NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2022-04

June, 2022

Enabling Robust and Secure Edge-to-Cloud Communications

Kaiyu Hou

Abstract

Availability and security, as two of the most important requirements for networks, are not always achieved due to misconfiguration or performance restrictions. This report introduces two of my projects. The former secures configurations for the cellular networks (the edge), while the latter secures and optimizes the communications for the serverless networks (the cloud). Both projects target next-generation network protocols, focusing on practical solutions for real-world networked systems.

Keywords

Networked System, Network Protocol, Cellular Network Emergency Call, Formal Methods, Serverless Computing, QUIC Protocol

NORTHWESTERN UNIVERSITY

Enabling Robust and Secure Edge-to-Cloud Communications

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Kaiyu Hou

EVANSTON, ILLINOIS

June 2022

ABSTRACT

Enabling Robust and Secure Edge-to-Cloud Communications

Kaiyu Hou

Since the invitation of ARPANet in 1969, network protocols and communication systems have continued to emerge. Especially in the past decade, the prosperity of mobile internet and cloud computing has resulted in a large number of network protocols and communication systems, which have become critical infrastructure for our society. Availability and security, as two of the most important requirements for networks, are, however, not always achieved due to misconfiguration or performance restrictions.

My dissertation focuses on enabling robust and secure edge-to-cloud communications in large networked systems. Generally, a networked system consists of two major parts: the edge access network and the centralized cloud network. In this dissertation, I will introduce two of my projects. The former project secures configurations for the cellular networks (the edge), while the latter secures and optimizes the communication channel for the serverless networks (the cloud). Both projects target next-generation network protocols, concentrating on practical solutions for real-world networked systems. They have been validated via extensive analysis, implementations, and real-world evaluations.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to my advisor and my committee chair, Prof. Yan Chen, for guiding me through this wonderful journey of my Ph.D. study. I am always impressed by his smartness, insight, and good taste in research. In these five years, I received unwavering support from him in both my research and daily life. I greatly appreciate the trust he had in me, which gives me the courage and strength to overcome challenges and helps me eventually find my own way.

Meanwhile, I would like to express my sincere gratitude to my committees, Prof. Fabián Bustamante, Dr. Vinod Yegneswaran from SRI International, and Prof. Hai Zhou. Prof. Fabián Bustamante and Prof. Hai Zhou also served as my qualifying exam committees. Their suggestions and comments greatly improve the quality of this dissertation.

Prof. Fabián Bustamante was one of the first computer science professors who also mentored my research. He was always approachable and helpful throughout my Ph.D. study. The idea of the QFaaS project was inspired while I was working with him.

Dr. Vinod Yegneswaran was my mentor during my research internships at SRI International and continued supervising me after my internships. I received tremendous help from him, not only including high-level idea discussion but also hands-on guidance and paper drafting. He has contributed a lot to the second part of this dissertation.

I worked very closely with Prof. Hai Zhou over the past five years and benefited greatly from his unique insights into defining scientific problems. He was always there willing to guide me. The first part of this dissertation was a joint work with him.

I would like to give special thanks to You Li, my colleague, collaborator, and friend, who made my road to Ph.D. no longer lonely. For countless mornings, afternoons, and nights, we were working together, exchanging thoughts, writing code, facing failures, and sharing joys. I cannot imagine how difficult this road would be without his accompany.

I have the fortune of working with many talented researchers and exploring new ideas with them, including Prof. Kai Bu, Prof. Simone Campanoni, Prof. Chunyi Peng, Prof. Xiaochun Wu, and Byungjin Jun, Xue Leng, Xing Li, Sen Lin, Bo Wu, Yinbo Yu.

In addition to my collaborators, there is also a long list of colleagues and friends, who made my life at Northwestern University colorful and joyful, specifically, Xue Chen, Xutong Chen, Xuechao Du, Yiding Feng, Weixin Jiang, Lin Jin, Mohammad Kavousi, Xiangmin Shen, Mary Truong, Lingzhi Wang, Runqing Yang, Guannan Zhao, Qingyang Zeng, and Tiantian Zhu.

I am grateful to the Computer Science Department, the beautiful Northwestern University campus, and the peaceful City of Evanston. I had a fairly good time there.

Finally, I thank all my family members. I would not have made it this far without the love and support from them. I dedicate this dissertation to them.

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Computer networks have become an irreplaceable part of our society. Since the invitation of ARPANet in 1969 [**66, 120**], network protocols and communication systems have continued to emerge. Especially in the past two decades, the prosperity of mobile internet and cloud computing has resulted in a large number of new network protocols, such as new generations of cellular network protocols (*e.g.*, LTE, 5G [**1**]), software-defined network (SDN) protocols (*e.g.*, OpenFlow [**96**], P4 [**31**]), transport protocols (*e.g.*, SPDY [**137**], QUIC [**115**]), security protocols (*e.g.*, TLS 1.3 [**50**]), and application protocols (*e.g.*, HTTP/3 [**29**]).

These network protocols make up a wide variety of large networked systems, typically represented by national cellular network systems, Internet of Things (IoT) network systems, and cloud-native network systems. With enormous market prospects, these large networked systems have become critical infrastructure in our daily lives. For instance, the total number of cellular network subscriptions is estimated to be 8.6 billion in 2021 [**123**]; 35 billion IoT devices have already been installed by 2021 [**59**], and its market share is projected to surpass $1,500 billion by 2025 [**132**]; the cloud computing market had a value of $133 billion in 2021 and is expected to reach $168 billion by 2025 [**142**].

Availability and security, as two of the most important requirements for communication systems, however, are not always achieved due to misconfiguration (§2) and performance restrictions (§3). For instance, there are always complaints about the availability

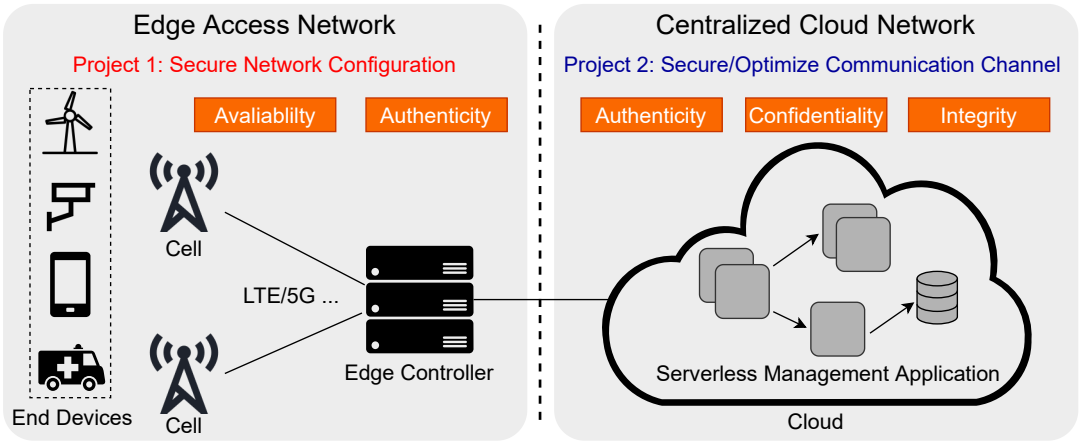


Figure 1.1. The architecture of IoT networked system.

of cellular networks, especially the emergency service [**138**]. The instability of emergency communication systems can lead to severe consequences [**103**]. Some cloud services lack encryption on internal communications, leading to publicized security disasters [**117, 78**].

In this dissertation, I will focus on the following question: *How can we enable robust and secure edge-to-cloud communications in real-world networked systems?* My dissertation tackles this question by addressing specific network availability and security challenges that we encounter in two projects from two different aspects of networked systems: securing configurations for the edge network, and securing and optimizing communication channels for the serverless cloud network.

In general, a large networked system can be divided into two major parts: the edge access system and the centralized cloud/data center system. The former provides ubiquitous connectivity for end-devices by utilizing mobile internet technologies, while the latter provides flexible control and management plane leveraging cloud computing and data center network (DCN) technologies. Taking the IoT system as an example (Figure 1.1), in

the IoT edge access network, with new wireless and mobile protocols, IoT end devices receive convenient, fast, and reliable network connections, continuously interacting with the central management cloud. On the cloud side, various microservice and serverless cloud applications offer centralized services to edges, such as big data processing and storage, identity management, and event analysis and triggering. Although the evolution of both parts has benefited from the development of next-generation network protocols, the use of new technologies has also introduced availability and security vulnerabilities.

In the first project, we aim to systematically address the availability and security problems in cellular networks. Formal methods have been widely adopted into network protocol studies by researchers [**141, 70, 72, 16, 26, 46, 2**]. However, most works suffer from modeling granularity and misrepresentation problems. To solve these difficulties, we propose a *seed-assisted specification* method. It combines prior knowledge and adaptive model construction in addition to protocol-based formal specifications. We take the cellular emergency call system as the case study and conduct the first research to thoroughly investigate its availability and security vulnerabilities. We build a formal model based on our proposed method. By running model checking, four public-unaware scenarios where emergency calls cannot be correctly routed are discovered, and two new attacks leveraging the privileges of emergency calls are found. We recommend a solution addressing all those failures and attacks and show its correctness. Our proposed seed-assisted specification method can be easily extended to analyze other communication systems which are described by protocols in general.

In the second project, we design and implement the *QFaaS* system. It can simultaneously improve performance and provide security to serverless cloud networks. Serverless

computing has greatly simplified cloud programming. Nevertheless, internal network connections between serverless functions are now initiated frequently, to support serverless features such as agile autoscaling and function chains, raising communication latency. To alleviate this cost, current serverless providers sacrifice security for network performance, keeping internal function communications unencrypted [**13, 23, 100, 63**]. To address this challenge, we first provide a new network-centric abstraction for serverless computing, filling in missing details about actual network flows in the widely used logic abstraction. This abstraction inspired the design of the QFaaS system, where the QUIC protocol can be seamlessly integrated into serverless platforms, to mitigate connection setup overheads and provide secure communications. Our design explicitly ensures that existing serverless applications can directly benefit from QFaaS without any application code modification. Experiments demonstrate that QFaaS can reduce the end-user response time of single functions and function chains by 28% and 40%, respectively.

Both projects target next-generation network protocols and aim to provide practical solutions to address availability and security problems in real-world networked systems. We emphasize the importance of real-world measurement and deployment in this dissertation. These two projects have been validated via real-world evaluations, and corresponding system implementations will be open-sourced to the public. Security is composed of four cornerstones: availability, authenticity, confidentiality, and integrity. The first project covers the *availability* and *authenticity* issues, while the second project provides *authenticity*, *confidentiality*, and *integrity* to the network.

We value the cooperation with the industry and are committed to making real-world impacts while conducting our research. Our study has attracted attention and received

positive feedback from the corresponding open-source communities. And we have actively transited our work to commercial cellular network carriers and cloud service providers. Three world-leading cellular network carriers have acknowledged our findings. And one of the world-leading cloud providers is working on a PoC (proof-of-concept) deployment of the proposed QFaaS system on its serverless platform.

1.1. Summary of the Contributions

This dissertation studies how to enable robust and secure communications for edge access networks and centralized cloud networks. The primary contributions are as follows:

- **Seed-assisted specification method for protocol-based communication systems.** We propose a *seed-assisted specification* method to solve both the modeling granularity and misrepresentation problems. It leverages the prior knowledge in model specification and integrates measurement results with a generalized formal model. This method can be applied to networked systems described by protocols in general.
- **First thorough investigation of cellular network emergency call systems.** We systematically explore availability and security pitfalls in cellular network emergency call systems. We find 4 scenarios where emergency calls cannot be routed and 2 new attacks that abuse emergency call privileges. A unified solution is proposed for cellular network carriers.
- **Network-centric modeling for serverless cloud computing.** We provide a network-centric abstraction model for serverless computing. It demonstrates the

actual network flows and bottlenecks in the serverless computing architecture, filling in missing details in the widely used logic model.

- **QFaaS system design, implementation, and evaluation.** We first extend the QUIC protocol into the domain of serverless platforms and propose the QFaaS system. It provides low latency serverless function communication with improved security. We implement the QFaaS prototype on the popular OpenFaaS platform. It requires no code modification for existing serverless applications.

1.2. Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, I first introduce the challenges and our solutions of applying formal methods to cellular network studies. I then present our thorough analysis of cellular emergency call systems, including specification, verification, and recommendation, as the case study to illustrate our proposed method. The seed-assisted specification method is formally explained at the end of this chapter. In Chapter 3, I first discuss the network challenges in serverless cloud computing. I then provide our modeling to the serverless network architecture. It is followed by the QFaaS system design, prototype implementation, and evaluation. Chapter 4 summarizes this dissertation and discusses future work and open research directions.

CHAPTER 2

Discovering Cellular Network Emergency Call Vulnerabilities with Formal Methods

2.1. Introduction

In this project, we aim to systematically address the availability and security issues in cellular emergency call systems and explain their underlying causal mechanisms in depth.

Public Safety Answer Points (PSAPs), such as the 911 call center in the U.S. and the 112 emergency service center in Europe, receive millions of emergency calls each year. Those calls are then dispatched to entities like police stations, fire stations, and ambulance centers. Cellular network systems play critical roles in this service. For example, in the U.S., there are an estimated 240 million emergency calls made to 911 each year, and more than 80% of those calls are from cellular devices [**103**]. Considering the huge number of emergency calls made through cellular networks and the potential impact of missing any of them, the correctness of cellular emergency call services deserves a close investigation. Unfortunately, as far as we know, there is no thorough research on this topic.

In 2019, a news report was widely spreading and had terrified the public in China. During an emergency medical situation, a cellphone (referred to as UE, user equipment) could not dial 120, the ambulance emergency number in China, even though a valid domestic SIM card was inserted (§2.3.1). Similar complaints about the availability of the cellular emergency call services can be found by skimming over public websites. In

fact, a cellular emergency call system can be unreliable and is subject to attacks. First, it is difficult to unify the system design, as different countries have different emergency numbers and different settings for historical reasons. Second, there exists no economic incentive for the cellular network carriers or the UE device manufacturers to optimize system implementation or correct existing errors.

Formal methods, including model checking [**141, 70, 72**] and symbolic analysis [**16, 26, 46, 2**], have been widely adopted by network researchers into network protocol studies. Given a set of security properties and a specified model, formal verification tools can either verify that the model upholds these properties or shed light on how the model violates them. However, most works suffer from two major problems in *formal specification*: modeling granularity and misrepresentation. First, the inappropriate level of granularity either requires enormous efforts in building models or can lead to false positives. Second, models specified from the protocols may not precisely represent the deployed systems, as a real-world system is just an instance from generally broad protocols.

To solve these difficulties, we propose a *seed-assisted specification* method. It combines prior knowledge and adaptive model construction in addition to protocol-based formal specifications. Using the above-mentioned case in China as a starting point, we first manually investigate its underlying causes. Instead of building a model of the whole system and aimlessly checking it for possible vulnerabilities, we limit the scope to the problem we want to explore: the routing failures of emergency calls. Therefore, we can include any related details into our model, slice away unrelated ones, and, most importantly, control the right level of granularity. Existing works aim to discover vulnerabilities in protocol designs. We believe that discovering vulnerabilities in deployed systems is equally or even

more critical. In that sense, we augment our model by configurations measured from deployed systems. As a benefit, we can ensure the counterexamples we found are practical in the real world.

The rest of this chapter is organized as follows.

In §2.2, I first present the challenges of applying formal methods to cellular network studies and our solutions to address these challenges.

In §2.3, following the proposed method, we specify the emergency call systems and conduct the first research to systematically study the availability and security issues in cellular emergency call systems. We have released our formal specifications as open source.

In §2.4, with this model, we systematically find 4 emergency call failure scenarios in China. One of the major carriers has confirmed our findings. The public is unaware of all these scenarios, while 2 of them are unknown to carriers. Moreover, we allow the flexibility to incorporate other configurations as model constraints to check other systems. We measure the emergency call configurations of two major U.S. carriers and find 2 new attacks abusing privileges of emergency calls. One attack is the first reported attack, which can bypass the screen password of UEs and make any calls. Another attack can block calls to the targeted numbers. It does not affect calls to other numbers and keeps effective longer than similar known attacks. We have reported them, and they are acknowledged by corresponding carriers.

In §2.5, we recommend a solution addressing all those failures and attacks, and show its correctness. We argue that lacking regulations or financial stimuli is another factor for the prevalent weaknesses in emergency call systems.

In §2.6, we summarize step-by-step procedures for the proposed *seed-assisted specification* method. This method can be easily extended to other systems which are described by protocols in general.

Discussions and related work are presented in §2.7 and §2.8, respectively.

Chapter summary, presented in §2.9, concludes this chapter.

2.2. Challenges and Our Solutions

As opposed to manual investigation [**85, 122, 71**], the use of formal methods brings a systematic and solid approach to cellular network research. In that sense, formal methods are introduced to protocol verification and have succeeded in cryptography-related analysis, such as authentication and key agreement (AKA) protocols [**2, 45**].

Nevertheless, problems emerge when applying formal methods to general cellular network protocols. Formal verification cannot be performed directly on human-language-based protocols. These protocols need to be translated into formal models first (*e.g.*, finite state machines). Meanwhile, a set of security properties are extracted from the protocol requirements. The process of constructing formal models and properties is called *specification*. The verification problem is to check whether models always follow their corresponding properties. However, meaningful attacks and vulnerabilities are unlikely to be found on an arbitrarily constructed specification without the guidance of strong prior knowledge.

Formal verification then executes in iterations. Each time the model checker finds a violation to the properties and returns a counterexample, researchers need to validate the

counterexample trace and determine whether it is feasible on a real-world system. Because there are always gaps between protocol definition, formal specification, and system implementation, a violation to a model built on protocols alone may not reflect a real-world problem. Researchers then have to exclude such an infeasible counterexample from the model and proceed to the next iteration. Without incorporating the information of real systems, the verification process can iterate forever until finding a non-trivial issue.

Therefore, there is still much room for improvement in the current approaches for verifying cellular network protocols. Several works [**70, 141, 140**] lie in this field, yet they all suffer from two major problems in specification: *modeling granularity* and *misrepresentation*.

- *Modeling Granularity*: In general, there is no golden rule to guide the granularity of modeling. Coarse-grained specification leads to false counterexamples because abstractions over-approximate the possible behaviors of a model. In contrast, the uniform fine-grained specification requires enormous efforts to build a model and has prohibitively large state space for model checkers.
- *Misrepresentation*: Protocols are generally broad and include many implementation options. Meanwhile, a real-world system is just an instance of protocols, and every possible option in protocols becomes a fixed assignment on the system. Therefore, models specified from protocols may not accurately represent the deployed systems.

From our observation, security analysis for cellular network systems has the following characteristics: *i)* an exposed critical security issue can attract widespread attention and is worthy of efforts to investigate the underlying causes; *ii)* a systematic search of potential

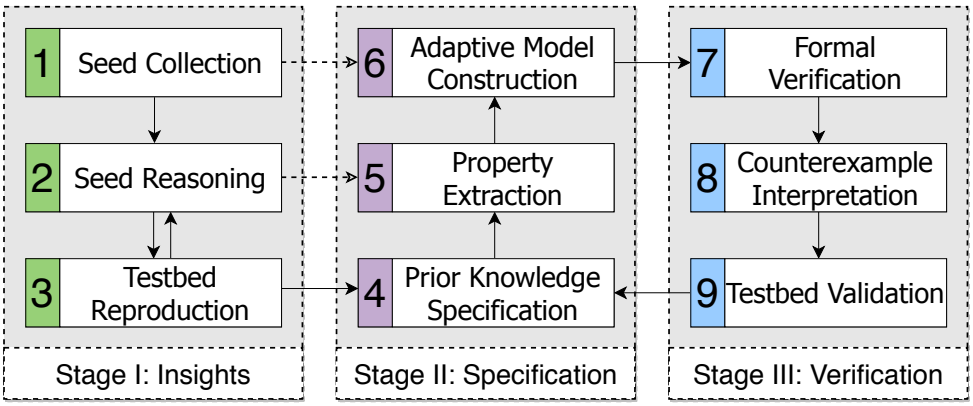


Figure 2.1. Framework of seed-assisted specification method.

vulnerabilities under similar causes is important; and *iii)* some configurations of deployed systems can be measured on-air or at the UE side. We believe that capturing these characteristics is the key of conducting formal analysis on cellular network systems.

Thus, we propose the *seed-assisted specification* method to solve both the modeling granularity and misrepresentation problems. Different from general protocol formal analysis, we leverage the prior knowledge, which is gained from investigating existing security issues and measuring real-world systems, to augment protocol-based specification. To this end, we use an exposed critical security issue as the "seed". The scope of verification, *i.e.*, what procedures to keep and their corresponding granularities, are decided by investigation and reasoning on the seed. When constructing a model from protocols, the implementation configurations and other flexibilities are distinguished and collected. In that way, real-world situations can later be incorporated into the model to reflect real-world systems.

Figure 2.1 shows the framework of the proposed method. It consists of 3 major stages in 9 steps. Prior knowledge is gained from and verified in the *Insights* stage. In the

Specification stage, we specify the model, design security properties, and adapt real-world configurations as model constraints. Counterexamples are generated in the *Verification* stage by model checkers. With prior knowledge, we can interpret counterexamples as availability issues or attacks on real-world systems and test their validities.

In the rest of this chapter, we first provide a case study, specifying cellular emergency call systems and discovering their pitfalls, to illustrate each of the steps by examples (§2.3, §2.4). We then formally explain each of the steps in §2.6. We believe it would be easier than before to secure protocol-based systems from vulnerabilities by following these steps.

