured by millions of operations per second, MOPS) of the on-device checking tools using the CenSys dataset on the Raspberry Pi 3 testbed and present the results in Fig. 6.9. The throughput reflects the performance of the checking tools when checking the CR status in batch for a large certificate set. From Fig. 6.9, the query throughput for TinyCR can be as high as a few millions per second for both revoked and legitimate certificate lookups on IoT devices, which can well support most batched CR checking applications in an efficient way.

6.3.6 Bandwidth vs. Dynamics

In Figs. 6.10 and 6.11, we show the delta message cost (each patch includes a 256-byte RSA signature) for keeping the verifier DASS synchronized under different updating scenarios and settings. Specifically in our experiments, we initialize DASS with 100 million certificates, with 1% revoked keys. Then we test two updating scenarios with different daily workloads: (1) 1 to 10^8 new certificates are added to the certificate

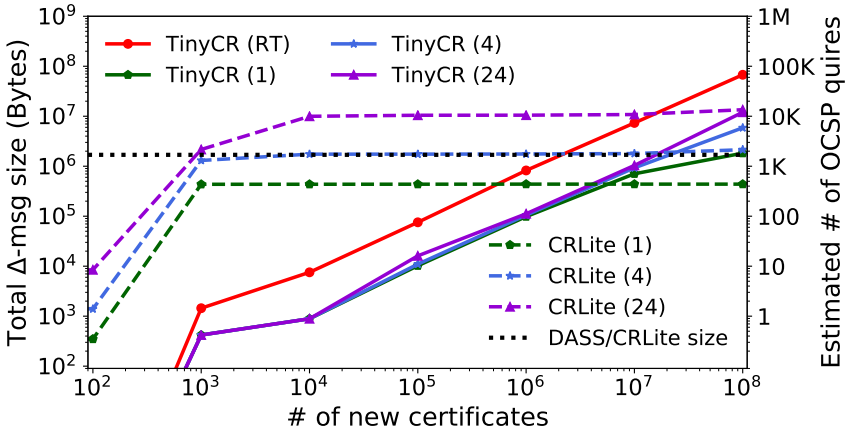


Figure 6.10: Total bandwidth cost for insertion.

universe; (2) 1 to 10^7 existing certificates are revoked. In the experiments, we assume the updates happen uniformly over the day. We deploy four different settings for TinyCR: TinyCR-(RT) sends the delta message immediately after each update happens; TinyCR-(1, 4, 24) means we only maintain 1, 4, 24 versions of TinyCR per day and use batching as in Sec. 6.2.5. Hence, the synchronization latency for TinyCR-(1) is up to one day and for TinyCR-(24) is up to one hour. Similarly, we implement the corresponding versions of CRLite as comparisons. The CRLite is updated for 1, 4, 24 times per day using a bsdiff [55] delta update message. The initial on-device memory costs of TinyCR and CRLite under this setting are both 1.7 MB. We also compare the protocols with OCSP, which has zero update cost on bandwidth and the device side but generates relatively constant traffic load (around 1KB according to prior measurement studies [41, 46]) for each query. Thus, on the y-axis in the right, we show the estimated number of OCSP queries that can be made using around the same amount of traffic load needed by the daily updating of TinyCR and CRLite.

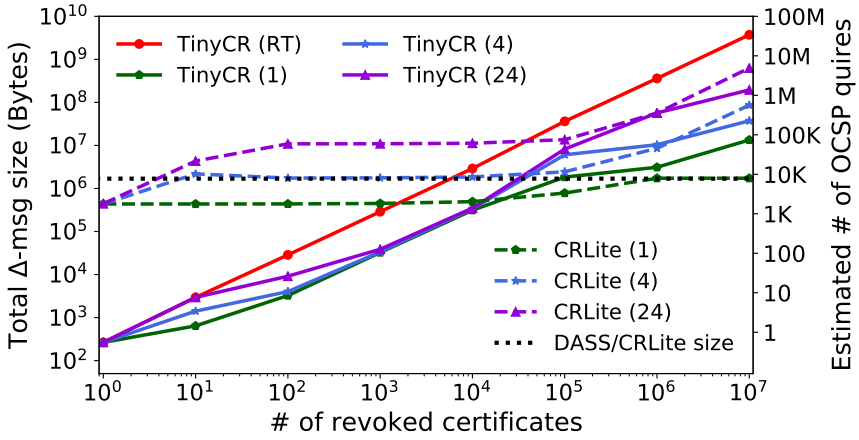


Figure 6.11: Total bandwidth cost for revocation.

