Smart Resource Allocation in Internet-of-Things:

Perspectives of Network, Security, and Economics

by

Ruozhou Yu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved June 2019 by the
Graduate Supervisory Committee:

Guoliang Xue, Chair
Dijiang Huang
Arunabha Sen
Yanchao Zhang

ARIZONA STATE UNIVERSITY

August 2019

ABSTRACT

Emerging from years of research and development, the Internet-of-Things (IoT) has finally paved its way into our daily lives. From smart home to Industry 4.0, IoT has been fundamentally transforming numerous domains with its unique superpower of interconnecting world-wide devices. However, the capability of IoT is largely constrained by the limited resources it can employ in various application scenarios, including computing power, network resource, dedicated hardware, etc. The situation is further exacerbated by the stringent quality-of-service (QoS) requirements of many IoT applications, such as delay, bandwidth, security, reliability, and more. This mismatch in resources and demands has greatly hindered the deployment and utilization of IoT services in many resource-intense and QoS-sensitive scenarios like autonomous driving and virtual reality.

I believe that the resource issue in IoT will persist in the near future due to technological, economic and environmental factors. In this dissertation, I seek to address this issue by means of smart resource allocation. I propose mathematical models to formally describe various resource constraints and application scenarios in IoT. Based on these, I design smart resource allocation algorithms and protocols to maximize the system performance in face of resource restrictions. Different aspects are tackled, including networking, security, and economics of the entire IoT ecosystem. For different problems, different algorithmic solutions are devised, including optimal algorithms, provable approximation algorithms, and distributed protocols. The solutions are validated with rigorous theoretical analysis and/or extensive simulation experiments.

Dedicated to my wife Rui Sun and my family

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my Ph.D. advisor, Dr. Guoliang Xue, for his continuous support and guidance throughout my study at Arizona State University, for his knowledge, patience, enthusiasm, and probity that have deeply affected me, and most importantly, for his standing care and friendship during my hard times. Having him as my advisor is one of the most fortunate things in my life. This dissertation would not have been possible without the academic guidance and freedom he has given me over the last six years.

My Ph.D. study has been supported by NSF grants 1461886, 1704092, and 1717197.

I would like to thank my dissertation committee members, Dr. Dijiang Huang, Dr. Arunabha Sen, and Dr. Yanchao Zhang, for their advices and comments. I am honored to have worked with an incredible group of co-authors and collaborators. I am specially thankful to Dr. Jun Huang and Dr. Dan Li, who led me onto the path of academic research. I want to thank my academic siblings, Jian Tang, Satyajayant Misra, Xi Fang, Dejun Yang, Lingjun Li, Xinxin Zhao, and Xiang Zhang, for their precious advices and support for my study, career and life. I am also very grateful to my colleagues and friends: Jian Cai, Xinhui Hu, Yiming Jing, Vishnu Teja Kilari, Bing Li, Liangyue Li, Yashu Liu, Jianqing Liu, Qiang Liu, Yaozhong Song, Yinxin Wan, Siqi Wei, Haiqin Wu, Haitao Xu, Rui Zhang, Ziming Zhao, and Zhuoyang Zhou. Their friendship and care are what have helped me survive over the years.

Finally and most importantly, none of my achievements would have been possible without the love and patience from my wife and my family. My most sincere thanks go to my wife Rui Sun, for her endless support and caring over the years. She is the reason, the why, for which I have strived hard in the past and will keep on striving in the future. I am very thankful to my parents, Bing Yu and Lianna Xie, for their unconditional and forbearing love and support throughout my life.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Coming into the 50-th year after its birth[1], the Internet has become just an indispensable part of most of our lives. We rely on it for work, for leisure, and for social activities; for making knowledge accessible to all, for building critical infrastructures that support our lives, and even for global capitalism, freedom of speech and digital democracy. The Internet is enabling fundamental transformations deep within economics, science and education, communications, politics, arts, and humanity. There has never been another invention that has such a direct yet profound impact on all aspects of human lives.

For long, the Internet has been providing us with an abundant world of digital services via our computing devices like servers, PCs, laptops, and mobile phones. However, it is now prevailingly recognized that the boundary of the Internet will not be constrained by only the computing devices. With the help of advanced and low-power communication technologies, physical objects with little to no computing capabilities are now remotely accessible and controllable via the Internet, comprising what we call the *Internet-of-Things* (IoT). Through the interconnection of "things", an incredible number of amazing new applications are coming into our lives, such as smart home, smart city, smart healthcare, smart industry, and many more. This technology (IoT) will have an enormous impact on our society. As a piece of preliminary evidence,

[1]Counted from the first successful experiment of the ARPANET in Dec. 1969.

the annual global IoT market has reached over $700 billion in 2018, and will soon surpass the one trillion mark in a few years (if not have already been so) [55]. But this is just the beginning. In the end, IoT will be the key technology that enables the merging of our digital and physical worlds.

While the big picture is bright and shining, if we take a closer look at this technology, IoT is still in its infancy. Like when any new technology emerges, IoT is facing many issues, challenges and concerns. These, especially the technical ones, are greatly hindering the deployability, applicability and acceptance of this technology. Without enumerating all the pros and cons of IoT, however, this dissertation is focused on one of the most critical issues present: the resource issue. The core conflict here is between the limited resources, in terms of computing, network, storage, power supply, etc., and the higher and higher resource demands of IoT-based applications. On one hand, capacity of the IoT is limited by various practical factors, including technology, cost, environmental concerns, regulations, etc. On the other hand, the fast growing IoT applications have more and more stringent and heterogeneous resource requirements in order to satisfy their performance goals, especially in terms of the quality-of-service (QoS) received by their end-users. A representative example of such an application is data analytics and utilization based on machine learning, The excessive computing resources required by advanced machine learning algorithms have basically prevented those from being used directly on commodity IoT devices. End-users are left with the options of either utilizing cloud-based services provided by tech giants, which leads to cost, reliability, security and privacy issues, or resorting to far less powerful local algorithms that sacrifice accuracy and performance.

While in the long run, resolving such an issue relies on continuous investment into infrastructure enhancement in IoT, such a solution will not fully resolve the issue in the near future. In this dissertation, let us take a different and more practical approach. By abstracting IoT systems using mathematical models, we enable the design of different smart resource allocation algorithms that apply in various IoT scenarios. Such algorithms aim to enhance system performance by improving resource utilization in the IoT, while considering various constraints posed by both the infrastructure and the applications.

1.2 Overview

In this dissertation, we investigate the resource issue in IoT along three basic dimensions: network, security, and economics. We start from the most fundamental problem: network resource allocation in IoT. Here, we concern the sharing of network resource, notably bandwidth, among different competing entities. This includes bandwidth allocation both among multiple data traffic flows, as well as among multiple high-level applications each with a number of flows. However, network resource allocation is more than pure bandwidth allocation, due to other requirements of the applications. A typical one is the sensitivity of network delay, which is very common in real-time IoT applications such as autonomous driving and virtual/augmented reality. Another one is the reliability of the transmission, measured by the capability of successful transmission in face of failures in the network. A third one is the policy requirement, where data traffic must be pre-processed by one or multiple intermediate processing services before reaching the application and/or the end-users, which corresponds to security checks in security-sensitive applications or data pre-processing in big data analytics. Further, the allocation of network bandwidth is sometimes subject to the flexible deployment of the applications themselves. We consider these requirements, and devise efficient optimization algorithms for different concrete problems that fall into this category.

We next investigate the deployment of security resources in IoT. Deploying security functions in the edge network can greatly strengthen system security, by preventing most attacks close to their origins and/or targets. However, due to the high cost of edge computing power and the limited budget of the IoT operator, we need a smart deployment algorithm that can maximize the overall security of the system subject to the budget bound. We formulate this problem as a security risk minimization problem. Adopting models from economical risk estimation and stochastic optimization, we address the simultaneous minimization of expected and worst-case security risks with tunable trade-offs.

Finally, we look into the economics perspective of IoT, specifically, how users can make fast and secure micropayments for using IoT services. Traditional payment methods such as online transfer require a central platform for the recording and charging of transactions, which leads to performance bottlenecks, security concerns and excessive service fees. Using cryptocurrencies, users and providers can distributedly settle micropayments for the use of IoT services, with minimal (and ideally no) third party involvement. Unfortunately, mainstream cryptocurrencies such as Bitcoin and Ether are still suffering from the scalability issue and the high transaction fees coming from the expensive blockchain operations. Instead, off-chain payment channel networks (PCNs) can achieve fast and economical payment settlement, while still retaining a similar level of security as the blockchain, through the use of smart contracts that minimize blockchain operations. We investigate the fundamental payment routing problem in PCN, aiming to improve both the payment success ratio and the routing efficiency over existing solutions. We first consider a simple scenario where users make time-sensitive payment requests, and the network tries to satisfy as many payment requests as possible. We further consider anonymous payment routing that does not leak the user identities and/or transaction details.

1.3 Summarized Contributions

Addressing the aforementioned problems can be very challenging. Most of the problems that we study are NP-hard, meaning that they basically do not admit exact algorithms unless P=NP. Some problems also require fully distributed solutions or a solution that considers extreme cases of the system. Fortunately, for most decision making and/or optimization problems that we study, we were able to devise efficient solutions that either closely approximate the optimal solution (within a given bound), or converge to the optimal solution eventually. For problems that require distributed solutions, we devise distributed protocols that always ensure the validity of the solution based on our routing and security goals. This section summarizes our main contributions. Note that the concrete definitions of terms used, such as problem definitions, formulation and algorithm classes, are delegated to the corresponding chapters.

1.3.1 Part I: Network Resource Allocation in IoT

In network resource allocation, we consider three problems concerning different layers in the network stack. The first problem concerns the scenario where the network provider wishes to optimize per-traffic flow routing and bandwidth allocation in terms of network congestion minimization in the network layer. We consider different QoS requirements of user traffic, including transmission bandwidth and delays for real-time IoT applications, reliability for critical IoT applications, and policy routing for security-sensitive IoT applications. Our contribution is three-fold [153], which will be expanded in Chapter 2:

- We define the QoS-aware and Reliable Traffic Steering (QRTS) problem, incorporating the bandwidth, delay, reliability and policy routing requirements all into the same formulation.
- We prove QRTS and its optimization version OQRTS to be NP-hard.
- We propose a fully polynomial-time approximation scheme (FPTAS) for OQRTS.

Based on the previous study, we further seek to jointly optimize the hosting of applications and the routing of data traffic in IoT, aiming at a cross-layer design combining network layer and application layer controls. In this problem, we specifically consider real-time applications with bandwidth, delay and reliability requirements, while the policy routing requirement can be easily added and hence is omitted for simplicity. An application requires one or multiple data streams from different sources in the network. The task here is to find one or multiple host nodes for each given application, modeled as a singleton processing unit, as well as routing and bandwidth allocation for all the data streams directing to each application's host node(s). Based on the characteristics of the applications, we divide them into *parallelizable* and *non-parallelizable* applications, with the former being able to balance processing load across multiple instances while the latter cannot. Depending on the network scenario, we further respectively consider scenarios both with a single application, and with multiple simultaneous applications. Combining the two application types with the two provisioning scenarios, we have four variants of this problem. Our contribution to this problem is also three-fold [151], [152], which will be expanded in Chapter 3:

- We define the Application Provisioning (AP) problem with four variants, namely Parallelizable Single Application Provisioning (P-SAP), Non-parallelizable Single Application Provisioning (N-SAP), Parallelizable Multi-Application Provisioning (P-MAP), and Non-parallelizable Multi-Application Provisioning (N-MAP).
- We prove all variants of AP and their corresponding optimization versions to be NP-hard.
- We propose FPTASs for the optimization versions of P-SAP, N-SAP and P-MAP, respectively, and a randomized algorithm for the optimization version of N-MAP.

In the third study, we bring the problem to purely the application layer. Instead of considering the simplified traffic patterns such as the basic traffic flow in the first problem or the star-like data streams of a singleton application in the second problem, we consider a complex distributed application scenario where the IoT application is comprised of a set of IoT microservices with interdependencies. These interdependencies among microservices, which define how data will flow across microservices for this application, are modeled as a general directed acyclic graph (DAG). This is a far more general model than a source-destination flow or a star-like application. The microservices are distributed among different edge computing nodes in the IoT, with each microservice having one or multiple deployed instances. Our substrate network here is the overlay network consisting of all deployed instances of the microservices, as well as the overlay links between these instances. The problem that we study here is to balance the data transmission and processing load across microservice instances, taking into account both the complex interdependencies among microservices and the structure of the overlay network. Again, we need to consider the capacity of the microservice instances, as well as the QoS of a given distributed application in terms of both the transmission delay in the overlay network and the processing delay at these microservice instances. Our contribution to this problem is also three-fold [147], which will be expanded in Chapter 4:

- We define both the QoS-agnostic Basic Load Balancing (BLB) problem and the QoS-aware Load Balancing (QLB) problem for IoT microservices, taking into consideration the complex interdependencies among microservices modeled as a DAG, as well as the QoS requirement of any given application.
- We prove the QLB problem to be NP-hard.
- We propose an optimal algorithm for the BLB problem, based on which we further propose an FPTAS for the optimization version of the QLB problem, utilizing our novel decomposition of a load balancing solution into *realization graphs* (to be defined in Chapter 4).

1.3.2 Part II: Robust Security Deployment in IoT

Regarding IoT security, we mainly focus on the problem of deploying security functions within the IoT edge network. The goal of the IoT operator is to ensure the highest level of security in the network, by deploying these security functions to process incoming traffic into the edge network. These functions can be deployed on distributed edge computing nodes within the edge network. Ideally, deploying security functions at every access point can ensure the maximum level of security, since every piece of traffic is checked by a function before entering the edge network. Unfortunately, due to both the limited budget of the IoT operator and the limited availability of edge computing nodes, the operator can only deploy a small number of security functions at designated locations. To find the best deployment plan, the operator needs to both model the security status of the system, especially in face of frequent system dynamics, and employ efficient optimization algorithms to find the optimal deployment plan. Our contribution to this problem is three-fold [150], which will be expanded in Chapter 5:

- We propose a stochastic model for the security risk of the IoT edge network. Utilizing risk modeling techniques from economics, our model captures both the expected security risk, and the worst-case security risk in terms of the rare cases with the most unfavorable system dynamics.
- We formulate the robust security deployment problem minimizing the expected and worst-case security risks, which is both stochastic and NP-hard.
- We propose an iterative algorithm to compute the optimal security deployment plan based on the model. To improve its efficiency, we propose a novel analytical solving technique based on the structure of this problem.

1.3.3 Part III: Micropayment Routing in Blockchain-based PCN

The ability to conduct service-to-service micropayments is the key to enable the IoT ecosystem. PCN is a perfect match to this demand: it is automatic, with no human intervention; it is secured by the blockchain; and finally it is fast in settling transactions. The routing problem, however, is perhaps the most cumbersome task in the current PCN. Due to the requirement of the underlying security protocol, a payment route must both satisfy the balance (bandwidth) and the timelock (delay) requirements. While this problem itself is already NP-hard as shown in Part I, PCN routing is further complicated due to both that the network is fully distributed, and that the balance information on channels are constantly changing with each on-going transaction. These have driven us to design distributed protocols to solve this problem, instead of centralized optimization algorithms as in the previous chapters.

We address two problems associated with PCN routing. The first one is to design a basic routing protocol, which must both satisfy all requirements including delay and balance (concurrency) and be as efficient and low-overhead as possible. The challenge here is to both gather instantaneous network information and make balance reservation at the same time. We make the following contribution [149], which will be expanded in Chapter 6:

- We formulate the PCN routing problem with both balance/concurrency and timelock/delay requirements.
- We propose an efficient distributed protocol which probes for network information and makes balance reservation simultaneously in a one-round process, in the case where a single payment path can be found to satisfy the payment request. Multiple rounds may be needed if a single path cannot satisfy the request, in which case our protocol uses as few payment paths as possible to reduce overhead and limit the transaction fee paid by the sender of the payment.

The messages exchanged in the proposed protocol contain sensitive information regarding the identity and/or location of the sender and recipient of a payment, as well as the payment value. This can be a big issue in PCN, as often times users may wish to hide their identity and other sensitive information from external observers when making private transactions. To remedy the privacy issue, we further propose an anonymous probing protocol, which complements the important probing phase of the routing protocol to provide privacy guarantee. This, combined with conventional privacy-preserving communication solutions such as Sphinx [26] and HORNET [18], can ensure full privacy of the transacting users during the routing process, which can further be used in conjunction with privacy-preserving payment protocols [80], [81], making the entire PCN payment process privacy-preserving. Our contribution regarding the privacy issue is summarized as follows [148], which will be expanded in Chapter 7:

- We describe the privacy issue in distributed probing algorithms, and establish the privacy goals.
- We propose an efficient and privacy-preserving probing protocol, which ensures the anonymity of both the sender and the recipient, while still being able to communicate sufficient information to establish a payment path should the routing succeed, or to acknowledge a routing failure, without knowing in advance which path(s) will be explored.

Part I

Network Resource Allocation in IoT

Chapter 2

QOS-AWARE AND RELIABLE TRAFFIC STEERING FOR SERVICE FUNCTION CHAINING IN MOBILE NETWORKS

2.1 Introduction

The recent years have witnessed a drastic growth on global mobile traffic, due to the prevalent use of personal mobile devices and the emergence of the internet-of-things (IoT). Billions of devices are connected via mobile networks, posing a severe challenge to current mobile infrastructures. Moreover, the greatly abundant mobile and IoT applications have very heterogeneous requirements, including quality-of-service (QoS), security, availability, etc. Satisfying these requirements is difficult for current mobile networks, largely due to their hierarchical nature: most QoS and security features are implemented at the gateway or in the cloud, in a centralized manner. The gateway, with the need to both serve the huge amount of traffic and provide fine-grained network control, becomes a severe performance bottleneck of the mobile network.

There have been many efforts in addressing this performance bottleneck. The key idea is to resolve as much traffic as possible within the mobile network, alleviating the load on the gateway. One promising method is to deploy capacity and performance enhancing network services, also called *middleboxes*, to provide in-network traffic processing before traffic reaches the gateway. These include security components such as firewall, intrusion detection/prevention system (IDS/IPS) and deep packet inspection (DPI), network optimization tools such as load balancer (LB) and TCP optimizer, network address translator (NAT), etc. Deploying network services can bring a lot of benefits, such as early resolution of useless or malicious traffic, load balancing, security enhancement, etc.

Traditionally, each network service is implemented via dedicated hardware pieces, hence can only be deployed at specific locations (most likely at the gateway) due to cost issue. Thanks to the recent advances in network function virtualization (NFV), many network services can now be implemented as software components hosted on general-purpose computation platforms at the network edge, such as fog computing nodes within the mobile network. Edge deployment of network services has several advantages. First, this alleviates the excessive traffic load at the gateway. Second, in-network processing can effectively reduce traffic size in many scenarios, *e.g.*, data preprocessing for big data analytics, or preventing distributed deny-of-service (DDoS) attacks. Third, this reduces the delay experienced by mobile traffic, especially those transmissions whose both end-points reside in the mobile network (*e.g.*, machine-to-machine communications). With the emergence of fog computing [11], network services can be flexibly distributed in the network, which further helps in network optimization to balance and resolve mobile traffic load.

Nevertheless, benefits often do not come without a cost. Along with the enhanced performance and enriched flexibility, comes the increased complexity for *service function chaining* (SFC). In SFC, each traffic class is assigned a *service function chain*, which is a sequence of network services (also called *service functions*) that the traffic needs to pass through before exiting the network. Different traffic classes may have different service function chains, due to their various QoS, security and reliability requirements. An important problem is to steer each class of traffic through its required network services in the given order, wherein both routing and bandwidth allocation need to be determined based on the traffic class's requirements.

In this study, we study the traffic steering problem in mobile networks. We take a software-defined approach, where a centralized controller collects global network information, and makes joint routing and allocation decisions for all traffic classes together. A software-defined approach commonly achieves better routing and resource optimization, compared to distributed or local optimization approaches. We formulate the QoS-aware and Reliable Traffic Steering (QRTS) problem in mobile networks, considering heterogeneous requirements of traffic classes, including QoS (throughput and delay), reliability, security and type-of-transmission constraints, etc. Both QRTS and its optimization version (OQRTS) are proved to be NP-hard. We then propose a Fully-Polynomial Time Approximation Scheme (FPTAS) for the optimization problem. Through extensive simulation experiments, we validate that our proposed algorithm produces near-optimal solutions, and greatly outperforms two baseline heuristic algorithms.

Our main contributions are summarized as follows:

- To the best of our knowledge, we are the first to formulate the traffic steering problem in mobile networks with QoS and reliability requirements, and prove its NP-hardness.
- We develop a Fully-Polynomial Time Approximation Scheme for the optimization problem.
- We evaluate the performance of our proposed algorithm via extensive simulation experiments, which validates the near-optimal performance of our algorithm.

The rest of this study is organized as follows. In Section 2.2, we introduce existing work related to this study. In Section 2.3, we present our network and service model. In Section 2.4, we formally define and formulate the QRTS problem and its optimization version, and prove that both problems are NP-hard. In Section 2.5, we then propose our algorithm for the problem, and analyze its performance guarantee and time complexity. In Section 2.6, we present our performance evaluation results. In Section 2.7, we conclude this study.

2.2 Background and Related Work

2.2.1 NFV and SFC

NFV has been recognized as one of the enabling technologies to next-generation mobile networks [51], [84]. Various network components can be implemented via virtualization [51], including gateway, mobility support, charging, etc. In addition, traditional services like firewall, IDS/IPS, network optimizer and NAT, can also be implemented in mobile networks based on operator and user demands. Recent advances in NFV enable flexible and cost-efficient deployment of various network services in mobile networks [39], [62], [83].

SFC is a problem arisen in network management in the presence of network services. Gember *et al.* [40] proposed a network orchestration layer, in which issues such as elastic scaling, flexible placement and flow distribution are addressed. Zhang *et al.* [159] studied SFC in the view of network protocols, and proposed a heuristic solution for SFC-aware network service placement. Bari *et al.* [6] studied service chain embedding (joint traffic steering and network service placement), and proposed another heuristic solution. Rost *et al.* [109] proposed the first approximation algorithm for service chain embedding, though the approximation is not constant-ratio. For solely the traffic steering problem, Cao *et al.* [13] proposed an FPTAS, which is similar to the result reported in this study. However, their problem does not consider traffic QoS, hence is not NP-hard. Our problem considers the heterogeneous QoS requirements of applications, and is NP-hard, hence an FPTAS is the best possible algorithm that we can expect unless P=NP. Guo *et al.* [47] studied traffic steering with pre-defined path sets, which are not assumed in our study. The applications of SFC in mobile networks have been summarized in [49].

Failure of network services can be frequent and can have a large impact on the performance of applications and services [99], hence is one of the major considerations in this study. Rajagopalan *et al.* [103] proposed a Software-Defined Networking (SDN) based replication framework for network services. Sherry *et al.* [114] proposed a log-based recovery model for network services or middleboxes, which can be used to fast recover failed network services. Fan *et al.* [34] and Ye *et al.* [144] studied the problem of reliable service chain embedding, and proposed different heuristic algorithms based on both dedicated backup and shared backup provisioning. Kanizo *et al.* [63] proposed a network-agnostic solution for network service backups based on bipartite matching. The above efforts all focus on providing full recovery of the failed network services. On the contrary, we argue that this "all-or-nothing" protection is an overkill for many applications, as shown in existing work [158]. Therefore we propose a "soft" reliability mechanism, such that only a bounded portion of throughput is affected during an arbitrary service failure.

2.2.2 Software-defined Mobile Networks

SDN has recently been applied to facilitate network configuration and management in various network environments, including the mobile networks. Different usages of SDN in mobile networks have been studied, including resource allocation in the Radio

Access Networks (RANs) [46], traffic control in the Mobile Core Networks (MCN) [59], topology reconfiguration [86], etc. Our approach utilizes the centralized control as in the above, but considers its application in QoS-aware and reliable traffic steering for SFC in mobile networks.

2.3 System Model

2.3.1 Network Topology

The SDN controller aggregates global network information. The network is modeled as a directed graph $G = (V, E)$, where V is the set of nodes, and E is the set of links. A mobile network consists of many heterogeneous nodes, including radio access points (RAPs), core switches, standalone fog nodes, the central cloud, and the gateway towards the Internet. The link set also consists of heterogeneous links, including high-speed fronthaul/backhaul fibre, digital subscriber lines, wireless and satellite channels, etc. Different links have different attributes including QoS, security, etc., and thus can carry different types of traffic. We consider two QoS parameters for each link: bandwidth and delay. For each link $e \in E$, we denote $b_e > 0$ as its capacity, and $d_e > 0$ as its transmission delay.

Note that either a physically centralized controller (*e.g.*, at the mobile gateway) or a logically centralized controller (with physically distributed controllers sharing the global view) can be used. A hierarchical control plane is helpful in distributing the computational load in the mobile network. Distributed implementation of our proposed algorithm is out of the scope of this study, and hence is omitted due to page limit.

2.3.2 Service Functions

The mobile network is deployed with heterogeneous network services, also called *service functions*, for in-network processing of user traffic. For example, signal processing may be virtualized and flexibly deployed on local fog nodes or in the cloud, due to recent advances in Software-Defined Radio (SDR) [60] and NFV. Different signal processing steps can be virtualized into independent functions, which can be deployed on different nodes. As modern RANs employ heterogeneous radio access technologies (RATs), different RATs may require different types and sequences of service functions for signal processing. On the other hand, the network can offer other network services for enhanced security and performance on the network edge [49], including firewall, intrusion detection, load balancer, etc. Utilizing NFV, these network services can also be implemented as virtualized software components, and flexibly deployed on fog nodes. Deployment of service functions at the network edge benefits from its low latency and high location flexibility, compared to the traditional cloud-based deployment.

Formally, we use $M = \{m_1, \cdots, m_{|M|}\}$ to denote the set of service functions provided by all nodes in the RAN. Each service function may have multiple instances, deployed at different locations. We use $V_m \subseteq V$ to denote the set of nodes that are deployed with service function $m \in M$, and $M_v \subseteq M$ as the set of service functions deployed on node $v \in V$. For each service function $m \in M_v$ deployed on node v, we are concerned with two attributes: $b_{v,m} > 0$ as its processing capacity, and $d_{v,m} > 0$ as its processing delay.

We assume available service functions have already been deployed in the network.

Service function placement is out of the scope of this study, and will be studied in future work.

2.3.3 Traffic Model

Traffic is aggregated and classified based on its access/exit points, QoS requirements, service function chain (*service chain* for short), types of traffic, and reliability requirement. In mobile networks, two most important QoS attributes are bandwidth and transmission delay. Denote all traffic classes (TCs) in the network as $\mathcal{C} = \{C_1, \cdots, C_{|\mathcal{C}|}\}$. Each TC is denoted as a 7-tuple $C_j = (s_j, t_j, B_j, D_j, \Pi_j, \mathcal{T}_j, r_j)$, where $s_j, t_j \in V$ denote the access and exit nodes respectively, $B_j > 0$ denotes the bandwidth demand, $D_j > 0$ denotes the maximum delay bound, Π_j denotes its service chain, $\mathcal{T}_j$ denotes the per-stage traffic type for each traffic stage defined in the service chain, and $r_j > 0$ denotes the reliability requirement. Explanation of the reliability requirement is deferred to Section 2.3.5.

Each TC's service chain is defined as a sequence of service functions, $\Pi_j = (\pi_1^j, \cdots, \pi_{\kappa_j}^j)$, where each $\pi_k^j \in M$ denotes a service function required by the TC. The *service function chaining* requirement specifies that each packet of the TC originates from s_j, passes through all required service functions in the order given in Π_j, and exits at t_j. We assume each service chain contains only distinct service functions.

The chaining requirement splits transmission of the TC into (κ_j+1) stages: $s_j \to \pi_1^j$, $\pi_k^j \to \pi_{k+1}^j$ for $k = 1, \ldots, \kappa_j - 1$, and $\pi_{\kappa_j}^j \to t_j$. Each stage of traffic may belong to a different traffic type, and can be carried on only a subset of links. For example, in Cloud-RANs, the uplink traffic enters the network as wireless radio signals; before signal processing, such traffic can only be transmitted along high-speed fronthaul fibre which supports the Common Public Radio Interface (CPRI). On the other hand, traffic already through some security functions can no longer be transmitted via potentially insecure links. These are expressed in $\mathcal{T}_j = \{T_1^j, \cdots, T_{\kappa_j+1}^j\}$, where $T_k^j \subseteq E$ is the subset of links that can carry the stage-k traffic of C_j.

2.3.4 Feasible Routing Graph

We first define the feasible routing paths for TCs. Given network G and a TC C_j, a path p in G is *feasible* for C_j iff

1. p originates from s_j and ends at t_j;
2. p visits all service functions $\Pi_j = (\pi_1^j, \cdots, \pi_{\kappa_j}^j)$ in the given order; and
3. p has total transmission and processing delay within D_j.

To better establish the feasibility constraints of a routing path, we construct a per-TC routing graph $G_j^{ext} = (V_j^{ext}, E_j^{ext})$ from the original graph G, as shown in Fig. 2.1. G_j^{ext} has $(\kappa_j + 1)$ layers, each layer k corresponding to one copy of the subgraph of G that contains all nodes in V and all links in T_k^j; this enforces the traffic type constraints. We denote $v_k^j \in V_j^{ext}$ as the copy of node $v \in V$ in layer-k of G_j^{ext}, and $e_k^j \in E_j^{ext}$ as the copy of link $e \in E$ in layer-k of G_j^{ext}. We call link e the *prototype* of e_k^j, denoted by $proto(e_k^j)$; we also call e_k^j an *extended link* of e. Link e_k^j has the same transmission delay d_e as its prototype. Further, we establish abstract links between consecutive layers. For each service function $\pi_k^j \in \Pi_j$, we establish an abstract link from the copy of each node $v \in V_{\pi_k^j}$ in the k-th layer, to its copy in the $(k+1)$-th layer. We denote this abstract link as $e_v^{j,k} = (v_k^j, v_{k+1}^j)$, and let it have delay d_{v,π_k^j}

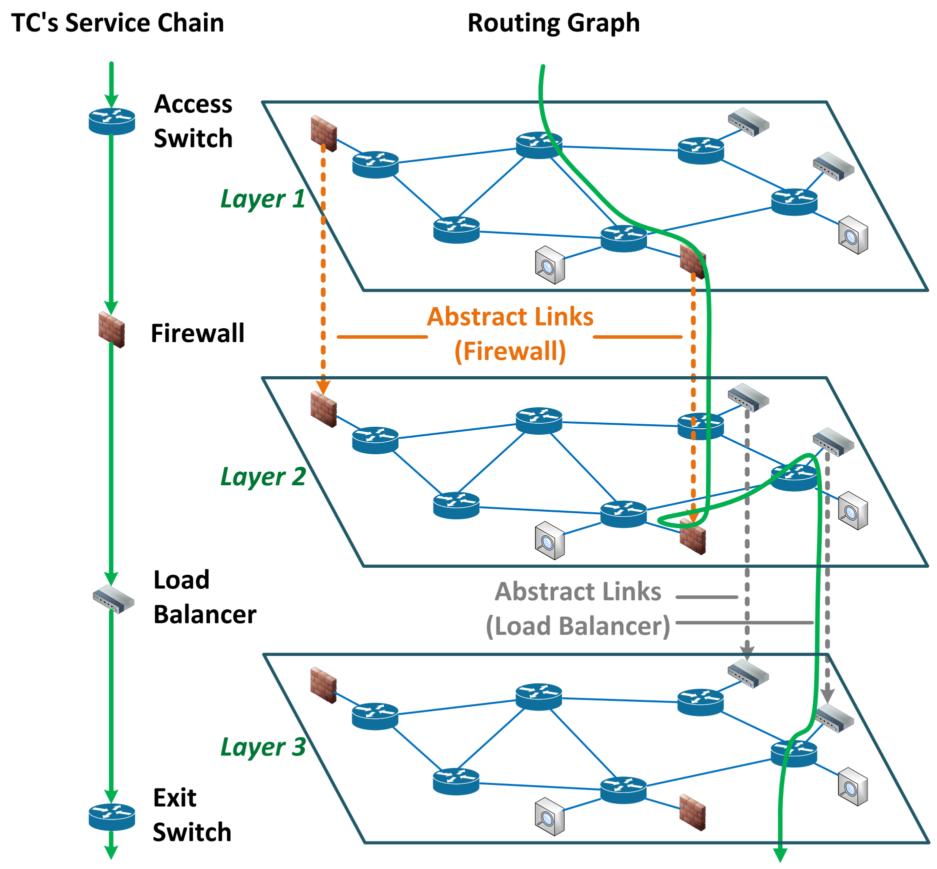


Figure 2.1: A TC's service function chain and routing graph.

(processing delay of the instance). We use the pair (v, π_k^j) to denote the *prototype* of link $e_v^{j,k}$, also denoted by $proto(e_v^{j,k})$; $e_v^{j,k}$ is thus an *extended link* of prototype (v, π_k^j).

Let s_0^j be node s_j at layer 0 of G_j^{ext}, and $t_{\kappa_j}^j$ be t_j at layer κ_j of G_j^{ext}. We assume that each routing graph G_j^{ext} is $(s_0^j, t_{\kappa_j}^j)$-*connected*, meaning that there is a routing path from s_0^j to every node $v \in V_j^{ext}$, and there is a routing path from every node $v \in V_j^{ext}$ to $t_{\kappa_j}^j$. Nodes not satisfying this condition can be safely removed from the routing graph, as it does not contribute to the connectivity between s_0^j and $t_{\kappa_j}^j$.

For simplicity, we aggregate all TCs' routing graphs into a giant one, denoted as $G^{ext} = (V^{ext}, E^{ext})$, where $V^{ext} = \bigcup_{C_j \in \mathcal{C}} V_j^{ext}$, and $E^{ext} = \bigcup_{C_j \in \mathcal{C}} E_j^{ext}$. Each TC's subgraph G_j^{ext} is *maximally* $(s_0^j, t_{\kappa_j}^j)$*-connected* in G^{ext}, meaning that adding any node $v \notin V_j^{ext}$ makes it not $(s_0^j, t_{\kappa_j}^j)$-connected.

Given G^{ext}, the *feasible path set* of C_j is defined as all paths from s_0^j to $t_{\kappa_j}^j$, each with the sum of link delays no greater than D_j. Note that the processing delays of service function instances have already been accounted for in their extended links' delays. We use $\mathcal{P}_j$ to denote the feasible path set for C_j, and let $\mathcal{P} = \bigcup_{C_j \in \mathcal{C}} \mathcal{P}_j$. Without loss of generality, we assume that each TC has a disjoint feasible path set $\mathcal{P}_j$.

The following notations are defined for simplicity. We denote $E_\nu = \{(v, m) \mid m \in M, v \in V_m\}$ as the set of all service function instances. $\mathcal{E} = E \cup E_\nu$, also called the *prototype* set, denotes the set of all original physical links and service function instances. We then use $E^{ext}(e)$ to denote all extended links of the same prototype $e \in \mathcal{E}$, *i.e.*, all links that share the same capacity bound b_e. We also use $\mathcal{E}(p)$ and $E_\nu(p) = \mathcal{E}(p) \cap E_\nu$ to denote the sets of all prototypes and only service function prototypes, respectively, used by path p. $\eta_p(e)$ denotes the number of times for which prototype e's extended links appear in path p. Note that $\eta_p(e) \leq 1$ if $e \in E_\nu$, as each service function chain contains only distinct service functions.

2.3.5 Reliability

Network service failures can downgrade or even halt the transmission of user traffic, which must be tackled to assure service continuity [99]. On the other hand, it has been revealed that the traditional "all-or-nothing" protection is actually an overkill for many data applications [158], due to the excessive resource consumption to provide such protection.

In this study, we follow existing work and seek a "milder" way for improving service availability [158]. Instead of providing full recovery, we seek to bound the amount of throughput loss due to an arbitrary *single service function instance failure* (*single service failure* for short). Specifically, each TC $C_j \in \mathcal{C}$ has a reliability parameter $r_j \in (0, B_j]$, denoting the *maximum tolerable throughput loss* that C_j may suffer from any single service failure; $r_j = B_j$ means no protection for C_j.

2.4 Problem Statement

2.4.1 Problem Description and Formulation

In this study, we study the traffic steering problem in mobile networks. Specifically, given the network G and the set of TCs $\mathcal{C}$, the network operator's goal is to find a subset of feasible routing paths, as well as allocate bandwidth for each path, to fulfill the bandwidth demand of each TC, meanwhile satisfying both capacity bounds and reliability requirements.

Definition 2.1 (Bandwidth allocation). *Let $P \in \mathcal{P}$ be a subset of feasible routing paths. A* bandwidth allocation *of P is defined by a mapping $\mathcal{L} : P \mapsto \mathbb{R}^+$, where $\mathbb{R}^+$ is the positive real number set. We say that $\mathcal{L}$ is a* feasible bandwidth allocation *of P iff for each prototype $e \in \mathcal{E}$, $\sum_{p \in P : e \in \mathcal{E}(p)} \eta_e(p)\mathcal{L}(p) \le b_e$. The* aggregate bandwidth *of P, denoted by $b(P)$, is the sum of bandwidth allocated on all paths in P:*

$$b(P) = \sum_{p \in P} \mathcal{L}(p). \qquad \square$$

Definition 2.2 (QRTS). *Given the network $G = (V, E)$, and the TC set $\mathcal{C}$, the* **QoS-aware and Reliable Traffic Steering (QRTS)** *problem in mobile networks* *is to find a tuple $\Gamma = (P, \mathcal{L})$, where $P \subseteq \mathcal{P}$ is a subset of feasible routing paths, and $\mathcal{L}$ is a feasible bandwidth allocation of P, such that*

1. *let $P_j \subseteq P$ be the set of feasible routing paths of TC C_j in P, then $b(P_j) \geq B_j$ for each TC $C_j \in \mathcal{C}$; and*
2. *during an arbitrary single service failure, at most r_j bandwidth is lost for each $C_j \in \mathcal{C}$.* □

2.4.2 Computational Complexity

Theorem 2.1. *QRTS is NP-complete.* □

Proof. First, QRTS is in NP, as checking all constraints takes polynomial time. Consider the special case of QRTS where there is only one TC with an empty service chain, no link excluded from E in $\mathcal{T}_j$, and no reliability requirement ($r_j = B_j$). Therefore, there is only one layer in its routing graph, which is the same as the original topology. In this case, we obtain the Multi-Path routing with Bandwidth and Delay constraints (MPBD) problem on a general graph, which has been proven NP-complete in [85]. As a known NP-complete problem is a special case of QRTS, the theorem follows. □

2.4.3 Optimization Formulation

The QRTS problem is an NP-complete decision problem. We further define the following optimization version of QRTS:

Definition 2.3 (OQRTS)**.** *Given the network $G = (V, E)$, and the TC set $\mathcal{C}$, the* **Optimal QoS-aware and Reliable Traffic Steering (OQRTS)** *problem in mobile networks is to find a tuple $\Gamma = (P, \mathcal{L})$, where $P \subseteq \mathcal{P}$ is a subset of feasible routing paths, and $\mathcal{L}$ is a feasible bandwidth allocation of P, such that*

1. *let $P_j \subseteq P$ be the set of feasible routing paths of TC C_j in P, then $b(P_j) \geq \xi \cdot B_j$ for each TC $C_j \in \mathcal{C}$;*
2. *during an arbitrary single service failure, at most r_j bandwidth is lost for each $C_j \in \mathcal{C}$; and*
3. *ξ is maximized.* $\square$

In the OQRTS problem, the network operator aims to maximize the *traffic scaling ratio* ξ, defined as the minimum ratio between the aggregate bandwidth and the demand of any TC, subject to the feasible path set, feasibility of bandwidth allocation and reliability constraints. If an OQRTS instance has an optimal solution of $\xi^* \geq 1$, the corresponding QRTS instance is feasible, and vice versa.