2.3. Specification of Emergency Call Systems

For our study, we take a security issue exposed in 2019 as the seed (§2.3.1) and aim to systematically address the availability and security issues in cellular emergency call systems. We give an in-depth explanation of their underlying mechanisms and provide a solution to these vulnerability issues. In this section, we first manually investigate all underlying causes of the seed (§2.3.2), then build a formal model for cellular emergency call systems (§2.3.3).

2.3.1. The Seed: A Piece of Shocking News

In 2019, news about the failure of emergency calls attracted public attention [**138**]. As reported by major Chinese media outlets, in an emergency medical situation, the victim tried to dial 120, the ambulance emergency number, using her Meizu MX6 UE. A SIM card with a valid subscription from carrier China Mobile was inserted in the UE. However, all 120 phone calls made through that UE, no matter the ones from the locked screen

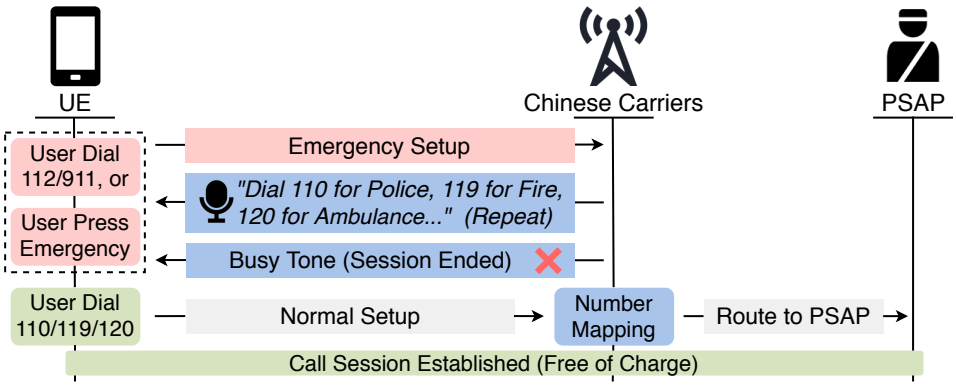


Figure 2.2. Calls initiated by the `Emergency Setup` signaling can only receive a recorded instruction. Chinese Carriers only route calls initiated by the normal `Setup` signaling.

emergency panel or the unlocked normal panel, could not reach the ambulance emergency center. What she heard was only a repeated dialing instruction, as shown in Figure 2.2.

2.3.2. Stage I: Insights

Step 1: Seed Collection. We performed the seed collection from three aspects: cellular network protocols, implementation details of UEs, and configurations of carriers.

The cellular network standard is developed by the 3GPP (3rd Generation Partnership Project), which consists of more than one thousand documents. By leveraging the seed, we do not need to investigate all of them. Instead, we narrow our reasoning and specification to call setup protocols [**8, 9, 10**] and emergency call-related protocols [**7, 11**].

From news interviews with the victim, we were able to collect the UE technical specification and system image (referred to as ROM) version. We were also able to record the procedure that she executed during the failure. Additionally, after this security issue was exposed, the Meizu corporation announced that they had solved this problem by releasing

a ROM update. This ROM was also collected for investigation. *Step 6: Adaptive Model Construction* presents more details about collecting information of UEs and carriers.

Step 2: Seed Reasoning. The 2G/GSM (Global System for Mobile Communications) emergency call service is inherited from the traditional landline system for public safety. In GSM, the normal `Setup` signaling is used to initiate normal phone calls, while the `Emergency Setup` signaling initiates emergency calls [**8**]. Two widely used emergency numbers, 112 (Europe) and 911 (North America), are listed as fixed emergency numbers [**11**]. When users dial them, UEs always send out emergency signaling.

When GSM was introduced in China in 1994, three separate emergency numbers, 110 (police), 119 (fire), and 120 (ambulance), had already been standardized and used for the landline system. All emergency calls to them were directly routed to their corresponding callees. The landline system uses the ITU (International Telecommunication Union) signaling system. It does not support `Emergency Setup` [**73**].

To deploy GSM in China, the following compromise was accepted (Figure 2.2). When UEs initiate the `Emergency Setup` signaling, (most likely when the users dial 112/911 or press the emergency button), carriers will not route these calls. Instead, carriers will loop sound recording of instructions to notify users of the correct emergency numbers in China. In contrast, call requests to 110/119/120 will be successfully routed if normal `Setup` initiates them. At a glance, such a scheme is backward compatible with the legacy processing system, while it does also "respond to" the `Emergency Setup` signaling to some extent. This setting has been inherited by 3G/4G networks and is still active today.

We therefore speculated that when the victim dialed 120, Meizu MX6 falsely initiated the call with the `Emergency Setup` signaling and was thus rejected by Chinese carriers.

Step 3: Testbed Reproduction. We reproduced the seed by using the same model UE with the same ROM. MX6 supports two major Chinese carriers. We dialed all emergency numbers with their SIMs under both the normal and the emergency panels. Calls from MX6 could not be successfully routed to PSAPs under any of these situations. Packet sniffer tools showed that MX6 used `Emergency Setup` to initiate these dials. We also used other UEs to initiate calls to 110/119/120 with `Emergency Setup`. All of them failed. These experiments proved that our seed reasoning is correct.

2.3.3. Stage II: Specification

Step 4: Prior Knowledge Specification. Following the idea of the *seed-assisted specification*, we exploited the prior knowledge to decide whether to keep, drop, or abstract a procedure or sub-procedure. Specifically, we are focusing on availability and security issues surrounding emergency call systems. From the seed, we learned that the most critical problem is related to routing: whether an emergency call can be routed and how exactly it is routed. The whole routing process is decided by the NAS (non-access stratum) layer protocol, which manages the communication session between the UE and the network [**10, 9**]. Procedures on the NAS layer depend on the bearers established on the RRC (radio resource control) layer [**6**]. However, we were not interested in modeling the RRC layer as any call will fail if the RRC layer fails.

Among all procedures on the NAS layer, *call control* procedures are most important. Meanwhile, the *attach* procedures are closely related to call control procedures. For instance, attach status and session contexts will afterward impact call setup and connection. The other procedures, such as *handover* procedures, *detach* procedures, and *identification*

procedures, do not have a direct impact on routing an emergency call. Therefore, the details of those procedures were abstracted away. We kept the skeletons of those procedures to ensure our model can still depict the whole call process.

We further analyzed in detail the call control and the attach procedures. The seed suggests that problems are likely to occur on occasions that normal procedures and emergency procedures are different. So, we distinguished the details that make normal and emergency calls different. For example, the available services are different depending on if the UE is attached to the network in emergency mode; the routing process and the response of the network are different depending on if the UE sends the call request in emergency mode.

We built our formal model in TLA+ [**81**]. The model has two major components: the UE and the network. Both components are flattened to avoid the hierarchical network structure between layers. A message channel synchronizes their interactions. A total of 36 configurable variables are included in our specification. Behaviors of the model are characterized by 20 TLA+ procedures. The original model with no constraints yields 10.59 billion distinct states and has a maximal diameter of 26 transitions from the initial states. The model, as well as corresponding model checking and counterexample interpretation utilities, has been open-sourced online.[1]

Step 5: Property Extraction. There are two major categories of properties: *safety* and *liveness*. Safety checking can guarantee the system never enters designated bad states, while liveness checking is typically used to check availability.

[1]https://github.com/FormalCellular/EmergencyCall

We elaborated the requirement that emergency calls *should* be routed to correct PSAPs with a *Liveness Property* ϕ_1: *If a user dials a local emergency number in China, the call should eventually be routed to the corresponding PSAP.* It states the basic availability requirement for emergency call systems in both 3GPP protocols [**11, 7**] and telecommunication regulations of China [**101, 102**].

We also detailed a *Safety Property* ϕ_2: *Any call should not be routed to a non-corresponding callee.* It has two implications. First, a call initiated by `Emergency Setup` shall not be routed to non-PSAP destinations. It eliminates the chance of adversaries leveraging emergency call privileges in normal dials. Second, a call made to a normal number shall not be routed to PSAPs. It prevents the possibility that emergency call systems interfere with normal calls.

Step 6: Adaptive Model Construction. We have found that using only the protocols is insufficient to discover or reproduce vulnerabilities in real, deployed systems. In many situations, a pitfall can only be reproduced on certain UEs and carriers. Their specific configurations should be modeled as *model constraints*. A formal model built on protocols is usually broad and lacking these details. Therefore, it is important to augment information from other sources to a general model.

First, it is necessary to locate the key configurations which can affect the "seed" problem. After *Seed Collection* and *Seed Reasoning*, we can locate a couple of key factors. Their assignments are determined by the literature survey, code analysis, or measurement. Next, if the model checking result is non-deterministic on the model, it usually indicates some key variables are missing. We should refine the model further.

Specifically, a UE can be considered to be one instance of the protocols. Following this idea, we analyzed the source files related to the *telephony* functionality from the Android Open Source Project (AOSP) [**62**] and Meizu MX6 ROMs.

Emergency calls have many privileges defined by the protocols, such as authentication-free registration and toll-free. Nevertheless, all of them rely on the configurations of carriers. Some detailed configurations of carriers were acquired indirectly from packet sniffing. We used QxDM [**114**], MTK Catcher [**97**], and MobileInsight [**86**] to sniff packets going between UEs and carriers. For directly testing a particular response, we programmed our UEs to send corresponding requests. For example, we programmed a UE without a valid subscription to test how carriers respond to an emergency attach request. Other configurations were partially inferred from the publicly available documentation by solution providers, *i.e.*, Cisco and Huawei [**42, 41, 69**]. In this chapter, we denote the original model as $\mathcal{M}$, and the adapted model as $\mathcal{M}^*$. More details about model construction are elaborated in the next section.

2.4. Verification of Emergency Call Systems

By performing the verification stage on our model (§2.4.1) and systematically find 4 emergency call failure scenarios similar to the seed (§2.4.2). Moreover, we measure the emergency call system configurations from two major carriers in the U.S. and incorporate them as model constraints. As a result, we find 2 new practical attacks (§2.4.3).

We verified all of the 4 failures and 2 attacks by real-world experiments on commercial UEs and public cellular networks. One of the major Chinese carriers has confirmed these

4 failures. We also reported to the corresponding U.S. carriers about the 2 attacks, and the carriers have acknowledged them.

2.4.1. Stage III: Verification

Step 7: Formal Verification. Model checking was executed with TLC [**152**] on an 8x3.6GHz machine with 64GB of RAM. Verifying the 4 failures took 195, 248, 694, and 328 seconds, respectively, while finding the 2 attacks took 508 and 309 seconds, respectively.

Step 8: Counterexample Interpretation. We took different approaches to interpret the counterexamples found by availability analysis (§2.4.2) and security analysis (§2.4.3). For availability analysis, we kept asserting that ϕ_1 fails. Assignments to configuration and condition variables were refined until we found the root cause of one issue. For security analysis, any violation of ϕ_2 could constitute a potential attack.

Step 9: Testbed Validation. For availability issues, we simply verified them with off-the-shelf UEs and real-world carriers. For potential attacks, we constructed a threat model and evaluated their feasibility using our hardware testbed.

§2.4.2 and §2.4.3 provide more details about the verification stage. In §2.4.2, we use our formal model to systematically study the emergency call availability issues in China. In §2.4.3, we augment system configurations of U.S. carriers as model constraints and discover security vulnerabilities on them.

2.4.2. Availability Study: Routing Failures of Emergency Calls

Our methodology for failure discovery is explained in §2.4.2.1. We found 4 emergency call failure scenarios in China. These scenarios are elaborated and discussed in §2.4.2.2.

2.4.2.1. Failure Discovery. The purpose of availability checking is not just to point out that there exist failures in some scenarios. Rather, it attempts to undermine the essential causes of the failures.

Initially, our model is augmented by the system constraints of carriers in China (*Step 6*). Here is how it looks like:

$$\begin{aligned} o \triangleq\ & \wedge\ \texttt{network\_route\_with\_number\_or\_type} = \texttt{number} \\ & \wedge\ \texttt{network\_emergency\_numbers} = \{110, 119, 120\} \\ & \wedge\ ... \end{aligned}$$

which says the network routes calls based on the callee number instead of the `Setup` message type; the network only recognizes `110,119,120` as emergency numbers.

Besides, the behavior of the model also depends on a set of condition variables, c, which is the abstraction of a scenario. For example,

$$\begin{aligned} c \triangleq\ & \wedge\ \texttt{ue\_sim\_present} = \texttt{False} && (c_1) \\ & \wedge\ \texttt{ue\_screen\_locked} = \texttt{False} && (c_2) \\ & \wedge\ \texttt{user\_dial\_panel} = \texttt{normal\_panel} && (c_3) \\ & \wedge\ ... \end{aligned}$$

The values of condition variables keep unchanged after model initialization. These values contain the root cause of a failure when the model violates properties.

Our initial liveness property, ϕ_1, states that: *If a user dials a local emergency number in China, the call should eventually be routed to the corresponding PSAP.* The strengthened negation of it, ϕ_1^*, becomes: *If a user dials a local emergency number in China, the*

call should never be routed to the corresponding PSAP. If ϕ_1^* is true, ϕ_1 should definitely be false. Checking the correctness of ϕ_1^* has several benefits. *i)* ϕ_1^* is now a safety property, which significantly benefits the execution time of the model checker. *ii)* By checking the safety property, we can avoid finding infinite loops in the model. Trivial loops in some local procedures can thwart liveness checking, *e.g.*, the case that users keep dialing and hanging up; safety checking can bypass such problems. *iii)* Most importantly, only then are we able to find the root cause, which always leads to emergency call failures.

We start by searching for a full assignment to all condition variables, $c = c_1 \wedge c_2 \wedge \cdots \wedge c_n$, and query the model checker on model $\mathcal{M}^*$ for ϕ_1^*. In practice, we can find such an assignment that satisfies ϕ_1^* from our insights. But the assignment is indeed the smallest cube, which leads to a very narrow real-world scenario. Then we attempt to remove a c_k from the current cube c. It can be removed if the cube after removal still satisfies ϕ_1^*. The process terminates when no more condition can be removed. The resulting cube, c^*, which describes an essential condition of a failure, is called *condition core*.

Then the condition core is ruled out from the model: $\mathcal{M}^* \leftarrow \mathcal{M}^* \wedge \neg c^*$. We iterate on the process in the last paragraph to extract the next condition core.

The order of removing c_k can decide the result of the current condition core. However, will the order of removal impact the set of found condition cores? No, because the other condition cores can still be found later, as any state $s \in \neg c^*$ is guaranteed to satisfy ϕ_1^* and violate ϕ_1.

2.4.2.2. Failure Scenarios. We found 4 meaningful scenarios that an emergency call cannot be routed to a PSAP, denoted as ***F-1***, ***F-2***, ***F-3***, and ***F-4***. We have provided a summary of condition cores and their real-world interpretations in Table 2.1. All these

Table 2.1. Four scenarios of emergency call routing failures in China. (found via TLC, verified in the real world)

Failure Scenario	SIM Inserted	Roaming	Localization	Voice Subscripted	Dialed from Normal Panel	When an emergency number is dialed in China, this call would **fail to be routed** to a PSAP if ...	Affected UEs
F-1	✗	-	-	-	-	*No SIM is inserted in the UE.*	All
F-2	✔	✗	✗	-	-	*The UE is not localized correctly.*	Partial
F-3	✔	(✔)	-	✗	-	*The SIM does not have a valid subscription.*	All
F-4	✔	✔	-	(✔)	✗	*The User dials from the emergency panel.*	Partial

"✔", "✗", "-" indicate True, False, no restriction, respectively. "(✔)" indicates no restriction but only a True value makes the case non-trivial.

scenarios are public unawareness, while ***F-3*** and ***F-4*** are unknown to carriers. Note that, we do not claim the search for failure scenarios is exhaustive.

F-1: *A call made in China cannot be routed to a PSAP if no SIM card is present.*

Explanation: Chinese cellular network carriers refuse to route a call with the `Emergency Setup` signaling. In the case that no SIM card is present, UEs will stay in *limited service state* [**6**] and can only provide "emergency calls only" service. Those calls initiated by `Emergency Setup` cannot be successfully routed to PSAPs in China.

Our experiment shows that all GSM/3GPP UEs we have tested fall into ***F-1*** (Table 2.2). We are not able to control the exact carrier a UE attaches to, as no SIM card is present. Therefore, we perform our testing in multiple locations in three different cities.

Public Unawareness: We initially thought ***F-1*** is common knowledge to the public. After seeing discussions online and surveys offline, we believe a vast majority of the public holds the opposite opinion.

This wrong impression might be due to the following reasons: *i)* most people do not have a real experience of dialing an emergency number; *ii)* UEs will show "emergency calls

Table 2.2. ***F-1*:** Availabilities for GSM/3GPP UEs to dial emergency numbers when no SIM card is present.

Number	110/119/120			112/911		
City	Beijing	Hangzhou	Wuhan	Beijing	Hangzhou	Wuhan
Available	✗	✗	✗	✗	✗	✗

only" on screens when users remove their SIMs, which misleads people that "emergency call is available" without SIMs.

F-2 *A call made in China cannot be routed to a PSAP if the UE does not have correct localized configurations.*

Explanation: Initially, UEs could not identify local emergency numbers, *e.g.*, 110/119/120. Hence, when the UE was in a geographic region where the subscribed carrier provides no coverage, the UE could not utilize emergency channels from other carriers. To solve this problem, starting from the era of 3G, 3GPP requires carriers around the world to store emergency numbers of home countries in SIMs [**5**]. Since then, UEs can recognize local emergency numbers when local SIMs are inserted.

This requirement negatively impacted the cellular emergency call system in China. Before that requirement took effect, a call to emergency numbers of China was made through normal `Setup` and could then be routed to PSAPs. However, after UEs identify a dialed number as the emergency number, this call is initiated by `Emergency Setup` and then fails in China. Consequently, UE device manufacturers have to take some compromised solutions, called *localization* in this project.

Localization: We unpacked several ROMs of Android UEs that did not fall in ***F-2*** and investigated their source codes. We will summarize our findings in the following. When

```
<!--Condition: there are following values:
  - 0: ecc only when no sim
  - 1: ecc always
  - 2: show ecc but send as normal   -->
<!-- Add for China CTA -->
<Ecc="110" Condition="2" Plmn="460FFF"/>
<Ecc="119" Condition="2" Plmn="460FFF"/>
<Ecc="120" Condition="2" Plmn="460FFF"/>
<!-- 3GPP 22.101 -->
<Ecc="112" Category="0" Condition="1" />
<Ecc="911" Category="0" Condition="1" />
```

Code 2.1. ***F-2*:** Excerpt of *ecc_list.xml* from Xiaomi Redmi 6A. Condition 2 is enforced when attached to a Chinese carrier (MCC 460) and dialed number 110/119/120 (L6-L8).

the UE identifies that the user is dialing an emergency number of China and the UE is attached to a Chinese carrier, the Android operating system (OS) will display "emergency dialing" on the screen. Meanwhile, the OS will command the hardware to make a call through normal `Setup`.

Different UE device manufacturers have their own implementations of this idea. Code 2.1 is a segment from the *ecc_list.xml* file of the Xiaomi Redmi 6A. The localized Android OS queries this file after the emergency number identification process. As shown in this segment, if the UE is attached to a PLMN[2] in China (MCC 460) and the dialing number matches any of the three entries (L6-L8), the system will enforce *condition 2*, sending normal `Setup` for this call. However, this solution has nothing to do with ***F-1***, because normal `Setup` is disabled when the SIM card is not present.

[2]PLMN (public land mobile network) consists of an MCC (mobile country code) and an MNC (mobile network code), corresponding to a carrier.

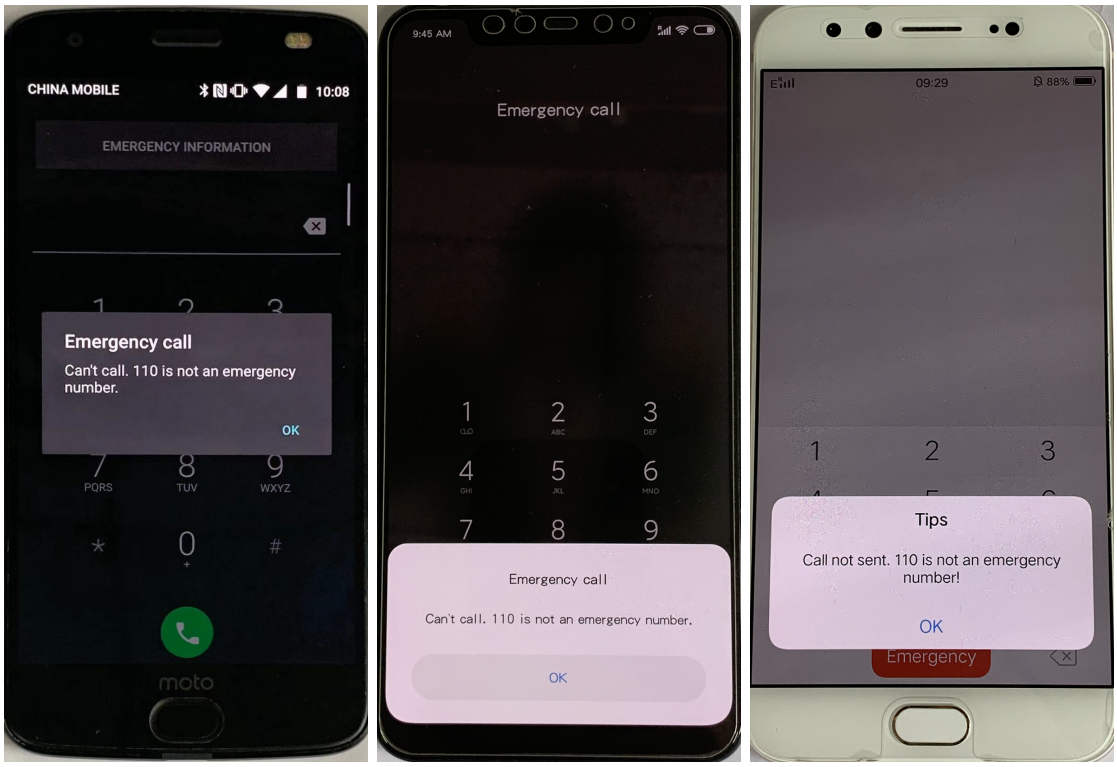
Table 2.3. ***F-3*:** Availabilities for roaming UEs to dial emergency numbers on the normal panel.