From Figs. 6.10 and 6.11, we can clearly observe that TinyCR costs less bandwidth by a few orders of magnitudes compared to CRLite, when the daily updating amount is moderate (for example, less than 1 million inserts or less than 1 thousand revocations per day). On the other hand, when the amount of daily updates is huge, TinyCR has similar total bandwidth cost as CRLite. More specifically, all versions of TinyCR have a similar raw delta message cost if DASS is not reconstructed, while the real-time TinyCR always causes more real-world traffic load due to the high cost of signing the delta messages. When the number of updates is large and DASS has to be reconstructed multiple times, the batching protocol with fewer batches has less bandwidth cost, since at most only one reconstructed and signed data structure needs to be sent in one batch. On the other hand, CRLite always needs a large delta message for synchronization whenever a false positive is found in its first layer of the filter cascades. The total message size of CRLite is in proportional to the updating frequency. When the daily update amount is huge, for example, the certificate universe is doubled or

more than 10,000 certificates are revoked per day, CRLite has a similar performance as batching TinyCR. In particular, with higher batching frequency, TinyCR is more efficient; while with lower frequency, CRLite is a better choice.

In addition, from Figs. 6.10 and 6.11, we find that the TinyCR cost is proportional to the number of updates while the cost of OCSP is proportional to the number of queries. Note that the TinyCR mainly consumes the downlink bandwidth while the OCSP mainly consumes the uplink bandwidth. Thus, it is easy to conclude that TinyCR is more bandwidth cost-efficient when the certificate universe and daily updating amount is small and querying is frequent, while OCSP is more cost-efficient in the opposite scenarios.

6.3.7 Mitigate Rebuilds

When TinyCR has to rebuild, the delta message and server resource cost is significantly higher. Therefore, if the CRL is rather dynamic, we could further optimize DASS to make it less likely to be rebuilt. If the certificate universe is smaller than what the devices can maintain with their memory capability, we could choose to slightly increase the DASS size to reserve spaces for future new certificates and revoked certificates. In particular, the two most important parameters that impact the probability of rebuild is the load factor α of the Cuckoo Filter and the table size coefficient β of Othello. The two parameters are set as $\alpha = 0.95$ and $\beta = 2.33$ recommended by the original studies [19, 85] to optimize memory. Thus, if memory allows, a smaller α and a larger β can be used to reduce the probability of rebuild.

In Fig. 6.12, we show how many updates can be handled by DASS without

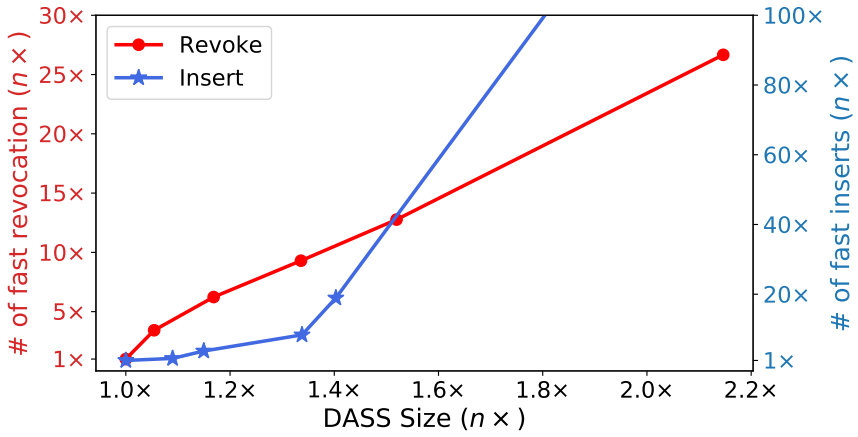


Figure 6.12: How many updates can be applied before the first rebuild.

rebuilding under different memory cost settings. The initialization setting is similar to the setting in Sec. 6.3.4. In the x-axis, $n\times$ means the memory cost of DASS under different settings compared with that of the memory-optimal setting, while in the y-axis, the $n\times$ means how many updates (insertion or revocation) can be processed without rebuilding, compared with that of the memory-optimal setting. For the memory-optimal setting, a rebuild will be triggered after 22 million insertions or 23 thousand revocations in average. From Fig. 6.12, we can find the capability of accommodating the updates can be significantly improved by increasing the memory cost slightly. For example, by using 1.5× memory, DASS can process more than 13× new revocations or 30× new insertions without reconstructions. This memory allocating strategy is rather effective for keeping the $O(1)$ updating cost in real deployment.

6.4 Discussion.

6.4.1 Application Scenarios for TinyCR

Based on evaluation results, TinyCR is ideal and optimal for the application scenarios where 1) users need fast or frequent on-device authentication, and low synchronization latency for security; 2) each user device has a limited size of memory, such as several MBs; 3) the dynamics of certificates are moderate. In addition, for other scenarios, TinyCR can be used as an alternative with proper configurations or as a complementary of other protocols.

Batching v.s. Real-time Updating.