With $\mathcal{L}(p)$ defined as the per-path variable of bandwidth allocation, and ξ as the minimum scaling ratio, we formulate the OQRTS problem as the following linear program (LP):

$$\begin{aligned}
\max \quad & \xi && && \text{(2.1a)}\\
\text{s.t.} \quad & \sum_{p\in\mathcal{P}_j} \mathcal{L}(p) \geq \xi B_j, && \forall C_j \in \mathcal{C} && \text{(2.1b)}\\
& \sum_{p\in\mathcal{P}: e\in\mathcal{E}(p)} \eta_p(e)\mathcal{L}(p) \leq b_e, && \forall e \in \mathcal{E} && \text{(2.1c)}\\
& \sum_{p\in\mathcal{P}_j: e\in E_\nu(p)} \mathcal{L}(p) \leq r_j, && \forall C_j \in \mathcal{C}, e \in E_\nu && \text{(2.1d)}\\
& \mathcal{L}(p), \xi \geq 0. && \forall p \in \mathcal{P} &&
\end{aligned}$$

Explanation: Objective (2.1a) is to maximize the traffic scaling ratio ξ. Constraint (2.1b) defines the scaling ratio for each TC: $(\sum_{p\in\mathcal{P}_j} \mathcal{L}(p))/B_j \geq \xi$. Constraint (2.1c) enforces per-prototype capacities. Constraint (2.1d) enforces the reliability for each TC C_j. Specifically, for each service instance $e \in E_\nu$, the amount of traffic of C_j through e must not exceed the maximum tolerable throughput loss, denoted by r_j. By this constraint, any single function instance failure will affect at most r_j bandwidth, hence satisfying the reliability requirements.

While the above formulation is a linear program (LP), it can have an exponential number of variables due to the potentially exponential number of feasible routing paths in a given graph. This prevents solving the problem using standard LP techniques. Note that since the decision problem QRTS is NP-hard, so is the optimization problem OQRTS. In the next section, we propose our approximation algorithm for OQRTS.

2.5 Fully Polynomial-Time Approximation Scheme

In this section, we design an FPTAS for the OQRTS problem. Since the problem is NP-hard, an FPTAS is the best algorithm one can hope for, unless P=NP.

Definition 2.4 (FPTAS). *Given a maximization problem Ω, an algorithm $\mathcal{A}$ is said to be a Fully-Polynomial Time Approximation Scheme (FPTAS) for Ω, iff for any instance of Ω with optimal objective value ζ^*, given any $\omega \in (0, 1)$, $\mathcal{A}$ can produce a feasible solution with objective value $\zeta \geq (1 - \omega) \cdot \zeta^*$, within time polynomial to both the input size and $1/\omega$.* $\square$

2.5.1 Dual Analysis

We first write the dual program of (2.1). Define dual variable $z(j)$ for Constraint (2.1b) with each $C_j \in \mathcal{C}$, $l(e)$ for Constraint (2.1c) with each $e \in \mathcal{E}$, and $\varphi(j, e)$ for Constraint (2.1d) with each $C_j \in \mathcal{C}$ and $e \in E_\nu$, the dual program is as follow:

$$\min \quad \sum_{e \in \mathcal{E}} b_e l(e) + \sum_{C_j \in \mathcal{C}} \sum_{e \in E_\nu} r_j \varphi(j, e) \tag{2.2a}$$

$$\text{s.t.} \quad \sum_{e \in \mathcal{E}(p)} \eta_p(e) l(e) + \sum_{e \in E_\nu(p)} \varphi(j, e) \geq z(j), \qquad \forall C_j \in \mathcal{C}, p \in \mathcal{P}_j \tag{2.2b}$$

$$\sum_{C_j \in \mathcal{C}} B_j z(j) \geq 1, \tag{2.2c}$$

$$z(j), l(e), \varphi(j, e) \geq 0. \qquad \forall C_j \in \mathcal{C},\ e \in \mathcal{E}$$

Explanation: Objective (2.2a) accounts for the constants in Constraints (2.1c) and (2.1d). Constraint (2.2b) is the dual constraint for primal variable $\mathcal{L}(p)$. Constraint (2.2c) is the dual constraint for primal variable ξ. For simplicity of notations, although $\varphi(j, e)$ is only defined for each $C_j \in \mathcal{C}$ and $e \in E_\nu$, we extend its definition to include $C_j \in \mathcal{C}$ and any $e \in \mathcal{E}$, and explicitly let $\varphi(j, e) = 0$ for $e \in \mathcal{E} \setminus E_\nu$.

Based on an observation similar to the one in [38], we have the following two lemmas:

Lemma 2.1. *At any optimal solution of Program* (2.2)*, Constraint* (2.2c) *is* binding*,* i.e.*, equality (rather than strict inequality) holds.* □

Lemma 2.2. *At any optimal solution of Program* (2.2)*, there exists at least one path* $p \in \mathcal{P}_j$ *for any* $C_j \in \mathcal{C}$*, such that Constraint* (2.2b) *with* C_j *and* p *is* binding*.* □

Proof. Assume that at an optimal solution, Constraint (2.2b) for any TC $C_j \in \mathcal{C}$ and path $p \in \mathcal{P}_j$ is not binding. It is obvious that we can reduce the value of any $l(e)$ or $\varphi(j,e)$ that has a positive value, by an arbitrarily small amount. The resulted solution is still feasible, but has a strictly smaller objective value than the optimal solution, leading to a contradiction. To prove Lemma 2.1, observe that if Constraint (2.2c) is not binding, then we can reduce the value of $z(j)$ for all $C_j \in \mathcal{C}$ by an arbitrarily small amount, which will make every Constraint (2.2b) to be unbinding, leading to the same contradiction. To prove Lemma 2.2, assume that there exists $C_j \in \mathcal{C}$ such that Constraint (2.2b) is not binding for any $p \in \mathcal{P}_j$, then we can increase the value of $z(j)$ by an arbitrarily small amount, which will make Constraint (2.2c) unbinding. This leads to the same contradiction as above. Hence both lemmas follow. □

Based on Lemma 2.2, it is now clear that at any optimal solution, $z(j) = \min_{p\in\mathcal{P}_j}\{\sum_{e\in\mathcal{E}(p)} \eta_p(e)l(e) + \sum_{e\in E_\nu(p)} \varphi(j,e)\}$. In other words, $z(j)$ is equal to the shortest path length in $\mathcal{P}_j$ regarding the per-link length function $\varsigma(\varepsilon) = l(\varepsilon) + \varphi(j,\varepsilon)$ for $\varepsilon \in E^{ext}$, where $l(\varepsilon) = l(proto(\varepsilon))$, and $\varphi(j,\varepsilon) = \varphi(j, proto(\varepsilon))$, respectively.

Lemmas 2.1 and 2.2 help us refine the dual program into a more concise form, removing variables $z(j)$. Define $D(l,\varphi) = \sum_{e\in\mathcal{E}} b_e l(e) + \sum_{C_j\in\mathcal{C}} \sum_{e\in E_\nu} r_j\varphi(j,e)$ (the dual objective function), and $\alpha(l,\varphi) = \sum_{C_j\in\mathcal{C}} B_j\delta_j(l,\varphi)$, where $\delta_j(l,\varphi) = \min_{p\in\mathcal{P}_j}\{\sum_{\varepsilon\in p}\varsigma(\varepsilon)\}$ is the shortest path length in $\mathcal{P}_j$ under length function ς. The dual problem is equivalent to finding length functions l and φ that minimize $D(l,\varphi)/\alpha(l,\varphi)$:

$$\min_{l,\varphi\geq 0} \quad \frac{D(l,\varphi)}{\alpha(l,\varphi)}. \tag{2.3}$$

2.5.2 Primal-Dual Algorithm

Our approximation scheme is based on a similar design as in [35], [38]. The intuitive is to greedily push flow along the dual-shortest feasible path for each TC, meanwhile updating the lengths such that the length of each prototype increases exponentially in the amount of its constraint violation. After a number of rounds, the flow is distributed approximately evenly in the network. By scaling the final flow with the bounded link lengths, we obtain a feasible solution that approximates the optimal. The algorithm is shown in Algorithm 2.1.

Lines 1–2 of Algorithm 2.1 initialize the length of each prototype, where γ is a value to be determined in Section 2.5.4. P_j and $\mathcal{L}$ denote the paths used by C_j and the bandwidth allocation over all paths respectively, both initialized to empty. After constructing the routing graph, we initialize the lengths of all extended links the same as their prototypes. The algorithm proceeds in phases (Lines 6–19), each phase consisting of $|\mathcal{C}|$ iterations (Lines 8–18). In the j-th iteration in each phase, the algorithm pushes B_j units of flow for TC T_j, which is done in steps (Lines 10–17). In each step, the algorithm finds the shortest feasible path $\tilde{p}$ for TC T_j under length function ς, and pushes ϕ units of flow through $\tilde{p}$, where ϕ is defined by the residual flow demand B'_j, the bottleneck capacity $\min_{e\in\mathcal{E}(\tilde{p})}\{b_e/\eta_{\tilde{p}}(e)\}$, and the reliability requirement r_j, whichever is the smallest. Since $\tilde{p}$ may pass through the same prototype for multiple times, the capacity b_e of each $e \in \mathcal{E}(p)$ is divided by $\eta_{\tilde{p}}(e)$, the number of times that it appears in $\tilde{p}$. After updating P_j, $\mathcal{L}(p)$ and B'_j,

Algorithm 2.1: Primal-Dual Algorithm for OQRTS

```
   Input: Topology $G$, TCs $\mathcal{C}$, tolerance $\omega$
   Output: Scaling ratio $\xi$, path sets $\{P_j\}$, bandwidth $\mathcal{L}$
 1 Initialize $l(e) \leftarrow \gamma/b_e$ for $\forall e \in \mathcal{E}$;
 2 Initialize $\varphi(j,e) \leftarrow \gamma/r_j$ for $\forall C_j \in \mathcal{C}, e \in E_\nu$, and $\varphi(j,e) \leftarrow 0$ for
     $\forall C_j \in \mathcal{C}, e \in \mathcal{E} \setminus E_\nu$;
 3 Initialize $P_j \leftarrow \emptyset$, $\mathcal{L} \leftarrow \emptyset$;
 4 Construct the routing graph $G^{ext}$, and let $l(\varepsilon) \leftarrow l(e)$, $\varphi(j,\varepsilon) \leftarrow \varphi(j,e)$ and
     $c_\varepsilon = c_e$ for $\forall \varepsilon \in E^{ext}, e = proto(\varepsilon)$;
 5 $\rho \leftarrow 0$;
 6 while $D(l,\varphi) < 1$ do
 7     $\rho \leftarrow \rho + 1$;
 8     for each TC $C_j \in \mathcal{C}$ do
 9         $B'_j \leftarrow B_j$;
10         while $B'_j > 0$ do
11             $\tilde{p} \leftarrow \arg\min_{p \in \mathcal{P}_j} \{\sum_{\varepsilon \in p} \varsigma(\varepsilon)\}$;
12             $\phi \leftarrow \min\{B'_j, \min_{e \in \mathcal{E}(\tilde{p})} \{\frac{b_e}{\eta_{\tilde{p}}(e)}\}, r_j\}$;
13             $P_j \leftarrow P_j \cup \{\tilde{p}\}$;
14             $\mathcal{L}(\tilde{p}) \leftarrow \mathcal{L}(\tilde{p}) + \phi$, $B'_j \leftarrow B'_j - \phi$;
15             $l(e) \leftarrow l(e)(1 + \epsilon \cdot \frac{\phi \eta_{\tilde{p}}(e)}{b_e})$ $\forall e \in \mathcal{E}(\tilde{p})$, and $l(\varepsilon) \leftarrow l(e)$ for $\forall \varepsilon \in E^{ext}(e)$;
16             $\varphi(j,e) \leftarrow \varphi(j,e)(1 + \epsilon\frac{\phi}{r_j})$ $\forall e \in E_\nu(\tilde{p})$, and $\varphi(j,\varepsilon) \leftarrow \varphi(j,e)$ for
                 $\forall \varepsilon \in E^{ext}(e)$;
17         end
18     end
19 end
20 $\xi \leftarrow (\rho - 1)/\log_{1+\epsilon} 1/\gamma$;
21 $\mathcal{L}(p) \leftarrow \mathcal{L}(p)/\log_{1+\epsilon} 1/\gamma$ for $\forall C_j \in \mathcal{C}, p \in P_j$;
22 return $(\xi, \{P_j\}, \mathcal{L})$.
```

the algorithm updates the per-prototype lengths $l(e)$ and the per-TC per-function instance lengths $\varphi(j, e)$, in Line 15–16. The value of ϵ is also to be determined in Section 2.5.4. The lengths of all extended links in the routing graph are then updated to reflect the change of lengths of their prototypes. Note that the resulting flow may exceed the capacity of each prototype. However, as we will show in Section 2.5.4, scaling the flow on each link by $\log_{1+\epsilon} 1/\gamma$ yields a feasible solution.

2.5.3 Approximating Shortest Feasible Paths

Algorithm 2.1 relies on finding the shortest feasible path for each TC under length function ς. However, since the feasible path set of each TC only contains delay-bounded paths, this task is non-trivial. In fact, finding the shortest delay-bounded path is known as the Delay Constrained Least Cost path (DCLC) problem, which is also NP-hard [139]. Nevertheless, there exist FPTASs for the DCLC problem, which output a path within $(1 + \omega')$ of the shortest delay-bounded path [75], [139]. In the next subsection, we will show that for the purpose of our algorithm, it is sufficient to find a $(1 + \omega')$-approximate ς-shortest path with strictly bounded delay.

2.5.4 Algorithm Analysis

Theorem 2.2. *Given G, $\mathcal{C}$ and ω, let $\omega' = \epsilon = \frac{\omega}{4}$, and $\gamma = \left(\frac{1-(1+\omega')\epsilon}{|\mathcal{E}|+|E_\nu||\mathcal{C}|}\right)^{\frac{1+\epsilon(1+\omega')}{\epsilon(1+\omega')}}$, then Algorithm 2.1 outputs a feasible solution that is within $(1 - \omega)$ times of the optimal solution, if the dual optimal objective value $\Delta \geq 1$.* □

Proof. We prove by bounding the primal-dual ratio for the solutions derived in the algorithm. The basic idea is that the primal value increases linearly with the flow pushed in each phase, but each link's dual lengths increase exponentially with the flow through it. After a polynomial number of phases, the primal-dual ratio is then within the desired bound.

We first define some notations. For any symbol υ (including $l, \varphi, \phi, \tilde{p}$), we use $\upsilon^s_{\rho,\tau}$ to denote its value after phase-ρ, iteration-τ, and step-s of the algorithm, $\upsilon_{\rho,\tau}$ to denote its value after phase-ρ and iteration-τ, and υ_ρ to denote its value after phase-ρ. For symbols in the form of $\upsilon(l, \varphi)$ (including D, α, δ_j), we use $\upsilon^s_{\rho,\tau}$ to denote $\upsilon(l^s_{\rho,\tau}, \varphi^s_{\rho,\tau})$, and similarly $\upsilon_{\rho,\tau}$ and υ_ρ. We then have

$$\begin{aligned} D^s_{\rho,\tau} \leq \sum_{e\in\mathcal{E}} b_e l^{s-1}_{\rho,\tau}(e) + \sum_{C_j\in\mathcal{C}} \sum_{e\in E_\nu} r_j \varphi^{s-1}_{\rho,\tau}(j,e) \\ + \epsilon\phi^s_{\rho,\tau}(1+\omega') \cdot \delta^{s-1}_{j,\rho,\tau} \\ \leq D^{s-1}_{\rho,\tau} + \epsilon\phi^s_{\rho,\tau}(1+\omega') \cdot \delta^s_{j,\rho,\tau}, \end{aligned}$$

where $j = \tau$ due to that in iteration-τ of each phase we only consider TC $C_j = C_\tau$; the first inequality is because path $\tilde{p}$ found in each step is a $(1+\omega')$-approximation of the shortest feasible routing path; the second inequality is due to that shortest feasible path length is monotonically non-decreasing. Summing up the flow pushed in all steps of iteration-τ where we push B_j flow in total, we have

$$D_{\rho,\tau} \leq D_{\rho,\tau-1} + \epsilon B_j(1+\omega') \cdot \delta_{j,\rho,\tau},$$

and hence

$$\begin{aligned} D_\rho &\leq D_{\rho-1} + \epsilon(1+\omega') \sum\nolimits_{j=1}^{|\mathcal{C}|} B_j \cdot \delta_{j,\rho} \\ &\leq D_{\rho-1} + \epsilon(1+\omega')\alpha_\rho. \end{aligned}$$

Let the optimal dual solution be $\Delta = \min\limits_{l,\varphi\geq 0}\{\frac{D(l,\varphi)}{\alpha(l,\varphi)}\}$, we know that $\frac{D_\rho}{\alpha_\rho} \geq \Delta$. Since we assume that $\Delta \geq 1$, we have

$$\begin{aligned} D_\rho &\leq \frac{D_{\rho-1}}{1-\frac{\epsilon(1+\omega')}{\Delta}} \\ &\leq \frac{D_0}{\left(1-\frac{\epsilon(1+\omega')}{\Delta}\right)^\rho} \\ &\leq \frac{D_0}{1-\epsilon(1+\omega')} e^{\frac{(\rho-1)\epsilon(1+\omega')}{\Delta(1-\epsilon(1+\omega'))}}, \end{aligned}$$

where the initial objective $D_0 = (|\mathcal{E}| + |E_\nu||\mathcal{C}|)\gamma$ due to the initial value of $l(e)$ and $\phi(j,e)$. The last inequality is due to $(1+x) \leq e^x$, where $x = \frac{\epsilon(1+\omega')}{\Delta-\epsilon(1+\omega')}$ in the inequality.

Now, assume the algorithm stops at phase ρ^*, hence $D_{\rho^*} \geq 1$ yet $D_{\rho^*-1} < 1$. Taking it into the above inequality, we have

$$\frac{\Delta}{(\rho^*-1)} \leq \frac{\epsilon(1+\omega')}{(1-\epsilon(1+\omega'))\ln\frac{1-\epsilon(1+\omega')}{(|\mathcal{E}|+|E_\nu||\mathcal{C}|)\gamma}}.$$

On the other hand, by the way we update lengths $l(e)$ and $\varphi(j,e)$ at Lines 15–16, each dual variable has its value increased by at least $(1+\epsilon)$ times when the corresponding primal constraint is *filled* for once, *i.e.*, when the flow through prototype $e \in \mathcal{E}$ increases by b_e, or when the flow for TC C_j through a function instance $e \in E_\nu$ increases by r_j, respectively. Since $D_{\rho^*-1} < 1$, we have $l_{\rho^*-1}(e) < 1/b_e$ and $\varphi_{\rho^*-1}(j,e) < 1/r_j$. Therefore, the flow after phase (ρ^*-1) scaled by $1/\log_{1+\epsilon} 1/\gamma$ is strictly feasible, which means the final scaling ratio $\xi = (\rho^*-1)/\log_{1+\epsilon} 1/\gamma$ is feasible.

The primal-dual ratio is then bounded by

$$\frac{\xi}{\Delta} \geq \frac{(1-\epsilon(1+\omega'))\cdot \ln\frac{1-\epsilon(1+\omega')}{(|\mathcal{E}|+|E_\nu||\mathcal{C}|)\gamma}}{\epsilon(1+\omega')\cdot\log_{1+\epsilon}\frac{1}{\gamma}}.$$

Given the selection of ϵ, ω' and γ, we have $\frac{\xi}{\Delta} \geq 1-\omega$. □

For time complexity, we define $O^*(f) = O(f \log^{O(1)}(\mathbb{L}))$, where f is a function of the input size, and $\mathbb{L}$ is the number of values in the input (independent of each value's magnitude).

Theorem 2.3. *The worst-case time complexity of Algorithm 2.1 is* $O^*(\frac{\Delta}{\omega^3}|V||\mathcal{E}|(|\mathcal{E}|+|E_\nu||\mathcal{C}|)\kappa_{\max}^2)$*, where* $\kappa_{\max} = \max_j\{\kappa_j\}$ *is the maximum service chain length of any TC.* □

Proof. As above, we have $\frac{\xi^*}{\Delta} > \frac{\xi}{\Delta} \geq \frac{(\rho^*-1)}{\Delta \log_{1+\epsilon} 1/\gamma}$. By strong duality of linear programming, we have $\frac{\xi^*}{\Delta} = 1$. Therefore, the number of phases is bounded by $\rho^* \leq \lceil \Delta \log_{1+\epsilon} 1/\gamma \rceil$. The number of iterations is $|\mathcal{C}|$ times the number of phases. In each iteration, every but the last step increases the length of at least one $l(e)$ or $\phi(j,e)$ by $(1+\epsilon)$ times, hence the number of steps exceeds the number of iterations by at most $(|\mathcal{E}| + |E_\nu||\mathcal{C}|)\log_{1+\epsilon}\frac{1+\epsilon}{\gamma}$. Thus totally there are $O^*(\frac{\Delta}{\omega^2}(|\mathcal{C}| + |\mathcal{E}| + |E_\nu||\mathcal{C}|)) = O^*(\frac{\Delta}{\omega^2}(|\mathcal{E}| + |E_\nu||\mathcal{C}|))$ steps by the choices of ϵ, ω' and γ. Each step incurs one approximate shortest feasible path computation, which by Xue *et al.* [139] is computed in $O(|V_j^{ext}||E_j^{ext}|(\frac{1}{\omega'} + \log\log\log|V_j^{ext}|))$ time in each TC's routing graph G_j^{ext}. Both the node set and the link set are bounded by $|V_j^{ext}| = O(|V| \cdot \kappa_{\max})$ and $|E_j^{ext}| = O(|\mathcal{E}| \cdot \kappa_{\max})$ respectively. The theorem follows. □

2.5.5 Feasibility and Demand Scaling

Theorem 2.2 relies on two facts: 1) the QRTS instance has a non-zero feasible solution, and 2) the optimal dual objective value $\Delta \geq 1$. On the other hand, the time complexity of Algorithm 2.1 is proportional to Δ, hence it should not be too large. In practice, these conditions may not be satisfied. Below we propose methods to tackle these issues.

Feasibility Checking: We first propose a method to check instance feasibility. Observe that as long as there is at least one feasible routing path $p \in \mathcal{P}_j$ for any TC $C_j \in \mathcal{C}$, the problem instance has a non-zero optimal objective value, as a multi-TC flow with $0 < \xi \leq \min_j \frac{r_j}{B_j}$ always exists. Therefore, as a feasibility check before running the algorithm, we first run a shortest path algorithm (regarding link delays) for each TC on the routing graph. If any TC C_j has the shortest path with delay larger than its delay bound D_j, we return that no feasible solution exists; otherwise, we proceed to the next step.

Demand scaling: The next step is to ensure $\Delta \geq 1$; otherwise the algorithm may not achieve the desired bound. Note that we can scale the demands of all TCs by a common factor in order to scale ξ^*, and equivalently Δ. Hence if we can derive a lower bound on ξ^*, we can scale all demands such that $\Delta \geq 1$.

Following the method proposed by Garg and Könemann [38] and later on improved by Fleischer [35], we derive both a pair of lower and upper bounds on Δ, by finding the *feasible routing path with maximum per-prototype capacity*, denoted by p_j^*, for each TC C_j, using a binary search on the per-prototype capacity. Given a capacity threshold $b > 0$, the computation first prunes all prototypes with capacity less than b, and then find a delay-shortest path in the remaining graph; if the path delay is bounded by D_j, we increase the threshold b; otherwise we decrease b. As there are at most $|\mathcal{E}|$ distinct capacity values, the binary search takes $O(\log(|\mathcal{E}|))$ shortest path computations. The time complexity of finding paths for all TCs is $O(|\mathcal{C}| \log(|\mathcal{E}|)(|\mathcal{E}| + |V| \log(|V|\kappa_{max}))\kappa_{max})$ if the Dijkstra's algorithm is used for shortest path computations. When $|\mathcal{C}|$ is large, this can be further reduced by computing a single round of *all-pair shortest paths* on the pruned original graph for each of the binary search iterations, and then utilize the auxiliary graph in [13] to compute the paths for all TCs.

Let $b_j = \min_{e \in \mathcal{E}(\tilde{p}_j)}\{b_e\}$ be the bottleneck prototype capacity of path p_j^*. Since a flow can saturate all prototypes at its maximum, an upper bound on the single-TC flow value is given by $|\mathcal{E}|\min\{b_j, r_j\}$, taking into account the reliability requirement of each TC. Hence an upper bound of the optimal objective value Δ is given by $\overline{\Delta} = \min_{C_j \in \mathcal{C}}\{\bar{b}_j/B_j\}$. On the other hand, given the bottleneck prototype capacity b_j, a flow that only contains path p_j^* and assigns $b_j/(\kappa_j+1)$ bandwidth to the path, is feasible for the TC itself, as a prototype can be used for at most $\kappa_j + 1$ times. Since there are $|\mathcal{C}|$ TCs sharing the network, $\underline{b}_j = \min\{b_j/(\kappa_j+1)|\mathcal{C}|, r_j\}$ yields a lower bound on the throughput received by C_j. Hence a lower bound of Δ is given by $\underline{\Delta} = \min_{C_j \in \mathcal{C}}\{\underline{b}_j/B_j\}$.

Given these bounds, we can scale all TCs' demands by a factor of $\underline{\Delta}$ (thus Δ by a factor of $1/\underline{\Delta}$), which ensures that the scaled dual optimal objective value $\Delta \geq 1$. But now Δ can be as large as $\widetilde{\Delta} = \overline{\Delta}/\underline{\Delta}$. We then use the same technique as in [38]: if Algorithm 2.1 does not terminate after $\lceil 2\log_{1+\epsilon} 1/\gamma \rceil$ phases, then we know that $\Delta \geq 2$. In this case, we double all demands B_j (thus halving the optimal solution Δ), and re-run Algorithm 2.1. Given the upper bound on Δ, this takes $O(\log(\widetilde{\Delta}))$ rounds of demand scaling.

Combined with Theorem 2.3, we have our main theorem:

Theorem 2.4. *Algorithm 2.1 (combined with feasibility checking and demand scaling) produces a* $(1-\omega)$*-approximation in time* $O^*(\frac{1}{\omega^3}|V||\mathcal{E}|(|\mathcal{E}|+|E_\nu||\mathcal{C}|)\kappa_{\max}^2 + |\mathcal{C}||\mathcal{E}|\kappa_{\max})$*, and hence is an FPTAS for OQRTS.* □

Proof. To compute the initial bounds, it takes $O(|\mathcal{C}|\log(|\mathcal{E}|)(|\mathcal{E}|+|V|\log|V|)\kappa_{max}\log(\kappa_{max}|V|))$ time. By the values of $\overline{\Delta}$ and $\underline{\Delta}$, we have $\widetilde{\Delta} \leq |\mathcal{E}||\mathcal{C}|\kappa_{max}$. Hence the number of demand scaling rounds is $O(\log(|\mathcal{E}||\mathcal{C}|\kappa_{max}))$. Each round consists of at most $\lceil 2\log_{1+\epsilon} 1/\gamma \rceil$ phases in Algorithm 2.1. By Theorem 2.3, each round runs in $O^*(\frac{1}{\omega^3}|V||\mathcal{E}|(|\mathcal{E}| + |E_\nu||\mathcal{C}|)\kappa_{\max}^2)$ time. Combining the above and omitting the logarithm terms, the final time complexity follows. □

2.5.6 Extension to Multiple QoS Requirements

Our proposed model and algorithm can be extended to incorporate other QoS requirements than delay, such as jitter, packet drop rate, etc. In general, assume each TC considers up to Q additive QoS parameters. We can simply replace the DCLC FPTAS in Section 2.5.3 with a Multi-Constrained Path (MCP) FPTAS [139]. The resulting algorithm is able to enforce one QoS requirement strictly, while approximating the other $Q-1$ requirements within a factor of $(1+\omega')$, as shown in [139].

2.6 Performance Evaluation

2.6.1 Experiment Settings

We implemented the following algorithms for comparison:

- PDA: Our primal-dual FPTAS (Algorithm 2.1). The accuracy parameter $\omega = 0.5$ by default. PDA has two variants, PDA-ND and PDA-NR, denoting the algorithms without delay and reliability requirements respectively.
- OND: An optimal algorithm for solving the OQRTS problem without consideration of TC delay constraints, obtained by solving an edge-flow multi-commodity flow LP. This yields an upper bound on the optimal solution.
- MPBDH: A flow-based heuristic which first computes a (delay-agnostic) maximum concurrent flow for all TCs, and then keeps finding feasible (delay-bounded) paths for each TC until no feasible path is left; extended from [85].

- SP: A baseline heuristic that decides the traffic scaling ratio based on shortest-path routing for each TC. As SP is a single-path routing algorithm, its solution can never exceed the minimum reliability ratio of any TC.

We randomly generated networks for evaluation. Topologies were generated based on the Waxman model [32]. By default, the network had 20 nodes. The network offered 10 types of service functions, each having 3 instances randomly deployed on nodes. Each instance had a random capacity within $[50, 100]$ Mbps, and a random delay within $[3, 30]$ ms. Connectivity parameters were set as $\alpha = \beta = 0.6$ in the Waxman model. Each link had a random capacity within $[10, 100]$ Mbps, and a random delay within $[1, 10]$ ms. In each experiment, we generated 20 TCs with random sources and destinations. Each TC had a random service chain with length within $[1, 5]$, a bandwidth demand within $[3, 30]$ Mbps, a delay bound within $[125, 250]$ ms, and a reliability requirement within $[0.35, 0.65]$ of its bandwidth demand. The above were the default parameters. In each set of experiments, we varied one control parameter for evaluation under different scenarios.

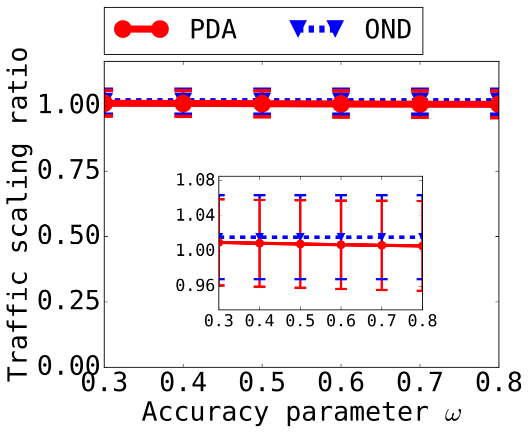
Two metrics were used to evaluate each algorithm. The **traffic scaling ratio** (objective function value) evaluates the algorithm's performance. The **average running time** evaluates the algorithm's overhead for producing the result.

We developed a C++-based simulator implementing all the above algorithms. For OND and MPBDH, we used the Gurobi optimizer [48] to solve the LPs. Each experiment was conducted on a Ubuntu Linux PC with Quad-Core 3.4GHz CPU and 16GB memory. Experiments were repeated for 20 times under the same settings to average out random noises. Each experiment was repeated for 20 times under the same setting, and results were averaged over all runs.

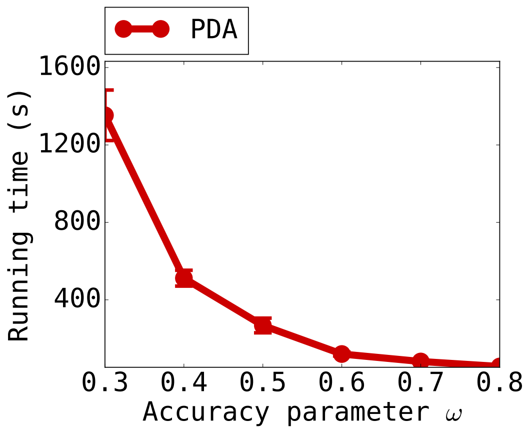
2.6.2 Evaluation Results

2.6.2.1 Comparison with theoretical upper bound

Fig. 2.2 shows the comparison between PDA and OND. Note that the error bars show the 95% confidence intervals around the mean. Since OND is delay-agnostic, its optimal value yields an upper bound on the optimal value of OQRTS. As shown in Fig. 2.2(a), the solution produced by PDA is extremely close to the upper bound produced by OND, much higher than the theoretical guarantee $(1-\omega)$. Also, though the solution degrades with looser accuracy parameter ω, the degradation is minor. The observed optimality gap is within 1%. The running time of PDA, shown in Fig. 2.2(b), is decreasing polynomially to $1/\omega$. In conclusion, a loose accuracy parameter ω, such as no less than 0.5, is typically sufficient for practical use.

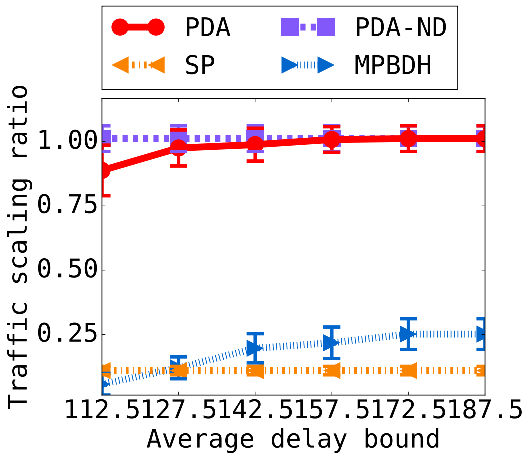


(a) Traffic scaling ratio vs. ω

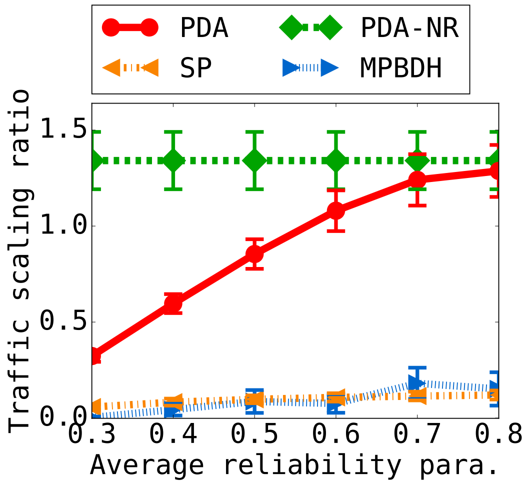


(b) Running time vs. ω

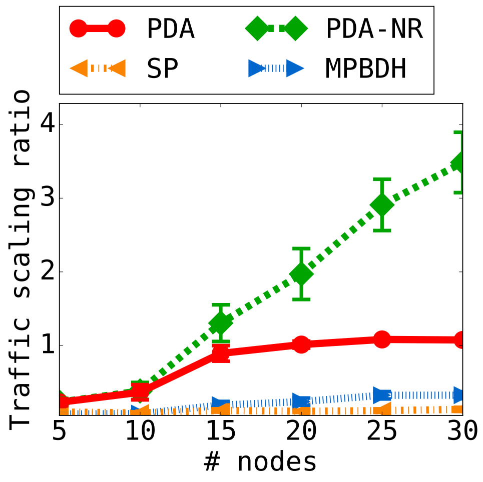
Figure 2.2: Comparison with upper bound, with varying accuracy.



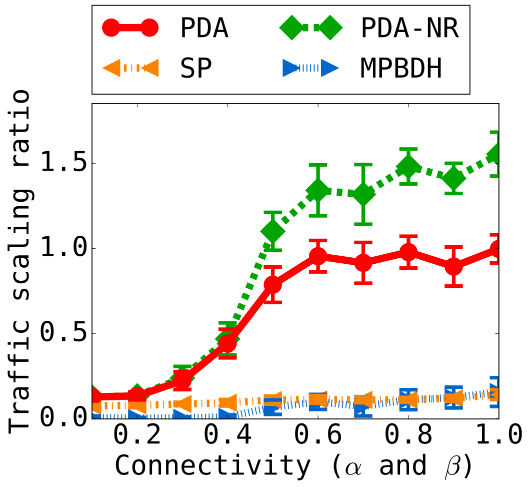
(a) Objective vs. delay bounds



(b) Objective vs. reliability



(c) Objective vs. # nodes



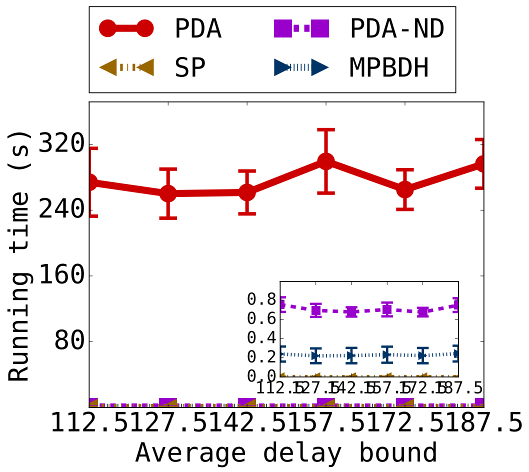
(d) Objective vs. connectivity

Figure 2.3: Traffic scaling ratio with varying delay bounds, reliability requirements, number of nodes, and connectivity parameters.

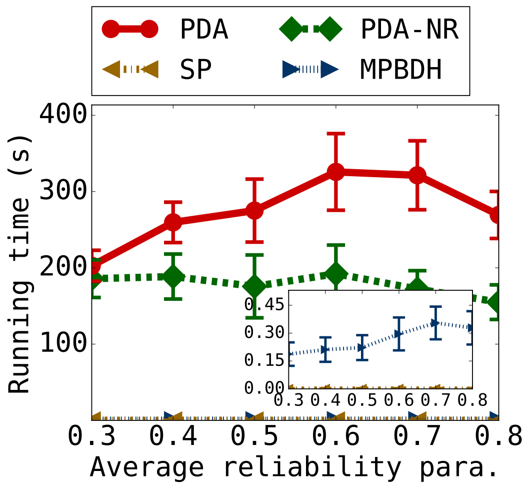
2.6.2.2 Comparison with baseline heuristics

Figs. 2.3 and 2.4 show the comparison of PDA (including PDA-ND or PDA-NR) with the two baselines, MPBDH and SP, under various scenarios.