Subscription	No				Yes	
SIM issued	US-A	US-T	US-S	US-V	Either US	
Attach To	*Emergency Call Only*			CN-T	CN-M	CN-U
Available	✗	✗	✗	✗	✔	✔

The default AOSP source code does not provide this special modification. Therefore, UEs using default AOSP or making mistakes in implementation (the seed case) are not correctly localized. Emergency calls from them cannot get routed in China. Public news shows that almost all major UE device manufacturers have made mistakes similar to the seed case one after another in the past decade.

F-3 *A call made in China from a roaming UE cannot be routed to a PSAP, if the UE does not have a valid subscription.*

Explanation: If a foreign SIM card is present, it is possible for the roaming UE to pass the authentication and then attach to the network. An emergency call can still not be made as normal call service is unavailable without a valid subscription. It means any users who have not activated their roaming services beforehand have no access to emergency call service in China. This scenario also applies to roaming UEs with the data-only roaming plan [**119**]. As opposed to ***F-1***, roaming UEs fell in ***F-3*** can use keys stored in the SIMs to authenticate partnered carriers in China.



(a) Moto Z2 (b) Xiaomi 8 (c) Vivo x9s

Figure 2.3. ***F-4*:** UEs which cannot identify emergency numbers of China (*e.g.*, 110) when foreign SIMs are inserted.

We tested SIMs from four major U.S. carriers[3] (Table 2.3). Among them, UEs with SIMs from carrier US-V can attach to carrier CN-T's network without activating roaming services. Emergency calls made from none of them can be routed.

F-4 *A roaming UE cannot initiate an emergency call in China by the emergency panel, even with a valid subscription.*

Explanation: In this scenario, a visitor, whose SIM is issued by a country which uses different emergency numbers from China, cannot make emergency calls on the emergency

[3]The four major carriers in the U.S. are AT&T (US-A), Verizon (US-V), T-Mobile (US-T), and Sprint (US-S). The three major carriers in China are China Mobile (CN-M), China Unicom (CN-U), and China Telecom (CN-T).

panel. Even though she has a valid SIM card inserted to avoid ***F-1***; is using a UE with localization to avoid ***F-2***; and has subscribed to roaming service to avoid ***F-3***, the emergency service is still unavailable when she dials without unlocking the screen. ***F-4*** best demonstrates the power of formal methods. It is hard to discover without systematic formal analysis. Yet it is easy to reproduce once found (Figure 2.3).

Reasoning: The purpose of the emergency panel is to allow users to dial emergency numbers without unlocking the screen. Nevertheless, emergency numbers differ from country to country. There are four sources for UEs to determine emergency numbers (Figure 2.4) [**11**]. First, two *fixed emergency numbers*, 112 and 911, are always identified as emergency numbers by UEs. Second, six common emergency numbers are stored in the *UE default emergency number set*. As shown in *Case 1*, if no SIM is present, those numbers are identified as emergency numbers. Third, *local emergency numbers* are stored on SIMs issued by local carriers. When a SIM is inserted, the *UE default emergency number set* will no longer be effective. Fourth, to notify a roaming UE with local emergency numbers, carriers can push the *local emergency number list* when UEs attach to them. Nevertheless, our measurement shows that none of the Chinese carriers push this list, which leads to the failure stated by ***F-4***. More details about the *local emergency number list* are discussed in §2.4.3.

Please notice this failure cannot be mitigated by simply pushing the *local emergency number list*. Otherwise, if roaming users call emergency numbers on the *normal* panel, these calls will be initiated by `Emergency Setup` and thus fail.

Outside China. Similar problems can happen beyond China. Any countries that have multiple emergency numbers or have an emergency number other than 112/911 can also

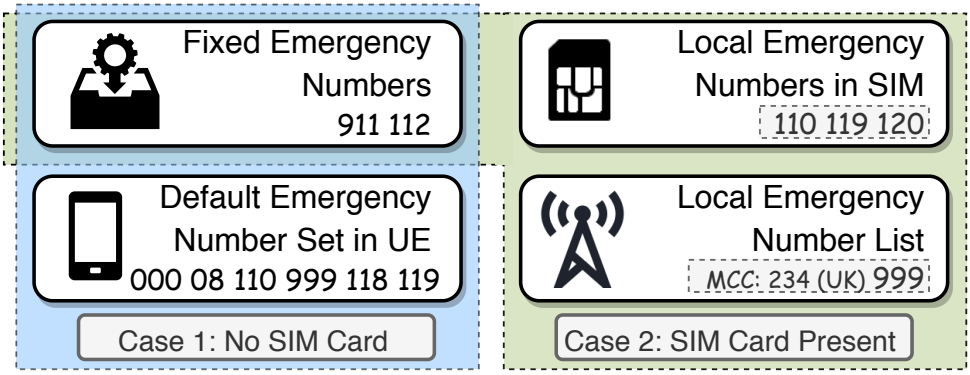


Figure 2.4. Four sources used by UEs to identify emergency numbers. Inserting a SIM card will invalidate the UE default emergency number set.

suffer from this problem. For instance, a thread on a Japanese forum discusses such a case: one cannot dial 110, the police emergency number in Japan, through *SoftBank*'s network [**76**].

2.4.3. Security Study: Abuse of Emergency Call Privileges

We leverage the formal model to discover potential attacks in §2.4.3.1 and define the threat model in §2.4.3.2. We find 2 new attacks. Both of them can have significant impacts on major U.S. carriers. They are introduced in §2.4.3.3 and §2.4.3.4. In §2.4.3.5, we demonstrate ways to deploy these attacks in the real world.

2.4.3.1. Attack Discovery. We augment the general model of emergency call systems, $\mathcal{M}$, with our measured configurations of two major U.S. carriers (denoted as US-I/US-II for anonymity). These two carriers use two distinct sets of configurations when routing calls with `Emergency Setup`. US-I determines the destination of a call only by the dialed number, while US-II routes a call to PSAPs provided `Emergency Setup` initiates it.

To find potential attacks, we assume an adversary who can impersonate one legitimate entity to inject messages within the channel between UEs and the network. Nevertheless, the adversary does not have any capability beyond a budget fake base station. For instance, the adversary cannot route calls from a victim UE to a real-world callee.

We use TLC to check whether those two augmented models satisfy the safety property ϕ_2. In fact, ϕ_2 is violated by both models. For carrier US-I, the adversary can bypass screen locking and SIM card locking to make phone calls. Nowadays, many individuals and companies use the incoming phone number to verify the identity of the caller. The adversary can impersonate the victim by launching this attack. For carrier US-II, the adversary can block calls from US-II's subscribers to any phone numbers in a specific area. These calls will not be routed to correct destinations. This violation can be used to launch a denial-of-service (DoS) attack.

Both of the two sophisticated attacks leverage the *local emergency number list* feature. We can precisely specify this feature in fine grain because we have prior knowledge coming from the reasoning of the seed case and measurement results on each of the two carriers. A general, abstract specification of protocols is unlikely to reveal these attacks.

2.4.3.2. Threat Model. We assume the adversary can set up a malicious base station, *i.e.*, eNodeB, to send sophisticated messages. We will discuss how to achieve this with existing techniques in §2.4.3.5. We also assume the adversary is geographically close to the victim's UE, where the adversary can impersonate the legitimate eNodeB of the targeted carrier by broadcasting messages with higher signal power. The message parameters related to carrier information can be learned by signal sniffing and analysis tools, such as QxDM and MobileInsight. In the UE screen lock bypassing attack, we assume the

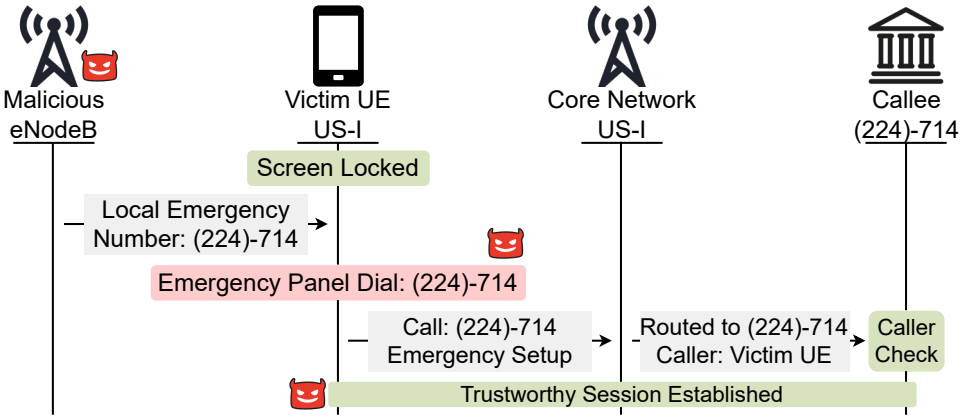


Figure 2.5. ***Attack-1***: UE Screen Lock Bypassing attack on carrier US-I. The adversary can bypass the password of the victim's UE and dial any number on behalf of the victim.

adversary can physically touch the victim's UE, while strong passwords protect both the UE and its SIM card.

2.4.3.3. Attack 1: UE Screen Lock Bypassing.

Attack-1 *The adversary can dial any normal number on the emergency panel of the victim's UE and get routed to the callee, if the UE is a subscriber of carrier US-I.*

Objective of the Adversary: The adversary wants to initiate a normal call from the victim's UE to impersonate the owner of this UE. From the callee's viewpoint, the caller ID (phone number) belongs to the victim. However, when the screen is locked, the UE will block any phone calls it believes not to be an emergency number. The adversary may not be able to simply put the victim's SIM into another compromised UE because either *i)* the victim uses a virtual SIM, or *ii)* a password protects this SIM.

Attack Description: Figure 2.5 shows the attack process. ① The adversary puts the desired number in the *local emergency number list* and pushes the list to the victim's UE through a malicious eNodeB. Possible ways to push this list will be discussed in §2.4.3.5.

② The adversary dials the desired number on the emergency panel without unlocking the UE. Now the OS will accept the adversary dialed number as an emergency number and command the hardware to send out an emergency call. ③ As for carrier US-I, calls are routed base on the dialed numbers. Consequently, this call will be routed to the desired callee as if it is a normal call.

Attack Consequence: Many customer service centers today use incoming caller IDs as the identification of callers. Now the adversary can impersonate the victim on those calls. Besides, financial institutions, as well as other online companies, rely on caller IDs as an important source of two-phase authentication. Now the adversary is possible to get the temporary identification code by making a phone call.

Attack Novelty: This attack can achieve similar consequences as the already known *caller ID spoofing* attack [**57, 49**]. The latter forges the phone number of a trusted caller by falsifying the information transmitted to callees. Nevertheless, the found attack is indeed a different attack, whether in principle or in effect. As for ***Attack-1***, the caller ID shown on the callee side is not forged. Therefore, the found attack can bypass *any* state-of-the-art defense mechanisms for caller ID spoofing, *i.e.*, callee-end defense [**49**] and in-network defense [**130**]. Additionally, the adversary could also receive call-backs from the callee for confirmation. To launch this attack, the adversary will need to access the victim's UE physically. However, to the best of our knowledge, this is the *first* attack that can bypass the screen password and make phone calls.

2.4.3.4. Attack 2: Call Service DoS.

Attack-2 *The adversary can block phone calls made to a set of any phone numbers in a specific area, if the caller UE is a subscriber of carrier US-II.*

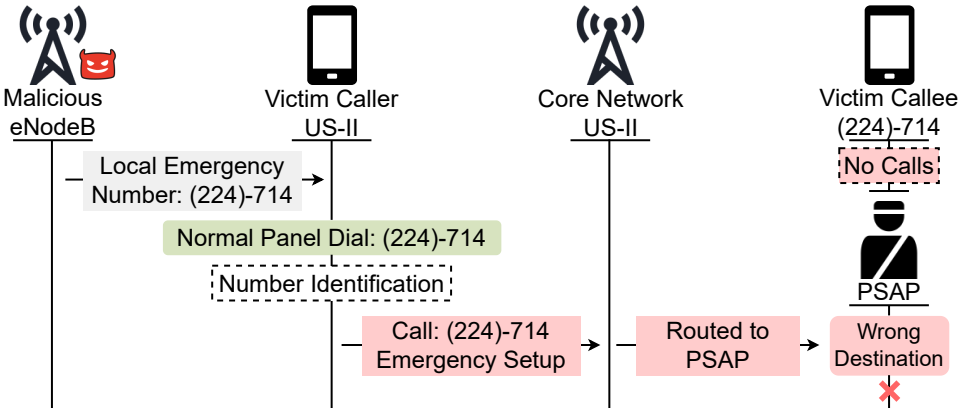


Figure 2.6. ***Attack-2***: Call Service DoS attack on carrier US-II. The adversary can block phone calls to target phone numbers in a specific area. Calls to these numbers will be falsely routed to PSAPs.

Objective of the Adversary: The adversary wants to block phone calls to designated normal numbers within a region. She controls a malicious eNodeB and can broadcast messages with higher signal power. Thus, victims' UEs in that region will attach to her eNodeB. Additionally, she wants to *i)* avoid blocking other numbers, and *ii)* prevent always turning on the eNodeB to reduce the chance of being discovered.

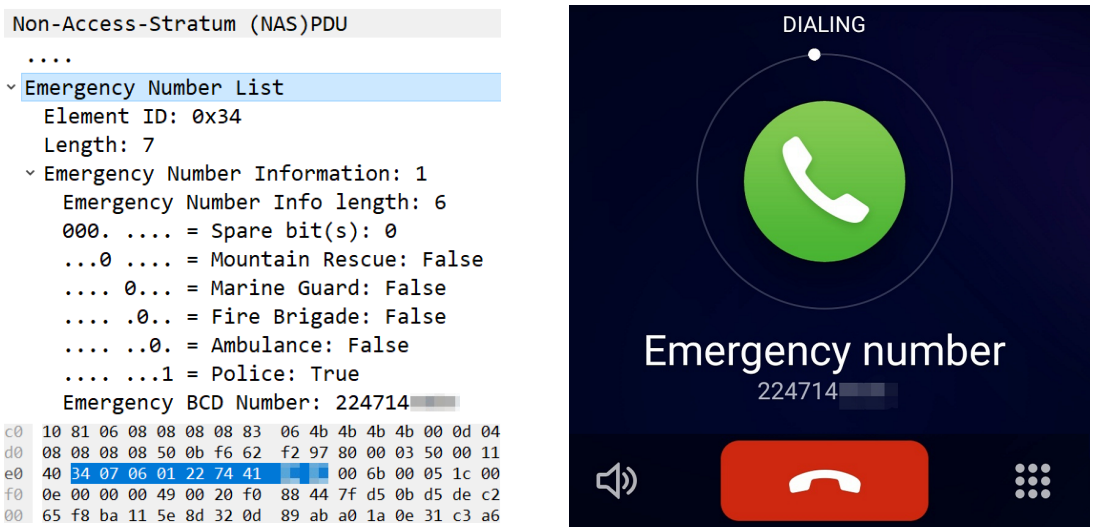
Attack Description: Figure 2.6 presents the attacking steps. ① Similar to *Attack-1*, the adversary pushes the fake *local emergency number list* to the victim's UE. Now the list stores the numbers which the adversary wants to block. ② Emergency number identification takes precedence over any other call-related processes [**11**]. When the victim dials these numbers on the normal panel, the OS will accept dialed numbers as emergency numbers and command the hardware to send them out with the `Emergency Setup` signaling. ③ Unlike carrier US-I, US-II disregards the dialed numbers. All phone calls through `Emergency Setup` will be routed to the PSAP. Thus, in effect, all calls made to these numbers are failed.

Attack Consequence: This attack can be used to obtain illegal economic benefits. For example, the adversary may want to disable phone calls from potential customers to business competitors. In addition, the adversary can get faster service by blocking others' competing calls to that service.

The adversary has an alternative way to leverage this vulnerability. She can broadcast a forged *local emergency number list* with popular numbers stored in it. All phone calls to those numbers from the subscribers of US-II will then be falsely routed to the local PSAP, which becomes a distributed denial-of-service (DDoS) attack to the local PSAP.

Attack Novelty: Comparing with existing call service DoS attacks [**85, 70, 60, 127**], the newly found attack has two different characteristics. First, it stays effective even after the malicious eNodeB is turned off. Second, the newly found attack only blocks the calls to a targeted set of phone numbers while keeps the calls to other phone numbers unaffected.

2.4.3.5. Deployment of Attacks. Background. NAS layer protocols are there to establish and maintain the communication session between the UE and the *core network*. Among those, the session establishment related procedures are called *attach* or *registration*. The *local emergency number list* is an optional Information Element (IE) in the `Attach Accept` message (for 2G-4G) and the `Registration Accept` message (for 5G). Within a *local emergency number list*, emergency numbers, together with their types and lengths, can occupy no more than 50 Bytes. The UE only stores the latest *local emergency number list* it receives from the network, meaning the previous list will be overwritten if a new list comes in [**10**].



(a) Wireshark Log (b) UE Screenshot

Figure 2.7. (a) Wireshark log of the fake *local emergency number list* we pushed. It contains (224)-714-*. (b) UE identifies the normal number (224)-714-* as an emergency number. We are dialing this number on the emergency panel without unlocking the UE.

Carrier US-I does not push the *local emergency number list*. Note the 911, the emergency number in the U.S., is always identified as an emergency number by UEs because it is a 3GPP fixed emergency number (Figure 2.4).

Implementation. We use USRP B210 [**104**] as the eNodeB hardware. It is driven by OpenAirInterface (OAI) [**110**], an open-source cellular network protocol stack emulator. The hardware and software suite supports the essential functionalities of the 4G core network and the eNodeB. The dedicated hardware costs about $1,200. The original OAI lacks implementation of the *local emergency number list*; only a stub interface is provided. We implemented this feature within the `Attach Accept` message. Figure 2.7 (a) shows the Wireshark decoded a NAS message we pushed to our UEs. This message contains the *local emergency number list* IE, which has one fake emergency number, (224)-714-*, in it.

Deployment. Both attacks are relying on the fake *local emergency number list* pushed by a malicious eNodeB. Although non-emergency `Attach Accept` messages are protected by encryption in 4G, ways to set up malicious cellular base stations and push fake messages are plenty [**68, 83, 127**]. However, enforcing any of them may violate the Federal Communications Commission's (FCC) regulation [**56**] and may block real-world phone services. For that reason, to show the proof of concept, we provide a new in-lab solution, called *dual-SIM leakage*, to launch these attacks. Only our controlled UEs will be affected by our malicious signal. Using existing ways of pushing fake messages, a real attacker can launch these attacks more easily. Next, we will discuss these ways.

The dual-SIM leakage of the *local emergency number list* is a vulnerability we find on the dual-SIM UEs. To be specific, the last received *local emergency number list* will overwrite the previous one, no matter which SIM we are using to make calls. Therefore, for our experiment, we utilize it by inserting two SIMs in a UE. One SIM is from the tested carrier (US-I or US-II), and the other is our customized SIM. The malicious *local emergency number list* is pushed on the experimental-licensed spectrum by our eNodeB. Only with our customized SIM can the UEs attach to our eNodeB. The broadcast signal will not affect other UEs. Then we make calls from the commercial SIM. Our experiment follows the setup mentioned in this section and validates the effectiveness of the proposed attacks. Figure 2.7 shows that the UE identifies the normal number (224)-714-* as an emergency number and dials it out without unlocking the screen.

An attacker can use other exposed cellular network vulnerabilities to launch the proposed attacks in the real world. *i) Force 2G fall back:* Lin [**68**] first demonstrated how to practically force a UE redirecting from a 4G eNodeB to a 2G base station. 2G does

not provide network side encryption. The attacker can then push the fake *local emergency number list* to the victim UE. After the UE reattach to a legitimate 4G eNodeB, the fake list is still effective, as carrier US-I does not send a new list to overwrite it. *ii) Force emergency attach:* Yu *et al.* [**151**] proposed a method that can force a UE to set up *emergency attach* procedures. Emergency attach procedures have the privilege to skip authentication. In this case, the `Attach Accept` message is not encrypted, and the *local emergency number list* can be forged into it. *iii) Malicious Wi-Fi and fake DNS:* 3GPP has allowed interconnections with non-3GPP access networks. Specifically, local emergency numbers can be provided through DNS queries within non-3GPP access [**4**]. By setting up a malicious Wi-Fi AP and forging fake DNS responses [**122**], fake local emergency numbers are then pushed to the victim UE. In the emerging 5G, such non-3GPP interconnections can be more prevalent.

Limitations of the Attacks. We noticed a misimplementation on Qualcomm SoCs (System-on-Chips): they truncate an emergency number of more than 6 digits to its first 6 digits. The protocol does not limit the length of each emergency number in the *local emergency number list*. Although most common emergency numbers are 3 digits long, longer emergency numbers are widely used for some special *local* services, such as maintain rescue and marine guard. It is indeed a bug and a violation of clearly defined protocols. However, this bug does weaken the effect of our attacks on those UEs using Qualcomm SoCs. The misimplementation only happens to Qualcomm SoCs. Huawei and MediaTek SoCs do not truncate emergency numbers.