Based on our analysis in Fig. 6.10 and 6.11, the real-time TinyCR updating policy is the optimal choice when CR updates are infrequent. This policy can minimize the synchronization delay and protect devices at any time with limited bandwidth cost. However, due to the high overhead of signing for the delta message, real-time TinyCR yields a high cost when the updating frequency is too high. Thus, we could choose the batching policy for TinyCR and keeping the updating frequency high enough (such as per hour or per 5 minutes) to trade between bandwidth cost and the worst synchronization latency. According to our results using batching, higher but bounded updating frequency does not introduce more bandwidth overhead other than the extra $O(1)$ signing cost. The batching policies are also friendly to the IDM servers if most IoT devices are sporadically connected, as it only needs to maintain a bounded number of DASS versions. In addition, DASS can use a slightly higher memory cost to reduce the reconstruction probability in practice.

When the certificate universe changes significantly every after a short period, TinyCR as well as all other push-based methods will have unacceptable bandwidth cost to keep the synchronization latency low. In such a scenario, we have to sacrifice security for efficiency by reducing the updating frequency, and CRLite is more efficient for one update per day. The on-demand-based methods (such as OCSP) are the other optional choice under this scenario despite its higher verification cost.

Moreover, if the CRL updates are non-uniformly distributed over the data and are predictable by the service providers, we can use a hybrid policy with the batching protocols and real-time protocol. For example, we can batch the updates in the peak hours when updating is frequent, and use the real-time protocol for the rest of the hours when the updating is sporadic.

TinyCR v.s. OCSP/OCSP-stapling.

TinyCR outperforms OCSP in that it is much faster for CR verification. In some IoT scenarios, the verification delay is critical since IoT devices usually have limited data (such as sensing data) to transmitted and the requirement for end-to-end data transmission delay is tight. For example, a smart vehicle is required to read the IoT sensors on streets for decision-making in a short delay while driving fast. In addition, OCSP is not suitable for many IoT applications as it leaks user privacy. This drawback becomes more severe as the IoT data access pattern may include not only the temporal context but also the location information of the user, such as when and where a user reads a static street sensor. Besides, many peer-to-peer communication patterns for IoT usually do not need access to the Internet, for example, IoT devices can be accessed using short-distance communication media, such as WLAN, Bluetooth, and visible light.

Hence, on-device CR checking protocols are more suitable for those scenarios. Still, for the rare cases when a new certificate cannot be verified by an outdated DASS, we can choose to fall back to OSCP.

OCSP-stapling is another practical design for CR checking in IoT scenarios as it does not rely on server access upon verification and can protect user's privacy. The major difference between TinyCR and OCSP-stapling is that TinyCR requires the device who verifies the other device to maintain DASS, while OCSP-stapling requires the device who is under verification to provide the time-stamped OCSP response extension. These OCSP-stapling devices have high bandwidth overhead. Thus, in IoT scenarios, if the device who needs to authenticate the other device has more memory/network resource (for example, a smartphone is required to authenticate a sensor), TinyCR is a better choice as it only requires the inquiring device to maintain an up-to-date DASS. On the contrary, if devices to be authenticated are more powerful (for example, a sensor needs to authenticate a smartphone), then OCSP-stapling can be used. If bi-directional authentications are necessary, we can use a hybrid method of DASS and OCSP-stapling to optimize the resource-security trade-off.

6.4.2 Security Analysis

We discuss the following attacking behaviors for TinyCR.

(1) *The attacker attempts to masquerade as a legitimate IoT client by using a revoked certificate.* Since the synchronization latency of TinyCR is only on the millisecond level plus the network latency, the attacker has very limited time to conduct such attacks. Compared to prior work that synchronize the devices on daily basis [41, 67],

TinyCR significantly reduces the chance of this attack. Note that it is also important for a CA to detect a comprised certificate as early as possible, although this topic is not the focus of this paper.

(2) *The attacker performs the MitM attacks between the IDM server and the IoT devices.* The current methods are sufficient to defend against MitM attacks between the IDM servers and the IoT devices. Each device can get the public keys of the IDM servers and CAs using offline methods during manufacturing or installation. With the public keys, the device can build trusted TLS sessions to IDM servers. Hacking an IDM server or a CA requires much more attacking power than hacking a device. In this paper, we do not consider the scenario where the IDM server is hacked.

(3) *The attacker attempts to manipulate the CRL, DASS or a delta message.* Since the CA-IDM channel can use trust TLS sessions, the integrity of the CRL can be protected. In addition, since the DASS messages are signed by the IDM servers, the attacker cannot manipulate the DASS installation or updating patches.

(4) *The attacker wants to infer private information of other devices, servers, or CAs from the TinyCR install and update messages.* An attacker can easily obtain the TinyCR install and update messages by compromising just one device. However, knowing these messages give the attacker no advantage because the CR information is public. DASS is not constructed for each particular device hence there is no device private information in the DASS messages.