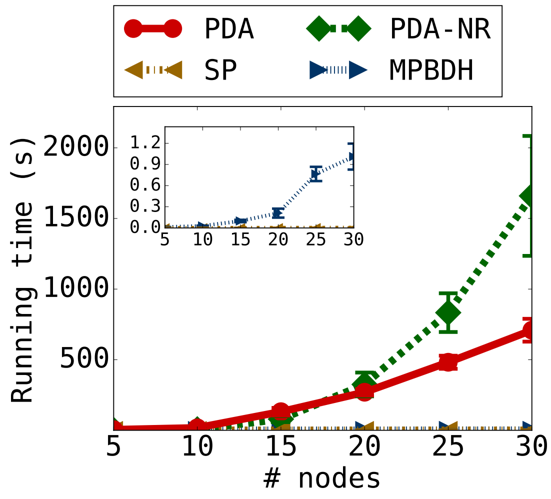
Figs. 2.3(a) and 2.4(a) show the objective values and running times respectively



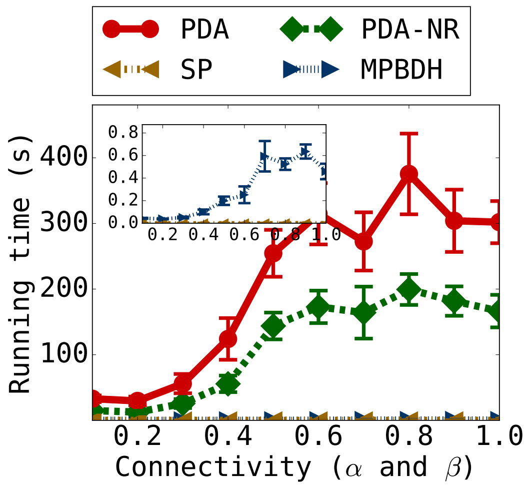
(a) Running time vs. delay bounds



(b) Running time vs. reliability



(c) Running time vs. # nodes



(d) Running time vs. connectivity

Figure 2.4: Running time with varying delay bounds, reliability requirements, number of nodes, and connectivity parameters.

with varying TC delay bounds. With increasing delay bounds, both PDA and MPBDH achieve better traffic scaling ratio. SP has consistent performance with varying delay bounds as it only considers the shortest path. However, PDA outperforms both heuristics drastically, with an average improvement of 5.9× compared to MPBDH and 7.9× compared to SP. The enhanced performance indeed comes with increased time complexity, as shown in Fig. 2.4(a). The delay bounds have limited impact on time complexity in Fig. 2.4(a). Finally, comparing PDA and PDA-ND, the delay constraints lead to both degraded throughput and much larger time complexity, the latter due to the computation of a delay constrained least cost path instead of a simple shortest path.

Figs. 2.3(b) and 2.4(b) show the objective values and running times respectively with varying TC reliability parameter (the average ratio of maximum tolerable loss over bandwidth demand). Increased tolerable loss results in increased traffic scaling ratio in general, due to more bandwidth available to each TC's traffic at each service function instance. PDA again outperforms both heuristics in terms of throughput, with an average improvement of $6.1\times$ and $8.5\times$ compared to MPBDH and SP respectively. Comparing PDA and PDA-NR, the latter has a better throughput due to the relaxation of the reliability requirements, and a lower time complexity due to the less number of constraints (thus the number of dual variables) when reliability is not considered.

Figs. 2.3(c) and 2.4(c) show experiments with varying number of nodes in the network. Increasing number of nodes leads to increased traffic scaling ratios. However, after a certain threshold, the scaling ratio derived by PDA saturates. This is because the number of instances of each service function remains the same, and hence the the scaling ratios are constrained by the reliability requirements instead of the link capacities when the number of nodes become large. The throughput achieved by PDA surpasses MPBDH and SP significantly, with an average improvement of $3.7\times$ and $8.2\times$ compared to MPBDH and SP respectively. The running times increase with the number of nodes, due to the increased number of links.

Figs. 2.3(d) and 2.4(d) show experiments with varying network connectivity, which is controlled by parameters α and β in the Waxman model. Increased connectivity leads to increased throughput. Comparisons among algorithms are similar to the above. On average, PDA outperforms MPBDH and SP by $6.5\times$ and $5.4\times$, respectively. MPBDH performs worse than SP, again due to the increased bandwidth on infeasible paths. The running times increase with network connectivity, due to the increase in both the problem size and the time for finding (approximate) shortest feasible paths.

To summarize, our findings are as follows:

- Our algorithm achieves near-optimal solutions even when the accuracy parameter is relatively loose. In general, the optimality gap is within 1%. Thus a loosely selected accuracy parameter is sufficient for most practical uses.
- Our algorithm outperforms both baseline heuristics (MPBDH and SP) significantly.
- The running time overhead of our algorithm is acceptable in practice, as network planning typically happens in much longer periods, for example, once per several hours.

2.7 Conclusions

In this study, we studied the QoS-aware and Reliable Traffic Steering problem for service function chaining in mobile networks. We formulated the problem in a software-defined approach, considering various requirements of different classes of traffic, including service chaining, QoS, reliability, and type-of-transmission constraints. The problem, along with its optimization version, was proved to be NP-hard. We then proposed an FPTAS for the optimization problem, which produces a $(1 - \omega)$-approximate solution within time polynomial to the input size and $1/\omega$. We evaluated our algorithm through extensive simulation experiments, which validated that our algorithm has near-optimal performance, and achieves much better throughput than the baseline heuristics.

Chapter 3

PROVISIONING QOS-AWARE AND ROBUST APPLICATIONS IN INTERNET-OF-THINGS: A NETWORK PERSPECTIVE

3.1 Introduction

Designed to connect the digital world and the real world, the Internet-of-Things (IoT) has been recognized as one of the enabling technologies of the next era of computing. Numerous applications have been developed utilizing IoT functionalities, enabling advances in a number of areas including smart cities, smart health, connected cars, etc. It has been anticipated that the global IoT market will exceed \$250B by 2020 [56].

One common type of IoT application is real-time processing applications, which process continuous data streams generated by IoT devices for pre-processing or analysis. These applications commonly have more stringent quality-of-service (QoS) requirements than traditional applications, including delay, throughput, etc., in order to ensure on-time delivery and analysis of real-time data and hence fast response to the users. An example is real-time sports analysis applications [44], [122], which analyze the status of live sports games, based on real-time data from cameras and/or other sensors.

Unfortunately, current IoT infrastructures are not built specifically for real-time processing applications. Current infrastructures use cloud computing as the underlying computing support. While cloud computing offers abundant and inexpensive computing power, it suffers from long end-to-end delay and high bandwidth usage, which greatly affect the performance of real-time processing applications. This situation is further aggravated by commonly used communication technologies in IoT, such as cellular networks and/or low-power wide-area networks (LPWANs), which offer only limited bandwidth for transmission.

Fog computing is one of the emerging technologies aiming to address these issues in current IoT. With fog nodes deployed near the IoT devices and end users, fog computing can reduce both the propagational delay and the bandwidth usage. However, ubiquitous fog node deployment is still unrealistic within the near future due to cost issues. Combined with the limited capacity of the IoT networks, this raises the problem of resource allocation in fog-enabled IoT. In particular, an infrastructure needs to allocate computing and network resources to support each application with proper QoS guarantees.

In this study, we study this problem from a network perspective, extending from our previous conference version [151]. Given a real-time processing IoT application, the infrastructure needs to decide both the fog node to host this application, and the channels along which the application's data streams will be transmitted. The channels must satisfy the QoS requirement of the application, including both the bandwidth demand of each data source, and the delay bound of the application. Compared to our conference version [151], we further consider the robustness requirement of each data source: how could the application survive an arbitrary network failure. We propose a robustness technique where given an arbitrary network failure, either a link, a network node or a host node, the data loss incurred on each data stream is bounded by a portion of the total data. This, combined with existing error correction coding schemes [25], can achieve lossless data transmission and processing for IoT applications, which is crucial for many critical application scenarios such as emergency handling.

Two types of applications are considered in this study. A *parallelizable application* is one that can be deployed as multiple distributed instances, possibly with certain data restrictions. A *non-parallelizable application* is one that must be centrally implemented on one host node. We further consider two provisioning scenarios: single-application provisioning, and multi-application provisioning. Combining the two types with the two provisioning scenarios, we have four versions of the provisioning problem. We formally define these problems, and prove that they are all NP-hard. We then propose fully polynomial-time approximation schemes (FPTASs) for three of the four versions, and a randomized algorithm for the last one. To validate our algorithms, we have conducted extensive simulations, comparing our proposed algorithms both to the theoretical upper bound and to several heuristic solutions. It has been shown that our algorithms achieve close-to-optimal performance, and largely outperform the heuristics in terms of both bandwidth and delay.

Our contributions are summarized as follows:

- To the best of our knowledge, we are the first to study the problem of IoT application provisioning with both QoS and robustness requirements.
- We formulate four versions of the provisioning problem, and proved all of them to be NP-hard.
- We propose FPTASs for three of the four versions, and a randomized algorithm for the last one.
- We use extensive simulations to evaluate the performance of our algorithms against both the theoretical bound and several practical heuristics.

The rest of this study is organized as follows. In Section 3.2, we introduce background and related work. In Section 3.3, we present our system model. In Section 3.4, we formally define the four provisioning problems we study, and present their complexity result. In Sections 3.5 and 3.6, we propose our algorithms for single application provisioning and multi-application provisioning, respectively. In Section 3.7, we present our performance evaluation results. In Section 3.8, we conclude this study.

3.2 Background and Related Work

3.2.1 Internet-of-Things and Fog Computing

While the concept of the "Internet-of-Things" can trace back to the last century[2], its power has barely been unleashed until recently, when several enabling technologies, including wireless networks, cloud computing, and data science, have witnessed drastic advances. Since then, extensive efforts have been put into IoT-related areas, including computing architectures [11], communications [122], radio-frequency identification (RFID) [19], etc. A survey on IoT can be found in [73].

Fog computing has been regarded as one of the key technologies that enable IoT [11]. Extending from cloud computing, fog computing deploys geographically distributed fog nodes in the edge network, providing computing power closer to both the IoT devices and end users. Fog computing can improve the performance and energy efficiency in many IoT applications, including crowdsensing [7], smart cities [41], etc.

The limited resources in IoT and fog have urged efforts on new resource allo-

[1]The term dates back to a talk by Kevin Ashton in 1999 [73].

cation methods. Zeng *et al.* [155] studied task scheduling and data placement to minimize I/O time, computing time and transmission delay in fog platforms. Many have studied workload offloading in edge/fog-cloud systems with different objectives, including power consumption [28], [136], delay minimization [28], [120], [123], quality-of-experience [136], etc. However, most of these do not consider the complex structure and limited capacity of the edge network; while Deng *et al.* [28] indeed considered network bandwidth constraints, they assumed that the transmission of each application's data will not interfere with each other, which does not capture the sharing nature of the IoT networks, and hence does not apply in many cases. Due to lack of existing work on network resource allocation in fog-enabled IoT, we study application provisioning from a network perspective, where we aim to guarantee the QoS of applications in terms of both transmission delay and bandwidth.

3.2.2 Network Service Provisioning

Stepping out of the IoT and fog computing domain, some related resource allocation problems have also been studied in different contexts, such as *virtual network/infrastructure embedding* (VNE/VIE) [21] and *service function chaining* (SFC) [70], [109]. The VNE/VIE problems aim to find an embedding of a virtual service topology onto the physical topology, which respects resource capacities in the substrate. The difference is that VIE allows virtual node consolidation while VNE does not. While these two problems can be viewed as a generalization of ours, they are harder to solve. To the best of our knowledge, there has yet been any non-trivial approximation ratio for VNE/VIE on general graphs. Assumptions on topologies and/or service models help in providing performance bounds [162], but are commonly too restrictive to handle the complex structures of the IoT networks.

SFC is another special case of the general VNE/VIE problems, where the virtual topology is restricted to line graphs. In this case, certain approximation bounds can be obtained, as shown by Rost *et al.* [109] and Kuo *et al.* [70]. In this study, we consider a different service model than SFC, where the virtual topologies are star graphs. We also propose solutions with non-trivial performance guarantees.

3.2.3 Robustness Applications and Networks

Robustness of computing and network services has long been studied in the literature. There are commonly two approaches for building and/or maintaining robust services. One is to provide *fault resilience* through redundancy, *i.e.*, provisioning redundant resources (computing, path, bandwidth, etc.) as backups to quickly recover a service when failure happens. For example, restoration based routing, bandwidth allocation and network embedding have been studied in [58], [101], [137], [138]. Similar approaches have also been used for service, application and virtual machine backups, such as [10], [63], [142], [146], [154]. Due to the need for redundancy, this approach commonly results in excessively reserved backup resources that remain idle during normal operations.

The second approach is to rather leverage the *fault tolerance* of the services themselves, and to minimize either the fault probability or the fault impact incurred on the services. Zhang *et al.* [157] modeled the fault probability of a virtual infrastructure based on the availability of all its physical components, and sought to minimize this probability during the embedding. Acharya *et al.* [1], Zhang *et al.* [158] and

Yallouz *et al.* [140] explored bounding the impact of a failure on the overall throughput of the system. This was termed *tunable survivability* in [140]. This idea was further leveraged in [141] and [153]. Compared to the redundancy-based approach, this approach results in much less resource consumption, yet providing reasonably good protection in practice. We therefore take this approach in our study, specifically due to the already scarce resources in the IoT environment.

3.3 System Model

3.3.1 Infrastructure Model

The IoT infrastructure is modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the node set, and $\mathcal{E}$ is the link set. The node set consists of both *facility nodes* (general-purpose servers, fog-enabled switches/routers, etc.) and *network nodes* (switches/routers). We use $\mathcal{F} \subseteq \mathcal{V}$ to denote the set of facility nodes, and $\mathcal{N} \subseteq \mathcal{V}$ to denote the set of network nodes. Note that these two sets are not necessarily disjoint, as some network nodes may also have computing capabilities [22]. Each link $e \in \mathcal{E}$ has a capacity, denoted by $c_e > 0$, and a transmission delay, denoted by $d_e > 0$. For simplicity, we use $\mathcal{E}_v$ to denote all links adjacent to node $v \in \mathcal{V}$, regardless of direction.

3.3.2 Application Model

An application receives continuous data from one or more data sources, and performs joint analysis of all received data. We assume each source generates data in a constant rate, *e.g.*, a video camera generating video footages. Given the application, the infrastructure needs to both find a facility node to host it, and establish transmission channels from each source to the host. Each application may require certain hardware resources, *e.g.*, video processing commonly requires strong GPU for efficient computation. Hence in many cases, only a subset of the facility nodes can host an application. The established channels need to satisfy at least two QoS requirements: 1) each source should receive bandwidth that meets its data generation rate, and 2) the transmission delay of each channel should be within the delay tolerance of the application.

Formally, an IoT application is denoted by a triple, $\Gamma = (\mathcal{S}, \mathbf{B}, D)$, where $\mathcal{S} \subseteq \mathcal{V}$ denotes the set of data sources of Γ, $\mathbf{B} = \{B_s \in \mathbb{R}^+ \mid s \in \mathcal{S}\}$ denotes the corresponding data generation rate of each data source in $\mathcal{S}$ ($\mathbb{R}^+$ is the set of positive real numbers), and $D > 0$ is the delay bound that must be enforced for transmission from each data source. Given an application Γ, we further use $\mathcal{F}_\Gamma \subseteq \mathcal{F}$ to denote its *candidate host set*, where each $v \in \mathcal{F}_\Gamma$ satisfies the hardware requirement of Γ.

3.3.3 Basic Provisioning Model

As aforementioned, application provisioning involves both finding the host node and data routing. Before defining the provisioning problems, we first make the following definitions.

Definition 3.1 (Feasible path set)**.** *Given network G and an application Γ, let $v \in \mathcal{F}_\Gamma$ be a candidate host of Γ and $s \in \mathcal{S}$ be a data source of Γ, the* feasible path set *of Γ regarding v and s, denoted by $\mathcal{P}^\Gamma_{v,s}$, is defined as the subset of all (s, v)-paths in G such that for each path $p \in \mathcal{P}^\Gamma_{v,s}$,*

$$\sum\nolimits_{e \in p} d_e \leq D. \tag{3.1}$$

We use $\mathcal{P}_v^\Gamma = \bigcup_{s \in \mathcal{S}} \mathcal{P}_{v,s}^\Gamma$ *to denote the feasible path set from all data sources of* Γ *to candidate host* v*, and* $\mathcal{P}^\Gamma = \bigcup_{v \in \mathcal{F}_\Gamma} \mathcal{P}_v^\Gamma$ *the feasible path set towards all candidate hosts of* Γ*.* □

Definition 3.2 (Bandwidth allocation)**.** *Let* P *be an arbitrary set of paths in* G*. A* bandwidth allocation *of* P *is defined as a mapping* $\mathcal{L} : P \mapsto \mathbb{R}^*$ *(*$\mathbb{R}^*$ *denotes the set of nonnegative real numbers), where* $\mathcal{L}(p)$ *denotes the bandwidth allocated on path* p *for any* $p \in P$*. We say that a bandwidth allocation* $\mathcal{L}$ *is* feasible*, iff for any link* $e \in \mathcal{E}$*,*

$$\sum\nolimits_{p \in P : e \in p} \mathcal{L}(p) \le c_e. \tag{3.2}$$

We use $b(P) = \sum_{p \in P} \mathcal{L}(p)$ *to denote the* aggregate bandwidth *of* $\mathcal{L}$ *over path set* P*.*

□

Before defining the expected outcome of application provisioning, we first distinguish between two types of applications. A *non-parallelizable* application has no parallelism capability, hence its logic must be centrally implemented on exactly one facility node. On the contrary, a *parallelizable application* can have its logic split over multiple distributed instances, with each instance processing a portion of the incoming data. However, implementing parallelism may require certain data splitting restrictions to be enforced during provisioning, such as data synchronization among data sources. An example of a parallelizable application is stateless sensor data fusion [30], where each instance can process an arbitrary portion of the incoming data as long as the same portion is received synchronously from every data source. We formalize the provisioning of these two types of applications in the following.

Definition 3.3 (Provisioning scheme)**.** *Given network* G *and an application* Γ*, a* provisioning scheme *is defined as a triple* $\Pi = (\mathbf{x}, P_{\mathbf{x}}^\Gamma, \mathcal{L}_{\mathbf{x}}^\Gamma)$*, where* $\mathbf{x} = \{x_v \mid v \in \mathcal{F}\}$ *is a decision variable vector with* x_v *denoting the fraction of data incoming to an instance* *of* Γ *on candidate host* v*,* $P^{\Gamma}_{\mathbf{x}} \subseteq \mathcal{P}^{\Gamma}$ *is a subset of feasible paths of* Γ *towards each candidate host* $v \in \mathcal{F}$ *with* $x_v > 0$*, and* $\mathcal{L}^{\Gamma}_{\mathbf{x}}$ *is a* feasible bandwidth allocation *of* $P^{\Gamma}_{\mathbf{x}}$*. We say that a provisioning scheme* Π *is* feasible *iff*

1. $\sum_{v \in \mathcal{F}} x_v = 1$,

2. *for* $\forall v \in \mathcal{F}$*, if* Γ *is non-parallelizable, then* $x_v \in \{0, 1\}$*, otherwise* $x_v \in [0, 1]$*, and*

3. *for* $\forall s \in \mathcal{S}$ *and* $\forall v \in \mathcal{F}$*, the aggregate bandwidth* $b(P^{\Gamma}_{v,s}) \geq B(s) \cdot x_v$*, where* $P^{\Gamma}_{v,s} = P^{\Gamma}_{\mathbf{x}} \cup \mathcal{P}^{\Gamma}_{v,s}$ *is the subset of selected paths from* s *to* v*.* $\square$

The last requirement of feasibility in Definition 3.3 ensures that the same portion of data generated by all data sources are received at the same instance, which can be used to enforce data synchronization for applications such as stateless sensor fusion. For simplicity, we assume that the processing results are consumed locally at the application host. However, it is trivial to add channels that transmit the results, and hence is omitted for simplicity.

3.3.4 Robustness Model

Robustness refers to the ability of an application to provide uninterrupted service when facing different types of failures in the physical infrastructure. Some failures have inevitable effects in service quality, such as failures at data sources; some other failures, however, can be avoided or at least alleviated by proper load and redundancy management. In this study, we focus on tackling failures that can be alleviated, including both link failures and node failures.

As stated before, instead of using a traditional "all-or-nothing" protection, we use a "soft" mechanism for robustness [153], [158], which ensures that each application incurs only bounded data loss due to an arbitrary single failure in the network. Specifically, each data source $s \in \mathcal{S}$ has a reliability parameter $r_s \in (0, 1]$, which denotes the *maximum tolerable data loss ratio* for correct processing of the data from source s. The idea is to ensure that the load is properly distributed such that the application incurs no more than $r_s \cdot B_s$ data loss from s due to any single link or node failure. We call this approach *robustness through load balancing.* This can be coupled with a proper application- or network-level coding technique [25] to achieve loss-resistance in failure scenarios.

Given the above, we can extend Definition 3.3 to incorporate robustness. We start with the definition of a *link-robust provisioning scheme*, which protects against a single link failure:

Definition 3.4 (Link-robust provisioning scheme)**.** *Given network G and an application Γ, a* link-robust provisioning scheme *is a provisioning scheme Π for Γ that satisfies: for $\forall s \in \mathcal{S}$ and $\forall e \in \mathcal{E}$, $\sum_{p \in P^{\Gamma}_{\mathbf{x}}: e \in p} \mathcal{L}(p) \leq r_s \cdot B_s$.* □

The idea behind Definition 3.5 is that the data loss for data source s due to a single link failure is essentially bounded by the amount of data transmitted on all paths through link e. Similarly, we can define a *node-robust provisioning scheme*:

Definition 3.5 (Node-robust provisioning scheme)**.** *Given network G and an application Γ, a* node-robust provisioning scheme *is a provisioning scheme Π for Γ that satisfies: for $\forall s \in \mathcal{S}$ and $\forall u \in \mathcal{V} \setminus \{s\}$, $\sum_{p \in P^{\Gamma}_{\mathbf{x}}: u \in p} \mathcal{L}(p) \leq r_s \cdot B_s$.* □

The source node s of each data stream is excluded from the protection, as failure of the source node will cause full blockage of data transmission, and hence cannot be alleviated through load balancing.

Before diving into the concrete problem definition, we want to highlight the different robustness capabilities of the two types of applications. Note that node-robustness is more difficult to achieve than link-robustness, as the former automatically guarantees the latter. For a non-parallelizable application, only link-robustness can be achieved. This is because a non-parallelizable application requires that its logic to be centrally implemented, which becomes a single point of failure. In this case, protection over node failures must be implemented in the form of redundancy rather than load balancing, and hence is out of the scope of this study. On the other hand, a parallelizable application can achieve node-robustness, due to its capability of balancing load across multiple instances. In the following of this study, we use the term "robust" to denote node-robustness for parallelizable applications, and link-robustness for non-parallelizable applications, depending on the context. It should be noted that $r_s = 1$ means no protection for data source s, hence the problem we study is a generalization of the problem studied in [151].

3.3.5 Notations

We define the following notations to facilitate illustration. $V = |\mathcal{V}|$ is the number of nodes. $E = |\mathcal{E}|$ is the number of links. $F_\Gamma = |\mathcal{F}_\Gamma|$ is the number of candidate hosts for application Γ, and $F = |\mathcal{F}|$ is the total number of facility nodes. $S_\Gamma = |\mathcal{S}_\Gamma|$ is the number of data sources for application Γ, and $S = \sum_{\Gamma \in \mathbf{\Gamma}} S_\Gamma$ is the total number of data sources for a set of applications $\mathbf{\Gamma}$.

3.4 Problem Statement and Complexity

We separately consider the provisioning for the two types of applications: parallelizable and non-parallelizable. As stated before, we consider two provisioning scenarios. In the first scenario, we consider the provisioning of a single application at a time, which can be applied, for example, when processing arriving application requests in an online manner. In the second scenario, we consider the joint provisioning of multiple applications (of the same type) at the same time. This can be useful both for batch provisioning of queued application requests, and for provisioning multiple inter-related application components. Combining the two types of applications with the two provisioning scenarios, we arrive at four versions of the provisioning problem, which are formally defined in the following.

Definition 3.6 (SAP)**.** *Given network G and an application Γ, the* ***Single-Application Provisioning (SAP)*** *problem is to find a* feasible and robust *provisioning scheme Π for Γ.*

Its optimization version, named ***O-SAP****, is to find a robust provisioning scheme Π, such that for every data source $s \in \mathcal{S}$, its aggregate bandwidth satisfies $b(P^{\Gamma}_{\mathbf{x},s}) \geq \lambda \cdot B(s)$, and the* traffic scaling ratio *λ is maximized.*

We use P-SAP/PO-SAP to denote the corresponding problem with a parallelizable application and node-robustness, and N-SAP/NO-SAP to denote the corresponding problem with a non-parallelizable application and link-robustness. $\square$

Definition 3.7 (MAP)**.** *Given network G and an application set $\mathbf{\Gamma} = \{\Gamma_1, \ldots, \Gamma_K\}$, the* ***Multi-Application Provisioning (MAP)*** *problem is to find a set of* feasible and robust *provisioning schemes $\mathbf{\Pi} = \{\Pi_1, \ldots, \Pi_K\}$, where $\Pi_k = (\mathbf{x}_k, P_k, \mathcal{L}_k)$ is the provisioning scheme for Γ_k for $k = 1, \ldots, K$, such that the shared capacity constraint* *is satisfied for any link $e \in \mathcal{E}$:*

$$\sum_{k=1}^{K} \sum_{p \in P_k} \mathcal{L}_k(p) \le c_e.$$

Its optimization version, named ***O-MAP****, is to find a set of* robust *provisioning schemes* $\mathbf{\Pi}$ *for* $\mathbf{\Gamma}$*, such that the minimum* traffic scaling ratio λ *of all applications, as defined in Definition 3.6, is maximized.*

We use $P_{k,s} = P_k \cap \mathcal{P}^{\Gamma_k}_{\mathbf{x}_k,s}$ to denote the subset of selected paths for data source s of application Γ_k.

We use P-MAP/PO-MAP to denote the corresponding problem with parallelizable applications and node-robustness, and N-MAP/NO-MAP to denote the corresponding problem with non-parallelizable applications and link-robustness. □

Theorem 3.1. *All problems defined above are NP-hard.* □

Proof. Since SAP problems are special cases of the corresponding MAP problems, it suffices to prove that P-SAP and N-SAP are NP-hard. Consider a special case of either P-SAP or N-SAP where Γ has one data source s, one candidate host t, and no protection ($r_s = 1$). In this case, SAP becomes finding a set of (s,t)-paths and a bandwidth allocation that satisfy the bandwidth demand $B(s)$ and the delay bound D. This turns out to be the Multi-Path routing with Bandwidth and Delay constraints (MPBD) problem, which is NP-hard [85]. Hence SAP is NP-hard, and the NP-hardness of the rest follows. □

Due to the NP-hardness of the above problems, we seek to design approximation schemes for the optimization problems, in order to provide as accurate judgements for the decision problems as possible. In the following sections, we show that three over four of the optimization problems admit FPTASs.

We start with the problem of provisioning one application at a time. Due to the two types of applications, parallelizable or non-parallelizable, we have two versions of this problem (PO-SAP and NO-SAP). In this section, we focus on the NO-SAP problem, and propose an FPTAS. We leave the parallelizable case to Section 3.6, where we propose an FPTAS for solving both PO-SAP and PO-MAP. In the rest of this section, we omit the term "non-parallelizable" without ambiguity.

Our algorithm to NO-SAP is based on the decomposition of NO-SAP into two subproblems: *Host Designation* (HD) that decides the host node of application Γ, and *Data Routing* (DR) that decides the routing paths and bandwidth from each data source to the host. For simplicity, we extend this decomposition method throughout the rest of this study, with HD denoting determination of the decision vector $\mathbf{x}$, and DR denoting the routing process, *i.e.*, determining $P_{\mathbf{x}}^{\Gamma}$ and $\mathcal{L}_{\mathbf{x}}^{\Gamma}$. For the NO-SAP problem, the relationship between this problem and its DR subproblem is stated in the following lemma.

Lemma 3.1. *If the DR subproblem admits a polynomial-time a-approximation algorithm, so does NO-SAP.* □

Proof. We construct an a-approximation algorithm to NO-SAP ($A_{\text{NO-SAP}}$) out of an a-approximation algorithm to DR (A_{DR}), as shown in Algorithm 3.1. The algorithm iterates over all candidate hosts to find the best solution for the application, using the a-approximation A_{DR}. To prove Algorithm 3.1 is an a-approximation to NO-SAP, let $\Pi^* = (\mathbf{x}^*, P^*, \mathcal{L}^*)$ be an optimal solution to NO-SAP with objective value λ^* and $x_{v^*}^* = 1$. Then $(P^*, \mathcal{L}^*)$ is indeed a feasible solution of DR given host node v^*. Let $\lambda_{v^*}^*$ be the optimal DR solution with v^*, we have $\lambda^* \le \lambda_{v^*}^*$. The DR solution

Algorithm 3.1: Approximation Algorithm $A_{\text{NO-SAP}}$

```
Input: Network $G$, application $\Gamma$
Output: Traffic scaling ratio $\lambda$, provisioning scheme $\Pi$
1 $\lambda \leftarrow 0$, $\mathbf{x} \leftarrow 0$;
2 for each candidate host $v \in \mathcal{F}_\Gamma$ do
3     $(\lambda_v, P_v^\Gamma, \mathcal{L}_v^\Gamma) \leftarrow A_{\text{DR}}(G, \Gamma, v)$;
4     if $\lambda_v > \lambda$ then
5         $\lambda \leftarrow \lambda_v$, $\mathbf{x} \leftarrow 0$, $x_v \leftarrow 1$;
6         $\Pi \leftarrow (\mathbf{x}, P_v^\Gamma, \mathcal{L}_v^\Gamma)$;
7     end
8 end
9 return $(\lambda, \Pi)$.
```

picked in Algorithm 3.1 during iteration v^*, denoted by $(P_{v^*}^\Gamma, \mathcal{L}_{v^*}^\Gamma)$, has scaling ratio $\lambda_{v^*} \geq a\lambda_{v^*}^* \geq a\lambda^*$. This leads to $\lambda \geq \lambda_{v^*} \geq a\lambda^*$. The lemma follows. □

It remains to solve the DR subproblem, which is still NP-hard due to the same argument as in the proof of Theorem 3.1. Yet, the DR subproblem turns out to be a special case of the Maximum Concurrent Flow (MCF) problem with delay bound and robustness requirement. It is not hard to see that this is a special case of the QoS-aware and Reliable Traffic Steering (QRTS) problem studied in [153], for which an FPTAS exists. Combining the FPTAS proposed in [153] with Lemma 3.1 leads to our final theorem for NO-SAP:

Theorem 3.2. *NO-SAP admits an FPTAS, as shown in Algorithm 3.1 combined with the FPTAS in [153].* □

3.6 Multi-Application Provisioning

In this section, we study the more general problem where multiple applications seek to share the IoT infrastructure. Again, the problem has two versions, one for parallelizable applications (PO-MAP), and one for non-parallelizable applications (NO-MAP). Note that PO-MAP is a generalization of PO-SAP, hence an FPTAS to the former automatically yields an FPTAS to the latter. However, we do not have an FPTAS for NO-MAP. We thus propose a randomized algorithm at the end of this section.

3.6.1 Problem Formulation for PO-MAP

Below, we first give an exact formulation of PO-MAP. For simplicity, we use k to denote Γ_k if no ambiguity is introduced. We use $\mathcal{P} = \bigcup_{k=1}^{K} \mathcal{P}^k$ to denote the set of all feasible paths of all applications[3]. We further use $\mathcal{P}_s^k \subseteq \mathcal{P}$ to denote all feasible paths for application k's data source s. For consistency of notation, we define variable $x(k, v) \triangleq x_v^k$ as the fraction of application k hosted on candidate host $v \in \mathcal{F}_k$, variable $\mathcal{L}(p)$ to denote the bandwidth allocation on path $p \in \mathcal{P}$, and variable λ still as the traffic scaling ratio. Then NO-MAP is formulated as follows:

$$\begin{aligned}
\max\quad & \lambda && && \text{(3.3a)}\\
\text{s.t.}\quad & \sum_{p\in\mathcal{P}^k_{v,s}} \mathcal{L}(p) \geq B^k_s \cdot \lambda \cdot x(k,v), && \forall k, v \in \mathcal{F}_k, s; && \text{(3.3b)}\\
& \sum_{v\in\mathcal{F}_k} x(k,v) = 1, && \forall k; && \text{(3.3c)}\\
& \sum_{p\in\mathcal{P}: e\in p} \mathcal{L}(p) \leq c_e, && \forall e \in \mathcal{E}; && \text{(3.3d)}\\
& \sum_{p\in\mathcal{P}^k_s: u\in p\setminus\{s\}} \mathcal{L}(p) \leq r^k_s \cdot B^k_s, && \forall k, s, u \in \mathcal{V}\setminus\{s\}; && \text{(3.3e)}\\
& x(k,v)\in[0,1], \mathcal{L}(p), \lambda \geq 0, && \forall k, v \in \mathcal{F}_k, p. && \text{(3.3f)}
\end{aligned}$$

Explanation: Constraint (3.3b) couples bandwidth allocation with the demands, the host designation, and the scaling ratio. Constraint (3.3c) ensures that each application is hosted on exactly one node. Constraint (3.3d) enforces link capacities. Constraint (3.3e) enforces the *node-robustness* requirement, such that the throughput over each node $u \in \mathcal{V} \setminus \{s\}$ is bounded by $r^k_s \cdot B^k_s$ for each data source s of each application k.

Program (3.3) seems like a Quadratic Program (QP) due to Constraint (3.3b). However, with a simple transformation shown below, it can be transformed into an

[3] W.l.o.g., if two applications have an overlapping feasible routing path, we still regard the same path for two different applications as two different paths.

equivalent Linear Program (LP). Define new variables $y(k,v) = \lambda \cdot x(k,v)$, we have

$$\begin{aligned}
\max \quad & \lambda && && \text{(3.4a)}\\
\text{s.t.} \quad & \sum_{p\in\mathcal{P}^k_{v,s}} \mathcal{L}(p) \ge B^k_s \cdot y(k,v), && \forall k, v \in \mathcal{F}_k, s; && \text{(3.4b)}\\
& \sum_{v\in\mathcal{F}_k} y(k,v) \ge \lambda, && \forall k; && \text{(3.4c)}\\
& \sum_{p\in\mathcal{P}:e\in p} \mathcal{L}(p) \le c_e, && \forall e \in \mathcal{E}; && \text{(3.4d)}\\
& \sum_{p\in\mathcal{P}^k_s:u\in p} \mathcal{L}(p) \le r^k_s \cdot B^k_s, && \forall k, s, u \in \mathcal{V}\setminus\{s\}; && \text{(3.4e)}\\
& y(k,v), \mathcal{L}(p), \lambda \ge 0, && \forall k, v \in \mathcal{F}_k, p. && \text{(3.4f)}
\end{aligned}$$

It is easy to observe that Programs (3.3) and (3.4) are equivalent. However, Program (3.4) may still have an exponential size due to the possibly exponential number of feasible paths in a graph, and hence it cannot be solved directly using standard LP techniques. Therefore, we next propose an FPTAS.

3.6.2 An FPTAS to PO-MAP

Our FPTAS to PO-MAP extends the ones to MCF reported in [12], [35], [38]. However, PO-MAP is more difficult than the above, due to the need for (fractional) host designation as well as the node-robustness constraint. We first write the dual of Program (3.4), where we define $z(k,v,s) \ge 0$ as the dual variable of Constraint (3.4b) for $\forall k, v \in \mathcal{F}_k, s \in \mathcal{S}_k$, $\varphi(k) \ge 0$ as the dual variable of Constraint (3.4c) for $\forall k$, $l(e)$ as the dual variable of Constraint (3.4d) for $\forall e \in \mathcal{E}$, and $\sigma(k,s,u)$ as the dual variable of Constraint (3.4e) for $\forall k, s \in \mathcal{S}_k, u \in \mathcal{V} \setminus \{s\}$:

$$\min \quad \Delta(l,\sigma) = \sum_{e \in \mathcal{E}} c_e l(e) + \sum_{k=1}^{K} \sum_{s \in \mathcal{S}_k} \sum_{u \in \mathcal{V}}^{u \neq s} r_s^k \cdot B_s^k \cdot \sigma(k,s,u) \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{e \in p} l(e) + \sum_{w \in p} \sigma(k,s,w) \geq z(k,v,s), \ \forall k,v,s,p \in \mathcal{P}_{v,s}^k; \tag{3.5b}$$

$$\sum_{s \in \mathcal{S}_k} B_s^k \cdot z(k,v,s) \geq \varphi(k), \quad \forall k, v; \tag{3.5c}$$

$$\sum_{k=1}^{K} \varphi(k) \geq 1; \tag{3.5d}$$

$$z(k,v,s), \varphi(k), l(e) \geq 0, \quad \forall k,v,s,e. \tag{3.5e}$$

Since the primal and dual are intrinsically different from the above references, we provide our full analysis for completeness of this study, starting from the observations below:

Lemma 3.2. *Constraint* (3.5b) *is binding,* i.e.*, equality holds instead of inequality at optimality, for at least one combination of k, v, s, p, where $k = 1 \ldots K, v \in \mathcal{F}_k, s \in \mathcal{S}_k, p \in \mathcal{P}_{v,s}^k$.* □

Lemma 3.3. *Constraint* (3.5d) *is binding.* □

Lemma 3.4. *For $\forall k$, Constraint* (3.5c) *is binding for at least one candidate host $v \in \mathcal{F}_k$.* □

Lemma 3.5. *For $\forall k, \forall v \in \mathcal{F}_k, \forall s \in \mathcal{S}_k$, Constraint* (3.5b) *is binding for at least one feasible routing path $p \in \mathcal{P}_{v,s}^k$.* □

Proof. Let ε be an arbitrarily small positive amount. If Lemma 3.2 is false, Constraint (3.5b) is not binding for every combination of k, v, s, p. Then we can reduce the value of $l(e)$ for an arbitrary e where $l(e) > 0$ by ε, and obtain a feasible dual solution with a strictly smaller objective value, contradicting our optimality assumption. If Lemma 3.3 is false, then we can reduce the value of $\varphi(k)$ for every k by ε. This will make every Constraint (3.5c) unbinding. Then we can reduce the value of $z(k, v, s)$ for every combination of k, v, s, which makes every Constraint (3.5b) to be unbinding, contradicting Lemma 3.2. If Lemma 3.4 is false for some k, then we can increase the value of $\varphi(k)$ by ε, which makes Constraint (3.5d) unbinding, contradicting Lemma 3.3. If Lemma 3.5 is false for some combination of k, v, s, then we can increase the value of $z(k, v, s)$ by ε, which makes Constraint (3.5c) unbinding for the corresponding k, contradicting Lemma 3.4. Therefore, we conclude that Lemmas 3.2–3.5 are all true. $\square$

Based on Lemmas 3.2–3.5, we have the following facts.