The carrier US-II has *local emergency number list* IE in the `Attach Accept` message with 911 in it; even 911 is a 3GPP fixed emergency number. Because of this setting,

US-II can sometimes escape from the proposed attack depending on the message pushing frequency and mechanism.

2.5. Recommendations

We propose a solution addressing all failures and attacks in §2.5.1 and show its correctness in §2.5.2. In §2.5.3, we argue that lacking regulations or financial stimuli is another factor for the prevalent weaknesses in emergency call systems.

2.5.1. Proposed Technical Solution

We devise a solution consisting of 4 stages. It will be the carriers' responsibility to take these actions. The overhead of the solution is marginal.

① *Pushing Local Emergency Number List.* We suggest that all official local emergency numbers should be included in the *local emergency number list*. Pushing this list to serviced UEs shall be mandatory for carriers. A list containing 3 emergency numbers only takes 12 extra Bytes during *attach* procedures. Upon receiving the correct list, the malicious list pushed by the adversary will be overwritten.

② *Accepting Emergency Setup.* Following the discussion above, all emergency calls that distinguishable by UEs should always raise the `Emergency Setup` signaling instead of the normal `Setup` signaling, to indicate the case of emergency. Hence, it is the carriers' responsibility to handle `Emergency Setup` properly. On the other hand, it is favorable that carriers can route calls to local emergency numbers sent through normal `Setup` properly in cases UEs cannot detect them. Besides, carriers should allow *emergency attach* regardless of the subscription status of UEs.

③ *Emergency Numbers in SIMs.* A traveler just roamed to a new country may not know the local emergency numbers there. Instead, she may dial an emergency number in her home country. If the home emergency numbers are hardcoded into the SIM issued by her home carrier, the UE will deem them equivalently as other emergency numbers. Local carriers can handle these calls properly by following the routing indications from her home carrier [**7**].

④ *Filtering Non-emergency Numbers.* Making an emergency call is a privilege and can bypass authentications of users to UEs or of UEs to networks. Only emergency traffics should be allowed on the emergency channel. Network carriers should apply filters to block other traffics on the emergency channel. A possible way would be binding the filtering rules to the location: only calls made to the fixed, local, or home emergency numbers are allowed to be routed to the corresponding PSAPs, while other calls initiated by the `Emergency Setup` signaling should be rejected.

2.5.2. Correctness of the Proposed Solution

We show the correctness of the proposed technical solution in principle, by our formal model, and by the testbed.

F-1/***F-2***: Now that carriers correctly route `Emergency Setup`, users in these scenarios can access the emergency service. Such an improvement is also backward compatible with already localized UEs because carriers are still able to handle emergency requests initiated by normal `Setup`.

F-3/***F-4***: According to our solution, UEs download the *local emergency number list* when they attach to the network. As a result, the local emergency number identification

is available. Users can now dial local emergency numbers no matter on the normal panel or the emergency panel. Besides, calls to emergency now can be routed to PSAPs, no matter roaming users dial home or local emergency numbers.

Attack-1: The adversary now cannot successfully dial any normal numbers from the emergency panel because of the added filter on the network side. Notice that there are no additional benefits for the adversary to dial an emergency number from the emergency panel.

Attack-2: Pushing the correct *local emergency number list* can overwrite the previously stored malicious list. In addition, the non-emergency filter rejects calls to PSAPs with normal numbers. It solves the potential DDoS threat to PSAPs.

We translated the solution into formal conditions. TLC proved that under these conditions, availabilities of emergency calls are now maintained in the 4 failure scenarios, and the 2 attacks are no longer possible. Please note, a formal specification cannot capture all information of real-world systems, so such a correctness proof is not complete. As a matter of fact, an interruption that happened to the physical layer can cause interruptions on any upper layers.

We also implemented a prototype on our testbed. Under the prototype, none of those availability issues and attacks can still affect the emergency call system. Nevertheless, as our testbed does not have the same capabilities to a real-world carrier, emergency calls on the testbed cannot really be routed to local PSAPs. We are collaborating with corresponding carriers regarding the deployment of the complete solution.

2.5.3. Social Economic Solutions

Cellular emergency call systems are technically complicated, yet this does not explain the extreme prevalence of attacks and reliability issues of these systems. We believe the root cause is the lack of motivation for carriers. Emergency call services are free of charge for end-users, which means the carriers may not put enough effort into testing and improving them. For those users who do not have a valid subscription or their subscribed carrier has no service in that region, it is even impossible for them to accuse other carriers.

We argue that cellular network features, which have high social impacts but make no profits, *e.g.*, emergency calls, shall be seriously considered and clearly defined by protocol designers. It is the social responsibility of the protocol committee to the public. Meanwhile, stronger regulations by authorities are also critical in solving this problem.

2.6. Seed-Assisted Specification Method

This section summarizes all steps in the framework of the *seed-assisted specification* method.

Stage I. Insights

Step 1: Seed Collection. A seed is an exposed issue on a security-critical system. In this step, all relevant information about this issue should be collected, such as the course of events and the circumstance when it happened. It can be collected from sources like official disclosures, news reports, and related protocols. In addition, specific modeling information is of great interest, including the system configurations, initial conditions, and execution procedures that lead to the issue.

Step 2: Seed Reasoning. The related parts in protocols should be looked through to find the execution path that raises this exposed issue. Although a protocol usually suggests a broad implementation and configuration space, the information in *Step 1* can help us to determine those configuration assignments. Sometimes, real-world measurements and investigations are also essential to portray the execution path. Reasoning can also help distinguish the essential procedures causing the issue and how they correlate to the whole system.

Step 3: Seed Reproduction. If the seed reasoning is correct, it would be possible to reproduce the security issue on the testbed. The test environment needs to be augmented with the configuration assignments to simulate the real-world system. If the issue cannot be reproduced following the reasoning, the reasoning result in *Step 2* needs to be revised.

Stage II. Specification

Step 4: Prior Knowledge Specification. With the prior knowledge from *Stage I*, the security researcher can then specify the model, $\mathcal{M}$, in the appropriate level of granularity. Instead of building a model for the whole protocol with all possible details, we suggest limiting the scope to just explore the similar security issues and only expatiate related state transactions. Nevertheless, the specification should follow the protocols and provide flexibility to support all the possible options provided by the protocols.

Step 5: Property Extraction. Model checkers can verify whether $\mathcal{M}$ satisfies a given security property ϕ: $\mathcal{M} \models \phi$. ϕ can be extracted from either the protocols or regulations and should be able to reveal the execution path of the seed issue. In other words, the seed is a violation of ϕ. One can also extract other security properties to find other vulnerabilities.

Step 6: Adaptive Model Construction. We treat real-world system configurations collected from *Step 1* as the observed model constraint, o. The adaptive model $\mathcal{M}^*$ is then the conjunction of $\mathcal{M}$ and o. This step assures the security issues reported by $\mathcal{M}^*$ to be practical for real-world systems. The general specification $\mathcal{M}$ can be reused on other verification tasks to the same protocol by re-applying model constraints with the configurations from other implementations. The benefits of adaptive model construction are more than being accurate and being universal. The construction can also control the size of searching space for model checkers, reducing the execution time of verification.

Stage III. Verification

Step 7: Formal Verification. The verification problem is to check whether $\mathcal{M} \models \phi$ holds. If it does, the model checker returns with no counterexamples. Otherwise, it returns with a counterexample π, which is a trace of state transitions. The *initial condition*, c, can be extracted from the first state.

Step 8: Counterexample Interpretation. Not all counterexamples are feasible and meaningful. To interpret and reproduce a counterexample, it needs a decomposition of the counterexample into procedures, and then needs a close look into every procedure. If a π can be interpreted and reproduced without external intervention, we conclude it is a *failure*. If it is not a failure, but we can assume a reasonable attacker to practice the external intervention, we conclude it is an *attack*.

Step 9: Testbed Validation. We should try to reproduce each failure or attack that is potentially feasible on the testbed. If it is not reproducible, it means either we have mistakes or have over-approximation in our specification, leading to a false-positive counterexample. In both cases, we need to go back to the specification stage to revise the

model and rerun verification. Finally, all failures and attacks reported by the model checker are valid in the real world.

Any systems characterized by human-language-based standards or protocols can benefit from our proposed method because inappropriate granularity and misrepresentation are inevitable in applying formal analysis. Therefore, the proposed method can be generalized to verify other security-critical systems and infrastructures [**33**], such as smart grids [**125**], intelligent transportation systems [**147**], and critical financial services [**52**]. In these systems, even small issues can have widespread consequences. In-depth investigations are always desired, including building formal models, running formal verification, and reasoning about deployed systems to reveal potential vulnerabilities.

2.7. Discussion

Protocols. We limit our study to the GSM/3GPP series cellular network protocols. In reality, 3GPP protocols have become the *de facto* and are the only solution for the 4G and the emerging 5G. The CDMA/3GPP2 series protocols have been announced their ending in the 3G era [**12**]. In the era of 2G/3G, 3 major carriers in China and the U.S., namely CN-T, US-V, and US-S, support CDMA, while all others support GSM/3GPP. Nevertheless, those three have also converged to GSM/3GPP series protocols in the era of 4G/5G and announced to terminate CDMA supports recently [**144, 131**].

UEs. CDMA based networks do not use the 3GPP `Emergency Setup` signaling. We noticed some UEs with only CDMA support can successfully connect to PSAPs in China without a SIM inserted. However, the problem remains for all UEs that are compatible

with both GSM and CDMA networks (include a vast majority of UEs on the market). Details about how the emergency call works for CDMA are out of the scope of this project.

Thanks to the AOSP project, for android UEs, we can investigate the source code to cross-validate the correctness of our findings from measurement and formal verification. The same methodology does not apply to Apple iPhones.

Ethics Concerns. Our work does not present ethical issues as we handle neither personal data nor human subjects. We run attack experiments in a responsive and controlled manner. All UEs and SIMs are under our control. Only UEs with our customized SIMs inserted can attach to our station.

2.8. Related Work

This section summarizes closely related work from two aspects. §2.8.1 introduces related work about applying formal methods to cellular networks. §2.8.2 discusses existing efforts to secure emergency call systems.

2.8.1. Formal Methods on Cellular Networks

Various formal verification techniques have been applied to security research on cellular network protocols and systems. We classify them into three categories.

Model Checking verifies correctness properties by exhaustively traversing the state space. Several previous works [**141, 140, 70**] have examined the security issues in 4G protocols with model checkers. Tu *et al.* [**141, 140**] focused on the reliability problems in protocol interactions. Random sampling was performed over all scenarios to cover a full permutation of usage scenarios in interaction space. Hussain *et al.* [**70**] exploited

vulnerabilities in the NAS procedures by abstracting and modeling NAS protocols. Their framework, *LTEInspector*, does not cover the emergency call systems with proper modeling granularity, and thus cannot find failures and attacks reported by this chapter. Both of them rely on manual model construction, using lots of standard documents as references.

The whole state space may be prohibitively large, especially for those systems involving cryptographic algorithms. The *Symbolic Analysis* employs predefined reduction rules to save efforts in verification. A lot of works [**2, 16, 26, 46**] applied modern symbolic provers, like ProVerif [**30**] and Tamarin [**25**], on AKA protocols used in 3G, 4G, and 5G. Nevertheless, cryptography-related procedures constitute only a small portion of cellular network protocols, and these methods cannot be generalized to other procedures.

Software Analysis aims to directly verify the implementations, as that can save time and efforts of building a model manually. For instance, Pi *et al.* [**112**] extracted binary codes from a Qualcomm baseband and performed static analysis and debugging. Yu *et al.* [**151**] ran software model checking on open-source cellular protocol emulators. However, one implementation is only a single instance of the protocols, so it can not reflect other implementations. In comparison, our approach is based upon protocols. It targets problems on a higher level and can be adapted to many instances.

2.8.2. Security of Emergency Call Systems

Emergency call systems have many privileges; they also have large impacts on society. Nevertheless, to the best of our knowledge, there is currently no work that formally analyzes the correctness or finds vulnerabilities of emergency call systems on either their designs or implementations.

Authorities usually make orders and standards to enforce local carriers to provide emergency call services. For example, FCC, the communication authority of the U.S., has issued orders [**54, 55**] to specify the requirements of wireless 911 calls. Ministry of Industry and Information Technology, the communication authority of China, has also published industry standards [**101, 102**], requiring the connectivity of emergency calls under the no-SIM condition. These documents, however, are more concentrated on functionalities than the security aspects of the system. Besides, these documents may state at a very high level, becoming ambiguous and incomplete.

Not much research literature focuses on emergency call systems. RFC 5096 [**135**] summarized the security threats that cellular emergency call systems might encounter in a conceptual manner. Those threats include leakage and falsification of location and personal information, as well as abuse of anonymity and priority privileges. However, no concrete attacks or defense approaches are discussed in it. The chance of the DDoS attack on 911 services by leveraging the anonymity privilege has been mentioned in [**65, 109**]. Based on the estimation in [**65**], with 6,000 bots, 911 emergency services in a U.S. state can be blocked for a whole day. Rebahi *et al.* [**118**] proposed an attack in the current 3GPP's scheme that an adversary can impersonate PSAPs.

The wireless emergency alert (WEA) system, also known as the public warning system (PWS) or the earthquake and tsunami warning system (ETWS), broadcasts alert to all UEs in a geographic area. This system is not within our research scope. It is worth to mention that the message authentication of WEA has been discussed for years [**3**]. However, this feature has not been fully settled in protocols even today, leading to multiple fake alert attacks [**83, 70**].

2.9. Chapter Summary

This work concentrates on how to use formal methods on cellular networks. In particular, we systematically explore availability and security pitfalls in cellular emergency call systems. We demonstrate in the chapter a novel way of specification, called *seed-assisted specification*, which can be applied to systems described by protocols in general. We emphasize the importance of prior knowledge in building the model, and we explain how it helps determine the critical processes and the granularity of the model. Then we describe how to integrate measurement results with a generalized formal model, such that a variety of scenarios can all be verified on real systems. From formal verification, we find 4 scenarios in China that emergency calls cannot be routed to PSAPs. Meanwhile, we find 2 new attacks in the U.S. that abuse emergency call privileges. We propose a unified solution for carriers. It can address the problems we have discovered and any similar problems we can foresee.

CHAPTER 3

Accelerating and Securing Serverless Cloud Networks with the QUIC Protocol

3.1. Introduction

In this project, we aim to design and implement a QUIC-based serverless computing framework, which can improve the serverless network performance and provide security to internal communications.

The rapid evolution of lightweight virtualization technology has spawned the rise of the serverless cloud computing paradigm [**24, 75, 36**]. In serverless computing platforms, cloud providers assume responsibility for all server-related management tasks, including both hardware resource allocation and software runtime preparation. Cloud software developers are thus free to simply focus on designing small discrete stateless functions and orchestrating them together for their high-level business logic.

Among the major allures of serverless computing is *agile autoscaling*. It allows service providers to quickly launch new function instances in response to end-user requests, while saving operational costs. Since auto-scaled instances can be quickly destroyed by cloud providers, tenants only pay for the actual function execution time and do not need to reserve resources for burst requests. Because of both efficiency and economic advantages, serverless computing garners extensive attention from industry and is expected to become

the dominant cloud computing paradigm [**75**]. Its market share is projected to surpass $21 Billion by 2025 [**93**].

Serverless computing is fundamentally a *network-based* cloud computing paradigm. Thus, optimizing the performance and security of serverless networking is arguably as crucial as existing research efforts on other aspects such as performance optimization of serverless platforms [**13, 149, 34, 128, 133, 74, 108, 146**] and security management of cloud functions [**124, 15, 47**].

Furthermore, the *zero trust* security model has gained considerable momentum among cloud security communities [**94, 121**]. Under this principle, any entities, even within the same internal network, should not be trusted by default. Therefore, fully encrypting all internal connections is now the best practice for major cloud providers [**21, 61, 99**]. Although initiating reliable transportation and encryption introduces extra delays, it is not the dominant performance bottleneck in traditional cloud computing: *i*) transmission delay within the data center is negligible compared to execution times; *ii*) connection setup latency of TCP and TLS can be simply mitigated by using persistent connections.

However, many leading commercial serverless providers and open-source serverless frameworks still use bare (unencrypted) TCP connections between functions, leaving a potential attack surface [**13, 23, 100, 63**]. This is due to new challenges that arise specifically in serverless networks. First, with serverless computing, a function instance can be initialized in milliseconds (less than 125 ms for cold-start on AWS [**13**]) and only processes a small sliver of the computational task (849 ms median execution times on Azure [**126**]). The latency introduced by TCP and TLS handshakes, even at the sub-millisecond-scale, should no longer be ignored [**74**]. Second, with the *scale-zero-to-infinity*

feature [**36**], function instances are quickly scaled up and down by cloud providers. It is thus tough to maintain persistent connections between ephemeral functions. Third, as serverless functions are commonly chained together to form task-specific workflows [**124**], cumulative handshakes exacerbate the end-to-end latency.

We raise the following question in this project: *Can we seamlessly enable secure and accelerated network communications for serverless cloud applications?* To address this question, we design and implement a novel solution based on the emerging QUIC protocol, called QFaaS, which can simultaneously improve performance and provide security to existing serverless platforms without the requirements of any tenant code modification.

QUIC [**115**] is a new transport protocol that has steadily gained popularity in wide-area Internet traffic [**145**], particularly for web and video streaming applications [**82**]. QUIC combines the advantages of both TLS 1.3 and UDP to provide a secure and reliable transport layer with 0-RTT (round-trip time) connection setup cost, *i.e.*, data packets may be sent without an explicit handshake. QUIC has also been successfully extended to some other scenarios, such as IoT meshes [**51, 80**], satellite communications [**139**], and Tor transports [**27, 28**]. Due to the inherent advantages of reduced handshake costs while providing a secure network, it is appealing to adapt this new protocol to securely address communication performance bottlenecks in emerging serverless networks.

The rest of this chapter is organized as follows.

In §3.2, I introduce the background of serverless computing and connection encryption, followed by the new challenge in serverless networks, connection setup latency.

§3.3 proceeds by first providing a network-centric view of serverless applications, filling in missing details about actual network flows in the widely used logic view. This inspired the design of our QFaaS system.

The QFaaS system design is presented in §3.4, where the QUIC protocol can be seamlessly integrated into serverless platforms, to mitigate connection setup overheads and provide secure communications. Our design explicitly ensures that existing serverless applications can be migrated to QFaaS without any code modification. In addition, serverless applications can be further accelerated by using our function chain library and always-on 0-RTT design.

In §3.5, we implement the QFaaS prototype into OpenFaaS, the most popular open-source serverless platform. The entire system code will be made publicly available. And the system is designed to be easily extensible to contemporary commercial and open-source serverless platforms.

Our experimental highlights in §3.6 include: *i)* QFaaS can reduce the single function and function chain response latency by 28% and 40% respectively compared with the state-of-the-art serverless platforms. *ii)* Upon deploying a real-world serverless application to QFaaS, the end-user response time is reduced by 50 ms. *iii)* In certain scenarios, QFaaS was even faster than other platforms using only insecure TCP connections.

Related work about serverless computing and QUIC is presented in §3.7.

Chapter summary, presented in §3.8, concludes this chapter.

3.2. Background and Challenges

Serverless computing and cloud computing connection encryption are introduced in §3.2.1 and §3.2.2, respectively. In §3.2.3, I discuss the connection setup latency as the new challenge in serverless networks.

3.2.1. Serverless Computing

In serverless computing, traditional applications are decomposed into small code slices, called *functions*. These *stateless* functions then can be orchestrated by tenants to perform their high-level business logic, which is also known as the *function-as-a-service* (FaaS) model. In comparison with the *infrastructure-as-a-service* (IaaS) model, in FaaS, it is no longer incumbent upon the tenants to manage the life cycle of virtual machines (VMs) or the deployment of software stacks. Cloud platform providers undertake all server-related tasks, such as launching VMs, provisioning container clusters, and preparing programming runtimes. From the tenants' perspective, the cloud development and deployment tasks are not server-centric, and thus called "serverless".

The rapid evolution of virtualization technology, especially container technology, is the basis for the emergence of serverless computing. The FaaS model introduces a novel capability to cloud computing: agile auto-scaling without explicit tenant provisioning. Specifically, stateless functions are usually deployed in lightweight virtualized environments, such as containers. They can be initialized within just a few milliseconds. Therefore, serverless providers can quickly scale the number of running function instances in response to changes in incoming request patterns, in a manner that is automatic, continuous, and completely transparent to tenants. This feature also provides certain inherent

economic advantages. Since the allocated resources can be released soon by the cloud providers when they are not in use, tenants will not be charged for idle time and only pay for actual function execution time, *i.e.*, *billing-based-on-usage* model. In effect, the price of handling one thousand burst requests in parallel is roughly the same as the price of handling one thousand requests at low density.

Meanwhile, *backend-as-a-service* (BaaS) is another important component of serverless computing [**75**]. Cloud providers can provide *stateful* services, such as data storage, event logging, and identity management, to cater to the rapid development of serverless applications.