(5) *The attacker can block the update messages between an IoT device and the IDM server, then use a revoked certificate to attack that device.* TinyCR has no specific design to prevent the attacks of blocking the communication to a device – and no other

CR method does. However, it is possible to detect such attacks. For example, the IDM server can send heartbeat packets to the devices periodically with the digest of the up-to-date DASS verifier and the current time. If the device does not hear the heartbeat after a period of time, it may detect such communication-blocking attack.

(6) *A compromised IDM server sends wrong DASS information and update messages.* All DASS install and update messages can be easily audited by another IDM server that knows all certificates and the revoked ones. "Auditable" means any party who knows the entire CRL can verify if another DASS version is maliciously modified. The device can forward the DASS messages with signatures to other IDM servers for auditing. The IDM servers can use their maintained certificate universe and the CRL to test the integrity of the DASS. If the DASS information is tempered, the other IDM servers can easily find the malicious IDM server by the signature.

(7) *The adversary acquires and causes a revocation with a strategy to trigger frequent rebuilds of DASS.* An attacker could learn which certificate revocation will trigger a rebuild of DASS (by running a simulation experiment) and then attacks that particular certificate and causes it to be revoked by the CA. To defend against such an attack, we can preallocate extra space in DASS to make it capable of learning more updates without rebuilding and reduce the probability to find a certificate that triggers a rebuild. From our analysis in Fig.6.12, we find this strategy is effective to defend the attacker with limited power. For example, by doubling the size of DASS upon initialization, it becomes more than 20 times harder to find a certificate that will trigger a rebuild.

6.5 Conclusion.

TinyCR is a new system and protocol to allow on-device CR checking for IoT. We develop DASS, a compact and dynamic data structure, to maintain the CR status of the entire IoT network, which costs each device very small memory. We also implement the two communication components of TinyCR: the tracker that run on an IDM server to construct and update DASS and sends the update messages to devices, and the verifier that can synchronize with the tracker and be queried for the CR status on IoT devices. The experiments show that TinyCR costs small memory, short CR checking time, low network bandwidth, and low synchronization latency.

Chapter 7

Summary.

This thesis studies two challenging problems regarding enhancing the reliability of large-scale mobile networks: (1) how to automatically and efficiently troubleshoot a individual customer service problem in cellular networks, and (2) how to perform low-cost and on-device certificate revocation checking for mobile and IoT network ecosystems. Both challenges require us to study the mobile network data and design effective data abstraction tools to properly represent the key knowledge from the vast data owned by the service providers. Specifically, we design two types of data abstraction approach for the two problems.

In the cellular customer-level service troubleshooting challenge, the key demand is how to automatically understand the network status and user experiences from the network data. The supervision for "understanding" the knowledge is the historical manual troubleshooting experiences recorded in the troubleshooting ticket data. Therefore, we design machine learning tools, such as deep neural networks and decision trees, to extract and abstract the valid and interpretable information from the network data.

The learning process is guided by the supervision from historical data and our domain knowledge. Although the ML-based data abstraction methods perform well according to our evaluation results based on real case studies, one most significant drawback of the ML-based methods is that it provides no theoretical lower-bound guarantee for the specific problem. In our automatic troubleshooting problem, there are currently no good ways to precisely estimate how bad the model might be in the worst cases. For example, we cannot precisely estimate how the model would perform if the network configuration is changed significantly in the future. Neither could we precisely tell whether the model can provide accurate troubleshooting results for a specific case, although we can estimate the global troubleshooting accuracy given a large historical dataset. Since no worst-case guarantee can be given while the cellular service providers have to be responsible for their customers, the ML-based troubleshooting systems cannot fully replace the current troubleshooting human agents. Instead, the best application scenario is using them as a "troubleshooting assistant" in the current manual troubleshooting framework. Therefore, the application scenario requires the ML model to provide not only the final troubleshooting results but also the interpretable data-driven insights about why the decision is made in the process.

For the on-device certificate revocation checking challenge, the goal is to tell the validity of a device certificate precisely. To solve the problem, we abstract the device certificate universe data by using the combination of probabilistic data filters and hash tables. Unlike the ML-based data abstraction methods, the hash-based methods can provide a theoretical worst-case guarantee for the data structure under different application scenarios. Therefore, those methods can be potentially used in the applications

with high responsibility demand, such as network security applications. On the other hand, current hash-based data abstraction methods cannot effectively model the high-level knowledge from the data, such as the network status and user experience analysis problem in the first challenge.

From the two major problems studied by this paper, we can learn that improving the reliability of mobile networks from the data-driven perspective requires not only the powerful but less verifiable data abstraction tools (such as the ML models) but also the verifiable but less representative tools (such as the hash-based methods). In addition to the data abstraction models, a comprehensive system design that considers real-world limitations and challenges is also necessary for real mobile networks.