1. At optimality, $z(k, v, s) = \min_{p\in\mathcal{P}^k_{v,s}}\{\sum_{e\in p} l(e) + \sum_{u\in p\setminus\{s\}} \sigma(k, s, u)\}$, *i.e.*, $z(k, v, s)$ equals the shortest feasible routing path length in $\mathcal{P}^k_{v,s}$ regarding length functions $l(\cdot)$ for links and $\sigma(k, s, \cdot)$ for nodes;
2. At optimality, $\varphi(k) = \min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B^k_s z(k, v, s)\}$, *i.e.*, $\varphi(k)$ equals the minimum (over all possible candidate hosts $v \in \mathcal{F}_k$) weighted (by B^k_s) sum (over all sources $s \in \mathcal{S}_k$) of shortest feasible routing path lengths in $\mathcal{P}^k$ regarding length functions l and σ.

Let $\zeta_{k,v,s}(l, \sigma) = \min\limits_{p\in\mathcal{P}^k_{v,s}}\{\sum_{e\in p} l(e) + \sum_{u\in p\setminus\{s\}} \sigma(k, s, u)\}$ be the shortest path length in $\mathcal{P}^k_{v,s}$ regarding length functions l and σ, and $\psi_k(l, \sigma) = \min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B^k_s\zeta_{k,v,s}(l, \sigma)\}$ be the minimum weighted sum of shortest path lengths of all data sources of k over any candidate host v. Further define $\alpha(l, \sigma) = \sum_{k=1}^K \psi_k(l, \sigma)$. Then, Program (3.5) is equivalent to $\min_{l,\sigma\geq 0} \Delta(l, \sigma)/\alpha(l, \sigma)$, *i.e.*, finding l and σ minimizing $\Delta(l, \sigma)/\alpha(l, \sigma)$.

Our FPTAS to PO-MAP is presented in Algorithm 3.2. A bold symbol denotes a vector of normal symbols hereafter. In the process, the algorithm keeps track of

Algorithm 3.2: Approximation Scheme $A_{\text{PO-MAP}}$

```
    Input: Network $G$, application set $\mathbf{\Gamma}$, tolerance $\omega$
    Output: Scaling ratio $\lambda$, decisions $\mathbf{y} = \{y(k,v)\}_{k,v}$, path sets $\mathbf{P} = \{P^k_{v,s}\}_{k,v,s}$,
            bandwidth allocation $\mathcal{L}$
1   Initialize $\epsilon = \omega' = \frac{\omega}{4}$, $\gamma = \left(\frac{1+\epsilon(1+\omega')}{E+(V-1)S}\right)^{1+\frac{1}{\epsilon(1+\omega')}}$, $l(e) = \frac{\gamma}{c_e}$ for $\forall e \in \mathcal{E}$,
     $\sigma(k,s,u) = \frac{\gamma}{r^k_s B^k_s}$ for $\forall k, s, u \in \mathcal{V} \setminus \{s\}$, $P^k_{v,s} = \emptyset$ for $\forall k, v, s$, $\mathcal{L} = \emptyset$;
2   $\rho \leftarrow 0$;
3   while $\Delta(l,\sigma) < 1$ do                                        // phase
4       $\rho \leftarrow \rho + 1$;
5       for $k = 1 \ldots K$ do                                          // iteration
6           $\eta \leftarrow 1.0$;
7           while $\eta > 0$ do                                          // step
8               $(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta}) \leftarrow \text{PrimUpdt}(G, \mathbf{\Gamma}, k, l, \sigma, \omega')$;
9               if $\tilde{\eta} > \eta$ then
10                  $\boldsymbol{\phi} \leftarrow \eta\boldsymbol{\phi}/\tilde{\eta}$; $\tilde{\eta} \leftarrow \eta$;
11              end
12              $y(k,v) \leftarrow y(k,v) + \tilde{\eta}$; $\eta \leftarrow \eta - \tilde{\eta}$;
13              for $s \in \mathcal{S}_k$ do
14                  $P^k_{v,s} \leftarrow P^k_{v,s} \cup \{\tilde{p}_s\}$;
15                  $\mathcal{L}(\tilde{p}_s) \leftarrow \mathcal{L}(\tilde{p}_s) + \phi_s$;
16              end
17              $l(e) \leftarrow l(e)(1 + \frac{\epsilon\phi_e}{c_e})$ for $\forall e \in \mathcal{E}_{\tilde{\boldsymbol{p}}}$, where $\mathcal{E}_{\tilde{\boldsymbol{p}}} = \bigcup_{s\in\mathcal{S}_k} \tilde{p}_s$, and
                 $\phi_e = \sum_{s\in\mathcal{S}_k : e\in\tilde{p}_s} \phi_s$;
18              $\sigma(k,s,u) \leftarrow \sigma(k,s,u)(1 + \frac{\epsilon\phi_u}{r^k_s B^k_s})$ for $\forall s \in \mathcal{S}_k, u \in \mathcal{V}_{\tilde{\boldsymbol{p}}} \setminus \{s\}$, where
                 $\mathcal{V}_{\tilde{\boldsymbol{p}}} = \bigcup_{s\in\mathcal{S}_k} \{v \in \tilde{p}_s\}$, and $\phi_u = \sum_{s\in\mathcal{S}_k : u\in\tilde{p}_s\setminus\{s\}} \phi_s$;
19          end
20      end
21  end
22  Scale $\mathcal{L}$ and $\mathbf{y}$ after phase $\rho - 1$ by $1/\log_{1+\epsilon} 1/\gamma$;
23  $\lambda \leftarrow (\rho - 1)/\log_{1+\epsilon} 1/\gamma$;
24  return $(\lambda, \mathbf{y}, \mathbf{P}, \mathcal{L})$.
```

both a primal solution, denoted by variables $(\boldsymbol{y}, \mathcal{L})$ (note that λ can be computed based on $\mathcal{L}$), and a dual solution, denoted by the length functions (l, σ) (note that both variables $\boldsymbol{z}$ and $\boldsymbol{\varphi}$ can be computed based on l and σ). Both solutions will be gradually updated. Initially, each link e's dual length is initialized to γ/c_e, and each

Algorithm 3.3: Algorithm PrimUpdt($G, \mathbf{\Gamma}, k, l, \sigma, \omega'$)

```
   Input: Network $G$, application set $\mathbf{\Gamma}$, index $k$, length functions $l$ and $\sigma$,
          tolerance $\omega'$
   Output: Paths $\tilde{\boldsymbol{p}} = (\tilde{p}_s)^{\mathrm{T}}_{s\in\mathcal{S}_k}$, bandwidth $\boldsymbol{\phi} = (\phi_s)^{\mathrm{T}}_{s\in\mathcal{S}_k}$, selected node $\tilde{v}$,
           fraction of flow $\tilde{\eta}$
   // path computation
 1 for $\forall v \in \mathcal{F}_k$ do
 2     for $\forall s \in \mathcal{S}_k$ do
 3         $\tilde{p}_{v,s} \leftarrow \arg\min_{p\in\mathcal{P}^k_{v,s}} \{\sum_{e\in p} l(e) + \sum_{u\in p\setminus\{s\}} \sigma(k,s,u)\}$;
 4     end
 5 end
 6 $\tilde{v} \leftarrow \arg\min_{v\in\mathcal{F}_k}\{\sum_{s\in\mathcal{S}_k} B^k_s \zeta_{k,v,s}(l,\sigma)\}$;
 7 $\tilde{p}_s \leftarrow \tilde{p}_{\tilde{v},s}$ for $\forall s \in \mathcal{S}_k$;
   // bandwidth allocation
 8 $\Upsilon_e \leftarrow 0$ for $\forall e \in \mathcal{E}$;
 9 $\Upsilon_{s,u} \leftarrow 0$ for $\forall s \in \mathcal{S}_k, u \in \mathcal{V} \setminus \{s\}$;
10 for $\forall s \in \mathcal{S}_k$ do
11     for link $\forall e \in \tilde{p}_s$ do
12         $\Upsilon_e \leftarrow \Upsilon_e + B^k_s$;
13     end
14     for node $\forall u \in \tilde{p}_s \setminus \{s\}$ do
15         $\Upsilon_{s,u} \leftarrow \Upsilon_{s,u} + B^k_s$;
16     end
17 end
18 $\Upsilon^1_{\max} \leftarrow \max_{e\in\mathcal{E}}\{\Upsilon_e/c_e\}$;
19 $\Upsilon^2_{\max} \leftarrow \max_{s\in\mathcal{S}_k,u\in\mathcal{V}\setminus\{s\}}\{\Upsilon_{s,u}/r^k_s B^k_s\}$;
20 $\tilde{\eta} \leftarrow 1/\max\{\Upsilon^1_{\max}, \Upsilon^2_{\max}\}$, $\phi_s \leftarrow B^k_s \cdot \tilde{\eta}$ for $\forall s$;
21 return $(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta})$.
```

node u's dual length (regarding application k's data source s) is initialized to $\gamma/r^k_s B^k_s$. The algorithm runs in *phases* (Lines 3–21), in each phase going through an *iteration* for each application k (Lines 5–20). In each iteration, the algorithm tries to push exactly B^k_s amount of flow for each data source s of application k. This is done in *steps* (Lines 7–19), where in each step, we push the same fraction of flow ($\tilde{\eta}$) to the same candidate host ($\tilde{v}$) from all data sources. This ensures that when we update the primal solution, the increment in variable $y(k, v)$ is proportional to the flow pushed to v from any data source $s \in \mathcal{S}_k$, thus satisfying both Constraints (3.4b) and (3.4c). This is achieved by first calling the PrimUpdt subroutine to get a feasible primal update, denoted by $(\tilde{\boldsymbol{p}}, \boldsymbol{\phi}, \tilde{v}, \tilde{\eta})$, and then updating the primal solution in Lines 9–16. After primal update, the algorithm then updates the dual lengths $l(e)$ based on the bandwidth ϕ_e pushed along each link e, in Line 17; it also updates $\sigma(k, s, u)$ based on the bandwidth at each node, in Line 18. It stops when $\Delta(l, \sigma) \geq 1$, after which it then scales the obtained flows to enforce the link capacity constraints in Lines 22–23.

A key building block is the PrimUpdt subroutine, which produces a primal update for application k that will be incorporated into the current primal solution. Its algorithm is shown in Algorithm 3.3. It starts from finding the dual-shortest feasible path from every data source $s \in \mathcal{S}_k$ to every candidate host $v \in \mathcal{F}_k$, denoted as $\tilde{p}_{v,s}$. The candidate host $\tilde{v}$ corresponding to the minimum value $\psi_k(l)$ is picked, along with the corresponding paths to $\tilde{v}$, denoted as $\tilde{\boldsymbol{p}}$. Next, it derives a bandwidth allocation, such that 1) each data source s's bandwidth (ϕ_s) is proportional to its demand B_s^k, 2) total bandwidth on every link e does not exceed e's capacity c_e, 3) the robustness requirement is also satisfied at each node u for each application's each data source, and 4) the minimum ratio $(\tilde{\eta})$ between any source's bandwidth and its demand is maximized. This is done in Lines 8–20 of Algorithm 3.3. Node $\tilde{v}$, paths $\tilde{\boldsymbol{p}}$ and bandwidth allocation $\boldsymbol{\phi}$ are then returned along with the resulting scaling ratio $\tilde{\eta}$.

PrimUpdt relies on finding the dual-shortest feasible routing paths, as in Line 12. However, this task itself is non-trivial, as it is equivalent to the Delay Constrained Least Cost path (DCLC) problem, which itself is NP-hard. Nevertheless, there exist FPTASs for DCLC [139], which can output a $(1 + \omega')$-approximation of the dual-shortest feasible path within time polynomial to the input size and $1/\omega'$. Combined with the selection of ϵ, ω' and γ, we can prove that such an approximation is sufficient for obtaining our desired performance guarantee. The performance of $A_{\text{PO-MAP}}$ is summarized in Theorem 3.3.

Theorem 3.3. *Given G, $\mathbf{\Gamma}$, and $\omega \in (0,1)$, $A_{PO\text{-}MAP}$ (with Line 3 of the PrimUpdt subroutine replaced by a DCLC FPTAS) can compute a $(1-\omega)$-approximation of the optimal PO-MAP solution, within time polynomial to both the input size and $1/\omega$, and hence is an FPTAS to PO-MAP.* $\square$

Proof. We first prove the approximation ratio of $A_{\text{PO-MAP}}$, and then prove its time complexity.

Part I (Approximation Ratio): We first assume the optimal primal objective $\lambda^* \geq 1$; this assumption will be removed later on. Due to the strong duality of LP, the optimal dual objective Δ^* is equal to λ^*. Let (ρ, k, τ) denote step τ of iteration k of phase ρ in the algorithm. Given a symbol used in the algorithm, $\nu \in \{l, \sigma, \zeta_{k,v,s}, \psi_k, \alpha, \Delta, \phi_s, \tilde{v}, \tilde{p}_s\}_{k,v,s,e}$, we use $\nu^{\rho,k,\tau}$, $\nu^{\rho,k}$ and ν^{ρ} to denote the corresponding values in/after the corresponding step, iteration and phase, respectively. We also use ν to denote $\nu(l, \sigma)$ if no ambiguity is introduced.

Based on the primal-dual updates, we have the following:

$$\begin{aligned}
\Delta^{\rho,k,\tau} &= \sum_{e \in \mathcal{E}} c_e l^{\rho,k,\tau-1}(e) + \epsilon \sum_{s \in \mathcal{S}_k} \phi_s^{\rho,k,\tau} \sum_{e \in \tilde{p}_s^{\rho,k,\tau}} l^{\rho,k,\tau-1}(e) \\
&\quad + \epsilon \sum_{s \in \mathcal{S}_k} \phi_s^{\rho,k,\tau} \sum_{u \in \tilde{p}_s^{\rho,k,\tau} \setminus \{s\}} \sigma^{\rho,k,\tau-1}(k, s, u) \\
&\leq \Delta^{\rho,k,\tau-1} + \epsilon(1+\omega') \textstyle\sum_{s \in \mathcal{S}_k} \phi_s^{\rho,k,\tau} \zeta_{k,\tilde{v}^{\rho,k,\tau},s}^{\rho,k,\tau},
\end{aligned}$$

due to that each path $\tilde{p}_s^{\rho,k,\tau}$ is a $(1+\omega')$-approximation of the dual-shortest feasible $(s, \tilde{v}^{\rho,k,\tau})$-path, and the dual-shortest feasible path lengths are non-decreasing during the algorithm.

As in each iteration k, we push exactly B_s^k flow for $\forall s \in \mathcal{S}_k$, we have the following by summing up for all steps:

$$\begin{aligned}\Delta^{\rho,k} &\leq \Delta^{\rho,k-1} + \epsilon(1+\omega') \min_{v\in\mathcal{F}_k} \sum\nolimits_{s\in\mathcal{S}_k} B_s^k \zeta_{k,v,s}^{\rho,k} \\ &\leq \Delta^{\rho,k-1} + \epsilon(1+\omega')\psi_k^{\rho,k}.\end{aligned}$$

Summing up for all applications (iterations), we then have:

$$\Delta^\rho \leq \Delta^{\rho-1} + \epsilon(1+\omega')\alpha^\rho.$$

Since we know that $\frac{\Delta^\rho}{\alpha^\rho} \geq \Delta^* \geq 1$, we further have:

$$\begin{aligned}\Delta^\rho &\leq \frac{\Delta^{\rho-1}}{1-\frac{\epsilon(1+\omega')}{\Delta^*}} \leq \frac{\Delta^0}{\left(1-\frac{\epsilon(1+\omega')}{\Delta^*}\right)^\rho} \\ &\leq \frac{\Delta^0}{(1-\epsilon(1+\omega'))} \exp\left(\frac{(\rho-1)\epsilon(1+\omega')}{\Delta^*(1-\epsilon(1+\omega'))}\right),\end{aligned}$$

where the last inequality is due to that $(1+x) \leq \exp(x)$.

The initial dual objective value is $\Delta^0 = (E+(V-1)S)\gamma$ given the initial l and σ. Let ρ^* be the last phase before the algorithm stops. We know that $\Delta^{\rho^*} \geq 1$ and $\Delta^{\rho^*-1} < 1$. Then we can bound the optimal dual objective value Δ^* as follows:

$$\Delta^* \leq \frac{(\rho^*-1)\cdot\epsilon(1+\omega')}{(1-\epsilon(1+\omega'))\ln\frac{1-\epsilon(1+\omega')}{(E+(V-1)S)\gamma}}.$$

To bound the optimal primal objective value λ^*, first observe that each primal update only increases the bandwidth on each link e by at most c_e, and the bandwidth at each node u by $r_s^k B_s^k$ for application k's data source s. Therefore, when the flow through a link e increases by exactly c_e, its dual length $l(e)$ is increased by at least $(1+\epsilon)$ times, due to the dual update in Line 17; similarly, when the flow of (k,s) through a node u increases by $r_s^k B_s^k$, the node's dual length $\sigma(k,s,u)$ is increased by at least $(1+\epsilon)$ times, due to the dual update in Line 18. Now, as

$\Delta^{\rho^*-1} < 1$, we have $l^{\rho^*-1}(e) < 1/c_e$ for $\forall e \in \mathcal{E}$, and $\sigma^{\rho^*-1}(k,s,u) < 1/r_s^k B_s^k$ for $\forall k,s,u$. Therefore, the final flow after phase $\rho^* - 1$ scaled by a factor of $1/\log_{1+\epsilon} 1/\gamma$ is strictly feasible. Since in each phase we push exactly B_s^k flow for each data stream, the scaling ratio after $\rho^* - 1$ phases is exactly $\rho^* - 1$. Scaled by $1/\log_{1+\epsilon} 1/\gamma$, the scaling ratio $\lambda = (\rho^* - 1)/\log_{1+\epsilon} 1/\gamma$ is strictly feasible.

Based on these, the primal-dual ratio is bounded as follows:

$$\frac{\lambda}{\Delta^*} \geq \frac{(1-\epsilon(1+\omega')) \cdot \ln \frac{1-\epsilon(1+\omega')}{m\gamma}}{\epsilon(1+\omega') \cdot \log_{1+\epsilon} \frac{1}{\gamma}}.$$

Given our selection of ϵ, ω' and γ, we have $\frac{\lambda}{\Delta^*} \geq 1 - \omega$.

It remains to remove our assumption that $\lambda^* \geq 1$. Based on [38], if we can obtain a pair of bounds $(\lambda_{\mathsf{LB}}, \lambda_{\mathsf{UB}})$ such that $\lambda^* \in [\lambda_{\mathsf{LB}}, \lambda_{\mathsf{UB}}]$, then we can guarantee $\lambda^* \geq 1$ by scaling all demands by $1/\lambda_{\mathsf{LB}}$. Following [35], we use a path-based method to find λ_{LB} and λ_{UB}. For each data stream (k,s), we use a binary search to find a *maximum-capacity feasible routing path* $\bar{p}_{v,s}^k$ to each candidate host $v \in \mathcal{F}_k$. Given v, the search sets a threshold β, and then finds a shortest (s,v)-path (*w.r.t.* delay) in G_β, a subgraph of G that has all links in $\{e : c_e < \beta\}$ pruned. If the path has delay no more than D_k, β is increased; otherwise it is decreased. Let $\bar{b}_{v,s}^k = \min_{e \in \bar{p}_{v,s}^k}\{c_e\}$ be the capacity of $\bar{p}_{v,s}^k$, and $\bar{\lambda}_v^k = \min_{s \in \mathcal{S}_k}\{\bar{b}_{v,s}^k / B_s^k, r_s^k\}$. For each k, we then select candidate host $\bar{v}_k = \arg\max_{v \in \mathcal{F}_k}\{\bar{\lambda}_v^k\}$, and let $\bar{\lambda}_k = \bar{\lambda}_{\bar{v}_k}^k$. Then, our upper bound is $\lambda_{\mathsf{UB}} = E \min_k\{\bar{\lambda}_k\}$, as each flow can be decomposed into up to E paths, with no contention among each other. A lower bound is $\lambda_{\mathsf{LB}} = \min_k\{\bar{\lambda}_k\}/S$, by scaling using the maximum number of competing flows.

Part II (Time Complexity): For simplicity, we define notation $O^*(f) = O(f \log^{O(1)} \mathbb{L})$, where f is a function of the input size $\mathbb{L}$. Based on [35], [38], the number of phases is bounded by $\rho^* \leq \lceil \Delta^* \log_{1+\epsilon} \frac{1}{\gamma} \rceil = O^*(\frac{\Delta^*}{\omega^2})$, each with K iterations, and the total number of steps is bounded by $(E + (V-1)S) \log_{1+\epsilon} \frac{1+\epsilon}{\gamma} = O^*((E + (V-1)S)\frac{\Delta^*}{\omega^2})$

plus the total number of iterations. Each step incurs one PrimUpdt call, which both finds (approximate) dual-shortest feasible paths for every (v, s) pair, and allocates bandwidth. According to Xue *et al.* [139], each path is found in $O^*(\frac{1}{\omega'}VE)$ time. Bandwidth allocation in PrimUpdt takes $O(SV)$ time, as each path consists of at most $V-1$ links. Combining the above, the time complexity of $A_{\text{PO-MAP}}$ is given by $O^*(\frac{\Delta^*}{\omega^3}SFVE(E+(V-1)S+K))$.

To remove the dependency on Δ^*, we employ the demand scaling technique in [38]. If the algorithm does not stop after $\lceil 2\log_{1+\epsilon}\frac{1}{\gamma}\rceil$ phases, we know that $\Delta^* \geq 2$. We then double all demands, hence halving Δ^*, and then re-run Algorithm 3.2. Now, we have $\Delta^* \in [1, SE]$ after the initial scaling in Part I. Hence at most $O(\log_2(SE))$ demand scaling rounds are needed to bring Δ^* within $[1, 2]$, each spending $O^*(\frac{1}{\omega^3}SFVE(E+K))$ time. Omitting the logarithm terms, the final complexity is $O^*(\frac{1}{\omega^3}SFVE(E+(V-1)S+K))$ combined with the initial scaling. The theorem follows. $\square$

3.6.3 NO-MAP Formulation and Randomized Algorithm

We now turn to the NO-MAP problem, which is the hardest among the four provisioning problems. Though the hardness of NO-MAP follows from that of NO-SAP, there are $O(F^K)$ possible HD solutions in the worst case for NO-MAP, instead of the linear number in NO-SAP. This prevents us from iterating over all possible HD combinations as in the last section. Below, we first give an exact formulation of NO-MAP. Similar as in the previous subsections, we define $x(k, v) \in \{0, 1\}$ as the indicator of whether an application k is hosted on node $v \in \mathcal{F}_k$, $\mathcal{L}(p) \geq 0$ as the bandwidth allocation on $p \in \mathcal{P}$, and $\lambda \geq 0$ as the traffic scaling ratio. Then NO-MAP is formulated as follows:

$$\begin{aligned}
\max \quad & \lambda && && \text{(3.6a)}\\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}^k_{v,s}} \mathcal{L}(p) \geq B_s \cdot \lambda \cdot x(k,v), && \forall k, v, s; && \text{(3.6b)}\\
& \sum_{v \in \mathcal{F}_k} x(k,v) = 1, && \forall k; && \text{(3.6c)}\\
& \sum_{p \in \mathcal{P}: e \in p} \mathcal{L}(p) \leq c_e, && \forall e \in \mathcal{E}; && \text{(3.6d)}\\
& \sum_{p \in \mathcal{P}^k_s: e \in p} \mathcal{L}(p) \leq r^k_s \cdot B^k_s, && \forall k, s, e \in \mathcal{E}; && \text{(3.6e)}\\
& x(k,v) \in \{0,1\}, \mathcal{L}(p), \lambda \geq 0, && \forall k, v, p. && \text{(3.6f)}
\end{aligned}$$

Explanation: Program (3.6) basically has the same form as Program (3.3). The only exception is Constraint (3.6e), which enforces the *link-robustness* requirement for each link e instead of the *node-robustness* for each node u. This is because NO-MAP cannot support node-robustness: its final selected host is always a single point of failure when the host node fails.

Due to binary variables $x(k,v)$ and Constraint (3.6b), Program (3.6) is a Mixed Integer Quadratic Program (MIQP), which is generally hard to solve. However, by relaxing the integer constraints on variables $x(k,v)$, we arrive at an QP that has almost the same structure as Program (3.3). We can then apply the same transformation as from Program (3.3) to Program (3.4), which also generates an LP, in other words, the *linear relaxation* of Program (3.6). The linear relaxation is written as follows:

$$
\begin{aligned}
\max \quad & \lambda && && \text{(3.7a)}\\
\text{s.t.} \quad & \sum_{p \in \mathcal{P}^k_{v,s}} \mathcal{L}(p) \geq B_s \cdot y(k,v), && \forall k, v, s; && \text{(3.7b)}\\
& \sum_{v \in \mathcal{F}_k} y(k,v) \geq \lambda, && \forall k; && \text{(3.7c)}\\
& \sum_{p \in \mathcal{P} : e \in p} \mathcal{L}(p) \leq c_e, && \forall e \in \mathcal{E}; && \text{(3.7d)}\\
& \sum_{p \in \mathcal{P}^k_s : e \in p} \mathcal{L}(p) \leq r^k_s \cdot B^k_s, && \forall k, s, e \in \mathcal{E}; && \text{(3.7e)}\\
& y(k,v), \mathcal{L}(p), \lambda \geq 0, && \forall k, v, p. && \text{(3.7f)}
\end{aligned}
$$

Due to the similar structure of Program (3.7) and Program (3.4), we can basically adopt the same method as in Algorithms 3.2 and 3.3 to obtain an FPTAS to Program (3.7), for which the details are omitted. Based on the FPTAS, we then propose a randomized algorithm to NO-MAP, as shown in Algorithm 3.4. It starts by solving Program (3.7) using a modified version of Algorithm 3.3. With the fractional solution, it then randomly selects a host $v \in \mathcal{F}_k$ with probability equal to $\tilde{y}(k,v)$ (normalized $y(k,v)$) for each application. After that, it solves the original NO-MAP program with fixed hosts $\mathbf{v} = \{v_k\}_k$ to ensure solution feasibility. This turns out to be a trivial generalization of the DR subproblem of NO-SAP, and hence can be solved using the FPTAS in [153].

The time complexity of Algorithm 3.4 is dominated by the complexity of the FPTAS to PO-MAP and the FPTAS for solving the DR subproblem with fixed HD. Therefore, it also runs in time polynomial to the input size and $\frac{1}{\omega}$. Unfortunately, the randomized algorithm does not have a constant approximation ratio. Non-constant performance bound can be obtained via conventional stochastic theorems such as the

Algorithm 3.4: Randomized Algorithm $A_{\text{NO-MAP}}$

```
Input: Network $G$, application set $\mathbf{\Gamma}$, tolerance $\omega$
Output: Scaling ratio $\lambda$, host selections $\mathbf{v}$, path sets $\mathbf{P}$, bandwidth allocation $\mathcal{L}$
1 $(\lambda, \boldsymbol{y}, \boldsymbol{P}, \mathcal{L}) \leftarrow A_{\text{PO-MAP}}(G, \mathbf{\Gamma}, \omega)$;
2 for $k = 1$ to $K$ do
3     $\tilde{y}(k, v) \leftarrow y(k, v) / \sum_{v \in \mathcal{F}_k} y(k, v)$ for $\forall v \in \mathcal{F}_k$;
4     Select $v \in \mathcal{F}_k$ with probability $\tilde{y}(k, v)$ as $v_k$;
5 end
6 Solve NO-MAP (Program (3.6)) with fixed HD solution $\mathbf{v} = \{v_k\}_k$, with
    accuracy $\omega$;
7 return $(\lambda, \{v_k\}_k, \{P_s^k\}_{k,s}, \mathcal{L})$.
```

Chernoff bound. Such a result, however, is far from providing a realistic performance bound that could be useful in practical settings. We thus omit the theoretical analysis of Algorithm 3.4 for sake of simplicity.

3.7 Performance Evaluation

3.7.1 Experiment Settings

We used randomly generated topologies and applications for performance evaluation. The random topologies were generated using the Waxman model [130]. Each random topology has 20 nodes, where 20% of all nodes were randomly selected as facility nodes. Links were created using parameters α and β in the Waxman model, where $\alpha = \beta = 0.6$. Link capacities were randomly generated in $[10, 100]$ Mbps, and delays were randomly generated in $[1, 10]$ ms. In each experiment, we generated 5 IoT applications. An application had $[3, 10]$ data streams, each from a different data source. Application delay bounds were randomly generated in $[15, 25]$ ms. For each data stream, its bandwidth demand were randomly generated in $[1, 25]$ Mbps. The default robustness (maximum tolerable data loss ratio) was 0.5 for all data streams. We set accuracy $\omega = 0.5$ for the approximation algorithms. Above were the default parameters. We varied one control parameter in each set of experiments in order for evaluation under various scenarios.

SAP	Our SAP algorithm. For parallelizable applications, this is our FPTAS to PO-MAP (Algorithm 3.2). For non-parallelizable applications, this is our FPTAS to NO-SAP (Algorithm 3.1).
MAP	Our MAP algorithm. For parallelizable applications, this is our FPTAS to PO-MAP (Algorithm 3.2). For non-parallelizable applications, this is our randomized algorithm to NO-MAP (Algorithm 3.4).
ODA	*Optimal Delay-Agnostic* algorithm. For parallelizable applications, this directly solves an edge-flow multi-commodity flow (MCF) LP. For non-parallelizable applications, this attempts all combinations of application HD, and for each combination solves an edge-flow MCF LP that neglects applications' delay bounds. ODA yields an upper bound on the optimal delay-bounded solution.
NS (HD)	*Nearest Selection* HD heuristic. For each application, this selects the host with minimum maximum delay from all data sources.
RS (HD)	*Random Selection* HD heuristic. For each application, a random candidate host is selected that is within the delay bound from every data source.
GH (DR)	*Greedy Heuristic* for DR. This works in rounds where in each round, the delay-shortest path with positive capacity is found for every data stream, and then bandwidth allocation is done as in Lines 8–20 of Algorithm 3.3; it stops when any data stream's shortest path exceeds the application's delay bound.
DA (DR)	*Delay-Agnostic* optimal DR solution. An edge-flow MCF LP, which neglects application delay bounds, is solved. This yields an upper bound on DR.

Table 3.1: Implemented Algorithms

Our comparison algorithms are shown in Table 3.1. Note that we proposed algorithms to solve HD and DR both jointly (SAP, MAP, ODA) and separately (NS and RS for HD, and GH and DA for DR). In the experiments, we further decomposed the entire MAP algorithm (Algorithm 3.4 for the non-parallelizable case) into its subroutines for solving HD (Lines 1–5) and DR (Line 6) respectively. Each combination of HD and DR algorithms was denoted by {HD}+{DR}, *e.g.*, NS+GH uses NS for HD and GH for DR.

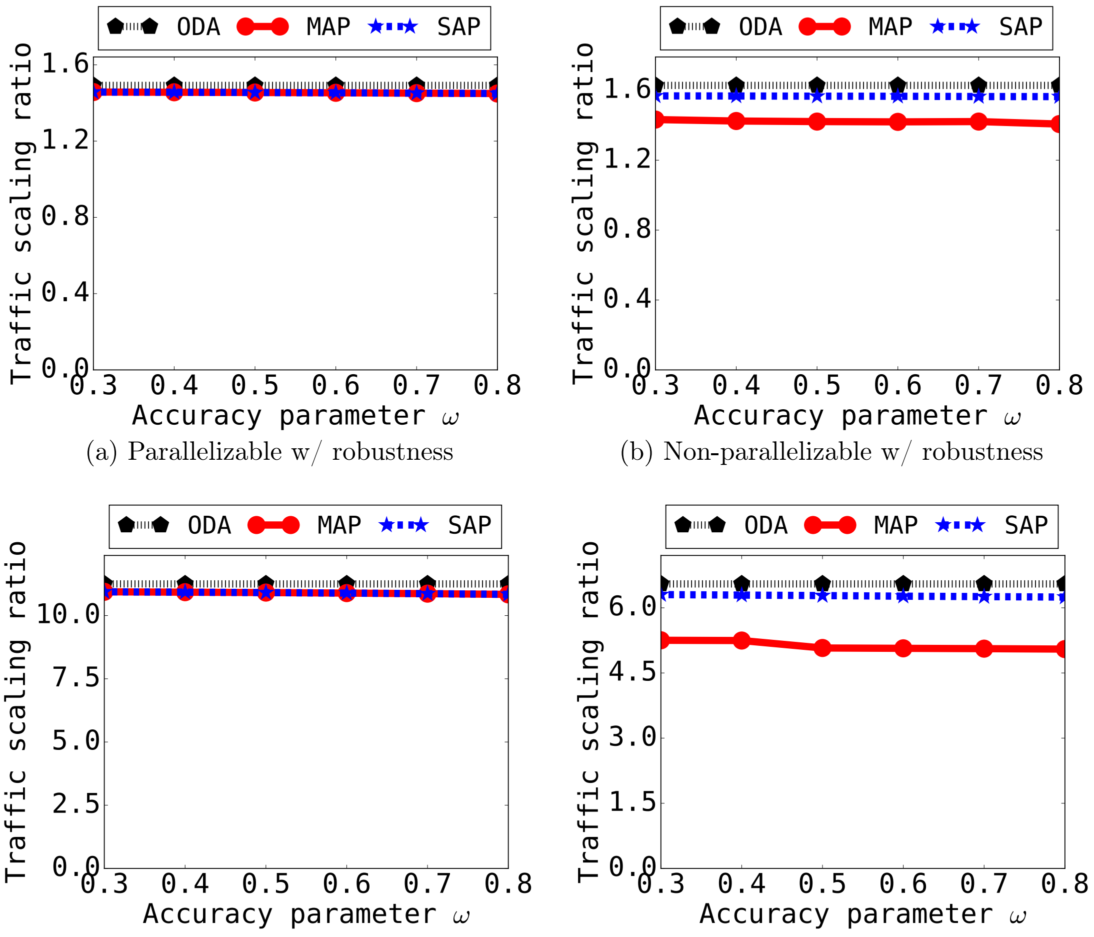
We used the following metrics in performance evaluation. *Traffic scaling ratio* is the optimization objective λ, which is the minimum ratio between the allocated bandwidth and the demand of every data stream. *Maximum delay ratio* is the average ratio between the maximum transmission delay received by any application and its delay bound. *Running time* is the average running time of an algorithm in an experiment.

We developed a C++-based simulator which implements all the above algorithms. The Gurobi optimizer [48] was used to solve the LPs. Experiments were conducted on a Ubuntu Linux PC with Quad-Core 3.4GHz CPU and 16GB memory. Each experiment was repeated for 50 times under the same setting, and results were taken as the average over all runs.

3.7.2 Evaluation Results

3.7.2.1 Single-Application Scenario

We use our single application experiments to show 1) that our algorithms are close-to-optimal through comparison with the theoretical upper bound (ODA), 2)



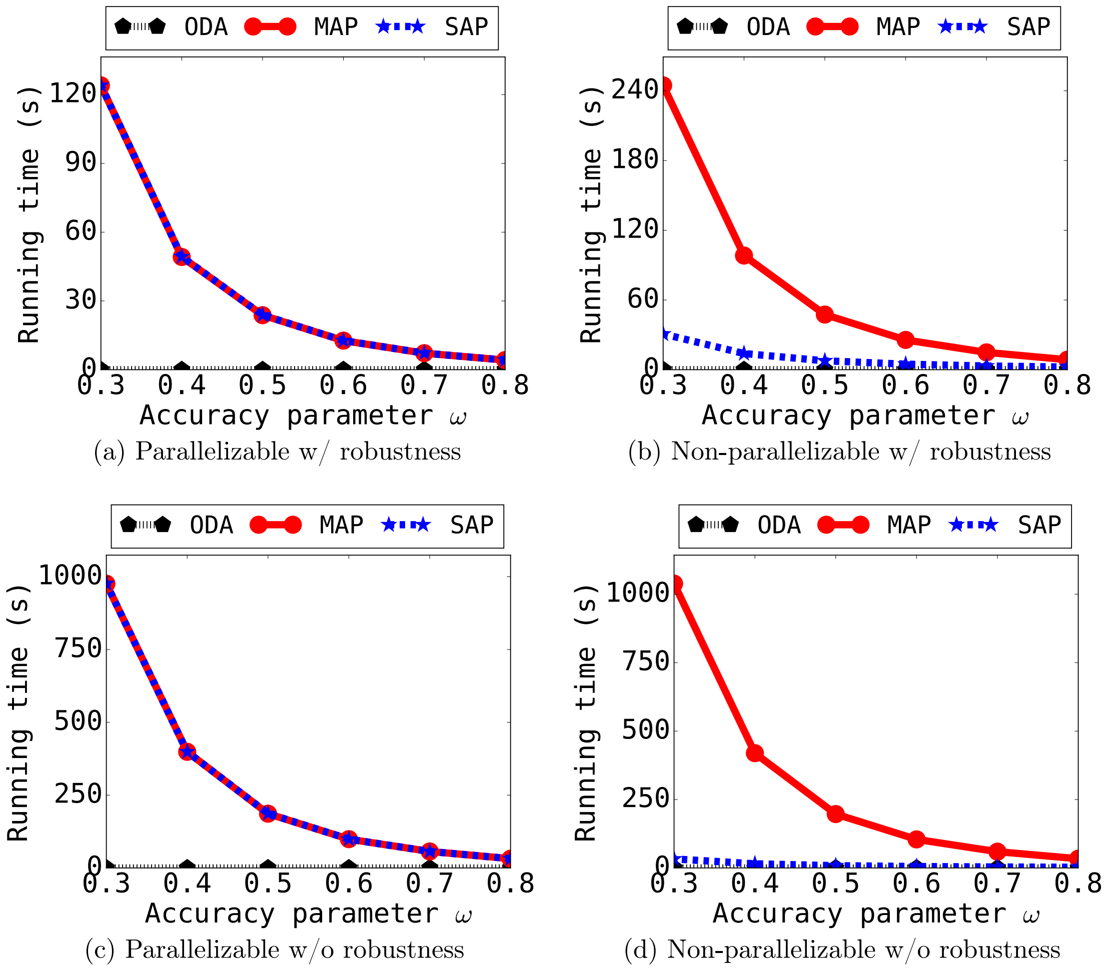
(a) Parallelizable w/ robustness (b) Non-parallelizable w/ robustness

(c) Parallelizable w/o robustness (d) Non-parallelizable w/o robustness

Figure 3.1: Single application: objective value against accuracy parameter ω.

the impact of robustness on the provisioning performance, and 3) the impact of parallelizability on our algorithms. The results are shown in Figs. 3.1–3.4. Note that for a single parallelizable application, SAP and MAP are essentially the same algorithm (Algorithm 3.2), and hence they have exactly the same performance.