3.2.2. Connection Encryption for Zero Trust

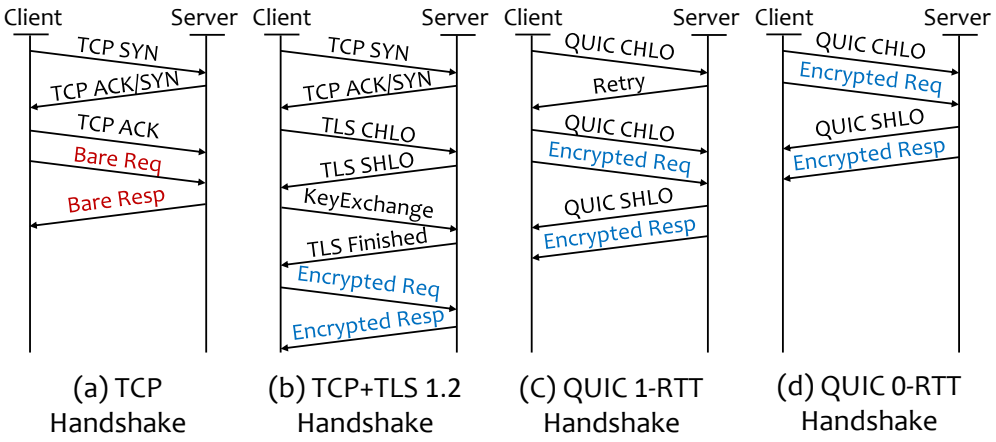
Cyber threats emanate not only from tenacious attackers outside the data center but also other insidious entities sharing infrastructure within the same data center. Due to the lack of encryption on internal communication, early-stage data centers exposed extra attacking surfaces to adversaries, leading to several publicized security disasters [**117, 78**]. Consequently, the zero trust security model came into being. In the zero trust model, no entities should trust each other by default; hence authentication and encryption are always desired. This model has gained popularity in most cloud computing paradigms, such as IaaS and newly-emerging microservices, which dedicates TLS encryption to all connections to be the best practice [**21, 61, 99**].

Nevertheless, even in those leading commercial serverless platforms, due to the network performance restriction (detailed in §3.2.3) and the early stage of development, the lack of traffic encryption between internal serverless communications is still the status quo.

For instance, dedicated traffic encryption is provided by default to most services within AWS data centers [**20, 21**]. However, for the Lambda serverless computing service [**18**], AWS only provides traffic encryption in connections between end-users and the Lambda function invoker. The connections between the function invoker and function workers remain unencrypted [**23, 13**]. Other popular commercial providers, such as Google Cloud and Azure, also do not encrypt the function invoker to function worker connections. In addition, Azure does not require encryption in the gateway to function invoker connections [**100**] and Google Cloud even allows end-users to trigger serverless functions through the gateway via insecure HTTP requests [**63**].

Traffic encryption is also not prevalent in open-source serverless platforms. For example, OpenFaaS [**88**], the most popular open-source serverless platform, disables all traffic encryption by default. Users must manually enable TLS 1.2 encryption which significantly impacts its network performance, as we show in the evaluation (§3.6).

Virtual Private Cloud (VPC) is the prevalent solution for IaaS network security. It provides a virtual isolated networking environment in public clouds. However, VPC has several drawbacks when applied to serverless platfroms. First, the initialization performance of VPC cannot meet the rapid scaling requirements of serverless compute environments. For instance, the initialization of AWS VPC interfaces takes 15 to 90 seconds, and this cost has to incur per cold-start serverless function call. With this realization, AWS disables tenant VPC for Lambda by default, and *hyperplane VPC interface sharing* was recently announced; however, that still incurs a one-second overhead [**19**]. We make the case that such delays are prohibitive for serverless functions that initialize and execute at millisecond scales. Additionally, the economic advantages of serverless computing come



Scheme	TCP	TCP + TLS 1.2	QUIC 1-RTT	QUIC 0-RTT
New Session	1	3	1	-
Recover Session	1	2	1	0

Figure 3.1. **Round-trips incurred by different transport protocols: (a)** insecure TCP incurs 1 extra RTT; **(b)** in TCP+TLS 1.2, the encrypted request is sent after 3 RTTs; **(c)** in QUIC 1-RTT mode (new session establishment), the encrypted request is sent after 1 RTT; **(d)** in QUIC 0-RTT (session resumption), the encrypted request is sent immediately.

from efficient hardware multiplexing among tenants. However, exploiting this architectural flexibility limits opportunities for pre-binding tenant VPCs to hardware resources. Finally, VPC is arguably targeted more toward traffic *isolation* than *encryption*. Specifically, cloud providers typically enforce traffic encryption between VMs (*e.g.*, AWS EC2) in the same VPC but not other services due to the hardware and design restrictions [**21, 61**]. Therefore, VPC is not the off-the-shelf solution for serverless encryption.

3.2.3. Connection Setup Latency: New Challenge

Internal communication delay was not a significant problem in traditional monolithic applications. Threads within the same process shared the same view of virtual address

spaces and inter-process communication (IPC) provided convenient mechanisms (*e.g.*, signals, pipes, and shared memory) for different processes to exchange data efficiently. Cloud computing paradigms increased flexibility in application design and deployment, by breaking up monolithic application stacks into independent services. Nevertheless, there was often some additional cost associated with such flexibility.

In practice, disparate services are commonly deployed in isolated VMs for ease of management. Internal messages between them now must go through a complete network stack, raising communication latency. Such distributed service orchestration also introduces connection setup latency. Specifically, TCP and TLS are used to provide reliable and secure connections between VMs. Both of them require extra round-trips when initiating new connections, as shown in Figure 3.1 (a) and (b). But such initialization delays were not a severe drawback in cloud computing until the advent of serverless computing. First, maintaining persistent connections among services can partially mitigate the connection setup latency. Though this solution cannot eliminate the delay after launching a new VM, the connection setup latency is still negligible when compared with the VM initiation time. Second, TCP, the dominant protocol for reliable network communication, has been ossified into the OS kernel and is not easily replaceable. Thus, other problems in cloud computing, like scheduling, elastic scaling, and storage, were prioritized over optimizing connection setup latency.

In contrast, the *scale-zero-to-infinity* feature of serverless exponentially magnifies the disadvantages of connection setup latency. To be specific, in serverless computing, *i)* since the initiation and execution times for function instances are minuscule, the connection setup time is no longer negligible. *ii)* In addition, the number of running function instances

rapidly scale up and down in response to the request changes. It is now not possible to maintain persistent connections between these stateless function instances. *iii)* Finally, as functions are usually chained together to form task-specific workflows, connection setup costs are incurred at each hop of the function chain, significantly compounding the non-negligible delay. To address these challenges, always keeping at least one instance of each function alive could be a compromise solution (which became an option on AWS Lambda recently). However, such a solution largely increases the tenants' cost and violates the serverless philosophy to some extent. Meanwhile, it still suffers from burst requests. We believe that there is now a greater urgency to prioritize the optimization of connection setup latency in cloud network communications.

3.3. Modeling Serverless Networks

To identify potential network bottlenecks in serverless computing, the first challenge is to identify network connections in its architecture. Though the logic abstract model of serverless platforms is commonly used, important details were missed with respect to network modeling (§3.3.1). We address the limitations by providing a new abstraction of the serverless architecture through the network-centric view (§3.3.2). This model will guide our QFaaS system design.

3.3.1. Logic Model

General discussion about serverless architectures is commonly framed in the context of the logic view, shown in Figure 3.2. Under this abstraction, a unified API Gateway continuously listens for end-user requests. Upon receiving a function invocation, Gateway first

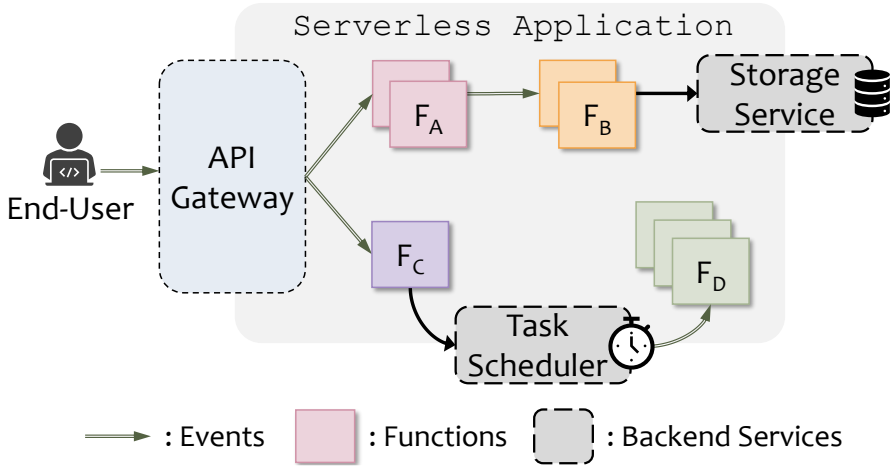


Figure 3.2. **Serverless architecture in logic view.** Gateway forwards end-user requests to corresponding functions. Functions chained together with backend services compose a serverless application. However, network details are missing: *i)* function results are returned through Gateway, *ii)* functions are chained through Gateway, *i.e.*, no direct connections between end-users and ephemeral function instances, and between two instances.

executes permission authentication and scales corresponding function instances. Gateway then forwards the user request to a function instance behind it. Functions chained together with backend services compose an integrated serverless application and perform cloud tenants' business logic.

While the logic model largely simplifies the connection details, such that one can quickly understand the essential concepts of serverless computing, it does not reflect actual network flows. There are two important details missed: *i)* after the end-user sends a function request to Gateway, the response of this function (F_A or F_C) is returned through Gateway instead of directly by the function; *ii)* in function chains, when a function (*e.g.*, F_A) sends a request to another function (F_B), this request must also go through Gateway. Because only Gateway can launch new instances of functions, and knows the destination address of their running instances.

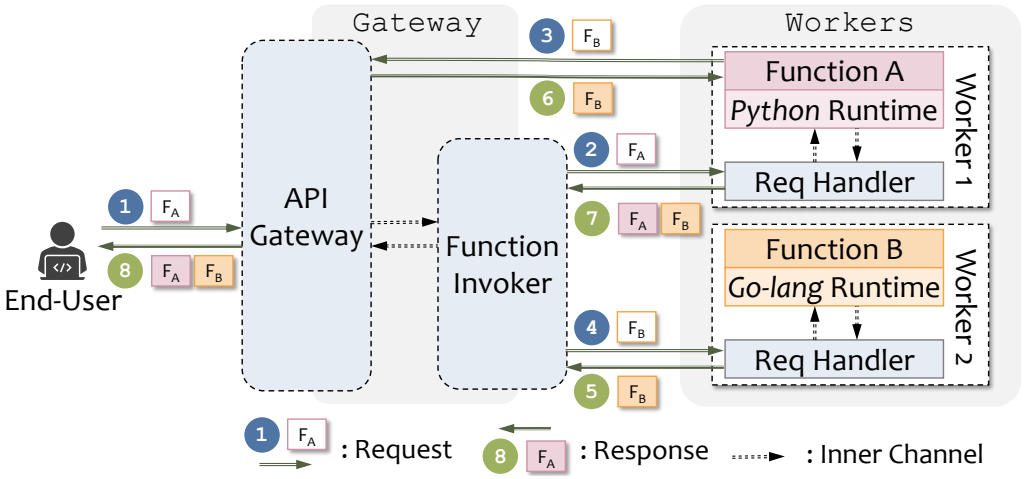


Figure 3.3. **Serverless architecture in network-centric view.** Gateway components run permanently, expose function interfaces, manage running workers, and dispatch requests. Workers run on ephemeral containers, host request handler and different language runtime for functions. The request handlers can only be connected by Gateway.

3.3.2. Network-centric Model

To address the aforementioned limitations, we provide a new abstraction of the serverless architecture through the network-centric view (Figure 3.3). In this model, the serverless architecture is divided into two parts: the gateway subsystem and the workers subsystem.

- Components in the gateway subsystem expose static function interfaces to end-users, manage running workers, and dispatch requests to corresponding functions. These services are all *stateful* and run on *permanent* machines. In existing serverless platforms, corresponding modules may have variant names. For example, in AWS, they are called frontend and worker manager; in OpenFaaS, they are called `api-gateway` and `faas-netes` controller. Regardless of the names, they provide the same functionality.

- Workers are *ephemeral* containers that comprise the request handler, the function runtime, and tenant functions. The request handler provides the internal communication ability for workers. It receives trigger requests from Gateway and sends function results back to Gateway. The function runtime provides isolated software stacks and programming language libraries to execute tenant functions. Therefore, tenant functions are decoupled from the management of ingress network connections. Hence, the request handler can be designed independently by serverless providers.

Figure 3.3 shows an example where an end-user requests the service of function F_A, while F_A chained together with F_B provides the service to the end-user. In this example,

- (❶—❽) the end-user sends F_A a request (❶) and receives responses of F_A—F_B (❽) from the direct connection with API Gateway. In the process, Gateway acts as a transport layer *server*, listening and responding to user requests. Message flow details behind it are transparent to the end-user.
- (❷—❼) API Gateway forwards the F_A trigger event to Function Invoker. After the F_A worker is initialized, Function Invoker sets a connection to the request handler in F_A worker, sends request data (❷), and receives responses (❼). In this process, Function Invoker plays the role of the transport layer *client* to initiate this connection. The request handler plays the role of the transport layer *server*.
- (❸—❻) F_A needs the response of F_B. Nevertheless, instead of sending a request directly to a worker of F_B, F_A will send the F_B request (❸) and receive responses of F_B (❻) from Gateway. F_A does not need to care about any scheduling details

of F_B. In this process, Gateway acts as a transport layer *server* again, even though the connection is internal.

- (❹—❺) Function Invoker initializes a worker for F_B, sets up a connection to its request handler, sends request data (❹), and receives responses (❺).

In current serverless platforms, HTTP (including REST-API and gRPC) is commonly used application layer protocols for connections to API Gateway (❶—❽, ❸—❻) and request handlers (❷—❼, ❹—❺). These application layer protocols rely on TCP and TLS protocols underneath to provide reliable and secure transport communications. For security concerns, connections involved API Gateway (❶—❽, ❸—❻), which exposes interfaces to outside, are mandatorily encrypted by most providers (§3.2.2). For other connections in Figure 3.3, the data exchange between request handlers and functions in the same worker is through IPC with negligible overhead. API Gateway and Function Invoker are usually deployed in different machines. But they can maintain a persistent connection to mitigate the connection setup overhead.

3.4. QFaaS: System Design

QFaaS leverages the emerging QUIC protocol to accelerate and secure serverless computing. §3.4.1 introduces QUIC and emphasizes its benefits for serverless. The QFaaS system architecture is described in §3.4.2. It requires no code modification for existing serverless applications. Designs in §3.4.3 and §3.4.4 further accelerate serverless networking.

3.4.1. QUIC Protocol for Serverless Networks

QUIC has been quickly and widely adopted in the wide-area Internet after demonstrating the ability to mitigate several drawbacks of TCP (*e.g.*, performance, evolvability). After 2017, more than 7% of Internet traffic (a major part of Google's egress traffic) is under QUIC [**82**]; in 2021, 5.1% of all websites over the world are using QUIC [**145**]. QUIC has been standardized by IETF (Internet Engineering Task Force) in RFC 9000 in May 2021 [**115**]. And IETF is working on "HTTP over QUIC" towards the next-generation HTTP, HTTP/3 [**29**]. With rising demand for low latency applications, QUIC gains growing popularity and is likely to outpace TCP on the Internet in the near future.

We think that QUIC can likewise evolve communications in serverless computing as it provides a robust pathway to improve security and performance. On the one hand, cloud users seamlessly benefit from the security of QUIC as it is coupled with the latest TLS 1.3 protocol to provide always-on encryption by design. On the other hand, QUIC can achieve 0-RTT shaving both transport and cryptography handshakes, meaning the first encrypted data packet could be sent before any handshake happens. First, QUIC mitigates the handshake overhead in the TCP protocol, as it provides a reliable multiplexing transport on top of UDP instead of TCP. Second, QUIC further leverages the 0-RTT resumption feature in TLS 1.3. Consequently, QUIC only requires 1 extra round-trip (1-RTT mode) to set up the connection when the client never connected to the server before (Figure 3.1 (c)). The first encrypted data packet can be sent immediately (0-RTT mode) if the client cached the server information in previous connections (Figure 3.1 (d)). Thus, QUIC has the potential to greatly reduce the connection setup latency in serverless computing,

especially when new function instances are instantly scaled up and chained together to support burst requests.

We notice that QUIC is even better suited for serverless computing environments than the Internet in some respects. First, enabling QUIC requires modifications on both client- and server-side software to install relevant libraries with compatible versions, which is challenging when the two sides are controlled by different entities. In serverless computing, as cloud providers prepare all the software stacks, including networking-related stacks, software compatibility is no longer an issue. We can further leverage this capability to ensure the always-on 0-RTT (§3.4.4). Second, the 0-RTT replay attack is another concern when using QUIC on the Internet [**35**]. If it is possible to monitor connections and sniff packets in the middle, the adversary can potentially resend the first client packet to trigger the related request twice on the server-side. Though performing this attack requires strict conditions, which are harder to achieve inside a data center, we further implement a more secure QFaaS prototype by sending all non-idempotent requests through 1-RTT to mitigate the threat of the 0-RTT replay attack (§3.5.2).

In addition to security and low latency, using QUIC in serverless networks can also provide many of the same benefits as using it on the Internet. For example, QUIC supports stream multiplexing within one connection. It avoids head-of-line blocking delay due to the TCP's sequential delivery. Besides, as QUIC runs in the user space instead of the kernel, the transport layer is now more malleable to meet evolving application demands with frequent updates, such as a notable recent breakthrough: *Pluginized QUIC* [**48**].

TCP Fast Open (TFO) [**148**] is a potential competitor of QUIC [**37**]. It allows the application data attached in the client SYN packet to avoid the TCP handshake latency

if the SYN packet contains an identifier (TFO cookie) from the last connection. Though TLS 1.3 (0-RTT) over TFO provides the same theoretically round-trip performance as QUIC, because of several deep-rooted privacy and performance flaws, TLS over TFO has been disabled on all modern browsers (e.g., Chrome, Firefox, and Edge) and is not yet actively used by most popular operating systems after 10 years [**134**]. First, TFO relies on an unencrypted unique cookie in the TCP header, which leads to severe tracking concerns on the public Internet. In addition, enabling TFO requires updating all middleboxes in data centers, such as firewalls, proxies, and security devices, to support a non-originally designed TCP option. Nevertheless, these network core devices are ossified in the network and rarely updated. Consequently, failed TFO requires an ordinary TCP SYN retry, leading to TFO actually increasing round-trips. In contrast, QUIC runs over UDP and only requests updates on end devices. Besides, only the first data packet (MTU) will benefit from the TFO, while the whole client initialized messages can benefit from the QUIC 0-RTT feature. Therefore, we use QUIC instead of TFO for QFaaS.

3.4.2. QFaaS System Architecture

We first identify connections that affect the serverless network performance and can be seamlessly optimized without tenants' code modification. In our network-centric view, API Gateway to Function Invoker connections and request handler to language runtime connections could be persistent or through IPCs. The connection between end-users and Gateway (❶—❽) is initialized by end-users and could also be persistent. The connection

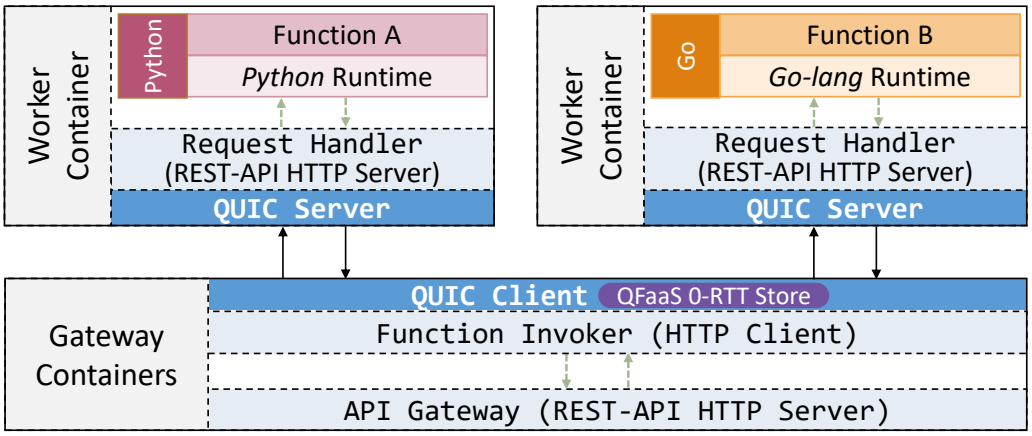


Figure 3.4. **System design of QFaaS.** QUIC client and QUIC servers are integrated into Function Invoker and worker request handlers to replace the TCP/TLS client and servers. This modification is transparent to cloud tenants and ensures the activation of the QUIC 0-RTT feature.

from the function to Gateway (❸—❻) is initialized by functions. In contrast, the connections from Gateway to workers (❷—❼, ❹—❺), which are fully controlled by providers, expose opportunities to optimize serverless networks.

First, function workers are instantaneously launched and teared down in response to requests. We cannot simply use persistent connections to mitigate the connection setup latency. Second, this overhead will be multiplied when functions are chained together or the number of running instances is quickly scaled up. Thus, these serverless-introduced bottlenecks drive major cloud providers to sacrifice security for performance, keeping ❷—❼, ❹—❺ unencrypted (§3.2.2).

To accelerate serverless networking while maintaining security, we introduce the design of QFaaS, as shown in Figure 3.4. In this design, we integrate the QUIC client into Function Invoker and also integrate the QUIC servers into the worker request handlers (to replace the TCP and TLS client and servers, respectively). All function requests that

go through Gateway to workers would now benefit from the efficiency and security of the QUIC protocol.