Chapter 8

Lessons Learned for Handling Large-Scale Networking Data

Recent decades have witnessed a vast growing of Networking Data including the service-level data, network measurement data, and network log data. For example, in a nationwide network database, the data for describing user-level network logs in each single day can be at trillion bytes scale. In addition, for a state-of-the-practice large-scale networking system, there can be variety of network functions that measure different aspects of the network at multiple levels. Those network functions provide us with heterogeneous networking data for the comprehensive understanding network operating status. However, handling the trillion-scale heterogeneous data is not a straightforward work although the algorithms themselves are theoretically efficient and scalable for the data, especially when the data processing modules are supposed to be deployed for a real-time system on resource-limited devices. In this chapter, I would like to share

some of my experiences I gained from the above projects for using and processing the large-scale network data in real-time systems.

Understanding the data first. Before designing the algorithms or applying the machine learning models for data processing, the very first step is understanding the data. Domain knowledge is always required for understanding the data. For example, how network measurements are performed, what are the key metrics to indicate network performance, and what are the key features of the studied network system architecture, etc. In addition to the domain knowledge, we also need to have a comprehensive understanding of the knowledge of the specific data sets. For example, we need to understand how the data is collected, what information we could learn from each individual attribute, how the values of the attributes make an impact and relate to the topic of the study, etc. It is particularly important to maintain and study from a detailed meta-data document for the data sets. After combining the domain knowledge and the dataset knowledge, we could learn what information set we can obtain from the data and how the information set can be helpful for the studied topic. Since processing the large-scale data is rather time-consuming, finding the correct directions for handling the data can minimize the time and computational resources wasted on "useless efforts".

Start with a subset of the data. After the direction of how to process the data is decided, we can move on to process the data with our algorithms. We can always start with a tiny subset of the database rather than the whole database to test our programs or algorithms. Processing the data can be extremely time-consuming. Thus, we do not hope to find that our programs or algorithms have some flaws after we finish processing the whole dataset. By preprocessing a small subset, we could find

and eliminate those flaws at the early stage and avoid those issues when handling the large data set. To examine the correctness of our data processing, we can visualize the attributes for the small data set and manually validate the feature values using our domain knowledge. If everything looks like what we expect, then we can work on the whole data.

Streaming data processing. Many types of network data are time-series data generated in a streaming manner. When processing the streaming time-series data in a real-time system, if the algorithms are not dependent on the information of the far history, it might be wise to schedule the data processing programs (for example, read and process a period of the real-time data in every time interval using Crontab in Linux) rather than to use one single program that runs for forever and always process the data when new data is available. The data feeding sources may not always be reliable. For example, the data feeds may suffer from power outages. By using the chunked feature processing programs, we can automatically recover the processing as soon as the data feeds are back online. In addition, the chunked programs can be more robust to the out-of-memory issues by recycling the memory in a timely way.

Real-time systems. For real-time systems, it is important to understand whether the data delay and processing delay can satisfy the requirements of the system. The data delay is naturally bounded by the delay and the granularity of the data feeds, while the processing delay is decided by the cost of the algorithms and the capabilities of the hardware. Sometimes, there is a trade-off between the data delay and the processing delay. For example, if we want the processing delay smaller, we may have to use more coarse-grind data feeds, which will increase the data delay and

scarifies the model accuracy. For heterogeneous networking data, different sources may have different data delays and granularity. Therefore, we may also have to synchronize the data (by padding, sampling, interpolation, etc.) and choose an optimal data delay and granularity that satisfy the system requirements while providing the optimal model performance.

Bibliography

[1] An Internet of Things Reference Architecture. White Paper, Symantec, 2016.

[2] Why Digital Certificates Are Essential for Managing Mobile Devices. White Paper, DigiCert, Symantec's Website Security business, 2019.

[3] Censys. `https://censys.io/certificates`, 2021. Accessed: 2019.

[4] Signal Features - MATLAB. `https://www.mathworks.com/help/predmaint/ug/signal-features.html`, 2022. Accessed: 2022.

[5] Arwa Alrawais, Abdulrahman Alhothaily, Xiuzhen Cheng, Chunqiang Hu, and Jiguo Yu. Secureguard: A Certificate Validation System in Public Key Infrastructure. *IEEE Transactions on Vehicular Technology*, 67(6):5399–5408, 2018.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010.

[7] Jessa Bekker and Jesse Davis. Learning from Positive and Unlabeled Data: A Survey. *Machine Learning*, 109(4):719–760, 2020.

[8] Burton H Bloom. Space/Time Trade-offs in Hash Coding With Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[9] Denis Charles and Kumar Chellapilla. Bloomier Filters: A Second Look. In *In Proceedings of the European Symposium on Algorithms (ESA)*, 2008.