Figs. 3.1 and 3.2 show the experiments in four combination scenarios: parallelizable application with robustness, non-parallelizable application with robustness, parallelizable application without robustness, and non-parallelizable application without robustness. First, we can see that our SAP FPTASs (MAP in Figs. 3.1(a) and 3.1(c)



(a) Parallelizable w/ robustness

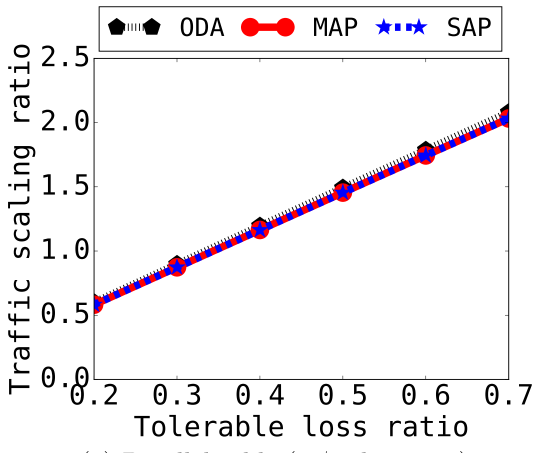
(b) Non-parallelizable w/ robustness

(c) Parallelizable w/o robustness

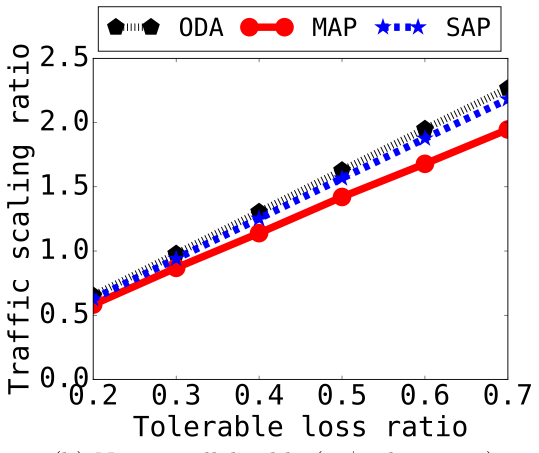
(d) Non-parallelizable w/o robustness

Figure 3.2: Single application: running time vs. accuracy parameter ω.

and SAP in Figs. 3.1(b) and 3.1(d)) achieve objective values extremely close to the upper bound ODA, much greater than their theoretical performance bounds ($(1-\omega)$ times the optimal). In Figs. 3.1(b) and 3.1(d), the MAP randomized algorithm achieves slightly worse performance than the FPTASs, yet its performance is still pretty close to ODA and even higher than $(1-\omega)$ times the optimal (although this is not theoretically guaranteed). On the other hand, we can observe that with decreasing accuracy parameter ω, little changes can be observed on the objective value, while great reduction in running time can be observed in Figs. 3.2(a)–3.2(d). This shows

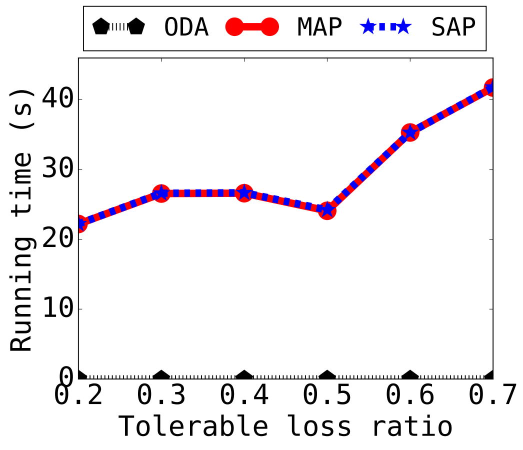


(a) Parallelizable (w/ robustness)

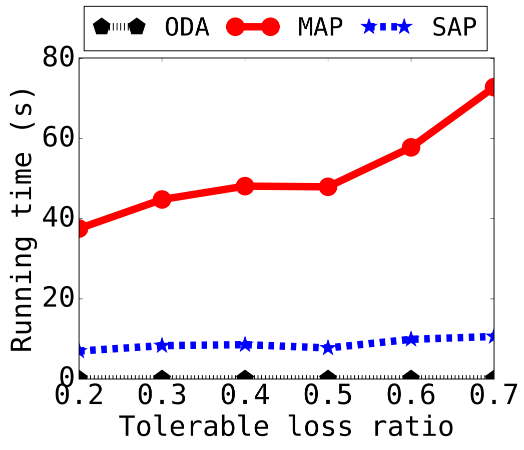


(b) Non-parallelizable (w/ robustness)

Figure 3.3: Single application: objective value vs. robustness parameter (maximum tolerable loss ratio).



(a) Parallelizable (w/ robustness)



(b) Non-parallelizable (w/ robustness)

Figure 3.4: Single application: running time vs. robustness parameter (maximum tolerable loss ratio).

that the proved theoretical bounds are pretty conservative in practice. An empirical setting of $\omega \geq 0.5$ can be used in practice to achieve close-to-optimal performance with reasonable computational overhead.

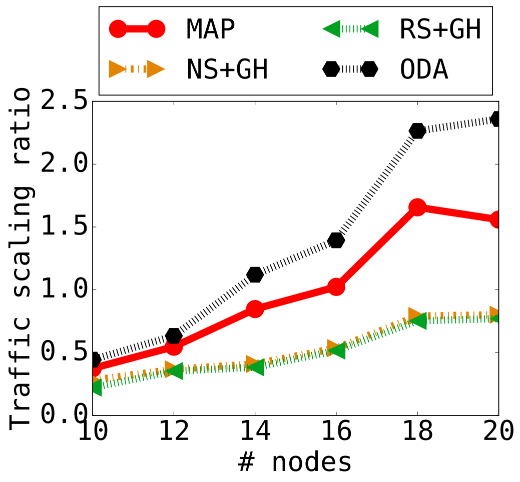
Comparing Figs. 3.1(a) and 3.1(c) (and similarly Figs. 3.1(b) and 3.1(d)), we can observe the impact of robustness. Enforcing robustness clearly reduces the objective value by great amounts. This shows that in practice, applications with robustness requirements can find it much harder to get accommodated when the system has limited resources. Looking at Fig. 3.2, running time increases when robustness is removed, which is due to that the complexity depends on the objective value, matching our previous analysis. Note that we did not use the polynomial-time demand scaling technique in our experiments, in order to better present this correlation.

Comparing Figs. 3.1(a) and 3.1(b), we can see the impact of parallelizability when robustness is enforced. Both ODA and SAP show that parallelizability reduces the objective value. This is because with parallelizability, the applications were able to enjoy the more strict *node-robustness*, while without parallelizability, the applications can only achieve *link-robustness*; clearly the former consumes more resources, as it guarantees the latter while providing additional protection. On the contrary, when robustness is not enforced, we see the opposite comparison in Figs. 3.1(c) and 3.1(d). Applications with parallelizability achieve better scaling ratios than those without. This is because without the restriction of robustness, the parallelizable problem is now an LP relaxation of the non-parallelizable problem, and hence the former represents an upper bound on the latter. Looking at the running times in Figs. 3.2(a) and 3.2(b), the time for the non-parallelizable case approximately doubles that for the parallelizable case. This is because in the non-parallelizable case, the same formulation is solved for two times, one for HD and one for DR; in the parallelizable case, both HD and DR can be solved in one round. Note that Algorithm 3.1 (SAP in Fig. 3.2(b)) is much faster than both Algorithm 3.2 (SAP/MAP in Fig. 3.2(a)) and Algorithm 3.4 (MAP in Fig. 3.2(b)), because Algorithm 3.1 solves a formulation that has fewer variables. Figs. 3.2(c) and 3.2(d) show similar comparisons, and the reason why the running time of MAP does not double in Fig. 3.2(d) is that the optimal objective value decreases greatly due to the non-parallelizability, and hence the running time of DR solving is dominated by the time of HD solving.

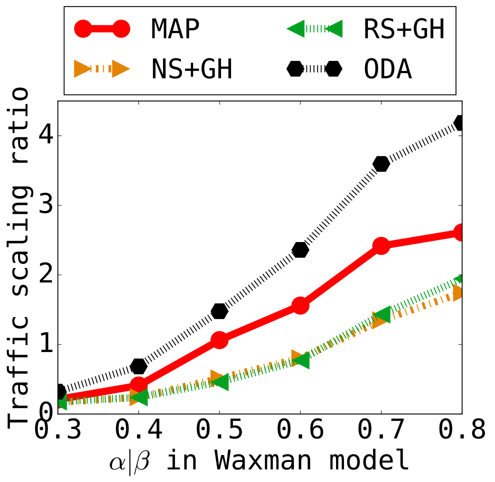
We further show in Figs. 3.3 and 3.4 the impact of different robustness parameters on the performance. With our formulation, the objective value should increase with the tolerable loss ratio of each application when the resources are relatively abundant, which is validated in Fig. 3.3. Figs. 3.3(a) and 3.3(b) show the same comparison as in Figs. 3.1(a) and 3.1(b), *i.e.*, parallelizable applications achieve worse scaling ratios than non-parallelizable ones due to the enforcement of node-based instead of link-based protection. The running time basically increases with increased objective value due to the relaxation of robustness requirement (larger tolerable loss ratio).

3.7.2.2 Multi-Application Scenario

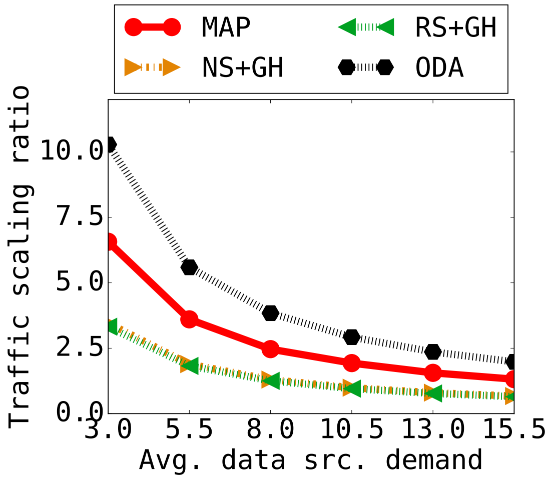
In the following experiments, we omitted the applications' robustness requirements, and focused on the non-parallelizable application case which is more common in practice. Figs. 3.5 and 3.6 show experiment results for multi-application provisioning, with varying number of nodes, connectivity, average bandwidth demand, and accuracy ω. First, MAP outperforms both RS+GH and NS+GH in relatively large scales. Specifically, MAP can serve up to $2\times$ the traffic that can be served by RS+GH or NS+GH in a majority of the experiments. The cost of its superior performance is its higher running time. MAP is slower than ODA mainly because the latter does not consider application delay bounds. Also, with more applications, the running time of MAP will soon beat that of ODA, as the former is a polynomial-time algorithm, while the latter's time complexity is exponential to the number of applications. The shown trends basically match our intuition, *e.g.*, increased nodes or links lead to increased



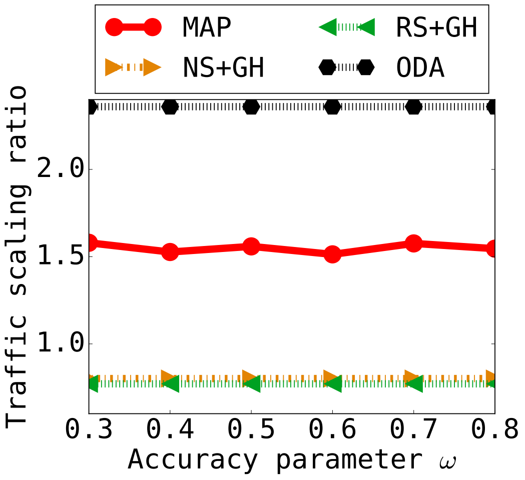
(a) Scaling ratio vs. # nodes



(b) Scaling ratio vs. connectivity



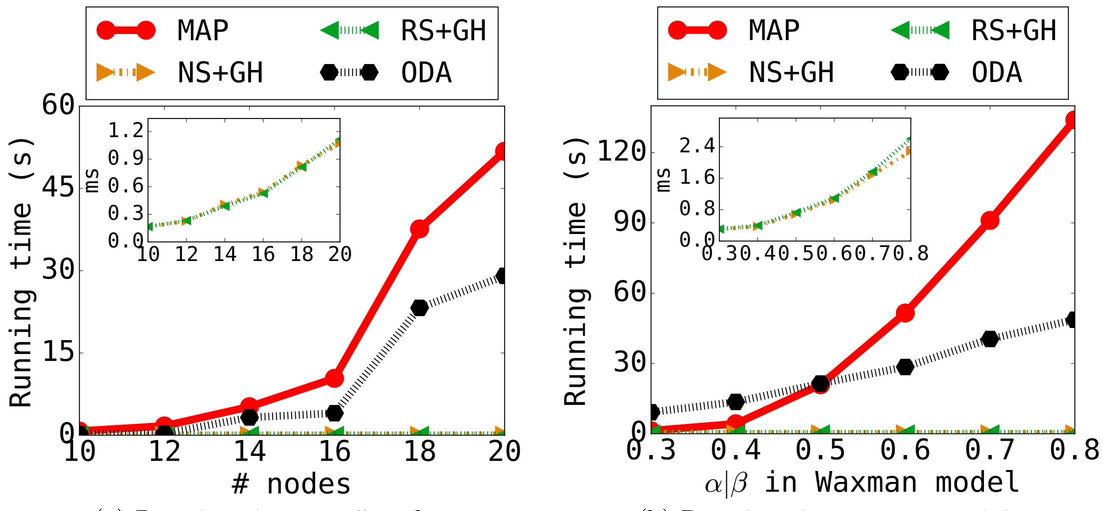
(c) Scaling ratio vs. demand



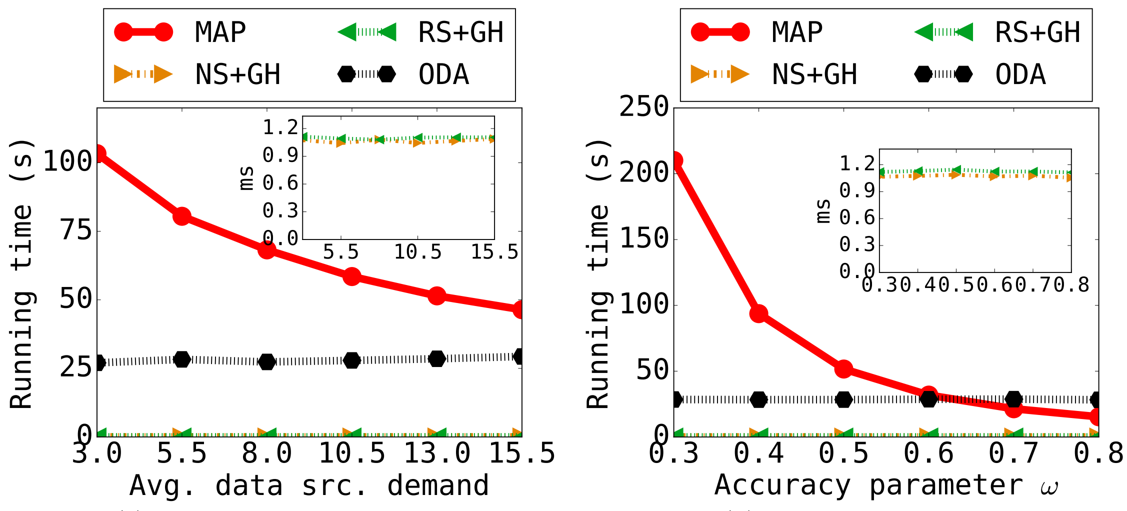
(d) Scaling ratio vs. ω

Figure 3.5: Multi-application: objective value vs. number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).

scaling ratios and running times, while larger bandwidth demands of the applications lead to smaller scaling ratios. In Fig. 3.6(c), the running time of MAP decreases with the scaling ratio. This matches their correlation described in the proof of Theorem 3.3. This correlation can be removed by the demand scaling technique illustrated in the proof. Finally, Figs. 3.5(d) and 3.6(d) show similar results as in Figs. 3.1 and 3.2 for



(a) Running time vs. # nodes (b) Running time vs. connectivity

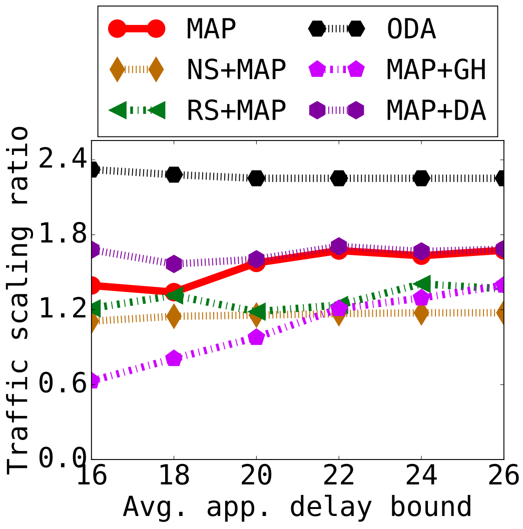


(c) Running time vs. demand (d) Running time vs. ω

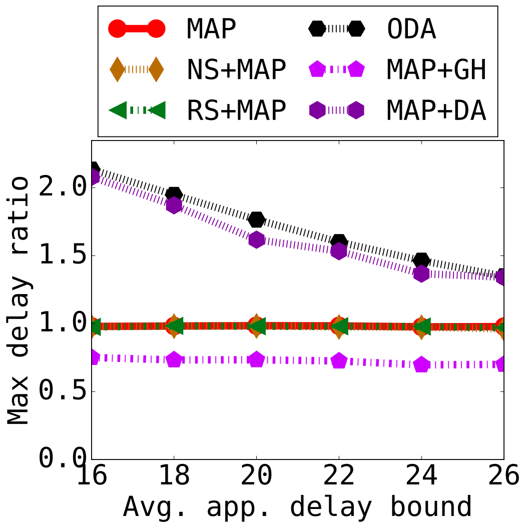
Figure 3.6: Multi-application: running time vs. number of nodes, connectivity (α, β), bandwidth demand, and accuracy (ω).

MAP, where a looser accuracy parameter ω does not lead to noticeable performance loss, but greatly reduces its running time.

The above experiments show the superior performance of our joint algorithm MAP. In Figs. 3.7, we further analyze its performance for HD and DR separately,



(a) HD comparisons



(b) DR comparisons

Figure 3.7: Multi-application: HD and DR with varying delay.

where we combined MAP's subroutines with different heuristics respectively. Shown in Fig. 3.7(a), delay-aware DR solutions (GH and MAP's DR subroutine) achieve better scaling ratios with larger delay bounds, while delay-agnostic solutions do not. Comparing different HD algorithms, MAP's HD subroutine still achieves much better traffic scaling ratios than either NS or RS. Interestingly, RS outperforms NS, which is because NS can lead to congestion when a host is significantly closer to most data sources than the others. Comparing different DR algorithms, MAP's DR subroutine also outperforms GH. On the other hand, since given fixed HD, the DA algorithm is optimal for delay-agnostic DR, we can see that MAP's DR subroutine is near the optimal when delay bounds are large. In Fig. 3.7(b), with MAP's DR subroutine, the maximum delay ratio is always bounded by but close to 1, meaning it utilizes paths of various delays, yet strictly sticks to the application delay bounds. GH also respects the delay bounds, yet it uses shorter paths, which leads to its low traffic scaling ratios in Fig. 3.7(a). Both ODA and DA are delay-agnostic, hence they can result in delays more than 2× the bounds, violating application QoS requirements. In summary, the superior performance of the MAP algorithm comes from the advantages of both its HD and DR subroutines, compared to the heuristic algorithms.

3.8 Conclusions

In this study, we studied the provisioning of real-time processing applications in IoT. For each application, we considered both the QoS and the robustness requirements. We considered two types of applications: parallelizable and non-parallelizable. For either type, we further studied both the provisioning of a single application, and the joint provisioning of multiple applications. We proved all four versions of the problem NP-hard. We then showed that three of the four versions admit FPTASs. For the last version, we proposed a randomized algorithm. We validated the advantages of our proposed algorithms over several heuristic solutions through extensive simulation experiments.

Chapter 4

LOAD BALANCING FOR INTERDEPENDENT IOT MICROSERVICES

4.1 Introduction

The Internet-of-Things (IoT) has drastically grown in size and capability in the recent years, owing to advances in broadband access networks, cloud/edge computing, big data analytics, machine learning, etc. In the near future, the global IoT can expand to tens of billions of devices, powering up numerous applications such as smart city, smart home, smart health, connected vehicles, etc. The global economic impact of IoT can be more than several trillion dollars in early 2020s [56].

Due to IoT's rapid development, the traditional monolithic architecture is no longer suitable for IoT applications. Instead, the microservice architecture is gaining support from both industry and academia. The architecture is built upon loosely coupled microservices, each with compact logic and well-defined interfaces. An IoT application is built as a collection of microservices with inter-microservice communications. In real deployment, an application can employ multiple instances of each microservice to achieve elasticity and robustness.

A key difficulty in microservice management is the interdependencies among microservices. Specifically, the input data to one microservice may depend on the output data of other microservices. To capture this, existing works adopt a graph-based approach, modeling an application as a directed graph where vertices represent microservices, and edges represent data flows between microservices. Fig. 4.1 shows an example.

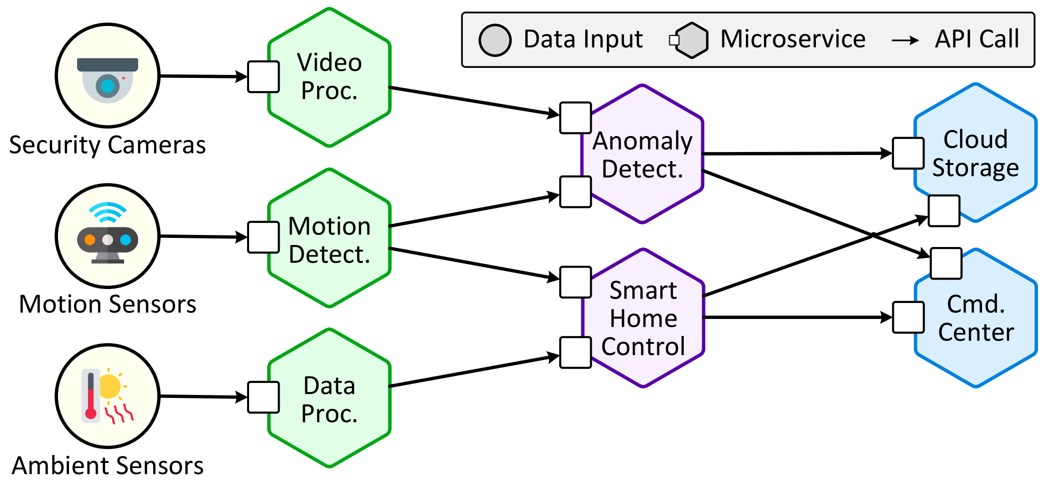


Figure 4.1: Graph representation of a smart home application.

In this study, we study an important problem in microservice management: load balancing across microservice instances. Lower load on instances can lead to better robustness and elasticity when facing instance failures or demand changes. Yet microservice load balancing is a complex problem due to the interdependencies. Specifically, changing the load distribution at one microservice instance could cause changes to the load on many other microservice instances. This situation is aggravated by the heterogeneous connectivity between microservice instances, which are distributed in the edge network.

Existing efforts have adopted simplified models to make the load balancing problem tractable, *e.g.*, by abstracting the application's processing logic as a chain of microservices [90]. Such a model, however, is insufficient to capture the rich interdependencies in modern IoT applications. In this study, we aim to provide a general solution to load balancing for interdependent microservices. We start with a directed acyclic graph (DAG)-based model for describing microservice interdependencies, which is rich enough to abstract a majority of IoT applications. We then propose a linear program formulation for the basic load balancing problem, where the application aims to minimize the maximum load among all instances.

The basic model neglects the quality-of-service (QoS) goal of the application, and may result in arbitrarily large end-to-end delay when answering user requests. This motivates us to study the more complex QoS-aware load balancing problem, where the application aims to optimize the end-to-end QoS while satisfying a desired load balancing goal. We first present a method to characterize the QoS of a load balancing solution, by abstracting a realization structure for the application graph. We show through a decomposition theorem that the realization structure can precisely represent QoS-aware load balancing solutions. Unfortunately, the QoS-aware problem is NP-hard. Hence we propose a fully polynomial-time approximation scheme (FPTAS) for the problem. We show through simulation experiments that our proposed algorithm indeed outperforms heuristic solutions in terms of QoS of the application.

Our main contributions are summarized as follows:

- To our knowledge, we are the first to study microservice load balancing with DAG-based interdependencies.
- We formulate both a basic and a QoS-aware load balancing problem. We prove the latter to be NP-hard.
- We show that the basic problem can be solved optimally, while the QoS-aware problem admits an FPTAS.
- Simulation experiments have shown that our algorithm can improve application QoS compared to baselines.

The rest of this study is organized as follows. In Section 4.2, we introduce background and related work. In Section 4.3, we present system model and the basic load balancing model. In Section 4.4, we describe our QoS-aware model, a formal statement of the problem, and its complexity. In Section 4.5, we propose the FPTAS for QoS-aware load balancing. In Section 4.6, we show our simulation results. In Section 4.7, we conclude this study.

4.2 Background and Related Work

4.2.1 Microservices and Application Graph Models

The microservice architecture, originally proposed for complex business applications in enterprises such as Amazon [131] and Netflix [132], has rapidly gained attention in the IoT domain, where it can largely reduce the cost and complexity of building IoT applications. With the help of distributed edge computing, a microservice-based application can achieve a number of benefits over a monolithic application, including robustness [68], elasticity [125], security [76], evolvability [33], and many more.

Graph-based approaches are commonly adopted to model and manage such decomposed IoT applications. Belli *et al.* [8] proposed an application architecture, which uses the processing graph to characterize QoS and improve infrastructure usage efficiency. Akkermans *et al.* [2] built a system for application orchestration based on application graphs. Erbs *et al.* [31] built another graph-based distributed processing system, which, in addition to orchestration across logical components, also considers the chronological dependencies among components. Lee *et al.* [71] used *tenant application graphs* to model cloud applications, and proposed bandwidth-aware application embedding in tree-like cloud networks.

Resource allocation for application graphs can be hard due to the complex structures of such graphs. Many heuristic solutions exist without performance bound [5],

[50]. To address this, existing approaches used simplified graph models. For example, Li *et al.* [72] and Niu *et al.* [90] used a chain-based model. Li *et al.* [72] proposed a heuristic for bandwidth-aware application chain deployment. Niu *et al.* [90] proposed a game theoretical approach to coordinate resources among competing application chains, in order to minimize their response times. A related area considers the embedding of *service function chains* (SFC), where similarly one or multiple chains of service functions are to be deployed in the network. Approximation algorithms exist, *e.g.*, due to Cao *et al.* [13], Kuo *et al.* [70], and Yu *et al.* [153]. Yu *et al.* [151] studied a different model where the application has a star structure, and proposed an FPTAS for network load balancing. Yet these simplified models greatly limit the expressiveness of this approach, and hence are not suitable for general IoT applications.

Outside of the IoT domain, there are problems with similar abstractions. An early area of research is virtual network embedding (VNE), where a request is given as a virtual network graph that is to be embedded on a physical network [21]. A related area is data center virtualization, where a tenant request is also given as a graph, which is to be embedded in the physical cloud network [157]. Since these problems are mainly studied in the network domain, they are different from resource allocation for application graphs. As an example, these problems commonly request that a virtual node is mapped to one and only one physical node, while a vertex (microservice) in an application graph can be implemented by a number of distributed instances at the same time.

4.2.2 Application-level Load Balancing

The load balancing problem has been studied in many different contexts, such as parallel computing [133], web applications [14], cloud applications [94], network load [3], [52], [121], microservices [90], etc. Due to the large body of work, we only do a brief review on the methods used.

Many existing works are based on the principle of *randomized load balancing*, where in-coming load is randomly assigned to different entities based on load information. For example, Equal-Cost Multi-Path (ECMP) [121] is one of the most widely used Layer-3 load balancing technique that has a lot of variants. In the computing domain, it has also been shown that using randomized load balancing can reduce the queueing delay at servers [145]. A drawback of the randomized approach is that it is difficult to consider the load interdependencies among entities, hence it may result in skewed load distributions at entities that significantly depend on others.

Contrary to randomized load balancing, *deterministic load balancing* can make decisions based on many different factors, such as entity interdependencies [5], [50], [90], QoS information [90], [151], energy consumption [94], failures and network asymmetry [52], etc. These solutions differ from the randomized ones in that commonly the former need global coordination to obtain system-wide information and to enforce load balancing policies. In network, this either requires a centralized network controller [52], or complex peer-to-peer information aggregation and dissemination [3]. In the computing domain, the task is easier, as most computing platforms already employ hypervisors to make centrally coordinated decisions. Among problems with different considerations, load balancing with complex interdependencies seems to be one of the most difficult, where only heuristic solutions exist [5], [50].

4.3 System Model and Basic Formulation

4.3.1 Application Model

An IoT application is built by selecting a number of microservices, and establishing proper inter-connections between their output and input APIs. Specifically, an application is modeled as a directed acyclic graph (DAG), denoted as $G = (V, E)$, where V is the set of *vertices* denoting microservices, and E is the set of *edges* denoting direct API calls between microservices. A microservice v is called a successor of microservice u if there is a directed edge $(u, v) \in E$; v is a predecessor of u if the opposite edge $(v, u) \in E$ exists. A microservice with no successor is called a *sink microservice.* For simplicity, we use $V_{\text{in}}(v)$ and $V_{\text{out}}(v)$ to denote the subsets of predecessors and successors of microservice v, respectively. G is called the *application graph* (app-graph) hereafter.

The set E defines how IoT data flow across microservices. For load balancing, it is important to capture how data are distributed at each microservice. For each edge $e = (u, v) \in E$, a *data distribution ratio* r_e is defined, which denotes the input data volume that microservice v would receive from u if u is fed with 1 unit of input data. The input data of a microservice thus depends on both the external data it directly receives, and the data distributed from its predecessors. r_e can be obtained via analysis of historical measurements. An app-graph example is shown in Fig. 4.2(a).

4.3.2 Infrastructure Model

The IoT application must be instantiated with microservice instances in the network. For each microservice $v \in V$, we define N_v as the set of *nodes* (microservice instances) that implement v. The physical connectivities between instances of a pair of microservices $(u, v) \in E$ are defined by *link* set $L_{uv} \subseteq N_u \times N_v$. Since we only care about connectivities between microservices that have direct API calls, the set L_{uv} is only well-defined for $(u, v) \in E$. The *infrastructure graph* (inf-graph) is denoted as $\Gamma = (N, L)$, where $N = \bigcup_{v \in V} N_v$ and $L = \bigcup_{(u,v) \in E} L_{uv}$. For simplicity, we let $v_n \in V$ be the microservice that node n belongs to; we then use the same notation, "predecessor/successor" of node n, to denote the corresponding predecessor/successor microservice of v_n. We further let $L_{\text{in}}(n)$ or $L_{\text{out}}(n)$ be the subset of links in-coming or out-going node n, respectively, and $L_{\text{out}}(n, v)$ be the subset of out-going links of node n that point to nodes belonging to microservice v. The inf-graph is also a DAG.

We next define a number of attributes for the inf-graph. First, each node $n \in N$ has a capacity c_n, which is the maximum load it can process to avoid congestion. In stream-analysis applications, c_n is commonly measured in terms of input data volume. Second, each node may have a processing delay d_n. The delay values can be obtained, *e.g.*, using the method outlined in [57]. Each link $l \in L$ may also have a delay d_l, denoting the data transmission latency between the two instances. For a path p, its delay is defined as the sum of node and link delays on p: $d(p) = \sum_{n \in p} d_n + \sum_{l \in p} d_l$. The application receives input data from external sources such as IoT devices, and the data may be fed into a certain node based on the data source locations, types of data, frontend distribution policies, etc. We use δ_n^{ext} to denote the volume of external data fed into node n, also called its *external demand*. A node with $\delta_n^{\text{ext}} > 0$ is called a *source* *node*. Note that node n may also receive input data from other nodes through API calls, which is different from its external demand, and is called the *internal demand* instead. An inf-graph example is shown in Fig. 4.2(b), corresponding to the app-graph in Fig. 4.2(a).

Note that for clarity of illustration, we use different terms for different graphs. We use "vertex" and "edge" for entities in the app-graph. We use "node" and "link" for entities in the inf-graph. In the next section, we will use "point" and "arc" for entities in the realization graph, to be explained later.

4.3.3 Basic Load Balancing Model

In our scenario, an application is instantiated by allocating external and internal demands to the microservice instances. In choosing how to instantiate the application, its owner aims to balance the load across different microservice instances, in order to achieve the best performance as well as to leave room for elastic scaling in the future. As a first step, we establish a formal model for the basic load balancing problem, which neglects the QoS requirement of the application.

Let δ_n be the total demand into node $n \in N$, which is the summation of both its external and internal demands: $\delta_n = \delta_n^{\text{ext}} + \delta_n^{\text{int}}$. The external demand δ_n^{ext} is regarded as a constant value, but the internal demand δ_n^{int} depends on the demands distributed from predecessor microservice instances of n, which in turn depends on the input demands of the predecessor instances, their corresponding data distribution ratios, and how they allocate their own output demands. Define $f(n_1, n_2)$ as the *demand allocation* from n_1 to n_2 if $(n_1, n_2) \in L$. We then have $\delta_n^{\text{int}} = \sum_{l \in L_{\text{in}}(n)} f(l)$, and hence:

$$\delta_n = \delta_n^{\text{ext}} + \sum_{l \in L_{\text{in}}(n)} f(l), \quad \forall n \in N. \tag{4.1}$$

For a demand allocation function $f : L \mapsto \mathbb{R}^*$ ($\mathbb{R}^*$ is the non-negative real number set) to be feasible, it must satisfy the following two constraints:

1. The total demand at each node should not exceed its desired capacity multiplied by a load factor ψ:
$$\delta_n \leq \psi \cdot c_n, \quad \forall n \in N. \tag{4.2}$$
2. The demand distributed from a node to all nodes of a successor microservice should satisfy the data distribution ratio between the two microservices:
$$\sum\nolimits_{l \in L_{\text{out}}(n,w)} f(l) = r_{(v_n,w)}\delta_n, \; \forall n, w \in V_{\text{out}}(v_n). \tag{4.3}$$

The load factor ψ essentially specifies the maximum load of any microservice instance. Commonly, the application would have a desired bound Ψ, such that the load on any instance does not exceed this bound. We then define the following problem:

Definition 4.1. *Given app-graph G with inf-graph Γ, and load bound $\Psi > 0$, the* ***Basic Load Balancing (BLB)*** *problem seeks for a demand allocation function $f : L \mapsto \mathbb{R}^*$, which satisfies Eqs.* (4.1), (4.2) *and* (4.3) *while ensuring $\psi \leq \Psi$. Its optimization version, instead of giving a load bound Ψ, is to minimize ψ, and is named* ***O-BLB*** *hereafter.* □

The following linear program (LP) formulates O-BLB:

$$\min_{f \geq 0} \quad \psi \tag{4.4}$$
$$\text{s.t.} \quad (4.1), (4.2) \text{ and } (4.3).$$

Theorem 4.1. *O-BLB can be solved in $O(|L|^3 \cdot \mathbb{L})$ time.* □

Proof. Program (4.4) is an LP with $(|L| + 1)$ variables. Based on [143], the LP can be solved optimally in $O(|L|^3 \cdot \mathbb{L})$ time ($\mathbb{L}$ is the input size). □

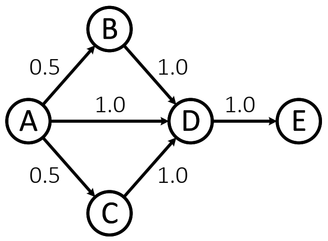
In Fig. 4.2(b), we also show a feasible solution to the load balancing problem. Bold links show links with positive demand allocation, and each bold link's allocation is equal to the cumulative load imposed on the link. For example, link $\mathrm{A}_1 \to \mathrm{B}_1$ has an allocation of $\delta \cdot 0.5 = 1.0$, while link $\mathrm{D}_1 \to \mathrm{E}_1$ has an allocation of $\delta \cdot (0.5 \cdot 1.0 + 0.5 \cdot 1.0) \cdot 1.0 = 2.0$ with the first half of demand coming from the path $\mathrm{A}_1 \to \mathrm{B}_1 \to \mathrm{D}_1$, and the second half from $\mathrm{A}_1 \to \mathrm{C}_1 \to \mathrm{D}_1$. A clearer view of the solution is shown in Fig. 4.2(c), which will be detailed using the real-graph abstraction in the next section. This basic model, however, neglects the QoS of the application, and may lead to arbitrarily long delay for serving user demands. In the next section, we propose a novel model for characterizing QoS.

4.4 QoS-aware Load Balancing

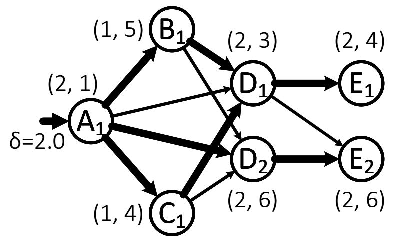
The above outlines a basic formulation of the load balancing problem. The model, however, merely reflects the numerical relationship between different instances, without describing the richer structural relationship in the app-graph. It is therefore intrinsically difficult to incorporate QoS information into the formulation. In this section, we model the QoS of an application through a novel realization graph abstraction.

We consider the following QoS goal of the application. In IoT, users usually ask for a guaranteed response time to ensure timely reception and handling of IoT events, such as traffic status or emergency events. We hence assume that the application's QoS goal is to bound or minimize the *maximum end-to-end delay* that any external demand would experience. To characterize this, we propose the following abstraction:

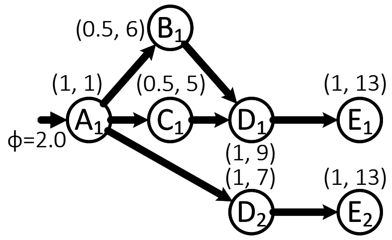
Definition 4.2 (Real-graph). *Given app-graph* $G = (V, E)$*, inf-graph* $\Gamma = (N, L)$



(a) App-graph $G = (V, E)$ (Section 4.3). Symbols in circles are microservices. Values on lines are distribution ratios.