On the one hand, QUIC embraces full encryption by default and employs the latest TLS 1.3 protocol. As a result, connections from Gateway to workers would benefit from security improvements provided by TLS 1.3. On the other hand, this QFaaS design can ensure the activation of the QUIC 0-RTT feature. To enable 0-RTT connection resumption, QUIC leverages *QUIC connection tokens* and *TLS session caches* stored on the client-side. In serverless architecture, Gateway runs on stateful machines and plays the role of the QUIC client in QFaaS. We integrate the *QFaaS 0-RTT Store* into the Gateway to maintain and manage connection-specific information. Therefore, serverless applications under this design can further benefit from the 0-RTT feature.

Moreover, QFaaS design does not request any changes to tenants' function code. In serverless computing, all running containers, as well as the code of Gateway and request handlers, are provided and controlled by cloud providers. Modifications are totally transparent to cloud tenants and end-users.

3.4.3. Function Chain Library

An inquisitive reader might wonder why we did not further replace the connection initiated from the function to Gateway (❸—❻) with QUIC? This is because such a connection is function code related and programming-language specific. Specifically, Gateway usually exposes URLs or REST APIs for functions. End-users can invoke a function by sending an HTTP request to the corresponding URL. Similarly, when a tenant wants to invoke a

function (F_B) by another function (F_A) to form a function chain, the tenant also must initiate an HTTP request by the function code in F_A and follows the programming language practices. For example, AWS, Azure, Google Cloud, and OpenFaaS all suggest Python users form function chains by leveraging the Python `Requests` library [**77**]. Therefore, such connections are not fully controlled by the cloud providers and cannot be optimized as described in §3.4.2.

One alternative way to enable QUIC at ❸—❻ is to provide QUIC server at Gateway and ask developers to integrate a QUIC client in their function code. Nevertheless, this design requires significant code modification and also requires developers to be familiar with QUIC client configurations.

Recent advancements in platform specific libraries allows for an improved design. For fine-grained access control and ease of use, some serverless platforms, such as AWS Lambda, now provide libraries under different languages for tenants to form function chains [**22**]. With such libraries, developers can directly call the platform API to invoke another function instead of explicitly sending an HTTP request by code.

Leveraging this idea, we provide a QFaaS function chain library to enable QUIC at ❸—❻, requiring slight tenant code modification. This chain library has QUIC as its underline transport layer protocol. We integrate the QUIC server into Gateway to accept function requests through QUIC. Thus, all function chain traffic invoked by the library will benefit from the short latency and security of QUIC. Currently, this library supports Python3 and Go-lang, which are two popular programming languages used in all major serverless platforms. The code modification to adapt this library is minimal. For instance,

the Python developers only need to import the library and switch their `Requests` call to the QFaaS chain library call, which are only 2 lines of code modification.

This design also has three side benefits. First, Gateway now has the ability to accept QUIC requests. It is now possible for end-users to initiate a request by QUIC and further accelerate the ❶—❽ connection. Second, this new ability will not interfere with existing TLS Gateway functionalities, as QUIC listens on the UDP port while TLS listens on the TCP port. Third, the system can now be more easily integrated with current serverless security and access control mechanisms [**15, 47, 124**].

3.4.4. Always-on 0-RTT QUIC

QUIC is initially designed for the wide-area Internet, where connection peers are controlled by different entities. On the contrary, in serverless networks, providers can fully administrate the platform. Leveraging this ability, we propose the *Always-on 0-RTT QUIC* design, which ensures the activation of 0-RTT even for completely cold-start functions.

In the QUIC protocol, if the client has never connected to the server, the first request will use 1-RTT mode, due to the lack of pre-knowledge with the server. QUIC clients rely on the QUIC connection token and TLS session cache from previous handshakes to enable 0-RTT. The token is used for servers to identify and verify the 0-RTT connection from clients. The TLS session cache is indeed the TLS pre-shared key (PSK) [**50, 90**], which is the basis for 0-RTT encryption.

We introduce a *QFaaS 0-RTT Generator* component. When launching a cold-start worker (QUIC server), the *Generator* will put a valid Function Invoker (QUIC client) token into the worker, so the 0-RTT connection from Function Invoker can be accepted.

In addition, the *Generator* will also produce a unique PSK and insert it into both sides, which will then be used for 0-RTT encryption. As this process is a part of workers' environment setup, it will not introduce extra delay. After the handshake process is complete, the server will provide a new token and session to the client for the next 0-RTT connection using the QUIC protocol. These will be stored in the *QFaaS 0-RTT Store* (§3.4.2). This design utilizes the advantages of serverless computing and is fully compatible with the QUIC protocol.

3.5. QFaaS: System Implementation

We implemented the QFaaS prototype on OpenFaaS (§3.5.1) and enabled the QUIC 0-RTT feature on the system (§3.5.2). The QFaaS design is easy to be extended to other platforms (§3.5.3).

3.5.1. QFaaS Prototype on OpenFaaS

We implemented our QFaaS into the popular OpenFaaS [**88**] serverless platform. OpenFaaS is currently the leading open-source serverless platform (sorted by the GitHub stars [**111**]). It uses Docker containers to host all components and Kubernetes (K8s) to simplify container deployment and management.

We extended `quic-go` [**89**] for our prototype. `quic-go` supports the recently standardized IETF QUIC [**115**]. It is implemented in Go-lang, the same language as OpenFaaS and K8s, which makes them easier to be integrated. `quic-go` also provides an HTTP/3 [**29**] implementation by assembling QUIC with the Go-lang HTTP package (`net/http`).

We primarily modified two components of OpenFaaS: `faas-netes` and `of-watchdog`.

`faas-netes` is the Function Invoker component in the OpenFaaS platform that resides on Gateway. It controls the life cycle of worker containers by sending commands to the K8s master. It also works as an HTTP client, forwarding function requests to corresponding workers through standard HTTP messages. We modified `faas-netes`, integrating a `quic-go` HTTP/3 client module and coordinating it with the remaining parts. All function requests then are proxied and encrypted by QUIC when they are forwarded to workers. All these modifications only introduce a minimal increase (15 MB) to the size of compiled `faas-netes` container images.

`of-watchdog` is a tiny HTTP server, working as the request handler inside the function worker container. `of-watchdog` uses an internal IP address and is only reachable within the K8s cluster. It receives incoming function requests from `faas-netes` and passes them on to the function. We reformed the HTTP server module in `of-watchdog` to the `quic-go` HTTP/3 server such that it can accept QUIC connections from `faas-netes` and decrypt HTTP/3 messages. After attaching related packages for HTTP/3 and QUIC, the size of `of-watchdog` executable file only increased by 3 MB.

Besides `of-watchdog`, an OpenFaaS worker container also contains a language runtime and the tenant function code. As the runtime is independent to `of-watchdog`, QFaaS inherently support tenant function code in various languages with its one-size-fits-all `of-watchdog` implementation. There is no need to modify a specific runtime.

To support QFaaS function chain library, we also installed a QUIC server into the OpenFaaS `api-gateway`. It will receive and respond to all function invoke requests from the function chain library and end-users using a QUIC client. This modification does not interfere with exiting Gateway functionalities as it listens on the UDP port.

3.5.2. QUIC 0-RTT Activation

To further enable the QUIC 0-RTT feature, we also implemented the *QFaaS 0-RTT Store* in `faas-netes` to maintain and manage the QUIC connection tokens and TLS session caches for QUIC 0-RTT connections. With this implementation, scenarios, such as the function warm-start, resuming suspended workers, and processing non-continuous requests, will benefit from the performance of QUIC 0-RTT. We further implemented the *QFaaS 0-RTT Generator* in the K8s control component. It generates and distributes the 0-RTT connection information when a cold-start worker is launching. Therefore, the cold-start scenario will be accelerated by QUIC 0-RTT.

The replay attack [**35**] is a major security threat when using QUIC's 0-RTT mode. Under sophisticated settings, an adversary could potentially replay the first 0-RTT message to trigger the corresponding action twice on the server [**50**]. To be specific, man-in-the-middle (MITM) sniffing and packet replay are two necessary conditions for the QUIC 0-RTT replay attack. On the one hand, internal data-center networks have different characteristics than the public wide-area Internet. Both conditions may be harder to achieve inside a data center. One the other hand, to provide a higher security level, following the suggestions of [**115, 50**], we provided a more secure QFaaS option, which mandatorily sends all non-idempotent [**116**] requests (*e.g.*, POST) through 1-RTT. This option further mitigates the threat of the 0-RTT replay attack. Users can make choices in QFaaS based on their security needs. Additionally, recent cryptography research [**64**] also shows that it might be possible to support perfect forward secrecy during the 0-RTT key exchange process. We will show in our evaluation that even when operating in 1-RTT mode, QFaaS

is still considerably faster than OpenFaaS because it still requires fewer RTTs than the TCP+TLS scheme.

3.5.3. Platform Universality

The QFaaS design is not only effective for OpenFaaS but can also be easily extended to other serverless platforms. This is because the network-centric model we provided is universal to prevalent serverless architectures. For instance, network flows in Lambda also follow this model. Specifically, AWS revealed the Lambda architecture in [**13**]. Unlike Google Cloud or OpenFaaS, Lambda uses microVMs instead of containers for function workers, where each worker also contains a request handler (called λ shim) that listens to HTTP requests from the Lambda Frontend. Thus the QFaaS design can be directly applied in the Lambda architecture, *i.e.*, by integrating the QUIC server and client into the λ shim and Frontend, and maintaining the *QFaaS 0-RTT Store* at Frontend. In addition, Lambda now provides libraries for access control and function chains [**22**]. QFaaS function chain library can be implemented into it to further accelerate chain communications.

QFaaS can also be easily migrated to other open-source serverless platforms. Taking Apache OpenWhisk [**136**] as an example, the ❷—❼ and ❹—❺ connections are essentially the connections between OpenWhisk Controller and Code Invoker that we can use QUIC to secure and accelerate. Additionally, we can apply the design of QFaaS function chain library into special *trigger events* supported by OpenWhisk.

Our work was built on the foundations of other open-source projects, such as OpenFaaS and `quic-go`. We will completely open-source the QFaaS system, including code, datasets,

trace generators, and measurement platforms upon publication. Meanwhile, we are working for integrating QFaaS as an official plugin in the OpenFaaS platform. Moreover, one of the world-leading cloud providers is working on a PoC (proof-of-concept) deployment of QFaaS on its serverless service.

3.6. Evaluation

This section answers the following questions about QFaaS.

- What are our testbed and experiment settings (§3.6.1)?
- Will QFaaS introduce extra overheads in building function images (§3.6.2)?
- How does QFaaS perform on a single function in comparison with TCP and TCP+TLS in cold-start and warm-start scenarios (§3.6.3)?
- How does the length of function chains impact QFaaS performance (§3.6.4)?
- And how well do the benefits transfer to real-world serverless applications (§3.6.5)?

3.6.1. Testbed and Experiment Settings

We evaluate the performance of QFaaS system using several synthetic serverless functions and a real-world commercial serverless application *Hello, Retail!* [**106, 105**]. These synthetic functions are designed to cover different scenarios independently, including simple echo functions, functions with large content data, and function chains with variant lengths [**150**]. *Hello, Retail!* is a popular open-source serverless application used in many recent serverless studies [**47, 124, 15, 107**]. We use it to assess the benefits that QFaaS can deliver in real world.

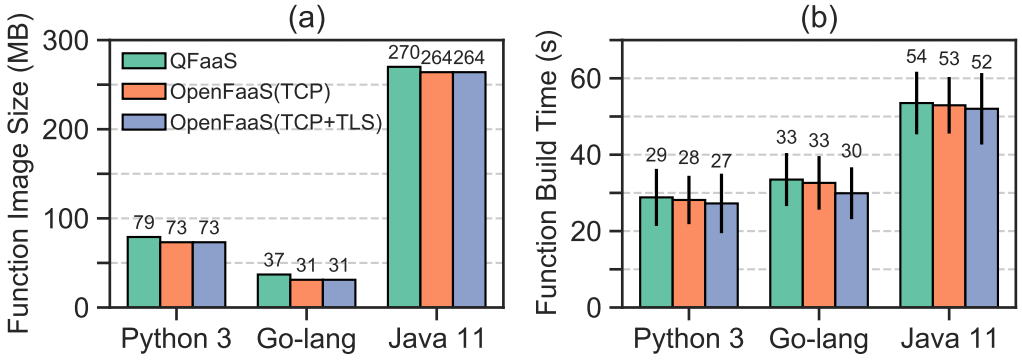


Figure 3.5. **(a) Function image sizes** and **(b) image build time** using different `of-watchdog`s and language runtimes.

We compare the performance of QFaaS with respect to OpenFaaS, using TLS 1.2, for inter-component communications. We also measure the performance of OpenFaaS using only bare TCP connections between components as a reference. Because TCP does not encrypt and decrypt packets, it has a much shorter OS processing delay in comparison to secure protocols. However, TCP-only OpenFaaS does not provide any security for the cloud platform. We find that our QFaaS design is even faster than TCP-only OpenFaaS in some scenarios, while providing additional security.

All experiments were performed on our K8s cluster with 1 master node and 3 follower nodes. The K8s system components were deployed in the master node. All OpenFaaS related components were running in the follower nodes and each node has a 4x2.9 GHz CPU with 8 GB of RAM. All Docker images were pre-pulled to avoid the influence of external network variations. We also enabled the K8s local DNS agent feature to improve cluster DNS performance.

The network delay within a typical data center has been found to be around 0.5 ms in prior studies [**87, 113, 67**]. Hence, in the following experiments, we set the default

delay between cluster nodes to 0.5 ms to simulate the data center network. In addition, intra network delay is also potentially very small. We thus specifically measure the case of zero network delays. We further note that the delay within a public cloud could be longer than 0.5 ms. For example, the real-time AWS intra-regional delay varies from 1 ms to 3 ms [**95**]. We will also show the benefits of QFaaS against different intra-cloud delays. AWS users might notice that the latency between two EC2 VMs could be as small as 0.1 ms. This is because AWS tends to deploy VMs belonging to the same tenant to the same host machine [**146**]. But in serverless computing, multi-tenants share the same Gateway. The internal delay is closer to the intra-regional average delay. We keep all MTU, TCP, TLS, and QUIC settings as default.

3.6.2. Function Image Overheads

Using QFaaS does not require any application code modification. To migrate existing OpenFaaS application functions to QFaaS, cloud providers only need to rebuild the function container image on top of the modified `of-watchdog`. This process can be done automatically and is transparent to tenants. The function image size and function build time overheads are given in Figure 3.5 (a) and (b), respectively.

Results: Figure 3.5 (a) indicates that QFaaS only slightly increases the container image size by 3 MB among each different language runtimes. The increased size is due to the additional libraries for QUIC server support. As shown in Figure 3.5 (b), QFaaS only introduces negligible addition time in building functions comparing with OpenFaaS.

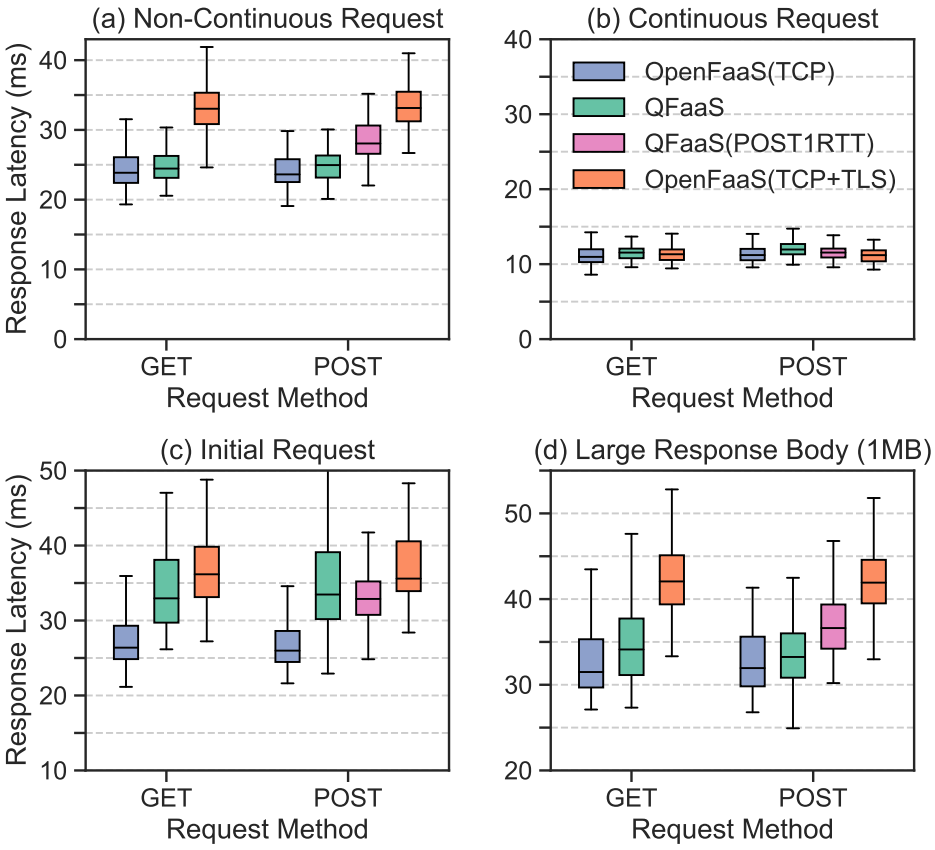


Figure 3.6. **Single function end-user response latency: (a) non-continuous requests** require connection resumption. QFaaS shows the same performance as insecure OpenFaaS (TCP) and is 28% better than OpenFaaS (TCP+TLS). **(b) continuous requests** have no connection setup overhead. They perform identically; when an **(c) initial request** is sent to a newly launched function, if no session caches, QFaaS is still 11% faster; for **(d) large responses**, QFaaS is slightly slower than OpenFaaS (TCP) due to encryption, but is 21% faster than OpenFaaS (TCP+TLS).

3.6.3. Single Function Performance

We measure the response latency of several synthetic single functions to show the benefits of QFaaS under different scenarios independently (Figure 3.6). Response latency represents the time interval between the end-user sending the request and receiving the

complete function response. For GET requests, we compare the latency between OpenFaaS (TCP), QFaaS, and OpenFaaS (TCP+TLS). For POST requests, we also measure the latency of QFaaS with mandatory 1-RTT enabled, *i.e.*, QFaaS (POST1RTT) (check §3.5.2 for more details). In Figure 3.6 (a), (b), and (c), we all use a simple echo function to avoid the jitter in function execution. The function used in Figure 3.6 (d) returns a large response body of 1 MB when called. We repeated each experiment 100 times.

Results: As shown in Figure 3.6 (a), we first measure the scenario that end-user requests sending in the no-continuous pattern. In this case, whether TCP, TLS, or QUIC, connection resumption is required. For GET requests, QFaaS performs the same as insecure OpenFaaS (TCP) and is 28% better than OpenFaaS (TCP+TLS). For POST requests, QFaaS working in 1-RTT mode is still 14% faster than OpenFaaS (TCP+TLS) and achieves the same benefits as it performs on GET in 0-RTT mode. In Figure 3.6 (b), end-users continuously send requests to avoid any connection resumption. All these implementations perform identically. It indicates that QFaaS does not bring additional overhead for this scenario.

In Figure 3.6 (c), we measure the first request latency when a function instance is newly launched. In this scenario, we let the QUIC client in `faas-netes` have no server session cache corresponding to the new function worker, thus working in 1-RTT mode. In this case, QFaaS is still 11% faster than OpenFaaS (TCP+TLS).

Figure 3.6 (d) shows the scenario when the function returning a large body. The end-user needs to wait for several packets before getting the complete response. In this

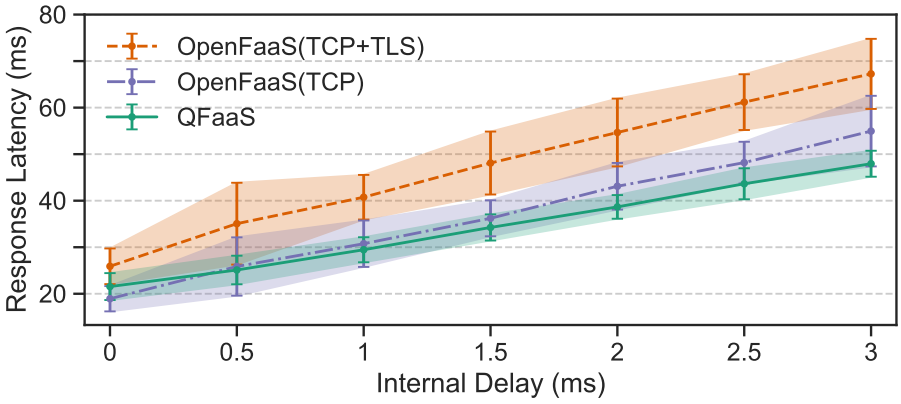


Figure 3.7. **Benefits of QFaaS under variant intra-cloud delays.** QFaaS is always faster (17% reduction) than OpenFaaS (TCP+TLS) when the internal delay is 0. The response latency difference increases as delays increase. QFaaS starts to be faster than OpenFaaS (TCP) when the delay is greater than 0.5 ms.

scenario, QFaaS is slightly slower than the insecure OpenFaaS (TCP) due to traffic encryption and decryption overhead. But it is still 21% faster than OpenFaaS (TCP+TLS) when the size of the response body is 1 MB.

Variant Intra-Cloud Delays. We show the benefits of QFaaS under different cloud internal delays in Figure 3.7. When the internal network delay is 0, QFaaS is still 17% faster than OpenFaaS (TCP+TLS) as it can reduce the number of upper layer handshakes. As the internal delay increases, the response latency will also increase. But the latency increase in QFaaS is always smaller than that of its opponents.