[10] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In *In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 30–39, 2004.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A Scalable Tree Boosting System. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[12] Xuxi Chen, Wuyang Chen, Tianlong Chen, Ye Yuan, Chen Gong, Kewei Chen, and Zhangyang Wang. Self-PU: Self Boosted and Calibrated Positive-Unlabeled Training. In *International Conference on Machine Learning*, pages 1510–1519. PMLR, 2020.

[13] Yi-Chao Chen, Gene Moo Lee, Nick Duffield, Lili Qiu, and Jia Wang. Event Detection Using Customer Care Calls. In *Proceedings of IEEE INFOCOM*, pages 1690–1698. IEEE, 2013.

[14] Marc Claesen, Frank De Smet, Johan AK Suykens, and Bart De Moor. A Robust Ensemble Approach to Learn From Positive and Unlabeled Data Using SVM Base Models. *Neurocomputing*, 160:73–84, 2015.

[15] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *Proceedings of ICML*, pages 933–941. PMLR, 2017.

[16] Ernesto Diaz-Aviles, Fabio Pinelli, Karol Lynch, Zubair Nabi, Yiannis Gkoufas, Eric Bouillet, Francesco Calabrese, Eoin Coughlan, Peter Holland, and Jason Salzwedel. Towards Real-time Customer Experience Prediction for Telecommunication Operators. In *Proceedings of IEEE BigData*, pages 1063–1072, 2015.

[17] Donald Eastlake et al. Transport Layer Security (TLS) Extensions: Extension Definitions. Technical report, RFC 6066, January, 2011.

[18] Charles Elkan and Keith Noto. Learning Classifiers from Only Positive and Unlabeled Data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220, 2008.

[19] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In *In Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2014.

[20] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip Yu, and Olivier Verscheure. Direct Mining of Discriminative and Essential Frequent Patterns via Model-based Search Tree. In *Proceedings of ACM SIGKDD*, pages 230–238, 2008.

[21] Jerome H Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of statistics*, pages 1189–1232, 2001.

[22] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Hongjun Lu, and Philip S Yu. Text Classification Without Negative Examples Revisit. *IEEE transactions on Knowledge and Data Engineering*, 18(1):6–20, 2005.

[23] Mark Goodwin. Revoking Intermediate Certificates: Introducing Onecrl. *Mozilla Security Blog*, 2015.

[24] Quanquan Gu, Zhenhui Li, and Jiawei Han. Generalized Fisher Score for Feature Selection. In *Proceedings of Uncertainty in Artificial Intelligence*, 2011.

[25] Phillip Hallam-Baker. X. 509v3 Transport Layer Security (TLS) Feature Extension. *RFC 7633*, 2015.

[26] William L Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.

[27] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining Sequential Patterns Efficiently by Prefix-projected Pattern Growth. In *Proceedings of ICDE*, pages 215–224. IEEE, 2001.

[28] Harri Holma and Antti Toskala. *LTE for UMTS: Evolution to LTE-advanced*. John Wiley & Sons, 2011.

[29] Russell Housley, Warwick Ford, William Polk, and David Solo. Internet X. 509 Public Key Infrastructure Certificate and CRL Profile. Technical report, RFC 2459, January, 1999.

[30] Jiyao Hu, Zhenyu Zhou, Xiaowei Yang, Jacob Malone, and Jonathan W Williams. CableMon: Improving the Reliability of Cable Broadband Networks via Proactive Network Maintenance. In *Proceedings of USENIX NSDI*, pages 619–632, 2020.

[31] Qinwen Hu, Muhammad Rizwan Asghar, and Nevil Brownlee. Certificate Revocation Guard (CRG): An Efficient Mechanism for Checking Certificate Revocation. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*.

[32] Anand Padmanabha Iyer, Li Erran Li, and Ion Stoica. Automating Diagnosis of Cellular Radio Access Network Problems. In *Proceedings of ACM MobiCom*, pages 79–87, 2017.

[33] Yu Jin, Nick Duffield, Alexandre Gerber, Patrick Haffner, Wen-Ling Hsu, Guy Jacobson, Subhabrata Sen, Shobha Venkataraman, and Zhi-Li Zhang. Making Sense of Customer Tickets in Cellular Networks. In *Proceedings of IEEE INFOCOM*, pages 101–105, 2011.

[34] Yu Jin, Nick Duffield, Alexandre Gerber, Patrick Haffner, Subhabrata Sen, and Zhi-Li Zhang. Nevermind, The Problem Is Already Fixed: Proactively Detecting and Troubleshooting Customer DSL Problems. In *Proceedings of ACM CoNEXT*, pages 1–12, 2010.

[35] Masahiro Kato, Takeshi Teshima, and Junya Honda. Learning from Positive and

Unlabeled Data with A Selection Bias. In *International conference on learning representations*, 2018.

[36] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *Proceedings of ICLR*, 2015.

[37] Ryuichi Kiryo, Gang Niu, Marthinus C Du Plessis, and Masashi Sugiyama. Positive-Unlabeled Learning with Non-negative Risk Estimator. *Advances in neural information processing systems*, 30, 2017.