(b) Inf-graph $\Gamma = (N, L)$ (Section 4.3). Symbols in circles are microservice instances. Values beside circles are (capacity, delay). Links have 0 delay. Bold links show a feasible load balancing solution.



(c) Real-graph $\pi = (X_\pi, A_\pi)$ (Section 4.4). Symbols in circles are instances mapped from the corresponding points. Values beside circles are (impact ratio, max cumulative delay).

Figure 4.2: App-graph (a), inf-graph (b) and a real-graph (c) of an example application with load bound $\Psi = 1$. Bold links in (b) show a feasible load balancing solution with max delay of 13, which is further shown in (c) as a single decomposed real-graph.

and a source node $n \in N$, a ***realization graph (real-graph)*** *is defined as a DAG $\pi = (X_\pi, A_\pi)$, coupled with a mapping $\sigma : X_\pi \mapsto N$, which satisfies that:*

1) x_π with $\sigma(x_\pi) = n$ is the only point with 0 in-degree;

2) $\forall x \in X_\pi$ and $\forall v \in V_{\text{out}}(v_{\sigma(x)})$, there is exactly one $y \in X_\pi$, such that $\sigma(y) \in N_v$, $(\sigma(x), \sigma(y)) \in L$, and $(x, y) \in A_\pi$. □

X_π is the set of *points* in π, and A_π is the set of *arcs*. We call x_π the root point of real-graph π. Real-graphs with roots mapped to source node $n \in N$ are denoted by set Π_n, and real-graphs of all source nodes are denoted by set $\Pi = \bigcup_n \Pi_n$. For simplicity, we use the same notation σ to map entities (nodes, links, paths or subgraphs) in π to the corresponding entities in Γ. We also use a point x to represent the node $\sigma(x) \in N$ when no ambiguity is introduced. $N_\pi \subseteq N$ and $L_\pi \subseteq L$ denote the subsets of nodes and links that are mapped from some points and arcs in π, respectively.

We now explain the intuition behind Definition 4.2. A real-graph can be viewed as a unitary structure that realizes every possible processing path starting with a source microservice (a microservice that has source nodes) in the app-graph. Each path is realized by a sequence of physical instances. To do this, we start from a source node, and recursively instantiate every successor microservice of the current node by assigning an instance to it, until we reach the sinks. Note that in this process, we may choose different instances of the same microservice, each as the successor of a different predecessor instance. Hence there can be multiple points in the real-graph mapped to the same node. We show an example of a real-graph in Fig. 4.2(c), which will be explained later.

We can define a number of attributes for π. First, each point/arc inherits the delay of its mapped node/link in Γ. Second, since each point x has exactly one neighbor y for each successor $v \in V_{\text{out}}(v_x)$, we can define the distribution ratio $r_{x,y} = r_{(v_{\sigma(x)}, v_{\sigma(y)})}$ for $(x, y) \in A_\pi$. We can further derive the *impact ratio* ρ^π_x for each point $x \in X_\pi$, defined as the *demand on x when one unit of demand is input at root x_π*. This can be computed by initially letting $\rho^\pi_{x_\pi} = 1$, and then traversing π from x_π, with each point adding its own impact ratio times the distribution ratio of each out-going arc to the impact ratio of the corresponding out-going neighbor. Similarly, the impact ratio ρ^π_a for each arc $a \in A_\pi$ can be computed. Then, we sum the impact ratios of all points mapped to a node $m \in N_\pi$ to compute the impact ratio ρ^π_m imposed on m by π, and similarly the impact ratio ρ^π_l on each link $l \in L_\pi$. The unitarity of π is guaranteed by assigning exactly one instance to every point's every successor microservice; in other words, each processing path in the app-graph corresponds to exactly one path in π.

Based on these, we can define a *source demand allocation* function $\phi : \Pi \mapsto \mathbb{R}^*$. For $\pi \in \Pi_n$, the value $\phi(\pi)$ denotes the external demand at the source node n that is allocated to be carried on real-graph π. Each node/link's demand under π can then be computed by multiplying $\phi(\pi)$ with the node/link's impact ratio. We highlight the importance of this real-graph abstraction in the following theorem, which is an analogy to the Flow Decomposition theorem for traditional network flow:

Theorem 4.2. *Any demand allocation f satisfying Eqs.* (4.1) *and* (4.3) *can be decomposed into at most $|N| + |L|$ real-graphs Π^{sel} with $\phi(\pi) > 0$ for $\forall \pi \in \Pi^{sel}$, such that the total demand incurred by ϕ on any link $l \in L$ is no more than $f(l)$.* □

Proof. We decompose f as follows. We first find an arbitrary real-graph $\pi \in \Pi_n$ for $n \in N$ with $\delta_n^{\text{ext}} > 0$, such that $f(l) > 0$ for $\forall l \in L_\pi$. We then calculate the maximum acceptable demand of π as $\delta(\pi) = \min\{\delta_n^{\text{ext}}, f(l)/\rho_l^\pi \text{ for } \forall l \in L_\pi\}$, *i.e.*, the minimum among δ_n^{ext}, and the total allocated demand on every link l that appears in π factored by the inverse of its impact ratio $1/\rho_l^\pi$. We let $\phi(\pi) = \delta(\pi)$. We then visit every link $l \in L_\pi$, deducting $\delta(\pi) \cdot \rho_l^\pi$ from $f(l)$; we also deduct $\delta(\pi)$ from δ_n^{ext}. After the deduction, the remaining f and demands still satisfy Eqs. (4.1) and (4.3), since we deduct the same amount from the lefthand and righthand sides of Eqs. (4.1) and (4.3). Due to our calculation of $\delta(\pi)$, at least one link's allocated demand is fully taken away during the deduction, or the total demand at a source node is deducted. We need at most $|N| + |L|$ steps to deduct all allocations from f. Also, we never deduct more than the demand allocated on any link, hence any capacity constraint satisfied by f is also satisfied by ϕ.

We now prove that we can always find a π with $f(l) > 0$ for $\forall l \in L_\pi$, if f is feasible and $\exists n \in N$ s.t. $\delta_n^{\text{ext}} > 0$. Let n be a node with $\delta_n > 0$ where δ_n comes from either external or internal demands. By Eq. (4.3), there exists $m \in N_v$ for $\forall v \in V_{\text{out}}(v_n)$ such that $f(n, m) > 0$, and hence $\delta_m > 0$. Therefore, we can start from any n with positive external demand, arbitrarily select a node $m \in N_v$ with $f(n, m) > 0$ for each successor microservice v of v_n, and then follow this process at each selected m until no successor to work on. This clearly generates a real-graph whenever f is feasible and has positive external demand, which completes our proof. $\square$

Fig. 4.2(c) shows a real-graph, which is also a decomposition of the load balancing solution shown in Fig. 4.2(b) (in practice, a solution may be decomposed into multiple real-graphs; in our example, only one is needed). We compute both the impact ratio and the cumulative delay for each point as shown in the figure. For example, the point mapped to D_1 has impact ratio $(r_{\text{AB}} \cdot r_{\text{BD}} + r_{\text{AC}} \cdot r_{\text{CD}}) = 1.0$. Delay is computed as the maximum delay from the root, *e.g.*, the delay at the point mapped to D_1 is $d_{\text{A}_1} + \max\{d_{\text{B}_1}, d_{\text{C}_1}\} + d_{\text{D}_1} = 9$. Note that although the real-graph realizes the app-graph, it may not be isomorphic to the app-graph, as it allows instantiating a microservice by multiple instances, such as microservice D instantiated by D_1 and D_2.

Theorem 4.2 is fundamental, as it enables us to use real-graphs as a basic structure for characterizing a load balancing solution. In other words, instead of defining a per-link allocation function f, we can define the allocation ϕ over the real-graphs from each source node, with the end-to-end delay of the application defined as the maximum delay from the source point to the leaf points of any real-graph with positive allocation. We can then define the QoS-aware load balancing problem. Let D be the application's delay bound. Define $d(\pi)$ as the maximum path delay from root x_π to any leaf point in π: $d(\pi) = \max\{d(p) \,|\, p \in \pi\}$. For brevity, we let $\Pi^m = \{\pi \in \Pi \,|\, m \in N_\pi\}$ be the subset of all real-graphs that include points mapped to node m.

Definition 4.3. *Given app-graph G, inf-graph Γ, load bound $\Psi > 0$, and delay bound $D > 0$, the **QoS-aware Load Balancing (QLB)** problem seeks for a subset of real-graphs Π_n^{sel} for each source node n, with $\Pi^{\text{sel}} = \bigcup_{n \in N} \Pi_n^{\text{sel}}$, and a source demand allocation function $\phi : \Pi^{\text{sel}} \mapsto \mathbb{R}^*$, s.t.:*

1) $\psi \leq \Psi$,

2) for node $\forall m \in N$, $\sum_{\pi \in \Pi^m} \rho_m^\pi \cdot \phi(\pi) \leq \psi \cdot c_m$,

3) for source node $\forall n \in N$, $\sum_{\pi \in \Pi_n^{\text{sel}}} \phi(\pi) = \delta_n^{\text{ext}}$, and

4) for real-graph $\forall \pi \in \Pi^{\text{sel}}$, $d(\pi) \leq D$.

*The optimization version, denoted as **O-QLB** hereafter, is to minimize the maximum delay of all selected real-graphs.* □

Proof of the following theorem is deferred to the appendix.

Theorem 4.3. *Both QLB and O-QLB are NP-hard.* □

4.5 Approximation Scheme Design

Due to the NP-hardness of O-QLB, we seek to develop an approximation algorithm. Below, we first show that if all delay values are positive integers, the QLB problem can be solved in pseudo-polynomial time. Such a problem is defined as Integral QLB (IQLB), with O-IQLB being its optimization version to minimize maximum delay. The pseudo-polynomial time algorithm is then used as a building block in the design of an approximation scheme for the general non-integral problem.

4.5.1 Pseudo-Polynomial Time Optimal Algorithm

Our pseudo-polynomial time algorithm for IQLB is based on a layered graph technique [85], [139]. Given inf-graph Γ and an integral delay bound D, we define an auxiliary inf-graph $\Gamma^D = (N^D, L^D)$. The node set $N^D = \{n^0, n^1, \ldots, n^D \,|\, n \in N\}$, *i.e.*, a node has $(D+1)$ copies each belonging to a layer. For source node $n \in N$, we let $\delta^{\text{ext}}_{n^{d_n}} = \delta^{\text{ext}}_n$; all other nodes have 0 external demand. Let $d^+_{nm} = d_n + d_{(n,m)}$ be the delay of link $(n,m) \in L$ plus the delay of m. The link set $L^D = \{(n^i, m^{i+d^+_{nm}}) \,|\, (n,m) \in L, i = 0, 1, \ldots, D - d^+_{nm}\}$, *i.e.*, each original link in L has $(D - d^+_{nm} + 1)$ copies. Each link copy inherits the distribution ratio of the original link. Due to page limit, we refer the reader to [85] (p. 4, Fig. 4) for an illustrative figure of the layered graph technique.

It is easy to see that each path or real-graph in Γ^D corresponds to exactly one path or real-graph in Γ, respectively. On the opposite direction, each path or real-graph starting with one source node n in Γ also corresponds to exactly one path or real-graph in Γ^D (since the external demand of each source node in Γ enters at exactly one node in Γ^D). As our focus is only on paths or real-graphs starting with source nodes, we use p or π to denote both a path or real-graph in Γ, and its correspondence in Γ^D, without introducing ambiguity.

The intuition behind this construction is to enforce that going through a node n or link l in Γ is equivalent to "climbing" d_n or d_l layers in the auxiliary inf-graph, respectively. The processing delay of source node n is encoded such that its external demand enters in the respective d_n-th layer. Since only $(D+1)$ layers (from 0 to D) present, any processing path must reach a sink node within D layers, thus bounding the maximum delay. Formally, we have the following observation:

Observation 1. *Any path p or real-graph π in Γ^D has delay $d(p) \leq D$ or $d(\pi) \leq D$ in Γ, respectively.* $\square$

Combining Theorem 4.2 and Observation 1, we are motivated to study the basic load balancing problem on the auxiliary inf-graph, which is formulated as the following LP:

$$\begin{aligned}
\min_{f \geq 0} \quad & \psi && \text{(4.5a)}\\
\text{s.t.} \quad & \delta_n = \textstyle\sum_{l \in L_{\text{in}}^D(n)} f(l) + \delta_n^{\text{ext}},\ \forall n \in N^D; && \text{(4.5b)}\\
& \textstyle\sum_{i=0}^{D} \delta_n \leq \psi \cdot c_n,\ \forall n \in N; && \text{(4.5c)}\\
& \sum_{l \in L_{\text{out}}^D(n,w)} f(l) = r_{(v_n,w)} \delta_n, \forall n \in N^D, w \in V_{\text{out}}(v_n). && \text{(4.5d)}
\end{aligned}$$

Program (4.5) is almost the same as Program (4.4), except that the capacity constraint (4.5c) now considers the demands entering all copies of the same node n. We then have the following:

Theorem 4.4. *IQLB can be solved in $O(D^4|L|^3\mathbb{L})$ time.* $\square$

Proof. Program (4.5) is an LP with at most $D|L|$ variables and input size of $O(D\mathbb{L})$, and hence it can be solved in $O(D^4|L|^3\mathbb{L})$ time. By Theorem 4.2, the feasible solution to Program (4.5) can be decomposed into at most $D(|N| + |L|)$ real-graphs on Γ^D. By Observation 1, each real-graph has delay bounded by D, thus the solution is feasible to IQLB if the optimal value of Program (4.5) satisfies $\psi \leq \Psi$. Now, assume IQLB has a feasible solution, by reversing the decomposition, we can construct a demand allocation f that satisfies all constraints in Program (4.5). The theorem follows. $\square$

4.5.2 Approximation Scheme for O-QLB

The basic idea of our algorithm is to find a sufficiently fine-grained discretization of the real-valued delays, such that the discretized solution is a good approximation of the optimal solution, and yet the time complexity is polynomial to the input size (and the inverse of an approximation factor ϵ). For this reason, we use a factor α to represent the granularity of discretization. We then define the discretized delay values given α: $d_n^\alpha = \lfloor \alpha \cdot d_n \rfloor + 1$ for $n \in N$, and $d_l^\alpha = \lfloor \alpha \cdot d_l \rfloor + 1$ for $l \in L$; we use similar symbols $d^\alpha(p)$ or $d^\alpha(\pi)$ to denote the delay of a path or a real-graph after discretization, respectively. The discretized delays satisfy the following lemma:

Lemma 4.1. *For any path p in Γ, we have*

$$\alpha \cdot d(p) \leq d^\alpha(p) \leq \alpha \cdot d(p) + 2|N| - 1. \qquad \square$$

Proof. The left inequality is clear. The right one is because each path has at most $|N|$ nodes and $|N| - 1$ links. $\square$

By selecting a proper factor α, we discretize the O-QLB problem to have only integral delay values. We want to solve the resulting O-IQLB problem to obtain an approximation to the original O-QLB problem. Let $\Delta^{\text{O-QLB}}$ be the optimal value to the original O-QLB problem, and let (UB, LB) be a pair of bounds such that $\text{UB} \geq \Delta^{\text{O-QLB}} \geq \text{LB}$. We define the discretization factor as $\alpha = \frac{2|N|-1}{\epsilon \text{LB}}$, given a small approximation factor ϵ. Let $\Delta^{\text{O-IQLB}}$ be the optimal value to the corresponding O-IQLB instance. We have the following:

Lemma 4.2. $\alpha \cdot \text{LB} \leq \Delta^{\text{O-IQLB}} \leq \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1.$ $\square$

Lemma 4.3. $\alpha \cdot \Delta^{\text{O-QLB}} \leq \Delta^{\text{O-IQLB}} \leq \alpha \cdot (1 + \epsilon) \cdot \Delta^{\text{O-QLB}}.$ $\square$

Proof of Lemmas 4.2 and 4.3. Let (Π^{sel}, ϕ) be an optimal solution to O-QLB. Since it is also feasible to O-IQLB, with Lemma 4.1, we have

$$\begin{aligned}\Delta^{\text{O-IQLB}} &\le \max\{d^{\alpha}(p) \,|\, \pi \in \Pi^{\text{sel}}, p \in \pi\} \\ &\le \alpha \cdot \max\{d(p) \,|\, \pi \in \Pi^{\text{sel}}, p \in \pi\} + 2|N| - 1 \\ &\le \alpha \cdot \Delta^{\text{O-QLB}} + 2|N| - 1 \\ &\le \alpha \cdot \text{UB} + 2|N| - 1.\end{aligned} \tag{4.6}$$

This implies the right inequality in Lemma 4.2, as $\Delta^{\text{O-IQLB}}$ is always an integer due to the discretization. From the third inequality in Eq. (4.6), we have

$$\begin{aligned}\Delta^{\text{O-IQLB}} &\le \alpha \cdot (\Delta^{\text{O-QLB}} + \frac{2|N| - 1}{\alpha}) \\ &= \alpha \cdot \big(\Delta^{\text{O-QLB}} + \epsilon \cdot \text{LB}\big) \\ &\le \alpha \cdot (1 + \epsilon) \cdot \Delta^{\text{O-QLB}}\end{aligned} \tag{4.7}$$

Based on the left inequality of Lemma 4.1, we have the left inequality in Lemma 4.3, which implies the left inequality in Lemma 4.2. This proves Lemmas 4.2 and 4.3. □

We now talk about the implications of Lemmas 4.2 and 4.3. Lemma 4.3 states that $\Delta^{\text{O-IQLB}}$ divided by α provides a $(1 + \epsilon)$-approximation to the value $\Delta^{\text{O-QLB}}$. Lemma 4.2 further provides a method to compute the value $\Delta^{\text{O-IQLB}}$, given a pair of bounds (UB, LB) on $\Delta^{\text{O-QLB}}$. A bisection method can be used to search the space $[\alpha \cdot \text{LB}, \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1]$ for the delay value $\Delta^{\text{O-IQLB}}$, each time solving an instance of IQLB by Program (4.5). The entire search requires $O(\log \frac{\lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1}{\alpha \cdot \text{LB}}) = O(\log \frac{|N|\text{UB}}{\epsilon \text{LB}})$ runs, each with time complexity of $O(|L|^3 (\frac{|N|\text{UB}}{\epsilon \text{LB}})^4 \mathbb{L})$. To summarize, Lemmas 4.2 and 4.3 give us a method for computing a $(1+\epsilon)$-approximation of the optimal solution to O-QLB within time polynomial to the input size, the parameter $1/\epsilon$, and the ratio $\frac{\text{UB}}{\text{LB}}$ between the upper and lower bounds.

To establish a polynomial-time algorithm, our next step is to find a pair of bounds $(\mathrm{UB}, \mathrm{LB})$ that is within a polynomial factor of each other. This can be done by finding a *bottleneck delay* that "determines" the feasibility of the problem instance. Let Γ_{-d} be the sub-graph of Γ where all nodes and links with delays larger than d are pruned. We wish to find the minimum value d_{bot} such that $\Gamma_{-d_{\text{bot}}}$ still admits a feasible BLB solution satisfying the load bound Ψ. Note that a source node n having delay $d_n > d$ directly eliminates the existence of a feasible solution in Γ_{-d}. Naturally, if the original O-QLB instance is feasible, which can be checked by solving Program (4.4) on the original Γ, then d_{bot} exists. Since there are at most $|N| + |L|$ distinct delay values, d_{bot} can be found also using bisection in $O(\log(|N| + |L|))$ iterations, each spending $O(|L|^3 \cdot \mathbb{L})$ time solving Program (4.4). We then have the following:

Lemma 4.4. $(2|N| - 1) \cdot d_{\text{bot}} \geq \Delta^{\text{O-QLB}} \geq d_{\text{bot}}$. □

Proof. By the definition of d_{bot}, it follows that the QLB instance is feasible on $\Gamma_{-d_{\text{bot}}}$ but is infeasible on $\Gamma_{-d_{\text{bot}}}$ with all nodes/links with delay equal to d_{bot} removed. This means that any feasible solution to QLB includes at least one node/link with delay no less than d_{bot}, which means d_{bot} is a lower bound of $\Delta^{\text{O-QLB}}$. On the other hand, since there is a feasible solution in $\Gamma_{-d_{\text{bot}}}$, and any path in $\Gamma_{-d_{\text{bot}}}$ has at most $|N|$ nodes and $(|N| - 1)$ links, the longest path delay in the feasible solution cannot exceed $(2|N| - 1) \cdot d_{\text{bot}}$. Hence $(2|N| - 1) \cdot d_{\text{bot}}$ is an upper bound of $\Delta^{\text{O-QLB}}$. This completes the proof. □

Lemma 4.4 provides us a pair of bounds $\mathrm{UB} = (2|N| - 1) \cdot d_{\text{bot}}$ and $\mathrm{LB} = d_{\text{bot}}$ with $\mathrm{UB}/\mathrm{LB} = 2|N| - 1$. Putting together the bounds with Lemmas 4.2 and 4.3, we now have an approximation scheme for O-QLB, shown in Algorithm 4.1.

Algorithm 4.1: FPTAS-O-QLB(G, Γ, Ψ)

```
1  Solve Program (4.4) on $\Gamma$ to test feasibility;
2  Ascendingly sort all delay values as $\mathbf{d} = (d_0, \ldots, d_K)$;
3  $lo \leftarrow 0, hi \leftarrow K$;
4  while $lo < hi - 1$ do
5      $mi \leftarrow \lfloor (hi + lo)/2 \rfloor$;
6      Construct sub-graph $\Gamma_{-d_{mi}}$;
7      Solve Program (4.4) on $\Gamma_{-d_{mi}}$ for optimal load $\psi$;
8      if $\psi > \Psi$ then $lo \leftarrow mi$;
9      else $hi \leftarrow mi$;
10 end
11 $d_{\text{bot}} \leftarrow d_{hi}, \text{LB} \leftarrow d_{\text{bot}}, \text{UB} \leftarrow (2|N| - 1)d_{\text{bot}}$;
12 Discretize delays with $\alpha = (2|N| - 1)/\epsilon\text{LB}$;
13 $D_{lo} \leftarrow \lfloor \alpha \cdot \text{LB} \rfloor, D_{hi} \leftarrow \lfloor \alpha \cdot \text{UB} \rfloor + 2|N| - 1$;
14 while $D_{lo} < D_{hi} - 1$ do
15     $D_{mi} \leftarrow \lfloor (D_{lo} + D_{hi})/2 \rfloor$;
16     Construct auxiliary inf-graph $\Gamma^{D_{mi}}$;
17     Solve Program (4.5) on $\Gamma^{D_{mi}}$ for optimal load $\psi$;
18     if $\psi > \Psi$ then $D_{lo} \leftarrow D_{mi}$;
19     else $D_{hi} \leftarrow D_{mi}$;
20 end
21 Do real-graph decomposition on the solution for $\Gamma^{D_{hi}}$;
22 return $(\Pi^{\text{sel}}, \phi)$ obtained by decomposition.
```

Lemma 4.5. *Given any $\epsilon > 0$, Algorithm 4.1 outputs a $(1 + \epsilon)$-approximation of the optimal O-QLB solution, within $O(\frac{1}{\epsilon^4}|L|^3|N|^8\mathbb{L}\log\frac{|N|}{\epsilon} + |L|^3\mathbb{L}\log|N|)$ time.* □

Proof. The approximation ratio is due to Lemma 4.3. The dominant time complexity comes from the two bisection searches along with an LP solving per search iteration. The first search takes $O(\log(|N|+|L|))$ iterations each with $O(|L|^3 \cdot \mathbb{L})$ time complexity. The second search takes $O(\log\frac{|N|}{\epsilon})$ iterations each with $O(\frac{1}{\epsilon^4}|L|^3|N|^8\mathbb{L})$ time complexity. Since $|L| < |N|^2$, we have the final complexity by adding them together. □

4.5.3 Efficiency Enhancement

While Algorithm 4.1 runs in time polynomial to input size and $1/\epsilon$, it has a time complexity as high as $O(|N|^{12}\log|N|)$ assuming $|L| = \Omega(|N|)$ and ϵ is a constant. Observe that the high complexity mainly comes from solving LPs with $D|L|$ variables and input size $D\mathbb{L}$, where $D = D_{mi}$ is in the order of $\frac{|N|\mathrm{UB}}{\epsilon\mathrm{LB}}$ and UB/LB is in the order of $|N|$. If we can reduce UB/LB to a constant, a reduction of order $|N|$ can be achieved on the program size, resulting in orders of reduction in the overall time complexity. We use a technique called *approximate testing* to achieve such a reduction [85].

Specifically, we define a test procedure $\mathrm{TEST}_\omega(\mathcal{D})$. Given $\omega > 0$ and $\mathcal{D} > 0$, we define a new discretization factor $\alpha = \frac{2|N|-1}{\omega\mathcal{D}}$ and discretize all delay values in Γ. We then define an instance of IQLB as (G, Γ, Ψ, D') where $D' = \lfloor\frac{2|N|-1}{\omega}\rfloor + 2|N| - 1$ and all delay values in Γ are discretized by α. Let $\mathrm{TEST}_\omega(\mathcal{D}) = \mathrm{True}$ if the discretized IQLB instance (G, Γ, Ψ, D') has a feasible solution, and $\mathrm{TEST}_\omega(\mathcal{D}) = \mathrm{False}$ otherwise. We then have the following lemma:

Lemma 4.6. *Given any $\omega > 0$ and $\mathcal{D} > 0$, we have*

$$\begin{aligned} \mathrm{TEST}_\omega(\mathcal{D}) = \mathrm{True} \quad &\Rightarrow \quad \Delta^{\text{O-QLB}} \le (1+\omega)\cdot\mathcal{D}; \\ \mathrm{TEST}_\omega(\mathcal{D}) = \mathrm{False} \quad &\Rightarrow \quad \Delta^{\text{O-QLB}} > \mathcal{D}. \end{aligned} \qquad \square$$

Proof. Assume $\mathrm{TEST}_\omega(\mathcal{D}) = \mathrm{True}$, then we have a feasible solution $(\Pi^{\mathrm{sel}}, \phi)$ for discretized $\mathrm{IQLB}(G, \Gamma, \Psi, D')$, which also translates to a feasible solution for O-QLB(G, Γ, Ψ). Let p be any processing path in any $\pi \in \Pi^{\mathrm{sel}}$, we have $d^\alpha(p) \le D'$.

Based on Lemma 4.1, we then have

$$\begin{aligned} d(p) &\leq d^{\alpha}(p)/\alpha \leq D'/\alpha \\ &\leq \left(\frac{2|N|-1}{\omega} + 2|N|-1\right) \cdot \frac{\mathcal{D}\cdot\omega}{2|N|-1} \\ &= \mathcal{D}(1+\omega) \end{aligned} \tag{4.8}$$

Since $\Delta^{\text{O-QLB}} \leq \max_{\pi\in\Pi^{\text{sel}}, p\in\pi}\{d(p)\}$ (as the solution is feasible to O-QLB), we have $\Delta^{\text{O-QLB}} \leq (1+\omega)\cdot\mathcal{D}$.

To prove the second statement, we can prove its contraposition, *i.e.*, if $\Delta^{\text{O-QLB}} \leq \mathcal{D}$ then $\text{TEST}_{\omega}(\mathcal{D})$ must output True. Let π be any real-graph in the optimal O-QLB solution, and let p be any path in π. Based on Lemma 4.1, we then have

$$\begin{aligned} d^{\alpha}(p) &\leq \alpha\cdot d(p) + 2|N|-1 \\ &\leq \frac{2|N|-1}{\omega\mathcal{D}}\Delta^{\text{O-QLB}} + 2|N|-1 \\ &\leq \frac{2|N|-1}{\omega} + 2|N|-1. \end{aligned} \tag{4.9}$$

This implies that $d^{\alpha}(p) \leq \lfloor\frac{2|N|-1}{\omega}\rfloor + 2|N|-1$ since again the delays are discretized to be integers. Therefore the optimal solution to O-QLB also translates to a feasible solution to the discretized IQLB instance, and hence $\text{TEST}_{\omega}(\mathcal{D})$ must output True based on Theorem 4.4. This completes the proof. □

Based on Lemma 4.6, we can use TEST_{ω} with $\omega = 1$ as a test procedure for carrying out a bisection search on the pair (UB, LB). Given an initial pair with $\text{UB}/\text{LB} = 2|N|-1$ (after Line 11 of Algorithm 4.1), we insert the procedure described in Algorithm 4.2, to obtain a pair of bounds within a constant factor of each other, before proceeding to Line 12 of Algorithm 4.1.

Theorem 4.5. *Given any $\epsilon > 0$, Algorithm 4.1 plus Algorithm 4.2 outputs a $(1+\epsilon)$-approximation for O-QLB within time $O(\frac{1}{\epsilon^4}|L|^3|N|^4\mathbb{L}\log\frac{|N|}{\epsilon} + |L|^3|N|^4\mathbb{L}\log\log|N|)$.*
□

Algorithm 4.2: RefineBound($G, \Gamma, \Psi, \text{LB}, \text{UB}$)

```
  // Between Lines 11 and 12 in Algo. 4.1
1 while $\text{UB} > 4 \cdot \text{LB}$ do
2     $\mathcal{D} \leftarrow \sqrt{\frac{\text{UB}\cdot\text{LB}}{2}}$;
3     if TEST$_1(\mathcal{D})$ = True then UB $\leftarrow 2 \cdot \mathcal{D}$;
4     else LB $\leftarrow \mathcal{D}$;
5 end
6 return (LB, UB).
```

Proof. First, observe that in Lines 3–4 of Algorithm 4.2, if $\text{TEST}_1(\mathcal{D})$ outputs True, then $2 \cdot \mathcal{D}$ is still a valid upper bound; if $\text{TEST}_1(\mathcal{D})$ outputs False, then $\mathcal{D}$ is still a valid lower bound; both due to Lemma 4.6. By the choice of $\mathcal{D}$, it satisfies that $\frac{\text{UB}}{\mathcal{D}} = \frac{2\cdot\mathcal{D}}{\text{LB}} = \sqrt{\frac{2\text{UB}}{\text{LB}}}$ in each iteration. Let $\beta = \frac{\text{UB}}{\text{LB}}$, and let $\beta[i]$ be the value of β after the i-th iteration. It is clear that $\beta[i] = \sqrt{2\beta[i-1]} = \cdots = 2^{1/2+1/4+\cdots+1/2^i}\beta[0]^{1/2^i} \leq 2 \cdot \beta[0]^{1/2^i}$. Therefore Algorithm 4.2 needs $O(\log\log\beta[0]) = O(\log\log|N|)$ iterations, each solving an IQLB instance with $D' = \lfloor\frac{|N|}{1}\rfloor + |N| = 2|N|$. It follows that Algorithm 4.2 has a time complexity of $O(|N|^4|L|^3\mathbb{L}\log\log|N|)$. Since we reduce UB/LB to be within 4 after Algorithm 4.2, the complexity of the second bisection search in Algorithm 4.1 now becomes $O(\frac{1}{\epsilon^4}|L|^3|N|^4\mathbb{L}\log\frac{|N|}{\epsilon})$. Combining these with the time complexity of the first bisection search in Algorithm 4.1, we have the claimed time complexity. □

4.6 Performance Evaluation

In this section, we present performance evaluation of our proposed algorithm with simulation experiments. Our algorithm is denoted as QLB. We implemented two heuristic algorithms. In the first algorithm denoted as BLB, the BLB formulation in Program (4.4) is solved, followed by a simple decomposition as in Theorem 4.2. In the second algorithm denoted as QHU (QoS-aware Heuristic), we solve a modified version of Program (4.4) where the objective is changed to minimizing the demand-weighted sum of delays of all nodes and links, meanwhile the load factor ψ has to be bounded by Ψ; we then use the same decomposition method. Therefore the first heuristic corresponds to load balancing that neglects QoS at all, and the second corresponds to a heuristic formulation where the total delay instead of the maximum delay of the solution is minimized in the modified LP.

We used randomly generated app-graphs and inf-graphs. The app-graph was organized as a random 5-layer DAG, with the lowest layer having 10% of the vertices and the other layers sharing the rest. Edges were generated from lower layers to all upper layers, with each vertex in the upper four layers having 4 in-coming edges on average. Distribution ratios of edges were also randomly generated, such that all out-going edges of a vertex had their ratios sum to 1. The inf-graph was generated such that each lowest-layer microservice had only one instance, as a source node, while each microservice in other layers had $[5, 15]$ instances. Links were built between instances of interconnected microservices with a base probability of 0.3, while additional links

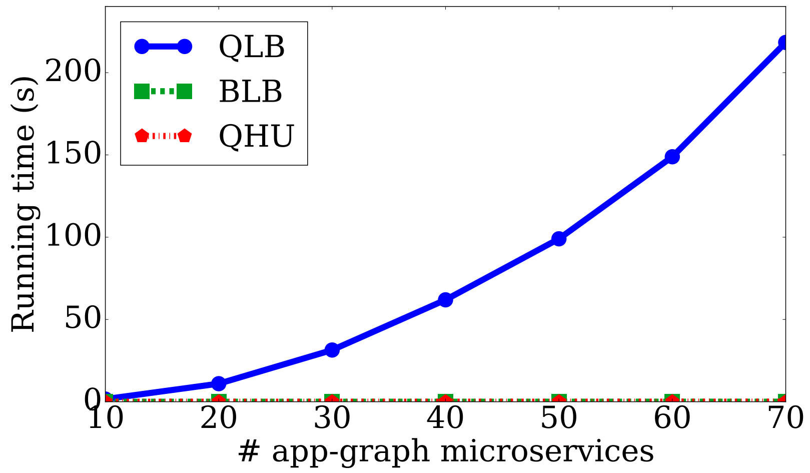


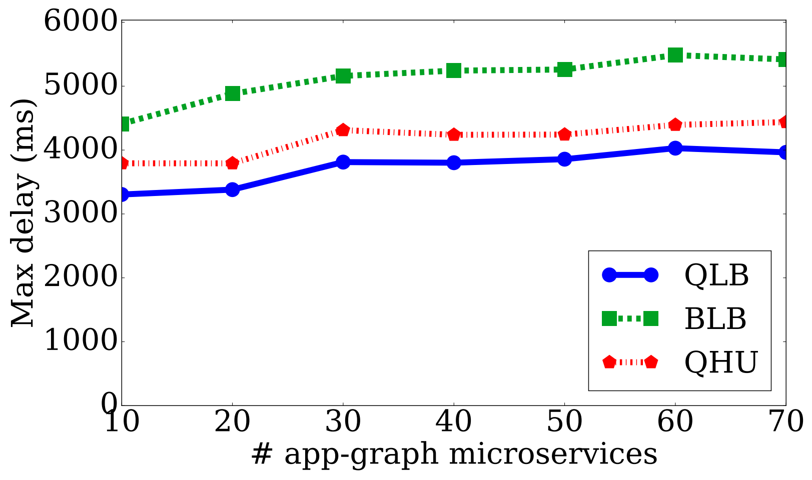
Figure 4.3: Running time vs. app-graph size

were added to ensure that each instance could find at least one out-going neighbor for every successor microservice. Each source node had a random demand in $[100, 900]$ and a large capacity such that it would not be a load balancing bottleneck. For all other nodes, capacities were generated in $[10, 90]$. Node and link delays were generated in $[0, 1000]$ ms and $[0, 500]$ ms respectively. Finally, we assigned the load bound Ψ to be one of the two values: it was either the optimal value ψ^* to BLB (Program (4.4)) or $2 \cdot \psi^*$. We used them to test the delay performance in heavy- or moderate-load scenarios respectively.

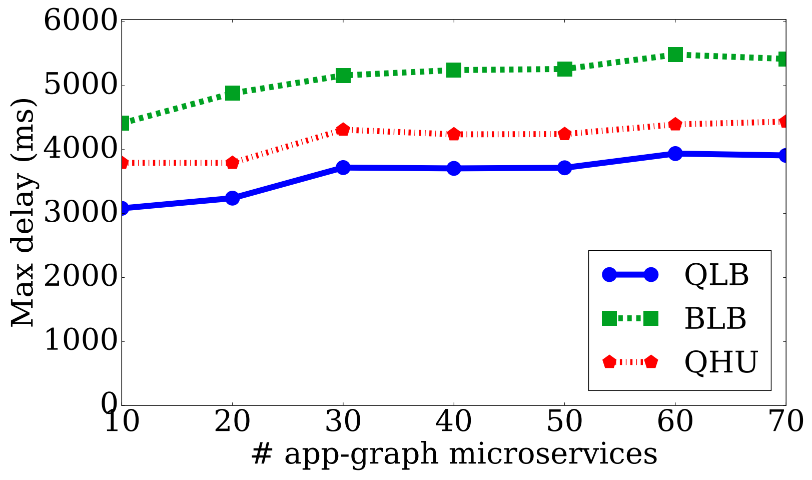
We set the accuracy parameter $\epsilon = 0.5$. Experiments were run on a Ubuntu PC with i7-2600 CPU and 16GB memory. To average-out random noise, we conducted 20 random experiments under each setting and took their average.

Fig. 4.3 shows the running time of the implemented algorithms with growth in the app-graph size. The running time of QLB is longer than the heuristics as expected. The growth in the running time is nevertheless polynomial to the growth in app-graph size (and hence the inf-graph size).

Fig. 4.4 shows the comparison of the delay values achieved by all three algorithms. We show the comparisons under two settings: heavy load ($\Psi = \psi^*$) and moderate load ($\Psi = 2\psi^*$). We note that QLB achieves the lowest delay in all scenarios, regardless of load. The QoS-aware heuristic QHU can improve delay performance over QoS-agnostic load balancing BLB, but cannot achieve an improvement as significant as QLB. Comparing Figs. 4.4(a) and 4.4(b), QLB has a larger advantage in terms of delay when the load is less (Fig. 4.4(b)). This is partly because when the load is moderate, QLB has more room for selecting nodes and links with lower delays, and thus it can further improve QoS. On the other hand, the results for BLB and QHU are almost the same with different loads, since their formulations commonly generate the same



(a) End-to-end delay with $\Psi = \psi^*$



(b) End-to-end delay with $\Psi = 2\psi^*$

Figure 4.4: Objective value vs. app-graph size

solutions. Differences may result from the decomposition process, which, as shown in the results, has little impact on the QoS of the final solutions.

To summarize, our algorithm always achieves advantageous QoS performance over the heuristics, with a polynomially bounded complexity. This supports our theoretical analysis that QLB has guaranteed performance while both heuristics do not.

4.7 Conclusions

In this study, we studied basic and QoS-aware load balancing across interdependent IoT microservices. A DAG-based model was used to abstract microservice interdependencies. We proposed an LP formulation for the basic problem, which can be solved in polynomial time. For the QoS-aware problem, we proposed a decomposition-based model where a realization of the application is expressed as a realization graph. Since the QoS-aware problem is NP-hard, we proposed an FPTAS, along with an efficiency enhancement technique that achieves several orders of speed-up. Simulations showed that our algorithm achieves enhanced QoS compared to heuristic solutions. We believe that the proposed method, aside from making theoretical contribution to the problem studied, more importantly provides insight in extending chain- or star-based application models to the more general DAG-based model, which is much more expressive and flexible in real-world scenarios.