Results: Specifically, QFaaS maintains its advantages over OpenFaaS (TCP+TLS) as the internal delay increases. Their latency difference reaches 19 ms when the internal delay increases to 3 ms. QFaaS becomes faster than insecure OpenFaaS (TCP) after the internal delay is greater than 0.5 ms and is 13% faster than OpenFaaS (TCP) when the internal delay is 3 ms.

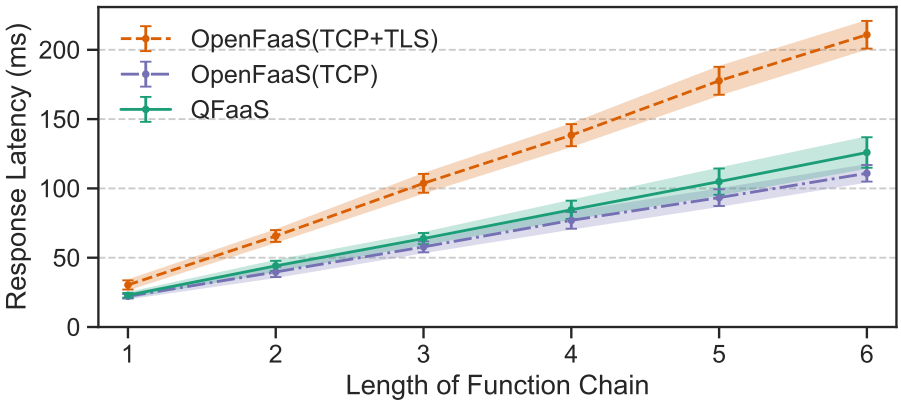


Figure 3.8. **Benefits of QFaaS with the function chain library.** The latency difference between QFaaS and OpenFaaS (TCP+TLS) increases as the chain's length increases and reaches 85 ms (40%) when the length is 6.

3.6.4. Function Chain Performance

In serverless applications, functions are commonly chained together to perform a complete task flow. We measure the benefits of using QFaaS function chain library in different lengths of function chains in Figure 3.8. The experiment design follows the nested function chain implementation in ServerlessBench [**150**]. In a function chain, the end-user invokes the ingress function; one function invokes another function and returns until got the response of the invoked function.

Results: As shown in the result, comparing with OpenFaaS (TCP), regardless of the length of the function chain, QFaaS always has the similar end-user response latency as the insecure OpenFaaS (TCP). QFaaS always performs better than OpenFaaS (TCP+TLS), and their latency difference increases as the chain's length increases. QFaaS is 85 ms (40%) faster than OpenFaaS (TCP+TLS) when the chain length is 6.

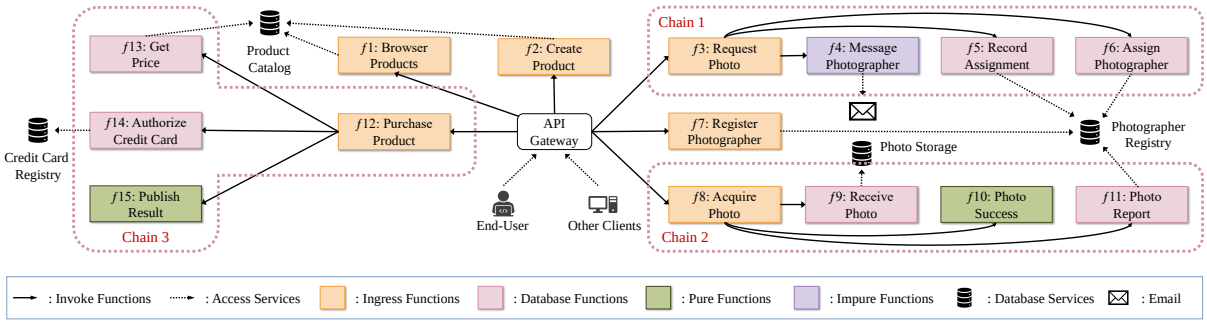


Figure 3.9. **A reference architecture of the *Hello, Retail!* application.** Rectangles represent serverless functions and are categorized into different colors by their attributes. They form 3 major function chains (see Table 3.1 for details).

3.6.5. Real-world Application Performance

To better understand how QFaaS works in production environments, we conducted additional experiments on a real-world serverless application *Hello, Retail!*. It implements a functional retail platform constructed by a set of serverless functions and back-end services. Figure 3.9 shows the reference architecture of *Hello, Retail!*. Please note, this figure uses the serverless login view instead of the actual network-centric view, to highlight the application's abstract structure. We ported the entire *Hello, Retail!* application to the OpenFaaS platform as described in prior work [**47, 124**]. It is also deployed into QFaaS without any code modification.

As shown in Figure 3.9, *Hello, Retail!* consists of 15 functions. These functions form 3 major function chains. Table 3.1 lists all scenarios we used in our experiments, covering the most representative scenarios in this application. In terms of whether a function accesses backend services and whether it invokes a function chain, we classify the scenarios as:

- **Pure Functions:** The function only communicates with the Gateway.
- **Database Functions:** The function will access back-end services, *e.g.*, database.

Table 3.1. Evaluated serverless function scenarios in the "*Hello, Retail!*" application.

Scenarios	**Function**	**Function Name**	**Method**	**Function**	**Function Name**	**Method**
Pure Functions	f10	Photo Success	GET	f15	Publish Result	POST
Database Functions	f1	Browse Products	GET	f7	Register Photographer	POST
Chain Functions	f3	Request Photo	POST	*Invoking a function chain:* f3 → f4 → f5 → f6.		
	f8	Acquire Photo	POST	*Invoking a function chain:* f8 → f9 → f10 → f11.		
	f12	Purchase Product	POST	*Invoking a function chain:* f12 → f13 → f14 → f15.		

- **Chain Functions:** The ingress function that sequences a function chain.
- **Chain Functions with Function Chain Library:** Chain functions adopting the QFaaS function chain library.

Table 3.1 lists all scenarios we used in our experiments, covering the most representative scenarios in this application. Following the preceding experiment setting in Figure 3.6 (a), we measured the end-user response latency by sending non-continuous function requests. Figure 3.10 shows the results.

Results: (a) For pure functions, f10 is invoked by GET messages, and f15 is invoked by POST messages. QFaaS and QFaaS (POST1RTT) can achieve a similar acceleration as they performed in single function evaluations (Figure 3.6). (b) For database functions, though the performance boosts are diluted by the extra connections with databases or third-party services, QFaaS can still achieve 7%-12% latency reduction against OpenFaaS (TCP+TLS) while keeping comparable performance as insecure OpenFaaS (TCP). (c) For the 3 chain functions, QFaaS remains to outperform OpenFaaS (TCP+TLS) by 14%-25% working in 0-RTT mode and 6%-10% working in 1-RTT mode. Additionally, QFaaS bonuses multiply to attain up to 50 ms latency reduction, which would perceptibly improve user experience. (d) To take full advantage of QFaaS, we integrated the function chain

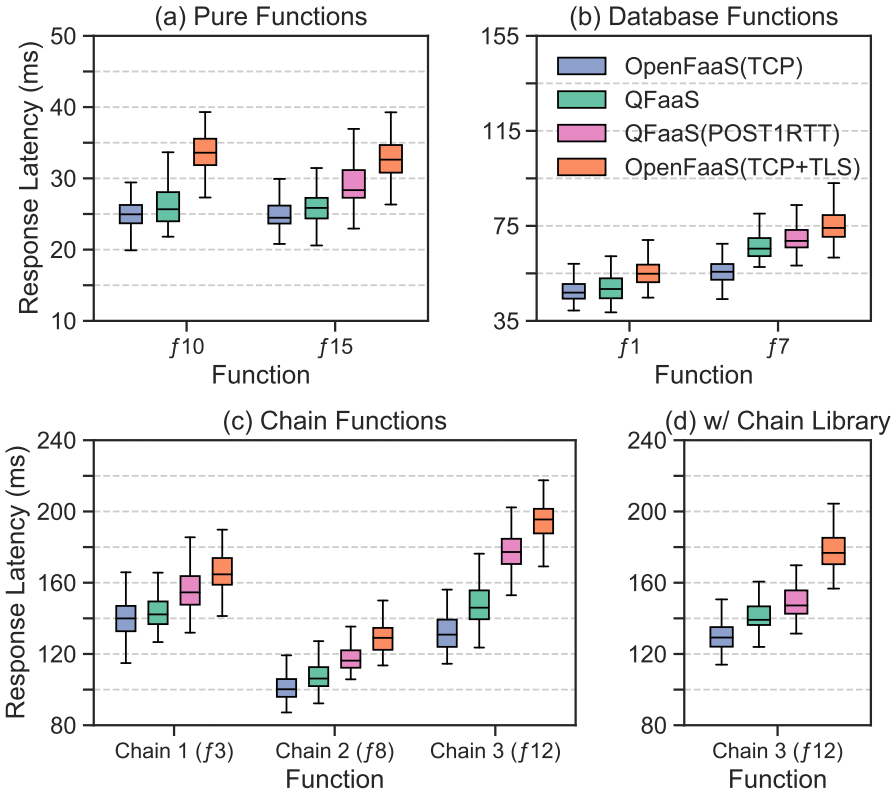


Figure 3.10. **End-user response latency in the *Hello, Retail!* application. (a) Pure Functions**: QFaaS achieved the same results as in Figure 3.6. **(b) Database Functions**: QFaaS is 12% faster than OpenFaaS (TCP+TLS); QFaaS (POST1RTT) is faster by 7%. **(c) Chain Functions**: 14%-25% latency reduction provided by QFaaS and up to 50 ms reduction in one request; 6%-10% latency reduction with QFaaS (POST1RTT). **(d) Chain Functions with Function Chain Library**: Chain 3 gaining a 21% performance boost.

library to f12. Since the natural language efficiency distinction, where the original *Hello, Retail!* is written in NodeJS and the QFaaS function chain library is implemented in Go-lang, for comparison fairness, we translated origin Chain 3 (f12) in Go-lang. It shows a 21% latency reduction on average.

The results on *Hello, Retail!* demonstrate that cloud tenants can instantly gain the benefits of QFaaS as synthetic serverless functions (§3.6.3). We therefore believe serverless computing platforms will be more attractive for existing cloud applications when the low-latency requirement can be met.

3.7. Related Work

This section summarizes closely related work from two aspects. §3.7.1 introduces research efforts about serverless computing. §3.7.2 provides a brief history of QUIC and existing projects about extending QUIC to other broader network scenarios.

3.7.1. Serverless Computing Research

The rising prominence of serverless computing has attracted recent research interest in wide-ranging topics. We summarize related work in the following closely related categories: security and access control, virtualization optimization, scaling and scheduling, and performance benchmarks.

Security and Access Control. Trapeze [**15**] uses a language-based dynamic information flow control (IFC) to secure the serverless functions. Each serverless function in Trapeze is wrapped by a security IFC shim to share data stores and exchange messages. Valve [**47**] employs function level information flow control to restrict unexpected function behaviors through the network. WILL.IAM [**124**] encodes absolute and conditional information flows into a graph to disallow access policy violations at the ingress. Nevertheless, these works all rely on a solid secure transport layer provided by the serverless platforms. We

believe that our work is complementary to existing research on serverless security and access control.

Virtualization Optimization. There are always trade-offs between efficiency and security. Docker containers impose minimal initial overhead but weaker resource isolation. In contrast, traditional virtual machine technologies provide strong resource isolation but require much higher overhead. Several research efforts have attempted to develop lightweight virtualization techniques to optimize the efficiency-security trade-off. AWS Firecracker [**13**] and SEUSS [**34**] devised lightweight VMs (microVMs) to accelerate function initialization. For instance, AWS Firecracker [**13**] removes unnecessary features like BIOS, PCI, and multi-OS support from traditional VMs. SEUSS uses unikernel snapshots to accelerate VM initialization. On the other hand, gVisor [**149**] is a new security-oriented container design to guarantee strong isolation between the host OS and containers. SCONE [**17**] utilizes the Intel SGX trusted computing to provide a secure container mechanism. All of these designs can initialize a serverless function at the millisecond scale but make the adverse impact of connection setup latency more significant. The approach adopted by QFaaS is fundamentally different but is also complementary to them.

Scaling and Scheduling. The cold-start problem is a major drawback of serverless computing [**75**]. To achieve the low cold-start latency, vigorous approaches are proposed. SAND [**14**] used fine-grained application sandboxing and hierarchical message bus mechanisms. FnSched [**133**] mitigated the resource contention between collocated functions by dynamically regulating the CPU shares. Nightcore [**74**] combined multiple techniques in platform design, including a fast path for internal function calls, efficient threading for I/O, and function executions with dynamically computed concurrency hints. Obetz

et al. [**108**] and Archipelago [**128**] proposed to use graph analysis to schedule function initialization in an efficient way. QFaaS solves the code start problem from a different angle from existing efforts. It can be combined with aforementioned approaches to better tame this problem.

Serverless Performance Benchmarks. Many benchmark suites, such as ServerlessBench [**150**], SPEC-RG [**143**], FAASDOM [**91**], and PanOpticon [**129**], were designed for serverless platforms to characterize metrics such as communication efficiency, stateless overhead, and performance isolation from different ways. Research literature, including [**79, 92, 84, 146, 58**], measured the performance differences about elasticity, latency, reliability, I/O, and cost for major commercial serverless platforms such as AWS, Google, Azure, and IBM. Our evaluation design mainly draws on their work to demonstrate the benefits of QFaaS in accelerating serverless networking.

3.7.2. QUIC: Evolution and Extensions

The first design of the QUIC protocol was released by Google in 2013 [**38**], that was informed by their experiments with the SPDY protocol [**137**]. This QUIC edition was later called gQUIC (Google QUIC) and brought to the IETF in 2015. Google joined the IETF team to provide a standardized protocol implementation called IETF QUIC, which has been incorporated into the Chrome browser since Oct. 2020 [**39**] and released as RFC 9000 [**115**] in May 2021. Major Internet service providers have all joined to provide their own QUIC implementation based on the IETF standard by different programming languages, such as `quic-go` (Go, Google) [**89**], `MsQuic` (C, Microsoft) [**98**], `mvfst` (C++, Facebook) [**53**], and `quiche` (Rust, Cloudflare) [**44**].

After its success in the wide-area Internet, such as web surfing and video streaming, QUIC has recently been extended to other broader network scenarios. Kumar et al. [**80**] utilized QUIC in IoT scenarios and have shown that QUIC largely benefits the connection migration for IoT devices. Thomas et al. [**139**] demonstrate that compared with TLS, QUIC can halve the page load time over the public satellite communication system. Cicconetti et al. [**40**] preliminarily evaluated the benefits of using QUIC in end-user to FaaS Gateway connections in the mobile network scenario. Research literature, such as [**27, 28**], integrated QUIC into the Tor network and provided empirical evaluation to show network acceleration. To the best of our knowledge, QFaaS is the first work to extend QUIC into the domain of serverless cloud platforms.

3.8. Chapter Summary

In this project, we raise the challenge of accelerating communications while providing security in emerging serverless cloud networks. To that end, we first abstract the network communications model for serverless computing systems and then propose an extension of the QUIC protocol, called QFaaS, that provides low latency serverless function communication with improved security. We implement the QFaaS prototype on the popular OpenFaaS platform such that it requires no code modification for cloud tenants to gain network performance boosts and security benefits. QFaaS function chain library and always-on 0-RTT designs can further accelerate serverless networking. Our evaluations on synthetic serverless functions and real-world serverless applications demonstrate that QFaaS can reduce the end-user response latency by 28% (in 0-RTT mode) and 14% (in 1-RTT mode) with negligible overheads compared to OpenFaaS using TCP+TLS.

We find that the performance benefits of QFaaS linearly increases with the length of the function chain, providing a reduction of 85 ms when the chain length is 6. This was also validated against the real-world *Hello, Retail!* application, where QFaaS obtained a maximum 50 ms reduction in latency. Overall, our findings validate that QFaaS delivers a compelling performance and security enhancement to the ecosystem of open-source serverless platforms.

CHAPTER 4

Conclusion

In this dissertation, I elaborate on a research question about *enabling robust and secure edge-to-cloud communications in real-world networked systems* and tackle this question by addressing specific research challenges encountered in two projects. These two projects cover two aspects of networked systems from edge to cloud. The first project secures the edge access network, while the second project secures the centralized cloud network. Both projects target next-generation network protocols and emphasize the importance of measurement, deployment, and validation on real-world networked systems.

Specifically, in the first project, we proposed a seed-assisted specification method to systematically study the availability and security problems in cellular network protocols. This proposed method combines prior knowledge and adaptive model construction to address the modeling granularity and misrepresentation problems in protocol specifications. We take the cellular emergency call system as the case study and conduct the first thorough research on it. As a result, we systematically find four availability issues and two new attacks. In the second project, we abstract the network communication model for serverless cloud computing and identify the network challenge in existing serverless platforms. We provide the QFaaS design to address the challenge, which requires no code modification for existing serverless applications. Evaluation of the QFaaS prototype shows that our design can significantly improve the serverless network performance with enhanced security.

4.1. Future Work

We will continue our research in both securing edge access networks and optimizing the serverless cloud networks.

Secure Configuration Space Search for Network Protocols. Large network protocols, such as cellular network protocols, have many flexible configuration options to fit different deployment scenarios and carrier requirements. But not every possible configuration combination is correct and secure. To meet the agile deployment and evolution of networked systems, configurations are updated frequently. Although it would be ideal to check each configuration in real-time before enforcement, it is impractical due to the time-consuming formal verification process. Meanwhile, network systems usually have multiple participants. One participant can check if its own system is correctly configured. But the correctness of the entire system depends on the configuration of all participants.

Therefore, we propose the idea called secure configuration space. Any configurations are guaranteed to be secure if they lie in this space. It has three advantages. First, the corresponding secure configuration space for a protocol can be generated offline. We can now verify a new configuration with negligible time by just SAT querying if this configuration resides in the space. Second, the verification can now support multi-participant network systems if the configuration of all participants meets the requirements of the secure configuration space. Third, if the current configuration is insecure, we can identify a secure configuration in the space with the minimum revision requirement to correct the insecure one. We plan to leverage recent breakthroughs ([**32, 43**]) in incremental induction-based model checking to search the secure configuration space, where the space can be transformed from the inductive invariant constructed in the verification process.

Predictive Clean-Slate Serverless Routing. As we discussed in §3.3, in current serverless computing platforms, function chains are formed by indirect connections through the unified gateway. Such a design greatly increases the internal function communication delay and affects the network's isolation. We plan to leverage data-plane programming and function chain prediction to develop an intelligent, clean-slate network architecture for serverless network communication. Data-plane programming can precisely control the message forwarding between function instances to avoid indirect connections through the unified gateway. The function chain performance thus benefits from direct connection and shortest-path routing. With function chain prediction, the forwarding rules are issued in advance to further improve performance by mitigating cold-start issues. This design also enhances traffic isolation and improves network security in multi-tenant clouds.

Specifically, the proposed research has two challenges: enabling direct connections by data-plane programming and pre-installing forwarding rules by function chain prediction.

For the first challenge, programmable network technologies, such as OpenFlow [**96**] and P4 [**31**], have flourished in the last decade. It can provide two capabilities to the serverless platform. First, it can quickly and dynamically adjust the network topology and forwarding paths while the network is running. Leveraging this capability, we can create a direct forwarding path when a function chain is created and destroy this path when the corresponding function instances are scaled down. Second, programmable networks can control the packet routing in a fine-grained. It provides a more powerful routing capability than traditional IP routing. Even if a function instance does not know the internal address of another ephemeral instance, it can still directly send packets to the instance with carefully designed routing algorithms, such as source address routing.

For the second challenge, the performance of serverless computing also suffers from the notorious cold-start problem [**14, 74, 34**]. When a new function instance is created in response to the end-user's request, the cloud providers need to allocate the hardware resource and prepare the software runtime for this new function instance. This process introduces extra delays to serverless applications. Similarly, installing dynamic forwarding rules to enforce direct forwarding paths also introduces delays. We propose pre-installing forwarding rules by function chain prediction to alleviate this drawback. Compared to existing work about resource predicting and scheduling in cloud computing, function chains usually have fixed patterns to follow the high-level business logic of serverless applications. Therefore, function chain prediction could be easier and yield more accurate results than traditional cloud computing resource prediction.

References

[1] 3rd Generation Partnership Project (3GPP). http://www.3gpp2.org.

[2] 3GPP. Formal Analysis of the 3G Authentication Protocol. Technical Report (TR) 33.902, 3rd Generation Partnership Project (3GPP), 2001. Version 4.0.0.

[3] 3GPP. Public Warning System (PWS) requirements. Technical Specification (TS) 22.268, 3rd Generation Partnership Project (3GPP), 2019. Version 16.3.0.

[4] 3GPP. Access to the 3GPP Evolved Packet Core (EPC) via non-3GPP access networks; Stage 3. Technical Specification (TS) 24.302, 3rd Generation Partnership Project (3GPP), 2020. Version 16.4.0.

[5] 3GPP. Characteristics of the Universal Subscriber Identity Module (USIM) application. Technical Specification (TS) 31.102, 3rd Generation Partnership Project (3GPP), 2020. Version 16.4.0.

[6] 3GPP. Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. Technical Specification (TS) 36.331, 3rd Generation Partnership Project (3GPP), 2020. Version 16.1.1.

[7] 3GPP. IP Multimedia Subsystem (IMS) emergency sessions. Technical Specification (TS) 23.167, 3rd Generation Partnership Project (3GPP), 2020. Version 16.2.0.