[38] D. Kumar, M. Bailey, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, and J. A. Halderman. Tracking certificate misissuance in the wild. In *In Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2018.

[39] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing Network-Wide Traffic Anomalies. *Proceedings of ACM SIGCOMM*, pages 219–230, 2004.

[40] Adam Langley. Revocation Checking and Chrome's CRL. *ImperialViolet (blog)*, 2012.

[41] James Larisch, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers. In *In Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 539–556. IEEE, 2017.

[42] Xiao-Li Li and Bing Liu. Learning from Positive and Unlabeled Examples with

Different Data Distributions. In *European conference on machine learning*, pages 218–229. Springer, 2005.

[43] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and Identification of Network Anomalies Using Sketch Subspaces. In *Proceedings of ACM IMC*, pages 147–152, 2006.

[44] Xin Li, Minmei Wang, Huazhe Wang, Ye Yu, and Chen Qian. Toward Secure and Efficient Communication for the Internet of Things. *IEEE/ACM Transactions on Networking*, 2019.

[45] Bing Liu, Yang Dai, Xiaoli Li, Wee Sun Lee, and Philip S Yu. Building Text Classifiers Using Positive and Unlabeled Examples. In *Third IEEE international conference on data mining*, pages 179–186. IEEE, 2003.

[46] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An End-to-end Measurement of Certificate Revocation in the Web's PKI. In *In Proceedings of the Internet Measurement Conference (IMC)*, pages 183–196. ACM, 2015.

[47] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting Deep Learning-based Networking Systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[48] Michael Mitzenmacher. Compressed Bloom Filters. *IEEE/ACM transactions on networking*, 10(5):604–612, 2002.

[49] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X. 509 Internet Public Key Infrastructure Online Certificate Status Protocol-OCSP. Technical report, RFC 2560, 1999.

[50] Alma Oracevic, Selma Dilek, and Suat Ozdemir. Security in Internet of Things: A Survey. In *In Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2017.

[51] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. In *In Proceedings of the European Symposium on Algorithms (ESA)*, pages 121–133. Springer, 2001.

[52] Lujia Pan, Jianfeng Zhang, Patrick PC Lee, Hong Cheng, Cheng He, Caifeng He, and Keli Zhang. An Intelligent Customer Care Assistant System for Large-Scale Cellular Network Diagnosis. In *Proceedings of ACM SIGKDD*, pages 1951–1959, 2017.

[53] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[54] Chunyi Peng. *Cellular Network for Mobile Devices and Applications: Infrastructure Limitations and Solutions.* University of California, Los Angeles, 2013.

[55] Colin Percival. Binary Diff/Patch Utility. *URL: http://www. daemonology. net/bsdiff*, 2003.

[56] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the JigSaw: Towards Automated Problem Inference from Network Trouble Tickets. In *Proceedings of USENIX NSDI*, pages 127–141, 2013.

[57] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. 2012.

[58] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of PCA for Traffic Anomaly Detection. In *Proceedings of ACM SIGMETRICS*, pages 109–120, 2007.

[59] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The Graph Neural Network Model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[60] Sundararajan Sellamanickam, Priyanka Garg, and Sathiya Keerthi Selvaraj. A Pairwise Ranking Based Approach to Learning with Positive and Unlabeled Examples. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 663–672, 2011.

[61] M Zubair Shafiq, Lusheng Ji, Alex X Liu, Jeffrey Pang, and Jia Wang. Characterizing geospatial dynamics of application usage in a 3g cellular data network. In *Proceedings of IEEE INFOCOM*, pages 1341–1349. IEEE, 2012.

[62] M Zubair Shafiq, Lusheng Ji, Alex X Liu, and Jia Wang. Characterizing and modeling internet traffic dynamics of cellular devices. *ACM SIGMETRICS Performance Evaluation Review*, 39(1):265–276, 2011.

[63] Zhihao Shen, Wan Du, Xi Zhao, and Jianhua Zou. Dmm: Fast map matching for cellular data. In *Proceedings of ACM MobiCom*, pages 1–14, 2020.

[64] Amit Sheoran, Sonia Fahmy, Matthew Osinski, Chunyi Peng, Bruno Ribeiro, and Jia Wang. Experience: Towards Automated Customer Issue Resolution in Cellular Networks. In *Proceedings of ACM MobiCom*, pages 1–13, 2020.

[65] Xiaofeng Shi, Shouqian Shi, Minmei Wang, Jonne Kaunisto, and Chen Qian. On-device IoT Certificate Revocation Checking with Small Memory and Low Latency. In *Proceedings of ACM CCS*, 2021.