4.8 Appendix

Proof of Theorem 4.3. We derive a reduction from the Partition problem to QLB, the former of which is a well-known NP-hard problem [37]. Given a set of objects $X = \{x_1, \ldots, x_\kappa\}$ and a positive integer a_x for $\forall x \in X$, the Partition problem seeks for a subset $Y \subset X$ such that $\sum_{x \in Y} a_x = \sum_{x \in X \setminus Y} a_x$. Given an instance X of Partition, we construct an instance (G, Γ, Ψ, D) of QLB as follows. $G = (V, E)$ has the vertex set $V = \{v_0, u_1, v_1, u_2, v_2, \ldots, u_\kappa, v_\kappa\}$, and edge set $E = \{(v_{i-1}, u_i), (u_i, v_i) \,|\, i = 1, \ldots, \kappa\}$. $\Gamma = (N, L)$ has the node set $N = \{n_i \,|\, \forall v_i \in V\} \cup \{m_i^0, m_i^1 \,|\, \forall u_i \in V\}$, and link set $L = \{(n_{i-1}, m_i^0), (n_{i-1}, m_i^1), (m_i^0, n_i), (m_i^1, n_i) \,|\, i = 1, \ldots, \kappa\}$. In summary, the app-graph is a line graph with $(2\kappa+1)$ vertices and 2κ edges, and the inf-graph has $(3\kappa+1)$ nodes and 4κ links. For the attributes, all edges in E have distribution ratio of 1. Node n_0 has external demand of 2, while all other nodes have no external demand. All n_i nodes have capacity of 2, while all m_i^j nodes have capacity of 1. All links, all n_i nodes and all m_i^0 nodes have delay of 0, while each m_i^1 node has delay $d_{m_i^1} = a_{x_i}$, for $i = 1, \ldots, \kappa$. Finally, we set the application delay bound $D = \frac{1}{2}\sum_{i=1}^{\kappa} a_{x_i}$, and the load bound $\Psi = 1$. An example is shown in Fig. 4.5.

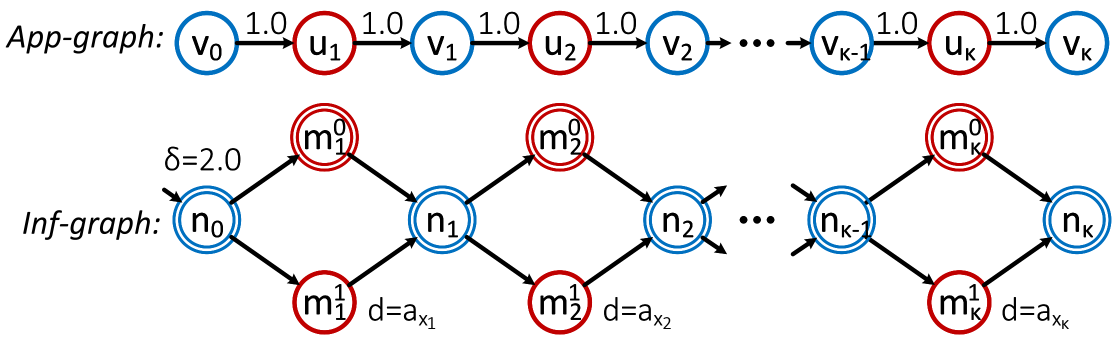


Figure 4.5: App-graph values are distribution ratios. Delays of m_i^1 nodes are shown beside them. Double lined nodes and all links have 0 delay. All n_i nodes have capacity of 2. All m_i^j nodes have capacity of 1. $\Psi = 1$. $D = (1/2)\sum_{i=1}^{\kappa} a_{x_i}$.

Since the app-graph G is a line graph, every real-graph of n_0 is also a line graph from n_0 to n_κ, and hence its delay is precisely the sum of delays of m_i^1 nodes included in the path. Now, suppose instance X of Partition has feasible solution Y. Then instance (G, Γ, Ψ, D) of QLB also has a feasible solution, which has two paths, one taking m_i^1 for $x_i \in Y$ and m_i^0 for $x_i \in X \setminus Y$, and the other taking the opposite m_i^j nodes, both having a demand allocation of 1. On the reverse direction, suppose instance (G, Γ, Ψ, D) has a feasible solution. The solution must also contain two paths since each m_i^j node can only accept half of the demands coming from the predecessor n_{i-1} node. A feasible solution to Partition instance X is then constructed by picking x_i

corresponding to all m_i^1 nodes taken by one of the paths. This proves the NP-hardness of QLB, and the NP-hardness of O-QLB follows. $\square$

Part II

Robust Security Deployment in IoT

Chapter 5

DEPLOYING ROBUST SECURITY IN INTERNET OF THINGS

5.1 Introduction

Despite its powerfulness and popularity, the current IoT is facing many challenges, among which a major one is the emerging concern on IoT security and privacy. In regard to security, the massive number of connected smart devices in IoT, which is indeed its biggest strength, is also a huge potential threat. On one hand, providing fine-grained security to a large number of geo-distributed devices is non-trivial, especially given the limited resources on each of these devices. On the other hand, these massive devices, once compromised, can be used as a powerful weapon against the system itself, for example, by launching large-scale distributed denial-of-service (DDoS) attacks. In fact, the digital world has already witnessed the devastating effect of such attacks in several recent incidents [126]. Privacy becomes part of the concern when IoT-connected devices infiltrate various private spaces, including homes, factories, and hospitals.

Enforcing security and privacy in IoT is difficult. The main reason, as aforementioned, is the limited resources (computing resource, memory, battery) on connected devices. Due to this, IoT devices can hardly run conventional cryptographic algorithms. Currently, there are two directions that deal with this issue. The first direction is to develop lightweight yet strong-enough cryptographic algorithms for IoT devices. Unfortunately, current advances along this direction is yet enough to conquer the overhead issue of cryptography in constrained environments [124]. The second direction is to offload security to the cloud [20], [161]. This is based on recent advances in network function virtualization (NFV), which can virtualize security mechanisms as software components to be run on general-purpose platforms [83]. However, cloud-based offloading has several drawbacks. First, cloud-based security leaves data transmission as a major vulnerability. In other words, the traffic from IoT devices to the cloud is unprotected, leaving room for data interception, manipulation and injection attacks, especially when the traffic needs to traverse the Internet before reaching the cloud. Second, cloud-based security cannot prevent saturation attacks at the early stage, such as DDoS attacks from IoT devices. In fact, a DDoS attack may even invalidate the cloud by saturating its bandwidth, rendering the whole security system unusable. Third, the cloud suffers from high latency which increases the probability of device-oriented oppotunistic attacks, for instance, the recently revealed Meltdown and Spectre hardware attacks [54] that can affect an extremely wide spectrum of IoT devices. The cloud is not capable of responding in real-time to incidents on IoT devices. Last but not least, the cloud becomes a single point of failure in the system. An attack compromising the cloud itself can have control over all the devices, including those in private spaces like homes or factories.

Powered by fog computing, a new approach to IoT security emerges. Fog computing offers in-network computing hardware in the edge network, which enables edge offloading of security mechanisms near the end devices. Hence it can largely realize fine-grained and real-time security, while avoiding high latency, high bandwidth usage and a single point of failure. That being said, fog-based security also has its limitations. First of all, the cost of using fog computing is generally higher than that of using the cloud. This is because fog nodes are commonly deployed in areas that are already dense with other devices, and hence will have higher deployment, operation and energy costs. Due to this, fog resources are still limited in each area, though much more abundant than IoT devices. Second, fog-based security still leaves some vulnerabilities in the early stage of data transmission. The unprotected and/or unmonitored traffic can cause various threats to the devices and the system, as we detail in Section 5.3.2.

An ideal architecture for IoT security should jointly make use of end device-, cloud- and fog-based security. Among these, fog-based security has the highest flexibility in the trade-off between security and cost, and hence should be carefully optimized by the IoT operator. In this work we focus on modeling and formulating the security offloading problem from the perspective of fog computing. Given a limited cost budget, the operator would want to deploy security functions on distributed fog nodes, in order to minimize the security risk experienced by the end devices. We mathematically formulate the security risk of the system in terms of the distance from each device to the nearest deployed security function, which is general enough to incorporate a wide range of risk measures in practice; see Section 5.3.2. A major challenge in the deployment problem is the dynamic nature of IoT, where both infrastructure and demand fluctuations could happen frequently, such as device mobility, maintenance, interference, failures, etc. To address this challenge, we propose a stochastic model to capture the uncertainties caused by dynamics. Leveraging a relevant concept in economics, our model can account for both the expected and the worst-case risks of the system, which is more robust than traditional stochastic models solely based on expectations. We then propose a novel decomposition-based framework, along with an efficiency-enhancement technique, to achieve accurate and efficient optimization of system security risk in the dynamic IoT environment. We evaluated our proposed model and optimization framework in extensive simulation experiments, and the results have shown the superb performance and efficiency of our proposed approach.

To summarize, our main contributions are as follows:

- To the best of our knowledge, we are the first to study, quantitatively model, and optimize the security risk in fog computing-based IoT security offloading. The problem is of both theoretical and practical importance.
- We propose a stochastic model to account for both the expected and the worst-case security risks of the IoT system with uncertainties, which is more robust than traditional stochastic models solely based on expectations, and is more suitable for security-related use cases.
- We propose a decomposition-based optimization framework, along with an efficiency-enhancement method that addresses the large overhead of stochastic programming.
- We conducted extensive simulations which show the superb performance of our approach compared to several other approaches.

5.2 Backbround and Related Work

5.2.1 IoT Security Challenges and Approaches

IoT security has yet attracted a lot of attentions, at least not until several recent attacks based on IoT devices [126]. It has then been recognized that security breaches in IoT can be as dominating and devastating as, if not more so than, any other known type of security incidents. Since then, more efforts have been put into addressing security challenges in IoT, which can be viewed roughly in two perspectives.

The first perspective is on protecting the IoT devices. The main challenge here is the limited resources on IoT devices, including computing power, memory, power supply, etc. Currently, there are two approaches to address the resource issue. The first one is lightweight cryptography and security, which aims to develop mechanisms that can provide good-enough security on resource-constrained devices, such as [61], [82]. Unfortunately, the current lightweight methods still can only be deployed on devices like smartphones or tablets, but can hardly be used on smaller devices like radio frequency identification (RFID) tags, largely due to the stringent power constraint on these devices [124]. Another approach is to offload security mechanisms to other platforms, including in-network fog nodes or the cloud [20], [66], [74], [135]. This approach addresses the overhead issue, but inevitably introduces some risk associated with the transmission to the offloaded security mechanisms, as detailed in the next section.

The second perspective is to protect the broader IoT system rather than only the end devices. An IoT system includes the end devices, network infrastructures in different levels, applications and their hosting fog or cloud nodes, and users of the IoT services. There are several challenges in this perspective, including the huge number of IoT devices, the heterogeneous and dynamic network environment, various requirements and characteristics of applications, etc. The *de facto* approach here is cloud-based security [111], which again has several issues. The cloud is generally far away from the IoT edge network, and hence is difficult to adapt to the fast changing environment of IoT; it is also vulnerable to saturation attacks like DDoS [119]. NFV [115] is a promising approach to addressing this issue, by enabling implementation of security mechanisms as in-network software components to resolve threats before they reach the cloud, other devices or the users.

It can be seen that both perspectives call for deployment of security mechanisms in the network. In both cases, security can be largely guaranteed when traffic has passed certain security functions, while the part of transmission before is not protected/monitored. Therefore, what we address in this study is to minimize the risk associated with the transmission before the security functions, by deploying these functions at optimal locations. A recent survey on IoT security is in [156].

5.2.2 Risk Management in Network Security

Traditionally, network security is mainly a 0–1 problem: a system is either secure or insecure. Recently, however, there has been a transition to providing best-effort network security, due to the enormous number and variety of attacks and the impractical amount of resources to prevent them all. The goal is either to make the difficulty or cost of launching an attack unacceptable to the attackers, or to make the probability or potential loss of undergoing an attack acceptable to the system.

Our work follows a number of existing works along the second direction. A major series of works have focused on modeling and optimizing network security risk using Attack Graphs (AG) [4], [95], [113]. An AG is a graph representation of all the possible attack paths into the system, and is used to derive various security risk measures via Bayesian theory [95], node ranking [113], or other mathematical tools [4]. The derived measures are then used to guide the deployment of security functions to harden the system [29], [91]. A main drawback of the AG representation is its scalability. The size of the AG can grow exponentially with the size of the network, and is further increased if each node have multiple vulnerabilities. Hence it is most suitable for simple and small-size environments like home networks, but is hardly applicable to IoT networks with even a few hundred devices. Furthermore, AG cannot handle fast dynamics in the network: it needs to be modified each time a new device joins or an old device leaves.

In face of these issues, some researchers have been seeking for simpler and more practical security measures for large-scale and dynamic environments like IoT. Rullo, Serra, Bertino, and Lobo [110] have proposed a novel model for security risk when monitoring geo-distributed IoT devices in an area. They considered the robustness of the system when facing dynamic user densities, an approach similar to what we use to address demand and topology fluctuations in the IoT network. In this study, we propose a security risk measure based on distances in the network, as well as an efficient optimization framework to minimize the security risk by flexibly deploying security functions on fog nodes.

5.2.3 Other Related Areas

Mobile Offloading (MO): The idea of offloading dates back to mobile cloud computing, where users offload their mobile applications to the cloud [69]. Yet security offloading differs from MO in two ways. First, MO is mostly ad hoc where each user makes its own decisions; however, security offloading is centrally controlled by the operator, who optimally coordinates the offloading for all users. Second, MO does not consider the network, since the destination is commonly the cloud. Security offloading needs to consider the edge network topology, and to minimize the risk associated with the offloading decisions.

NFV and Service Function Chaining (SFC): SFC is arisen in the context of NFV. SFC considers the interplay between different network services or security functions, modeling them as a chain of virtual functions. Existing work has focused on embedding service chains for each traffic flow [13], [70], [109]. Unfortunately, this method is neither scalable for the massive IoT devices, nor robust to the high dynamics. A good method should account for both the expected scenarios, and possible unfavorable scenarios due to system fluctuations.

Security with Robustness: Our work uses a stochastic approach to ensure system robustness. Similar methods have also been used in other security scenarios [110], [134]. Another potential approach is robust optimization, which deterministically optimizes the worst possible performance of the system, such as [17]. This approach has two drawbacks. First, it is hard to represent all unfavorable scenarios deterministically. Second, it only plans for the worst possible performance of the solution, which is commonly an overkill and wastes precious resources. A good approach should be able to balance the expected and worst-case performances, and give operator the flexibility to specify the desired objective.

5.3 Problem Description and Formulation

5.3.1 System Model

The IoT network is modeled as a directed graph $G = (V, E)$. V is the set of network nodes, including access points (APs), switches, routers, gateways, and various fog nodes. We use $F \subseteq V$ to denote the set of fog nodes, each being able to host security functions. E is the set of links between nodes.

The IoT faces two types of uncertainties. The first one is demand uncertainty, which comes from dynamics such as device deployment, maintenance, user load variation, device and user mobility, etc. Let $A \subseteq V$ be the set of APs of the IoT. To model demand uncertainty, each AP $a \in A$ is associated with a random variable $d_a \in \mathbb{R}^*$ ($\mathbb{R}^*$ is the non-negative real number set), denoting the amount of demand at a. We use $D = \{d_a \,|\, a \in A\}$ to denote the demand uncertainty set, and $d_{\text{sum}} = \sum_{a \in A} d_a$ to denote the total demand in the IoT. In practice, the demand can refer to (but is not limited to) the number of devices, number of flows, traffic volumes, etc.

The second type is topology uncertainty due to unexpected failures, maintenance, interference, etc. To model it, we associate each link $e \in E$ also with a random variable $y_e \in \{0, 1\}$, where $y_e = 1$ means e is operational, and $y_e = 0$ means e is down. This model is also able to account for node dynamics: if a node fails, all its adjacent links have $y_e = 0$. We use $Y = \{y_e \,|\, e \in E\}$ to denote the topology uncertainty set.

Let $\overline{D} = \{\overline{d}_a\}$ and $\overline{Y} = \{\overline{y}_e\}$ be a specific realization of the demand and topology uncertainty sets, respectively. We define a realization of the system (called a *scenario*) as $\Pi = (\overline{D}, \overline{Y})$.

5.3.2 Threat Model and Defense Mechanism

In an IoT network with security offloading, data is first transmitted from IoT device to the offloaded security function before heading to its final destination. Below, we analyze the threat associated with such transmission in several scenarios:

- *Highly constrained devices:* these devices cannot run any cryptographic procedure, hence offload all procedures to the network. All transmitted data is unprotected before passing the security functions, which is subject to leakage, manipulation, injection and other types of attacks.
- *Moderately constrained devices:* these devices can run lightweight cryptographic procedures such as secure computation outsourcing [20], [66], which only offloads

computation-intensive operations but keeps the data private. Yet in this case, the data stays unprotected in device memory until encryption is complete, during which it can be compromised by opportunistic attacks leveraging vulnerabilities of the devices themselves, such as the recent Meltdown and Spectre hardware attacks [54]. Probability of an attack is determined by the latency between device and the offloaded location, as the process can involve multiple rounds of back-and-force messaging [66].

- *IoT network:* the operator may deploy security functions (*e.g.*, intrusion detection, firewall, deep packet inspection) to protect the system from malicious/compromised devices. However, traffic not processed by certain security functions can be a threat to the network. For example, a DDoS attack may still overwhelm some of the nodes before being tackled by a detection and resolution function.

It can be seen that in these various scenarios, it is essential that the operator can flexibly deploy security functions to minimize the threats brought by the unprotected/unmonitored transmission of offloading. Unfortunately, the operator may have a tight budget for deploying the functions. We next define the operator's deployment plan to optimize such threats.

We consider the deployment of a single type of security function to prevent a specific type of attacks. Formally, we use binary variable set $X = \{x_v \mid v \in F\}$ to denote a *deployment plan*, where $x_v = 1$ means a security function is deployed at node v and 0 otherwise. Let $c_v \in \mathbb{R}^+$ be the deployment cost at node $v \in F$. A deployment plan is said to be *feasible*, iff it satisfies the operator's budget b:

$$\sum\nolimits_{v \in F} c_v x_v \leq b. \tag{5.1}$$

Definition 5.1 (**Security risk**). *Given network $G = (V, E)$, the* ***security risk*** *of a deployment plan X is the average distance that a unit of demand has to traverse before reaching the nearest security function from its AP, denoted as $R(X, D, Y)$. In other words, the security risk is a function of the deployment plan X and the uncertainty sets D and Y.* □

In Definition 5.1, the distance is a cumulative measure that can be selected based on different use cases. For instance, it can be as simple as the number of hops (vulnerable links or nodes) or transmission latency (time window of possible attacks), or as complex as the negative logarithm of the "safe" probability of traversed links or nodes (the "safe" probability is one minus the probability that a node/link/path is attacked; it is cumulated multiplicatively along the transmission path but is to be maximized instead of minimized). We do not assume a specific distance measure for sake of generality. We focus on the average distance of demands for ease of illustration, although our approach can be trivially extended to minimizing the maximum distance of demands. Note that the average distance is the average over demands from all APs in a fixed scenario, rather than over all possible scenarios of the system. For measurement over scenarios, we use both the expectation and a worst case-oriented metric, as detailed next.

5.3.3 Measuring Security Risk with Uncertainty

Given a fixed scenario Π, measuring the security risk of deployment plan X is as simple as finding shortest paths between AP-fog node pairs; minimizing it with flexible deployment plans, though, is NP-hard due to a reduction from the facility location problem [53]. Furthermore, the IoT environment is rather volatile, with uncertainties in both the demands and the topology. One approach is to measure and minimize the *expected risk*, which reflects the average level of threat of the system. However, such an approach is not robust enough when applied in security, as the system may experience arbitrarily high security risk in unfortunate scenarios.

To account for the worst-case performance, we adopt the concept of *Conditional Value-at-Risk* (CVaR), a risk measure widely used in economics and finance. The concept has been applied in several security-related use cases [110], [134]. Here "risk" refers to the investment risk an investor encounters when facing market variations, *i.e.*, potential loss due to unfavorable market trends. Formally, let random variable R be the loss of an investment. The following terms are defined:

$$\mathrm{VaR}_\alpha(R) = \min\{c \mid P(R \le c) \ge \alpha\}, \tag{5.2}$$

$$\mathrm{CVaR}_\alpha(R) = \mathbb{E}[R \mid R \ge \mathrm{VaR}_\alpha(R)]. \tag{5.3}$$

Here $\mathbb{E}[\cdot]$ is the expected value of a random variable. $\mathrm{VaR}_\alpha(R)$ (Value-at-Risk, also called α-VaR) is the minimum value such that the actual loss will not exceed it with α confidence, while $\mathrm{CVaR}_\alpha(R)$ (also called α-CVaR) is the expected loss of all scenarios where the loss can actually exceed $\mathrm{VaR}_\alpha(R)$. In other words, $\mathrm{CVaR}_\alpha(R)$ denotes the expected loss of the worst $(1-\alpha)$ percent scenarios in terms of investment loss.

Given uncertainty sets D and Y, $R(X, D, Y)$ is the security risk of deployment plan X, which is also a random variable. We use R to denote $R(X, D, Y)$ if no ambiguity is introduced. Next, we formally model and minimize the security risk of security offloading, utilizing the CVaR definition.

5.4 Security Deployment with Uncertainty

5.4.1 Problem Description and Formulation

Given the network and a limited budget, the IoT operator wants to ensure system security to the best extent. This includes both the overall system security in expectation, as well as the potential security risk in the $(1-\alpha)$ percent most unfavorable scenarios. We model these two goals as a multi-objective optimization problem as follows:

$$\min_{X \in \mathcal{X}} \quad \mathbb{E}[R],\ \mathrm{CVaR}_\alpha(R), \tag{5.4}$$

where $\mathcal{X}$ is the feasible deployment plan set satisfying Eq. (5.1).

Optimizing two objectives can be hard, as they may conflict with each other in their own optimality points respectively. A common technique is to *scalarize* the multiple objective functions into a single objective function. We scalarize Program (5.4) as the following single-objective program:

$$\min_{X \in \mathcal{X}} \quad \mathbb{E}[R] + \rho \cdot \mathrm{CVaR}_\alpha(R). \tag{5.5}$$

where ρ is a chosen balancing parameter.

In Eq. (5.3), the formulation of α-CVaR requires the computation of α-VaR beforehand, which is hard to incorporate in the above program. Fortunately, Rockafellar and Uryasev proved in [106] that the α-CVaR can be computed as follows without knowing the α-VaR beforehand:

$$\mathrm{CVaR}_\alpha(R) = \min_{c \in \mathbb{R}} \left\{ c + \frac{1}{1-\alpha} \mathbb{E}\left[(R-c)^+\right] \right\}, \tag{5.6}$$

where $\mathbb{R}$ is the real number set, and $(z)^+ = \max\{z, 0\}$. Therefore, Program (5.5) can be re-written, by incorporating Eq. (5.6), as the following program:

$$\min_{\substack{X\in\mathcal{X}\\ c\in\mathbb{R}}} \mathbb{E}[R] + \rho\left(c + \frac{1}{1-\alpha}\mathbb{E}\left[(R-c)^+\right]\right). \tag{5.7}$$

The random variable R is a function of the deployment plan X and random variable sets D and Y. Unfortunately, writing it as a closed-form equation is also a difficult task. Instead, we formulate it as the following program:

$R(X, D, Y) =$

$$\min_{t} \quad \frac{1}{d_{\text{sum}}}\sum_{a\in A} d_a \sum_{v\in F} \text{dist}_a(v) t_a(v) \tag{5.8a}$$

$$\text{s.t.} \quad \sum_{v} t_a(v) = 1, \quad \forall a; \tag{5.8b}$$

$$t_a(v) \le x_v, \qquad \forall a, v; \tag{5.8c}$$

$$t_a(v) \in [0,1], \quad \forall a, v. \tag{5.8d}$$

In Program (5.8), $\text{dist}_a(v)$ is a random variable, denoting the distance between AP a and node v. It is determined by the distance metric used (number of hops, latency, safe probability, etc.), as well as the topology uncertainty set Y. Objective (5.8a) is to minimize the demand-weighted average security risk (distance) of all APs. Constraint (5.8b) bounds the node selection variable $t_a(v)$. Constraint (5.8c) defines the relationship between the deployment variable x_v and the node selection variable $t_a(v)$. Note that $t_a(v)$, which can be viewed as the probability of selecting v for AP a, is a continuous variable in $[0, 1]$; its upper bound 1 is explicitly expressed in (5.8d) for clarity, but can be omitted due to Constraint (5.8b) when solving the program. If multiple nodes have the same minimum distance from a, the node selection can be arbitrarily split among them without affecting the objective value.

5.4.2 Scenario-based Stochastic Optimization

Program (5.7) is a stochastic optimization problem. Even with the characteristic functions of D and Y, the problem is still hard to solve since $R(X, D, Y)$ cannot be written in a closed form (and is not even convex as Y is discrete). A common approach is to approximate the expectations using sampling. Specifically, N sample scenarios are obtained from the underlying distributions of D and Y, denoted as $\mathbf{\Pi} = \{\Pi_1, \ldots, \Pi_N\}$. Let $\overline{R}_i \triangleq R(X, \overline{D}^i, \overline{Y}^i)$ be the security risk in scenario Π_i, which can still be expressed by Program (5.8) with the random variables replaced by the deterministic values in Π_i. The expected security risk $\mathbb{E}[R]$ is then approximated by the sample average function $\frac{1}{N}\sum_{i=1}^{N} \overline{R}_i$, while the α-CVaR is approximated by $\min\limits_{c\in\mathbb{R}} \left\{c + \frac{1}{1-\alpha}\frac{1}{N}\sum_{i=1}^{N}(\overline{R}_i - c)^+\right\}$.

Program (5.7) is then approximated by the following:

$$\min_{\substack{X\in\mathcal{X}\\ c\in\mathbb{R}}} \frac{1}{N}\sum_{i=1}^{N} \overline{R}_i + \rho\left(c + \frac{1}{1-\alpha}\frac{1}{N}\sum_{i=1}^{N}\left(\overline{R}_i - c\right)^+\right). \tag{5.9}$$

We next re-write Program (5.9) to resolve $(\cdot)^+$, $\overline{R}_i$ and $\mathcal{X}$. To resolve $(\cdot)^+$, we introduce additional variable $z_i \in \mathbb{R}^*$ for $i = 1 \ldots N$, and constrain the inner term of $(\cdot)^+$ using z_i. To resolve $\overline{R}_i$, note that both Program (5.8) and Program (5.9) are minimization programs, and hence can be merged. $\mathcal{X}$ is resolved by bringing Eq. (5.1) into the program. We then arrive at the following Mixed Integer Linear Program

(MILP):

$$
\begin{aligned}
\min \quad & \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\overline{d}^{i}_{\text{sum}}}\sum_{a\in A}\overline{d}^{i}_{a}\sum_{v\in F}\text{dist}^{i}_{a}(v)t^{i}_{a}(v)+ \\
& \qquad\qquad\qquad \rho\left(c+\frac{1}{1-\alpha}\frac{1}{N}\sum_{i=1}^{N}z_i\right) && (5.10a)\\
\text{s.t.} \quad & \frac{1}{\overline{d}^{i}_{\text{sum}}}\sum_{a\in A}\overline{d}^{i}_{a}\sum_{v\in F}\text{dist}^{i}_{a}(v)t^{i}_{a}(v)-c\le z_i, \quad \forall i; && (5.10b)\\
& \sum_{v} t^{i}_{a}(v)=1, \quad \forall i,a; && (5.10c)\\
& t^{i}_{a}(v)\le x_v, \qquad \forall i,a,v; && (5.10d)\\
& \sum_{v\in F} c_v x_v \le b; && (5.10e)\\
& x_v\in\{0,1\}, t^{i}_{a}(v)\in[0,1], z_i\ge 0, c\in\mathbb{R}, \ \forall i,a,v. && (5.10f)
\end{aligned}
$$

In Program (5.10), note that the random variable distances $\text{dist}_a(v)$ are also instantiated by the deterministic distances $\text{dist}^{i}_{a}(v)$, which can be computed beforehand for all scenarios. There may be scenarios where an AP cannot reach all the fog nodes due to disconnectivity. In this case, we set the distance to the disconnected nodes to a large value, in order to prefer security function deployment at other fog nodes instead.

Program (5.10) can be solved using optimization solvers such as Gurobi [48]. Yet, there are two reasons that Program (5.10) is extremely hard to solve in practice. First, the program is non-convex due to integer variables x_v. Second, the program size is linear to N. To get a good approximation of the distributions of D and Y, the number of samples needed is commonly at least thousands. An MILP of such size is largely unsolvable in practice. Therefore, we resort to some optimization techniques, which can drastically reduce the complexity of solving Program (5.10) to a practical level.

5.4.3 Two-stage Optimization with Benders' Decomposition

At a closer look, Program (5.10) can be viewed as a typical two-stage optimization problem, with a small number of master variables but a huge number of slave variables. In the first stage, the program seeks to fix deployment plan X, which is called the *master problem*. In the second stage, given fixed deployment plan, it then computes the security risk of all scenarios by selecting the optimal fog nodes that actually deploy the security functions; this stage is called the *slave problem*. A difficulty in solving a two-stage program is that when fixing the first-stage master solution, there is no clue on how such decisions will affect the second-stage slave solution, until the slave problem is actually solved. A natural choice is thus an iterative algorithm that progressively solves both the master and the slave, until an optimal solution is found. In this subsection, we apply a well-known algorithm of such kind: the *Benders' decomposition* due to Benders [9].

Formally, Program (5.10) can be re-written as follows:

$$\begin{aligned}
\min_{X,c} \quad & \rho \cdot c + Q(X,c) && \text{(5.11a)}\\
\text{s.t.} \quad & \text{(5.10e)}, \\
& x_v \in \{0,1\}, c \in \mathbb{R}, \qquad \forall v. && \text{(5.11b)}
\end{aligned}$$

where the slave problem $Q(X,c)$ is given by

$Q(X,c) =$

$$\begin{aligned}
\min_{z,t} \quad & \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{\overline{d}^i_{\text{sum}}}\sum_{a\in A}\overline{d}^i_a\sum_{v\in F}\text{dist}^i_a(v)t^i_a(v) + \frac{\rho}{1-\alpha}z_i\right) && \text{(5.12a)}\\
\text{s.t} \quad & \text{(5.10b)}, \text{(5.10c)}, \text{(5.10d)}, \\
& t^i_a(v) \in [0,1], z_i \geq 0, \quad \forall i,a,v. && \text{(5.12b)}
\end{aligned}$$

An intriguing property of the slave is that it is further decomposable regarding each scenario Π_i, as there are no coupling variables or constraints over i. Hence the slave problem can also be written as $Q(X,c) = \frac{1}{N}\sum_{i=1}^{N} Q_i(X,c)$, where $Q_i(X,c)$ is the slave subproblem for each scenario Π_i, defined by the inner term of the objective function and all the constraints for the specific i in Program (5.12).

To employ Benders' decomposition, we further need to study the dual program of $Q_i(X,c)$, defined as follows:

$\Delta_i(x,c) =$

$$\max_{\lambda,\phi,\mu} \quad \sum_{a\in A}\left(\phi_i(a) - \sum_{v\in F} x_v \mu_i(a,v)\right) - c\cdot\lambda_i \tag{5.13a}$$

$$\text{s.t.} \quad \lambda_i \le \frac{\rho}{1-\alpha}; \tag{5.13b}$$

$$\phi_i(a) - \mu_i(a,v) \le \frac{\overline{d}^{\,i}_a \mathrm{dist}^i_a(v)}{\overline{d}^{\,i}_{\mathrm{sum}}}(1+\lambda_i), \forall a,v; \tag{5.13c}$$

$$\lambda_i, \mu_i(a,v) \ge 0, \phi_i(a) \text{ unbounded}, \quad \forall a,v. \tag{5.13d}$$

In Program (5.13), dual variables λ, ϕ and μ correspond to primal constraints (5.10b), (5.10c) and (5.10d), respectively.

Given the above, the key idea of Benders' decomposition is to, instead of considering all constraints as a whole, progressively add constraints (also called *cuts*) that may help in approaching the optimal solution in an iterative manner, thus avoiding to consider the large number of constraints together. The algorithm starts with a feasible master solution (*e.g.*, with all $x_v = 0$ and $c = 0$ in our problem), a pair of lower and upper bounds LB $= -\infty$ and UB $= \infty$, and the master problem that contains only the master constraints (in our case, Constraint (5.10e) is the only master constraint). In each iteration, the algorithm first solves the dual slave problem given the current master solution. It then updates the master problem by adding cuts based on the dual slave problem solution. The updated master problem is then solved to optimality to obtain a new master solution. Note that although the master problem is still an MILP, it has a much smaller size compared to the original, and hence can be efficiently solved using a standard solver. The LB and UB are updated based on the master and slave solutions, respectively. The whole algorithm is shown in Algorithm 5.1.

Algorithm 5.1: Benders' Decomposition for Program (5.10)

```
   Input: Network $G$, scenarios $\mathbf{\Pi}$, quantile $\alpha$, tolerance $\epsilon$
   Output: Deployment plan $X$
1  $X \leftarrow \{x_v = 0 | v \in F\}$, $c \leftarrow 0$, LB $\leftarrow -\infty$, UB $\leftarrow \infty$;
2  while $UB - LB > \epsilon$ do
3      Solve dual slave problem $\Delta_i(x, c)$ for $\forall \Pi_i$;
4      if $\Delta_i(x, c)$ is unbounded then
5          Get unbounded ray $(\tilde{\lambda}, \tilde{\phi}, \tilde{\mu})$;
6          Add feasibility cut to the master problem:
           $\sum_{i=1}^{N} \left( \sum_{a \in A} \left( \tilde{\phi}_i(a) - \sum_{v \in F} x_v \tilde{\mu}_i(a, v) \right) - c\tilde{\lambda}_i \right) \leq 0$;
7      else
8          Get optimal point $(\lambda^*, \phi^*, \mu^*)$;
9          UB $\leftarrow \min\{\text{UB}, \Delta(x, c)\}$;
10         Add optimality cut to the master problem:
           $\sigma \geq \sum_{i=1}^{N} \left( \sum_{a \in A} \left( \phi_i^*(a) - \sum_{v \in F} x_v \mu_i^*(a, v) \right) - c\lambda_i^* \right)$;
11     end
12     Solve master problem $\min\{\sigma + \rho \cdot c \,|\, \text{cuts}, x \in \mathcal{X}\}$;
13     LB $\leftarrow \sigma + \rho \cdot c$;
14 end
15 return $X$.
```

In the algorithm, an unbounded ray $(\tilde{\lambda}, \tilde{\phi}, \tilde{\mu})$ in Line 5 is essentially a direction (in other words, a solution vector) to which the dual objective value goes to infinity. If the dual slave problem is unbounded, the primal slave problem is infeasible, and hence a *feasibility cut* is added in Line 6 to drive the master problem back into the feasible domain. If the dual slave has an optimal solution, the optimal dual point is incorporated into the master by adding an *optimality cut*, which drives the algorithm to search for master solutions with higher objective values. By adding cuts progressively, the algorithm avoids considering all the slave constraints together, but instead only considers those "promising" constraints that are likely to be active in the optimal solution. This can drastically reduce the overhead, especially for scenario-based optimization where a huge number of scenarios are considered. The UB is updated as the best feasible solution ever found, while the LB is updated when a new master solution is obtained. Optimality can be claimed when LB and UB converge.

5.4.4 Speeding-up Per-iteration Optimization

Even with the decomposition technique, Algorithm 5.1 is not efficient enough. The main reason is that it needs to solve a large number of dual slave linear programs (LPs) in each iteration, which is a slow process due to the cubical complexity for solving LPs [143]. In this subsection, we revisit the dual slave subproblems $\Delta_i(x, c)$ in (5.13), and show that they can be solved analytically due to their special structure.

Without loss of generality, we assume that the dual slave problem is feasible and bounded, *i.e.*, the corresponding primal slave problem is also feasible and bounded. Note that the primal problem is infeasible (hence the dual is unbounded) only when some AP is disconnected from any fog node in a certain scenario, which can be detected in the shortest path pre-computation phase. In the disconnected case, the *security risk* is ill-defined, as the AP cannot even communicate with the Internet. To tackle this, one way is to simply assign a uniform distance for all fog nodes, meaning that in this scenario it has no effect on the node selection. The primal problem cannot be unbounded (security risk is lower bounded by 0).

For scenario Π_i and AP a, let $v_a^i[1]$ and $v_a^i[2]$ be the two fog nodes with $x_v = 1$ and with the minimum distances from a, and let $\mathrm{dist}_a^i[1]$ and $\mathrm{dist}_a^i[2]$ be the corresponding distances respectively (with $\mathrm{dist}_a^i[1] \leq \mathrm{dist}_a^i[2]$). Also let $\delta_a^i = \overline{d}_a^i / \overline{d}_{\mathrm{sum}}^i$. We then have the following optimal solution:

$$\lambda_i = \begin{cases} \dfrac{\rho}{1-\alpha} & \text{if } \sum_a \delta_a^i \mathrm{dist}_a^i[1] \geq c \\ 0 & \text{otherwise} \end{cases} \tag{5.14a}$$

$$\phi_i(a) = \delta_a^i \mathrm{dist}_a^i[2](1+\lambda_i) \tag{5.14b}$$

$$\mu_i(a,v) = \begin{cases} \delta_a^i(\mathrm{dist}_a^i[2] - \mathrm{dist}_a^i(v))(1+\lambda_i) & \begin{array}{l}\text{if } v = v_a^i[1] \\ \text{or } x_v = 0\end{array} \\ 0 & \text{otherwise} \end{cases} \tag{5.14c}$$

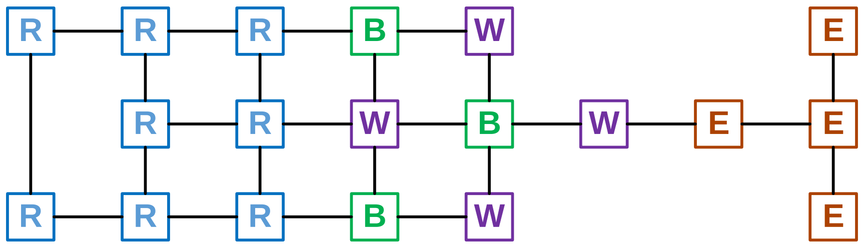
The following theorem states the optimality of the above solution, whose proof is delegated to the appendix.