[8] 3GPP. Mobile radio interface Layer 3 specification; Core network protocols; Stage 3. Technical Specification (TS) 24.008, 3rd Generation Partnership Project (3GPP), 2020. Version 16.5.0.

[9] 3GPP. Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3. Technical Specification (TS) 24.501, 3rd Generation Partnership Project (3GPP), 2020. Version 16.5.1.

[10] 3GPP. Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3. Technical Specification (TS) 24.301, 3rd Generation Partnership Project (3GPP), 2020. Version 16.5.1.

[11] 3GPP. Service aspects; Service principles. Technical Specification (TS) 22.101, 3rd Generation Partnership Project (3GPP), 2020. Version 17.2.0.

[12] 3GPP2. 3rd Generation Partnership Project 2. `http://www.3gpp2.org`, 2008.

[13] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *Proceedings of the 17th USENIX symposium on networked systems design and implementation (NSDI)*, 2020.

[14] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. Sand: Towards high-performance serverless computing. In *2018 Usenix Annual Technical Conference (USENIX ATC)*, 2018.

[15] Kalev Alpernas, Cormac Flanagan, Sadjad Fouladi, Leonid Ryzhyk, Mooly Sagiv, Thomas Schmitz, and Keith Winstein. Secure serverless computing using dynamic information flow control. *Object-Oriented Programming, Systems, Languages and Applications (OOPLSA)*, October 2018.

[16] Myrto Arapinis, Loretta Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In *Proceedings of the 2012 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 205–216. ACM, 2012.

[17] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[18] AWS. Lambda – Serverless Computing. `https://aws.amazon.com/lambda/`, 2014.

[19] AWS. Announcing improved VPC networking for AWS Lambda functions. `https://aws.amazon.com/blogs/compute/announcing-improved-vpc-networking-for-aws-lambda-functions/`, 2019.

[20] AWS. Amazon Web Services: Overview of Security Processes. `https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf`, 2020.

[21] AWS. Data protection in Amazon EC2 - encryption in transit. `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/data-protection.html#encryption-transit`, 2021.

[22] AWS. Lambda, AWS Boto3. `https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/lambda.html`, 2021.

[23] AWS. Security Overview of AWS Lambda: An In-Depth Look at AWS Lambda Security. `https://d1.awsstatic.com/whitepapers/Overview-AWS-Lambda-Security.pdf`, 2021.

[24] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 2017.

[25] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. Symbolically analyzing security protocols using tamarin. *ACM SIGLOG News*, 4(4):19–30, 2017.

[26] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1383–1396. ACM, 2018.

[27] Lamiaa Basyoni, Aiman Erbad, Mashael Alsabah, Noora Fetais, and Mohsen Guizani. Empirical performance evaluation of quic protocol for tor anonymity network. In *Proceedings of the 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE, 2019.

[28] Lamiaa Basyoni, Aiman Erbad, Mashael Alsabah, Noora Fetais, Amr Mohamed, and Mohsen Guizani. Quictor: Enhancing tor for real-time communication using quic transport protocol. *IEEE Access*, 9:28769–28784, 2021.

[29] M. Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). RFC Draft, IETF, February 2021.

[30] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

[31] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4:

Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[32] Aaron R Bradley. Sat-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011.

[33] Elgin M Brunner and Manuel Suter. *International CIIP handbook 2008/2009: An inventory of 25 national and 7 international critical information infrastructure protection policies*. Center for Security Studies (CSS), ETH Zurich, 2008.

[34] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. Seuss: skip redundant paths to make serverless fast. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys)*, 2020.

[35] Xudong Cao, Shangru Zhao, and Yuqing Zhang. 0-rtt attack and defense of quic protocol. In *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019.

[36] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The rise of serverless computing. *Communications of the ACM*, 62(12):44–54, 2019.

[37] Shan Chen, Samuel Jero, Matthew Jagielski, Alexandra Boldyreva, and Cristina Nita-Rotaru. Secure communication channel establishment: Tls 1.3 (over tcp fast open) vs. quic. In *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2019.

[38] Chromium Blog. Experimenting with QUIC. `https://blog.chromium.org/2013/06/experimenting-with-quic.html`, 2013.

[39] Chromium Blog. Chrome is deploying HTTP/3 and IETF QUIC. `https://blog.chromium.org/2020/10/chrome-is-deploying-http3-and-ietf-quic.htmll`, 2020.

[40] Claudio Cicconetti, Leonardo Lossi, Enzo Mingozzi, and Andrea Passarella. A preliminary evaluation of quic for mobile serverless edge applications. In *22nd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2021.

[41] Cisco. MME Administration Guide, Emergency Bearer Services. `https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21-5_N5-8/MME/21-5-MME-Admin/21-5-MME-Admin_chapter_010010.html`, 2016.

[42] Cisco. MME Administration Guide, Local Emergency Numbers List. `https://www.cisco.com/c/en/us/td/docs/wireless/asr_5000/21/MME/b_21_MME_Admin/b_21_MME_Admin_chapter_011111.pdf`, 2016.

[43] Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In *2012 Formal Methods in Computer-Aided Design (FMCAD)*, pages 52–59. IEEE, 2012.

[44] Cloudflare. quiche: Savoury implementation of the QUIC transport protocol and HTTP/3. `https://github.com/cloudflare/quiche`, 2018.

[45] Piergiuseppe Bettassa Copet, Guido Marchetto, Riccardo Sisto, and Luciana Costa. Formal verification of lte-umts and lte–lte handover procedures. *Computer Standards & Interfaces*, 50:92–106, 2017.

[46] Cas Cremers and Martin Dehnel-Wild. Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion. In *Proceedings of the 26th Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[47] Pubali Datta, Prabuddha Kumar, Tristan Morris, Michael Grace, Amir Rahmati, and Adam Bates. Valve: Securing function workflows on serverless computing platforms. In *Proceedings of The Web Conference*, 2020.

[48] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. Pluginizing quic. In *Proceedings of the annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2019.

[49] Haotian Deng, Weicheng Wang, and Chunyi Peng. CEIVE: Combating Caller ID Spoofing on 4G Mobile Phones Via Callee-Only Inference and Verification. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2018.

[50] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, IETF, August 2018.

[51] Lars Eggert. Towards securing the internet of things with quic. In *Workshop on Decentralized IoT Systems and Security (DISS)*, 2020.

[52] European Central Bank. Standards for Securities Clearing and Settlement in the European Union. Technical report, CESR, 2004.

[53] Facebook. mvfst: An implementation of the QUIC transport protocol. `https://github.com/facebookincubator/mvfst`, 2019.

[54] Federal Communications Commission, USA. Revision of the commission's rules to ensure compatibility with enhanced 911 emergency calling ssystems, September 2002. 17 FCC Rcd 19012 (25).

[55] Federal Communications Commission, USA. Wireless E911 location accuracy requirements, April 2015. 30 FCC Rcd 2990 (4).

[56] Federal Communications Commission, USA. 800 MHz Cellular Service. `https://www.fcc.gov/wireless/bureau-divisions/mobility-division/800-mhz-cellular-service`, 2019.

[57] Federal Communications Commission, USA. Caller ID Spoofing. `https://www.fcc.gov/consumers/guides/spoofing-and-caller-id`, 2020.

[58] Kamil Figiela, Adam Gajek, Adam Zima, Beata Obrok, and Maciej Malawski. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience*, 30(23):e4792, 2018.

[59] Nick Galov. How Many IoT Devices Are There in 2021? `https://techjury.net/blog/how-many-iot-devices-are-there`, 2021.

[60] Nico Golde, Kévin Redon, and Jean-Pierre Seifert. Let me answer that for you: Exploiting broadcast information in cellular networks. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security)*, pages 33–48, 2013.

[61] Google. Encryption in Transit in Google Cloud. `https://cloud.google.com/security/encryption-in-transit`, 2017.

[62] Google. Android Open Source Project. `https://source.android.com`, 2020.

[63] Google. Google Cloud Security Whitepapers. `https://cloud.google.com/security/overview/whitepaper`, 2021.

[64] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-rtt key exchange with full forward secrecy. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*. Springer, 2017.

[65] Mordechai Guri, Yisroel Mirsky, and Yuval Elovici. 9-1-1 ddos: attacks, analysis and mitigation. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 218–232. IEEE, 2017.

[66] Michael Hauben. History of arpanet. *Site de l'Instituto Superior de Engenharia do Porto*, 17, 2007.

[67] Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey. Early observations on the performance of windows azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC)*, 2010.

[68] Lin Huang. LTE REDIRECTION: Forcing Targeted LTE Cellphone into Unsafe Network. In *Proceedings of the 7th Annual HITB Security Conference (HITBSecConf)*, 2016.

[69] Huawei. CloudEC V600R019C00 Feature Guide, Emergency Call. `https://support.huawei.com/enterprise/en/doc/EDOC1100059085/e0804ebc/emergency-call`, 2019.

[70] Syed Hussain, Omar Chowdhury, Shagufta Mehnaz, and Elisa Bertino. LTEInspector: A systematic approach for adversarial testing of 4G LTE. In *Proceedings of the 25th Network and Distributed Systems Security (NDSS) Symposium*, 2018.

[71] Syed Rafiul Hussain, Mitziu Echeverria, Omar Chowdhury, Ninghui Li, and Elisa Bertino. Privacy attacks to the 4g and 5g cellular paging protocols using side channel information. In *Proceedings of the 26th Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[72] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 5GReasoner: A Property-Directed Security and Privacy Analysis Framework for 5G Cellular Network Protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 669–684. ACM, 2019.

[73] ITU-T. ISDN user-network interface layer 3 specification for basic call control. Itu-t recommendation, International Telecommunication Union, 1998. Q.931.

[74] Zhipeng Jia and Emmett Witchel. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[75] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint 1902.03383*, 2019.

[76] Kakaku BBS. I cannot make emergency calls such as 110 (in Japanese). `https://bbs.kakaku.com/bbs/J0000024343/SortID=21105988/`, 2017.

[77] Kenneth Reitz. Requests: HTTP for Humans. `https://docs.python-requests.org/en/master/`, 2011.

[78] Hannah Kuchler. Hackers find suppliers are an easy way to target companies. `https://www.ft.com/content/b4807a14-5097-11e4-8645-00144feab7de`, 2015.

[79] Jörn Kuhlenkamp, Sebastian Werner, Maria C Borges, Dominik Ernst, and Daniel Wenzel. Benchmarking elasticity of faas platforms as a foundation for objective-driven design of serverless applications. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC)*, 2020.

[80] Puneet Kumar and Behnam Dezfouli. Implementation and analysis of quic for mqtt. *Computer Networks*, 150:28–45, 2019.

[81] Leslie Lamport. *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[82] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017.

[83] Gyuhong Lee, Jihoon Lee, Jinsung Lee, Youngbin Im, Max Hollingsworth, Eric Wustrow, Dirk Grunwald, and Sangtae Ha. This is Your President Speaking: Spoofing Alerts in 4G LTE Networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 404–416. ACM, 2019.

[84] Hyungro Lee, Kumar Satyam, and Geoffrey Fox. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018.

[85] Chi-Yu Li, Guan-Hua Tu, Chunyi Peng, Zengwen Yuan, Yuanjie Li, Songwu Lu, and Xinbing Wang. Insecurity of voice solution VoLTE in LTE mobile networks. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 316–327. ACM, 2015.

[86] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 202–215, 2016.

[87] Shuhao Liu, Hong Xu, and Zhiping Cai. Low latency datacenter networking: A short survey. *arXiv preprint 1312.3455*, 2013.

[88] OpenFaaS Ltd. Openfaas: Serverless functions, made simple. `https://www.openfaas.com/`, 2016.

[89] Lucas Clemente, et. al. quic-go: A QUIC implementation in pure Go. `https://github.com/lucas-clemente/quic-go`, 2016.

[90] M. Thomson, S. Turner. Using TLS to Secure QUIC. RFC 9001, IETF, May 2021.

[91] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. Faasdom: A benchmark suite for serverless computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS)*, 2020.

[92] Maciej Malawski, Kamil Figiela, Adam Gajek, and Adam Zima. Benchmarking heterogeneous cloud functions. In *European Conference on Parallel Processing (Euro-Par)*. Springer, 2017.

[93] Markets and Markets. Serverless Architecture Market by Service Type (Automation and Integration, Monitoring, API Management, Security, Analytics, and Design and Consulting), Deployment Model, Organization Size, Vertical, and Region - Global Forecast to 2025. `https://www.marketsandmarkets.com/Market-Reports/serverless-architecture-market-64917099.html`, 2020.

[94] Stephen Paul Marsh. Formalising trust as a computational concept. *University of Stirling*, 1994.

[95] Matt Adorjan. AWS Latency Monitoring. `https://www.cloudping.co/grid`, 2021.

[96] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[97] MediaTek. MTK Catcher. `https://www.finetopix.com/showthread.php/40844-MTK-Catcher`, 2014.

[98] Microsoft. MsQuic: Cross-platform, C implementation of the IETF QUIC protocol. https://github.com/microsoft/msquic, 2019.

[99] Microsoft. Azure encryption overview. https://docs.microsoft.com/en-us/azure/security/fundamentals/encryption-overview, 2021.

[100] Microsoft. Azure security baseline for Azure Functions. https://docs.microsoft.com/en-us/security/benchmark/azure/baselines/functions-security-baseline, 2021.

[101] Ministry of Industry and Information Technology, China. Technical requirement of routing and implementation for inter-network emergency call service, September 2005. YD/T 1406-2005.

[102] Ministry of Industry and Information Technology, China. Technical requirement and testing methods for general function of mobile telecommunication terminal, December 2011. YD/T 2307-2011.

[103] Nation Emergency Number Association. 9-1-1 Statistics. https://www.nena.org/page/911Statistics, 2018.

[104] National Instruments, Ettus Research. Universal Software Radio Peripheral (USRP) B210 SDR Kit (70 MHz - 6GHz). https://www.ettus.com/all-products/UB210-KIT/, 2020.

[105] Nordstrom. Towards a serverless event-sourced nordstrom. https://youtu.be/WcCErxLKR7g, 2017.

[106] Nordstrom Technology. Hello, retail! https://github.com/Nordstrom/hello-retail, 2019.

[107] Matthew Obetz, Anirban Das, Timothy Castiglia, Stacy Patterson, and Ana Milanova. Formalizing event-driven behavior of serverless applications. In *European Conference on Service-Oriented and Cloud Computing (ESOCC)*. Springer, 2020.

[108] Matthew Obetz, Stacy Patterson, and Ana Milanova. Static call graph construction in aws lambda serverless applications. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[109] Andreea Ancuta Onofrei, Yacine Rebahi, and Thomas Magedanz. Preventing Distributed Denial-of-Service Attacks on the IMS Emergency Services Support through Adaptive Firewall Pinholing. *The International Journal of Next Generation Network (IJNGN)*, 2(1), 2010.

[110] OpenAirInterface Software Alliance. Openairinterface 5G Wireless Implementation. https://www.openairinterface.org/, 2020.

[111] OpenFaaS Ltd. GitHub: openfaas/faas. https://github.com/openfaas/faas, 2021.

[112] Peter Pi, XiLing Gong, and Gmxp. Exploring Qualcomm Baseband via ModKit. In *CanSecWest conference*, 2018.

[113] Diana Popescu, Noa Zilberman, and Andrew Moore. Characterizing the impact of network latency on cloud-based applications' performance. *Computer Laboratory technical reports*, 2017.

[114] Qualcomm. QUALCOMM eXtensible Diagnostic Monitor (QxDM). https://www.qualcomm.com/documents/qxdm-professional-qualcomm-extensible-diagnostic-monitor, 2020.

[115] QUIC Working Group. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, IETF, 2021.

[116] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.

[117] Raphael Satter, Christopher Bing, Joseph Menn. Hackers used SolarWinds' dominance against it in sprawling spy campaign. https://www.reuters.com/article/global-cyber-solarwinds/hackers-at-center-of-sprawling-spy-campaign-turned-solarwinds-dominance-against-it-idUSKBN28P2N8, 2020.

[118] Yacine Rebahi, Andreea Ancuta Onofrei, and Thomas Magedanz. Security in the Emergency Services Support for the IP Multimedia Subsystem (IMS). *5th International Week on Management of Networks and Services, Venice, Italy*, 2009.

[119] Red Pocket Mobile. Red Pocket Global Internet Data Plans. https://www.redpocket.com/global, 2020.

[120] Larry Roberts. The arpanet and computer networks. In *A history of personal workstations*, pages 141–172. 1988.

[121] Scott W Rose, Oliver Borchert, Stuart Mitchell, and Sean Connelly. Zero trust architecture. *Special Publication, National Institute of Standards and Technology (NIST SP), Gaithersburg, MD*, 2020.

[122] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Breaking LTE on layer two. In *Proceedings of the 2019 IEEE Symposium on Security & Privacy (S & P)*, 2019.

[123] S. O'Dea. Number of mobile (cellular) subscriptions worldwide from 1993 to 2021. `https://www.statista.com/statistics/262950/global-mobile-subscriptions-since-1993/`, 2021.

[124] Arnav Sankaran, Pubali Datta, and Adam Bates. Workflow integration alleviates identity and access management in serverless computing. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[125] Takuro Sato, Daniel M Kammen, Bin Duan, Martin Macuha, Zhenyu Zhou, Jun Wu, Muhammad Tariq, and Solomon Abebe Asfaw. *Smart grid standards: specifications, requirements, and technologies*. John Wiley & Sons, 2015.

[126] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC)*, 2020.

[127] Altaf Shaik, Ravishankar Borgaonkar, N Asokan, Valtteri Niemi, and Jean-Pierre Seifert. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. In *Proceedings of the 23rd Annual Network and Distributed System Security (NDSS) Symposium*, 2016.

[128] Arjun Singhvi, Kevin Houck, Arjun Balasubramanian, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. Archipelago: A scalable low-latency serverless platform. *arXiv preprint 1911.09849*, 2019.

[129] Nikhila Somu, Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. Panopticon: A comprehensive benchmarking tool for serverless applications. In *2020 International Conference on Communication Systems & NETworkS (COMSNETS)*. IEEE, 2020.

[130] Jaeseung Song, Hyoungshick Kim, and Athanasios Gkelias. iVisher: Real-Time Detection of Caller ID Spoofing. *ETRI Journal*, 36(5):865–875, 2014.

[131] Sprint. What this means to you after April 30, 2019. `https://www.sprint.com/en/support/account/oma-slot.html`, 2019.

[132] Statista. Forecast end-user spending on IoT solutions worldwide from 2017 to 2025. https://www.statista.com/statistics/976313/global-iot-market-size/, 2019.

[133] Amoghvarsha Suresh and Anshul Gandhi. Fnsched: An efficient scheduler for serverless functions. In *Proceedings of the 5th International Workshop on Serverless Computing*, 2019.

[134] Erik Sy, Tobias Mueller, Christian Burkert, Hannes Federrath, and Mathias Fischer. Enhanced performance and privacy for tls over tcp fast open. *Proc. Priv. Enhancing Technol.*, 2020(2):271–287, 2020.

[135] T. Taylor, H. Tschofenig, H. Schulzrinne, and M. Shanmugam. Security Threats and Requirements for Emergency Call Marking and Mapping. RFC 5069, IETF, January 2008.

[136] The Apache Software Foundation. OpenWhisk, Open Source Serverless Cloud Platform. https://openwhisk.apache.org/, 2016.

[137] The Chromium Projects. SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper, 2010.

[138] THE PAPER. Father fell to the ground with cerebral haemorrhage, mother's mobile phone cannot make emergency calls. Meizu said: possible a system problem (in Chinese). https://www.thepaper.cn/newsDetail_forward_3749664, 2019.

[139] Ludovic Thomas, Emmanuel Dubois, Nicolas Kuhn, and Emmanuel Lochin. Google quic performance over a public satcom access. *International Journal of Satellite Communications and Networking*, 37(6):601–611, 2019.

[140] Guan-Hua Tu, Yuanjie Li, Chunyi Peng, Chi-Yu Li, and Songwu Lu. Detecting problematic control-plane protocol interactions in mobile networks. *IEEE/ACM Transactions on Networking*, 24(2):1209–1222, 2016.

[141] Guan-Hua Tu, Yuanjie Li, Chunyi Peng, Chi-Yu Li, Hongyi Wang, and Songwu Lu. Control-plane protocol interactions in cellular networks. In *Proceedings of the annual conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 223–234. ACM, 2014.

[142] Lionel Sujay Vailshery. Cloud applications market size worldwide from 2013 to 2025. https://www.statista.com/statistics/475670/cloud-applications-market-size-worldwide/, 2022.

[143] Erwin Van Eyk, Joel Scheuner, Simon Eismann, Cristina L Abad, and Alexandru Iosup. Beyond microbenchmarks: The spec-rg vision for a comprehensive serverless benchmark. In *Companion of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2020.

[144] Verizon Wireless. CDMA Network Retirement. `https://www.verizonwireless.com/support/knowledge-base-218813/`, 2019.

[145] W3Techs. Usage statistics of QUIC for websites. `https://w3techs.com/technologies/details/ce-quic`, 2021.

[146] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference*, 2018.

[147] Bob Williams. *Intelligent transport systems standards*. Artech House, 2008.

[148] Y. Chen, J. Chu, S. Radhakrishnan, A. Jain. TCP Fast Open. RFC 7412, IETF, December 2014.

[149] Ethan G Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. The true cost of containing: A gvisor case study. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[150] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC)*, 2020.

[151] Yinbo Yu, You Li, Kaiyu Hou, Yan Chen, Hai Zhou, and Jianfeng Yang. CellScope: Automatically Specifying and Verifying Cellular Network Protocols. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, pages 21–23. ACM, 2019.

[152] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking tla+ specifications. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 54–66. Springer, 1999.