[66] Ahmed Shokry, Marwan Torki, and Moustafa Youssef. DeepLoc: A Ubiquitous Accurate and Low-overhead Outdoor Cellular Localization System. In *Proceedings of ACM SIGSPATIAL*, pages 339–348, 2018.

[67] Trevor Smith, Luke Dickinson, and Kent Seamons. Let's Revoke: Scalable Global Certificate Revocation. In *In Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020.

[68] Pawel Szalachowski, Laurent Chuat, Taeho Lee, and Adrian Perrig. RITM: Revocation in the Middle. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016.

[69] Pang-Ning Tan, Hannah Blau, Steve Harp, and Robert Goldman. Textual Data Mining of Service Center Call Records. In *Proceedings of ACM SIGKDD*, pages 417–423, 2000.

[70] Samuel Tanner Lindemer. Digital certificate revocation for the internet of things. Master's thesis, KTH Royal Institute of Technology, 2019.

[71] Yang Tong, Dongsheng Yang, Jie Jiang, Siang Gao, Bin Cui, Lei Shi, and Xiaoming Li. Coloring Embedder: a Memory Efficient Data Structure for Answering Multi-set Query. In *In Proceedings of the IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1142–1153. IEEE, 2019.

[72] Shobha Venkataraman and Jia Wang. Towards Identifying Impacted Users in Cellular Services. In *Proceedings of ACM SIGKDD*, pages 3029–3039, 2019.

[73] Daryl Walleck, Yingjiu Li, and Shouhuai Xu. Empirical Analysis of Certificate Revocation Lists. In *In Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy*, pages 159–174. Springer, 2008.

[74] Minmei Wang, Chen Qian, Xin Li, and Shouqian Shi. Collaborative Validation of Public-key Certificates for IoT by Distributed Caching. In *In Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 847–855. IEEE, 2019.

[75] Minmei Wang, Mingxun Zhou, Shouqian Shi, and Qian Chen. Vacuum Filters: More Space-Efficient and Faster Replacement for Bloom and Cuckoo Filters. In *In Proceedings of International Conference on Very Large Databases (PVLDB)*, 2020.

[76] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust Face Recognition via Sparse Representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227, 2008.

[77] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A Comprehensive Survey on Graph Neural Networks. *IEEE transactions on neural networks and learning systems*, 2020.

[78] Fengli Xu, Yong Li, Huandong Wang, Pengyu Zhang, and Depeng Jin. Understanding Mobile Traffic Patterns of Large Scale Cellular Towers in Urban Environment. *IEEE/ACM transactions on networking*, 25(2):1147–1161, 2016.

[79] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? 2019.

[80] Yixing Xu, Yunhe Wang, Hanting Chen, Kai Han, Chunjing Xu, Dacheng Tao, and Chang Xu. Positive-Unlabeled Compression on The Cloud. *Advances in Neural Information Processing Systems*, 32, 2019.

[81] Yixing Xu, Chang Xu, Chao Xu, and Dacheng Tao. Multi-Positive and Unlabeled Learning. In *IJCAI*, pages 3182–3188, 2017.

[82] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How Transferable Are Features in Deep Neural Networks? *Proceedings of NIPS*, 2014.

[83] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. *Proceedings of IJCAI*, pages 3634–3640, 2018.

[84] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations. In *In Proceedings of the ACM 5th*

International Conference on Emerging Networking Experiments and Technologies (CoNEXT), pages 313–324, 2009.

[85] Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. Memory-Efficient and Ultra-fast Network Lookup and Forwarding using Othello Hashing. *IEEE/ACM Transactions on Networking*, 26(3):1151–1164, 2018.

[86] Chaoyun Zhang, Marco Fiore, Cezary Ziemlicki, and Paul Patras. Microscope: Mobile service traffic decomposition for network slicing as a service. In *Proceedings of ACM MobiCom*, pages 1–14, 2020.

[87] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of SSL Certificate Reissues and Revocations in The Wake of Heartbleed. In *In Proceedings of the Conference on Internet Measurement Conference (IMC)*, pages 489–502, 2014.

[88] Ying Zhang, XuChan Ju, and YingJie Tian. Nonparallel Hyperplane Support Vector Machine for PU Learning. In *2014 10th International conference on natural computation (ICNC)*, pages 703–708. IEEE, 2014.

[89] Peifang Zheng. Tradeoffs in Certificate Revocation Schemes. *ACM SIGCOMM Computer Communication Review*, 2003.

[90] Dong Zhou, Bin Fan, Hyeontaek Lim, David G Andersen, Michael Kaminsky, Michael Mitzenmacher, Ren Wang, and Ajaypal Singh. Scaling Up Clustered Network Appliances with ScaleBricks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.

[91] Zhi-Hua Zhou. A Brief Introduction to Weakly Supervised Learning. *National science review*, 5(1):44–53, 2018.

[92] Xiaojin Jerry Zhu. Semi-supervised Learning Literature Survey. 2005.