Theorem 5.1. *If the dual slave problem has bounded optimal value, Eq.* (5.14) *is an optimal solution to Program* (5.13)*, and can be computed in linear time.* □

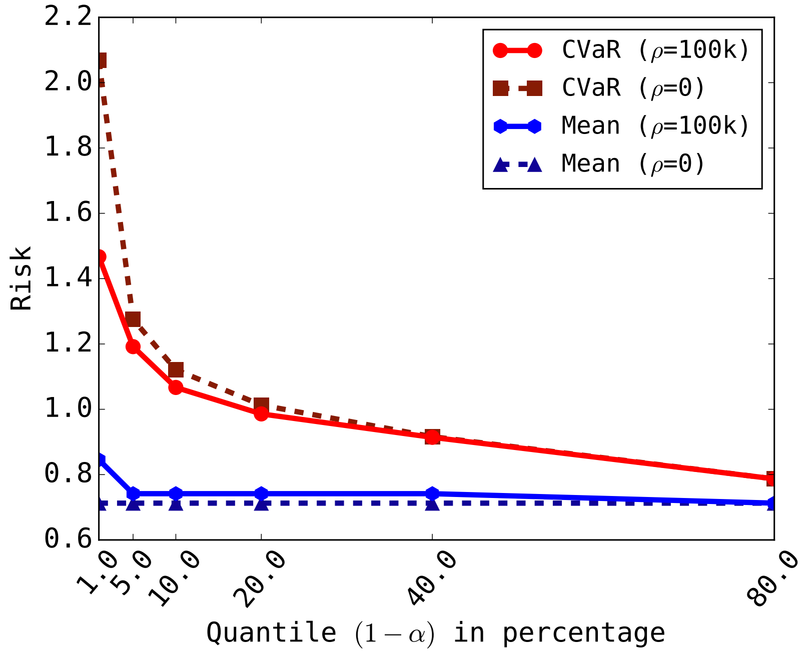
5.5 Performance Evaluation

5.5.1 CVaR vs. Expectation

In this subsection, we show simulation results on comparing the expected case and the worst case up to a small tail probability. Our topology is in Fig. 5.1(a), which is based on the IoT framework in [89]. The topology had four types of nodes representing different street blocks: residential (R), work (W), business (B) and entertainment (E).



(a) Simulation topology



(b) CVaR and expectation with $\rho = 100\text{k}$ (min CVaR) and $\rho = 0$ (min expectation) respectively

Figure 5.1: Simulation on designated topology.

Each node was both an AP and a fog node with uniform cost. The operator had a 30% budget over the sum of fog node costs. The scenarios were generated from a 3-month time period sliced into 15-minute intervals. Each link had an independent reliability of 99%. Depending on time and day of week, demand at each node was drawn from a Gamma distribution $\Gamma(a, b)$ where a is the shape and b is the scale, as shown in Table 5.1.

	Mon. – Fri.	Sat. & Sun.		Other (Sleep)
	8am–6pm	12pm–6pm	6pm–10pm	
R	$\Gamma(0.5, 1.5)$	$\Gamma(0.5, 1.5)$	$\Gamma(0.5, 0.5)$	$\Gamma(0.5, 3.0)$
W	$\Gamma(0.5, 2.0)$	$\Gamma(0.5, 0.2)$	$\Gamma(0.5, 0.2)$	$\Gamma(0.5, 0.8)$
B	$\Gamma(0.5, 0.5)$	$\Gamma(0.5, 2.0)$	$\Gamma(0.5, 1.0)$	$\Gamma(0.5, 0.3)$
E	$\Gamma(0.5, 0.2)$	$\Gamma(0.5, 0.5)$	$\Gamma(0.5, 2.5)$	$\Gamma(0.5, 0.1)$

Table 5.1: Probability Distribution of Demands

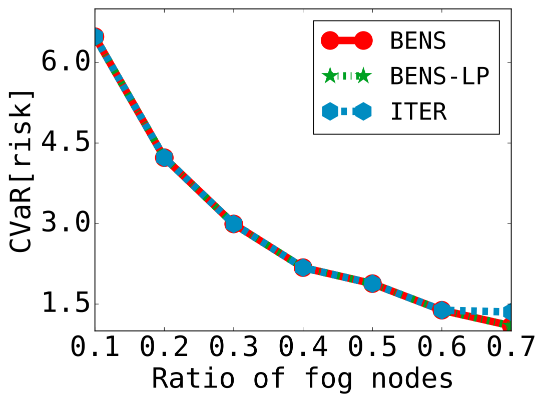
We compared between using Algorithm 5.1 with a large enough ρ value (where only CVaR is considered) and with $\rho = 0$ (where only the expectation of risk is considered). We varied the quantile α to see the CVaR of different confidence levels. For example, a large quantile $\alpha = 99\%$ means we only look at the 1% most unfavorable scenarios, while $\alpha = 0\%$ means that we are looking at all the scenarios in CVaR, which by definition is equivalent to the expectation itself. To solve the MILP master problem, we used the Gurobi solver [48].

Fig. 5.1(b) shows the optimal solutions (with error tolerance of $\epsilon = 10^{-5}$, same hereinafter) obtained by Benders' decomposition. The quantile $(1-\alpha)$ is the percentage of scenarios included in the computation of CVaR. We can see that optimizing CVaR and optimizing expectation indeed achieves different solutions in most of the cases. With increase in $(1 - \alpha)$, more scenarios are included in the CVaR, hence the CVaR value approaches the mean. However, with $(1 - \alpha) = 1\%$, there is a great difference in minimizing CVaR ($\rho = 100$k) and minimizing expectation ($\rho = 0$), where the CVaR of security risk can be almost $1.5\times$ larger in the latter case than in the former. This suggests that in security systems where a few worst cases need to be carefully tackled, it is important to apply CVaR-aware optimization to account for the worst-case performance rather than for the average case alone.

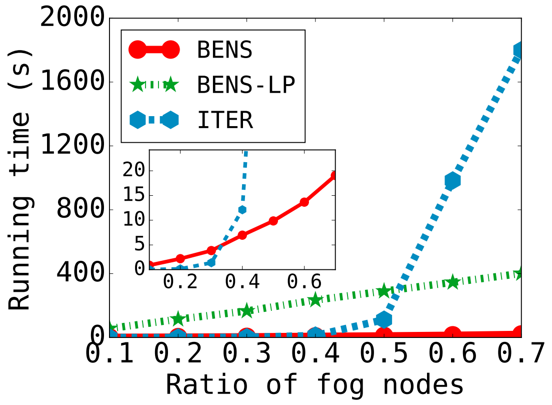
5.5.2 Optimality and Efficiency

Next, we used randomly generated network to show how our proposed algorithm was able to achieve vast speed-up compared to the brute-force approach. Each topology had 30 nodes, and was generated using the Waxman model [32] with parameters $\alpha = \beta = 0.3$. We randomly picked half of the nodes to be APs, and ψ of the nodes to be fog nodes where $\psi \in [0, 1]$ is the *fog node ratio*. Deployment costs were generated uniformly from $[10, 100]$, while the operator had a budget of 50% of the sum of costs. In total 10k scenarios where generated for each experiment. Link reliability was uniformly 99%. The demands at APs were generated from an Erlang$(1, 2)$ distribution. We set quantile $\alpha = 95\%$. To average out noise of topology randomization, we generated 20 different topologies for each experiment setting, and took the average over all runs. Simulations were conducted on a Linux PC with 3.4GHz Quad-Core CPU and 16GB memory.

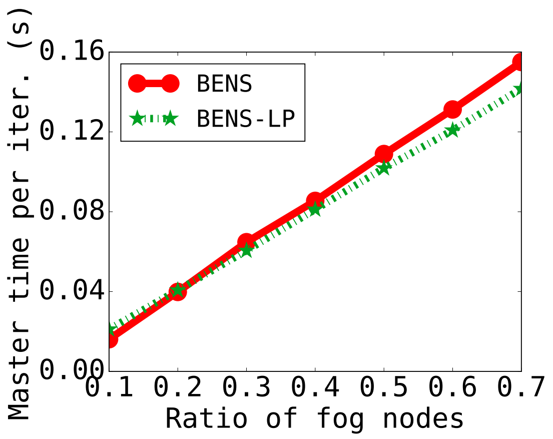
Fig. 5.2 shows the results of our algorithm over a brute-force algorithm. BENS is our proposed algorithm with our analytical model, while BENS-LP is our proposed algorithm with dual slave LPs solved by the Gurobi solver [48]. Brute-force algorithm ITER iterates over all possible deployment plans $X \in \mathcal{X}$, and returns the best solution found after all iterations or after a running time limit of 1800 seconds; ITER *returns the optimal solution unless exceeding the time limit.* Fig. 5.2(a) shows the optimality of our algorithm. Note that when the fog node ratio exceeds 0.7, ITER could not iterate over all possible solutions in the time limit, and hence returns suboptimal solutions. Fig. 5.2(b) shows the overall running times. Our BENS algorithm is the fastest in most cases. ITER is faster than BENS when the fog nodes are few, but then its time grows and quickly tops-up the pre-set time limit of 1800 seconds, due



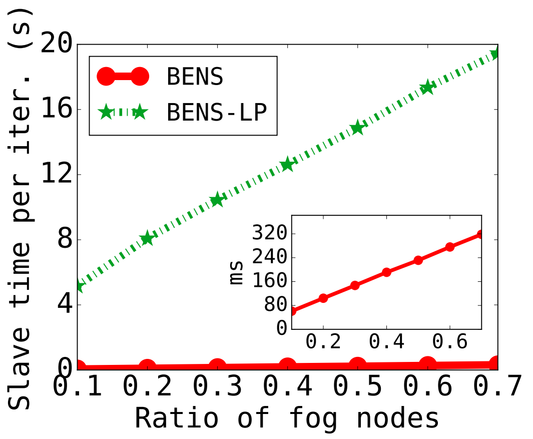
(a) CVaR vs. fog nodes



(b) Running time vs. fog nodes



(c) Master time vs. fog nodes



(d) Slave time vs. fog nodes

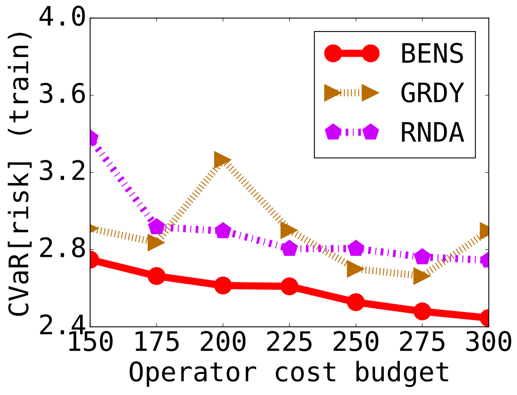
Figure 5.2: Simulation results with varying number of fog nodes over total nodes. ITER (optimal algorithm) exceeds the time limit (1800 s) when the fog node ratio is 0.7, and hence is terminated before finishing all iterations.

to the exponential increase in the solution space. Comparing BENS and BENS-LP, the former achieves a drastic speed-up. We further plot the average master and slave solving times per iteration in Figs. 5.2(c) and 5.2(d), respectively. It can be seen that BENS and BENS-LP do not differ much in master solving time. However, BENS achieves several orders of speed-up in slave solving time, further validating the effectiveness of our analytical model. Finally, an interesting finding is that, although the master problem is an MILP, it actually solves faster than the slave problem, due to its small size compared to the large number of scenarios.

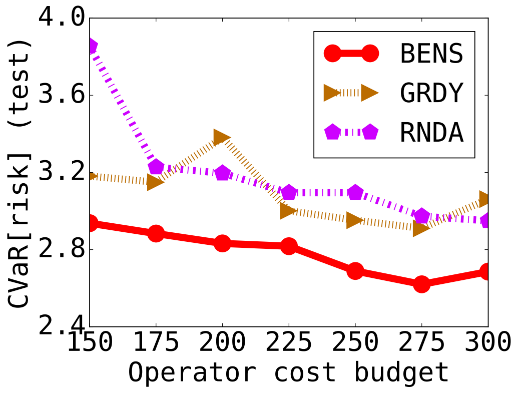
5.5.3 Comparison with Fast Baseline Heuristics

In the last set of experiments, we show the performance of our algorithm compared to fast heuristics. For this comparison, we used a synthesized dataset derived from real IoT device traces collected in the Dartmouth College [67]. The original dataset contained the location data of over 600 APs and the connectivity data of over 13000 user devices for more than 5 years. We synthesized the dataset as follows. First, we aggregated APs based on buildings, and regarded each building as a single AP. Second, we further divided buildings into street blocks based on a publicly available map of the campus. Due to lack of campus network topology, we adopted a topology where each building's AP is directly connected to a central node within each block, and all blocks are connected in a ring topology. All block nodes and 10% of the buildings were selected as fog nodes, with deployment costs of 100 and 10, respectively. Finally, since the entire data set contains many intervals that have unstable and erroneous measurements, we only used a one-year subset of data (09/2002–08/2003) that is relatively stable according to the collectors [67].

To characterize the demand distribution, we regarded the user connectivity data as samples from the underlying realistic distribution. We used 4 months of data (09/2002–12/2002) as training set, and the next 8 months of data (01/2003–08/2003) as the test set, both sliced into intervals of 15-minute length. We used the training data to deploy the security functions, and then calculated the security risks of the deployed security functions on both the training set and the test set. The number of devices were aggregated and averaged over every 15-minute interval for each building AP. All links had 99% reliability. We again set ρ to a large value, and let $\alpha = 95\%$.



(a) Training set performance



(b) Test set performance

Figure 5.3: CVaR of risk on synthesized Dartmouth data.

Fig. 5.3 shows the training and testing security risks using the synthesized data. We compared our proposed algorithm to two heuristics: RNDA and GRDY. RNDA is a random algorithm which randomly picks fog nodes to deploy the security function, until the budget is exceeded. GRDY is a greedy algorithm which iteratively selects fog nodes that result in the maximum reduction in the objective function, until the budget is exceeded. From the figure, we can see that our algorithm outperforms both GRDY and RNDA. The greedy heuristic generally achieves better performance than RNDA, but may result in poor performance with certain budget values such as a budget of 200. Also, it is interesting to see that an optimal training solution may not be optimal on the test set. For example, when the budget increases from 275 to 300, the training CVaR of BENS (the optimal) decreases, but the test CVaR increases. This is commonly due to changes in the underlying distribution, and should be tackled by periodically re-optimizing the deployment decisions based on new inputs.

5.6 Discussions and Future Work

Advanced deployment and risk measurement: Our current distance-based risk measure can be extended to more complex cases. For example, each link may have a routing capacity and each security function may have a processing capacity in practice. In this case, demands may be split over multiple paths and/or security functions if some are overloaded. Also, different links and/or nodes may have different contributions to the security risks, as some may be more vulnerable to others. These considerations can mostly be integrated with minimal modifications into our CVaR-based and scenario-based optimization framework. Derivation of efficient solutions in these more complex cases is delegated to our future work.

Further optimization speed-ups: The proposed algorithm uses the basic Benders' decomposition, with the only speedup being the problem-specific analytical model for the dual slaves. Other techniques may also improve the efficiency of Benders' decomposition, such as enhanced cuts [24], the three-phase method [24], trust regions [78], etc. While these techniques can hardly beat the speed-up achieved by our analytical model (and most of them are not trivially compatible with our analytical model), they are helpful in general when our model and framework are extended to other more complex cases as aforementioned. An orthogonal technique is to employ parallelization in each iteration, which is natural through the dual slave decomposition in our algorithm.

5.7 Conclusions

In this study, we studied the security risk associated with offloading security from IoT devices to fog or cloud nodes in the network. To maximize system security and robustness, the operator would want to deploy in-network security functions to minimize the security risk of all users, given various scenarios including varying demands and network failures. We made the following contributions. First, we proposed a stochastic model for uncertainties in IoT. Second, we used an economic model (CVaR) to capture the worst-case security risk of the system, in addition to the conventional expectation-based model. Third, we developed a decomposition-based optimization framework for optimizing both the expected security risk and the its CVaR in scenario-based stochastic programming. We then enhanced the framework with an analytical model tailored to drastically reduce its optimization overhead. Finally, we showed, through simulations, that the proposed model well captures system security risk up to a small tail probability, and that the proposed framework achieves optimal security deployment with limited overhead compared to other algorithms.

5.8 Appendix

Proof of Theorem 5.1. For simplicity, we omit subscript i, since the dual slave subproblem is independent for each scenario. We call a node *active* if it has $x_v = 1$ in the current iteration, otherwise it is *inactive*. First, for any inactive node v, we can assume that $\mu(a, v)$ takes an arbitrarily large value to enforce Constraint (5.13c), as it has a zero coefficient in the objective function. For each AP a, a node v can have a positive $\mu(a, v)$ only when the corresponding Constraint (5.13c) is binding, *i.e.*, equality holds instead of inequality at any optimal point, for this node; otherwise, the objective value can be solely increased by decreasing $\mu(a, v)$ without violating Constraint (5.13c). We claim that there is at most one active node v with positive

$\mu(a, v)$; if multiple active nodes have positive $\mu(a, v)$, then we can reduce $\phi(a)$ along with all the positive $\mu(a, v)$s by a small amount, reducing objective value without violating Constraint (5.13c) for any node. Further, we claim that the node with positive $\mu(a, v)$ must be the one and only one active node v^* who has the minimum distance from a; if more than one node has the same minimum distance, they must all have $\mu(a, v) = 0$. Otherwise, say if an active node v' with non-minimum distance has a positive $\mu(a, v')$ and the corresponding Constraint (5.13c) is binding, then clearly $\mu(a, v^*) > \mu(a, v') > 0$, because $\text{dist}_a(v^*) < \text{dist}_a)(v')$. Then we have two positive $\mu(a, v)$ values, which conflicts with the above. In the case of a single minimum-distance active node $v^* = v_a^i[1]$, both $\phi(a)$ and $\mu(a, v^*)$ can take positive values up to the values specified in Eq. (5.14), without violating Constraint (5.13c) for the second minimum-distance node, $v_a^i[2]$. We pick the upper bounds to motivate the master problem to search for new solutions in each iteration.

Now, for the inactive nodes, though they can take arbitrarily large values, we pick their lower bounds, such that reducing any μ value in this class will lead to constraint violation. This gives Eq. (5.14) for $x_v = 0$. This choice is for simplicity of the equation, but can also help in numerical stability in practice.

Based on the above, we know that in the optimal solution, the first term of the objective function is always equal to $\xi(1 + \lambda)$ where $\xi = \sum_a \delta_a \text{dist}_a[1]$. Then, it is clear that the λ value is based on the comparison between ξ and c. If $\xi \geq c$, we set λ to the maximum value $\frac{\rho}{1-\alpha}$ as bounded by Constraint (5.13b); otherwise, we set λ to 0. This completes the proof. $\square$

Part III

Micropayment Routing in Blockchain-based PCN

Chapter 6

COINEXPRESS: A FAST PAYMENT ROUTING MECHANISM IN BLOCKCHAIN-BASED PAYMENT CHANNEL NETWORKS

6.1 Introduction

Ever since the invention of Bitcoin by Satoshi Nakamoto in 2008 [88], we have well witnessed the blooming of thousands of altcoins[4], which (along with Bitcoin) jointly support a digital payment market with over \$800B of capitalization at its peak[5]. It is envisioned that, in addition to the digital world where digital payments already prevail, sectors such as banking, international trading, manufacturing, healthcare, taxation and many more will also enjoy extensive benefits from this trend. Furthermore, the underlying blockchain technology has inspired (and will continue to do so to) numerous innovations, fundamentally transforming the business models of internet-of-things, supply chain, auditing etc., and even exerts political and human right influences with applications in governance transparency, democratic voting and freedom of speech.

However, current mainstream cryptocurrencies such as Bitcoin and Ethereum, although achieving strong security through decentralization, bear some severe limitations that greatly hinder their applications in our daily life. A significant one is the *scalability issue* brought by the requirement of global consensus and the large-overhead of security assurance through expensive consensus algorithms. Both Bitcoin and Ethereum employs the Proof-of-Work consensus algorithm, which can only generate

[1]Altcoin refers to cryptocurrencies that are *alternative* to Bitcoin.

[2]Data is based on CoinMarketCap (https://coinmarketcap.com/).

blocks in a pre-specified speed, and each block can only include a limited number of transactions to avoid centralization. As a consequence, the Bitcoin blockchain can only process up to 7 transactions per second (tps) [97], while the number for Ethereum is around 15 tps [23], compared to over 45000 peak tps handled by Visa [97].

Facing this issue, there have been extensive efforts on improving blockchain scalability in several orthogonal directions. Among them, a promising proposal is to construct off-chain *payment channels*, which carry out transactions with minimal involvement of the blockchain itself. Specifically, users build peer-to-peer (P2P) channels with pre-deposit funds, and transfer values by re-adjusting fund allocation on the channels for each on-going transaction. Each transaction is protected by a smart contract, such that any non-cooperative behavior will be punished by granting all fund on the channel to the counter-party. In such a scenario, all transactions via a channel are stacked, and will be jointly published to the public blockchain upon channel expiration. Therefore the involvement of expensive blockchain operations is limited to only channel establishment, close-out, and the rare events of dispute arbitration in case of non-cooperative behaviors.

It has been envisioned that a distributed network comprised of these payment channels, namely a *payment channel network* (PCN), can take most of the transactions off-chain, hence drastically reducing payment overhead and increasing scalability of the payment system [97]. Both leading cryptocurrencies are actively seeking the deployment of PCNs alongside their main blockchains; see Section 6.6. However, PCN has its own problems to be addressed. Maintaining a payment channel basically requires locking a certain amount of funds within the channel for an extended amount of time. Hence each normal user only has the capability to maintain a small number of channels with closely related parties. When paying to an arbitrary recipient in the world, most likely an indirect payment is needed, which spans multiple channels in the network. Such indirect payments can cause a number of problems. First, one or multiple payment paths from sender to recipient need to be provisioned before the payment starts. In network terminology, the payment must be *routed* before going through. Second, a contract is needed to guarantee non-repudiation at each intermediate node. Other issues include denial-of-service attacks, privacy, node availability, transaction fees, etc.

In this study, we investigate routing for indirect payments in PCN. PCN routing has a number of distinct characteristics, which renders it intrinsically different from routing in traditional computer networks. First, PCN routing focuses on finding routes with *sufficient capacity (fund)* to serve a payment, rather than finding the shortest transmission paths as in traditional network routing. Second, PCN routing is fully distributed, as no central administrative operator exists in PCN; even if such an operator exists, it would not be trusted by any user, since its existence already defeats the spirit of decentralization that has set the foundation of cryptocurrencies. Third, PCN routing is more sensitive to dynamics in the network, due to the unique property of payment channels that the fund in a channel is commonly consumptive and non-recoverable unless other events happen; see Section 6.2. Last but not least, privacy plays an important role; minimizing privacy leaks of the payer/payee (and possibly the involved nodes) can be a priority task in certain scenarios.

In face of these challenges, we propose *CoinExpress*, a routing mechanism that efficiently finds "express lanes" for cryptocurrency-based digital payments in PCNs. As a first step in PCN routing, CoinExpress focuses on high-performance and efficient payment routing, while leaving the privacy issue as an orthogonal enhancement to be addressed in the next chapter. We make the following contributions:

- We investigate important design goals of routing in PCN, and propose a practical model for PCN routing based on network flow and concurrent flows.
- We propose a distributed approach for PCN routing, which, in addition to finding routes that fulfill the payment, providing guarantee to the timeliness and availability regarding user's payment deadline and the expiration times of the underlying channels, respectively.
- We have shown, through simulations, that CoinExpress not only achieves superior payment successful ratio, but also drastically reduces overhead over existing work.

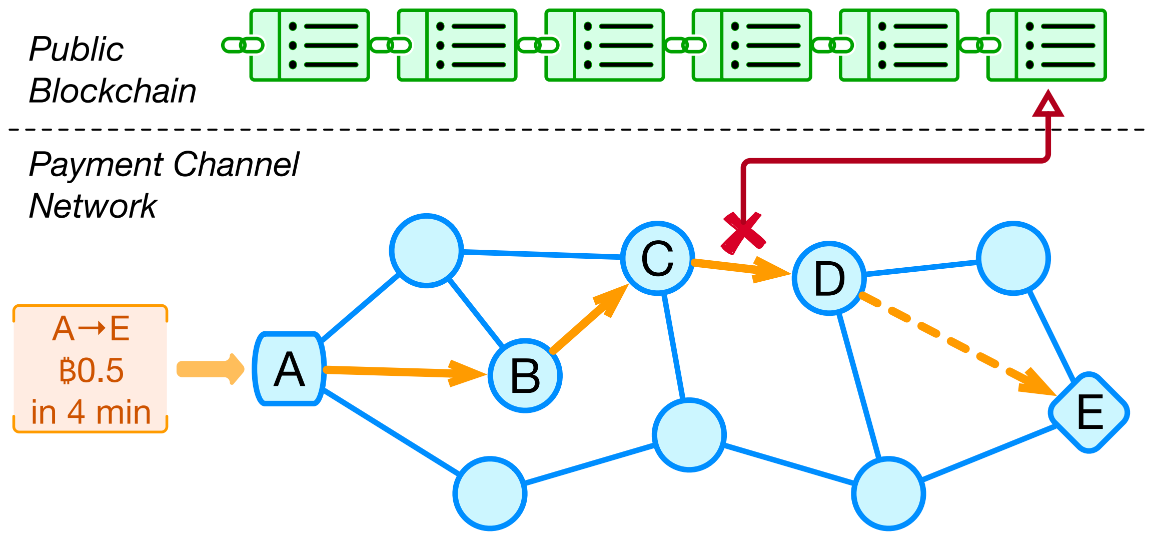
The rest of this study is organized as follows. In Section 6.2, we introduce the background of PCN and routing, and state the design goals. In Section 6.3, we present our system model. In Section 6.4, we propose our distributed PCN routing design, and describe the detailed algorithms for each involved party in routing. In Section 6.5, we show performance evaluation results of our design compared to a state-of-the-art PCN routing scheme and other baselines. In Section 6.6, we review existing work in related areas. In Section 6.7, we conclude this study.

6.2 Background and Overview

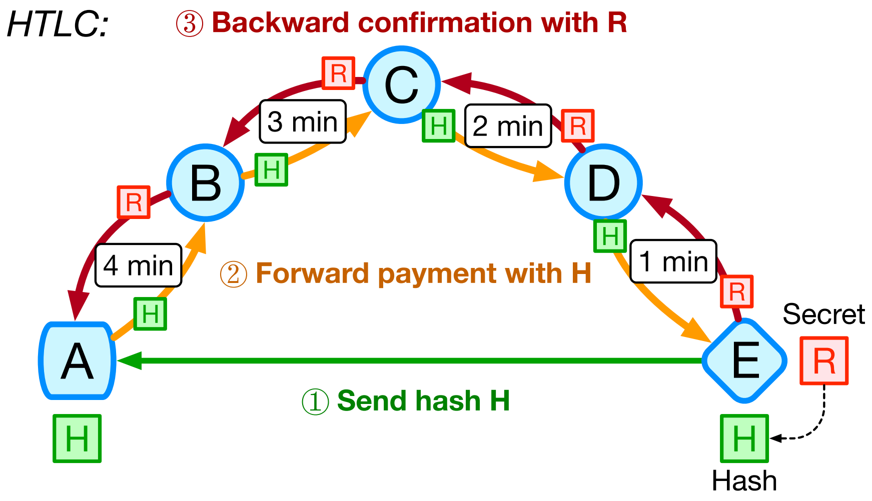
6.2.1 System Overview

A PCN consists of several components, as shown in Fig. 6.1(a). Their detailed functions are explained below.

In PCN, a user is identified by a unique account address, usually also the public key of the account. A payment channel can be viewed as a temporary joint account



(a) In a PCN, user A requests payment of 0.5 Bitcoin to E within 4 minutes. A finds payment path A→B→C→D→E which has sufficient balance and can settle within 4 minutes. A sends initiate payment to B for forwarding. Each hop forwards payment. If intermediate node D wants to steal the payment by not forwarding the payment, C will then publish this situation to the public blockchain, who will judge on D's dishonesty and grant all funds on channel C→D to C as a punishment to D. Assuming all parties are honest, no blockchain operation is involved.



(b) For request A→E, recipient E generates secret R and its hash H, and sends H to A. A encodes H and deadline 4 minutes into its contract with B, such that B cannot spend the payment without providing R to A within 4 minutes. Each node forwards payment similarly, and employs a decreasing deadline. Upon reception, E will provide R to D to spend D's payment, while D and so forth will do the same to previous hops, until A receives R.

Figure 6.1: PCN overview and Hashed TimeLock Contract (HTLC)

between two users, whose balance is divided by the two parties and the division can be adjusted based on agreement of both users. A channel is established by both parties each depositing a certain amount into the joint account. The total deposit amount is called the *channel capacity*, which defines the maximum amount of value that can be transferred via this channel. A *unidirectional channel* only allows monotone balance adjustments, while a *bidirectional channel* allows adjustments in both directions.

In PCN, a *transaction* is essentially a channel balance update agreed upon by both parties. A channel is protected by multi-signature smart contracts, which ensure validity, non-equivocality and non-repudiation of the on-going transactions. When one party publishes obsolete balance history to reverse settled transactions or to double-spend, the contract guarantees that the dishonest party is punished by granting all its remaining channel balance to the other party. This economically prevents an adversary from utility gain via dishonest behaviors.

A *payment* from sender to recipient is performed via a number of transactions in different channels, organized as either a *payment path* or a *payment flow*. A *direct payment* can be made between two parties who share a channel, and is settled immediately after the corresponding transaction is completed. If two parties do not share a channel, an *indirect payment* is needed, which requires balance updates on multiple channels. This, however, can lead to issues if an intermediate node denies performing a subsequent transfer after receiving a preceding one, or the recipient denies receiving the payment.

Hashed TimeLock Contract (HTLC) [97] is introduced to solve the multi-hop problem, as shown in Fig. 6.1(b). An HTLC consists of both a *hash lock* and a *time lock*. In the hash lock, recipient generates a random value R with hash value H, and sends H to the sender. Sender, as well as any intermediate node, includes

H in the transaction contract, such that the transferred fund is spendable by the transferee only when the secret R used to generate H is provided to the transferor. This ensures non-repudiation of transfer reception, as no one can spend its received amount without acknowledging the reception. In the time lock, each transaction is restricted by a completion deadline, such that if the transferor does not receive R by the deadline, the transferred fund will be refunded to the transferor. In an indirect payment, each forward hop employs a decreasing deadline, taking into account the time for completing its own balance update as well as the time the two parties wish to wait to tolerate delay fluctuations in other hops. Any non-cooperative behavior will cause the corresponding transaction to be published on the public blockchain, which will grant all funds in the channel to the counter-party as a punishment to the non-cooperative party. By employing a sequence of time-outs in the opposite direction of the payment path, it is guaranteed that no transferee or recipient can hold up a channel until any preceding channel expires (in which case it can repudiate the reception and steal the fund).

From the above discussion, the key in performing a successful indirect payment is to find a *route* with sufficient balance, in the form of a path or a set of paths (a flow), such that an end-to-end time lock can be established in each path that respects the expiration time of each channel in the path. Next, we elaborate a number of unique challenges of routing in PCN, and a set of desired design goals of any routing solution.

6.2.2 Challenges in Routing Design

At a first glance, routing in PCN is just a variant of the widest path problem or the maximum flow problem, for which many efficient algorithms exist. Unfortunately, the problem is not as simple as it seems due to many practical constraints.

First, due to the time lock in HTLC, each payment path needs to satisfy a sequence of time delay constraints. For example, the last hop's channel expiration time needs to be lower bounded by the cumulative update and transmission delay of all hops in the path; the second last hop's expiration time needs to be lower bounded by the same cumulative delay plus the backward delay of the last hop; so on and so forth. Moreover, a user may require fast payment settlement, by specifying a deadline before which the payment needs to be settled. For the payment flow problem, it is well-known that the Multi-Path routing with Bandwidth and Delay constraints problem (MPBD) is NP-hard, corresponding to our routing problem that only considers the last hop's expiration time.

Second, the remaining balance on each directional payment channel is *non-interfering monotone*. Here, a directional payment channel refers to either a unidirectional channel, or one direction of a bidirectional channel. For each directional payment channel, its remaining balance is *consumed* after each transaction. *Non-interfering monotonicity* means that unless the channel is explicitly interfered with, *e.g.*, is recharged via external funding or opposite-direction transactions, the consumed balance cannot be used by other transactions on the same direction. We use the term *monotonicity* for abbreviation.

One consequence of channel monotonicity is the unpredictability of balances in the network, which is due to the highly dynamic nature of the network. The available balance of each channel is constantly changing with every transaction. Thus it is impractical or even impossible for every node to keep track of the real-time balances of all channels in the network, especially when modern payment networks typically scale to billions of nodes [97], [104]. To find a route with sufficient balance, the sender could either estimate channel balances based on empirical data, but with a high risk of payment failure due to estimation inaccuracy, or actively probe for available balances, which incurs probing overhead.

Moreover, high network dynamics and channel monotonicity can cause concurrency issues. When multiple concurrent payments compete on one or more channels, they may block each other from progressing. In the worst case, deadlocks could happen, locking up funds on all involved channels [80].

6.2.3 Design Goals

Addressing these challenges require routing mechanisms that satisfy a set of design goals, which we elaborate below.

- **Timelock-compatibility:** In an HTLC-guarded PCN, a major requirement of payment routing is to ensure that a feasible end-to-end time lock can be successfully established along each path, which guarantees the commitment of honest processing at any involved node.
- **Distributedness:** The routing mechanism must not rely on a centralized trusted party. Centralized routing is subject to single point of failure upon external attacks, and hence cannot be trusted by users. Instead, nodes need to communicate with each other and conduct local computations to find routes for payments.

- **Concurrency:** A routing mechanism should be non-blocking in that at any time, at least one payment can progress without waiting for concurrent payments.
- **Goodput:** The mechanism should maximize system goodput, measured by the number or total value of successful payments in a given time window. This is not equivalent to maximizing system throughput, which is the total value that can be delivered in the period. A partially fulfilled payment is viewed as failed if the recipient does not receive the full amount within the deadline.
- **Efficiency:** First the mechanism should minimize the routing and payment latency incurred by users. In more time-sensitive scenarios, the mechanism should guarantee payment settlement within a user-specified deadline. Second, the mechanism should only incur limited overhead on both the end-points and the intermediate nodes.
- **Privacy:** The mechanism should preserve secrecy of various information in the network, which is distinguished into the following types:
  - *Sender/Recipient Privacy:* An adversary should not be able to determine the sender/recipient of any payment between non-compromised parties.
  - *Value Privacy:* An adversary should not be able to learn the exact value of any payment. Moreover, the adversary should learn as little information as possible about the range of value of any payment.
  - *Path Privacy:* An adversary should not be able to learn the path(s) of any payment, other than the nodes it has already compromised.
  - *Channel Balance Privacy:* An adversary should not be able to learn the exact balance of a payment channel at any given time, unless the channel connects to a node it has already compromised.
  - *Channel Load Privacy:* An adversary should not be able infer the load on

a payment channel connecting non-compromised nodes, from sources such as the settlement delay of the channel.

As a first step, in this study we aim to achieve the first five goals: *timelock-compatibility*, *distributedness*, *concurrency*, *goodput* and *efficiency*. We address privacy preservation as a separate and orthogonal task in the next chapter.

6.3 System Model

6.3.1 Network Model

A distributed PCN is modeled as a weighted directed graph $G = (V, E)$. V is the set of nodes, *i.e.*, users each of whom has established at least one payment channel with a peer user. E is the set of links. A link denotes either a unidirectional channel from one user (the transferor) to another (the transferee), or one direction of a bi-directional channel between two users.

Each link has several attributes. First, each link $e \in E$ has a *channel capacity* c_e, denoting the total amount of value deposit into the underlying payment channel by both parties. Second, each link $e \in E$ also has a current *balance* b_e. For a unidirectional channel (represented by a single link in G), the capacity c_e is the maximum amount of value that the transferor can send to the transferee before the channel expires, while the balance b_e is the remaining amount of value that the transferor can send. For a bidirectional channel (two collateral links in opposite directions between two users), the two links have the same capacity, equal to the sum of their balances, *i.e.*, $b_{(u,v)} + b_{(v,u)} = c_{(u,v)} = c_{(v,u)}$. A link e always has $b_e \leq c_e$. For simplicity, we assume the set E only contains links with positive balances at any time. A link with zero balance is temporarily removed from the graph (*i.e.*, the views of the nodes it connects), until its balance gets recharged by new deposits or payment transactions on the opposite direction.

Payment through a channel is not instant. First, each channel needs to complete arriving payments sequentially. Second, it requires time for the two parties to agree on the balance update, which is the forward processing time of the current hop. Third, due to the time lock in HTLC, the channel further needs to release the locked transferred value after receiving acknowledgement from the next hop, which requires a certain amount of waiting plus backward processing time. For simplicity, we omit the transmission delay between parties, which can be incorporated into the forward and backward processing times. We use d_e^1 to denote the forward wait time plus the forward processing time of a link e, and d_e^2 to denote the backward wait time plus the backward processing time, at any given time. We let $d_e = d_e^1 + d_e^2$. Each link further has an expiration time η_e, denoting the time when the underlying channel becomes unavailable. For a payment via e to be successful, it must settle before the channel expires.

We assume that each user only has local knowledge on all its in-coming and out-going links, including their capacities, expiration times, as well as instantaneous balances and delays. Each sender/recipient additionally knows the other party's address, but does not know the other party's location in the network. In general, each node may have rough estimation of the overall network status based on local information, but cannot know the instantaneous balance or delay of any remote link, due to network asynchrony and dynamics.

6.3.2 Payment Model

A *payment request* is denoted by a quintuple $R = (s, t, a, st, dl)$, where s and t are the sender and recipient respectively, a is the amount to be paid, and st and dl are the start time and deadline respectively. Let $\mathcal{P}$ be the set of all paths in the PCN. A payment request R is realized by a *payment plan*, *i.e.*, a set of paths $P_R \in \mathcal{P}$, where each $p \in P_R$ is an (s,t)-path in the network. Each path $p \in P_R$ is associated with a payment value, denoted by v_p. For a payment plan P_R to be successful, it needs to satisfy the following conditions:

- P_R is **feasible**, iff for any link $e \in E$ where $P_R(e) \subseteq P_R$ is the set of paths through e, we have
$$\sum\nolimits_{p \in P_R(e)} v_p \le b_e. \tag{6.1}$$
- P_R is **available**, iff for any $p \in P_R$ and $e \in p$, let $p_e^+ \subseteq p$ be links including and after e in p, then we have
$$st + \sum\nolimits_{e \in p} d_e^1 + \sum\nolimits_{e \in p_e^+} d_e^2 \le \eta_e. \tag{6.2}$$
- P_R is **timely**, iff for any path $p \in P_R$, we have
$$st + \sum\nolimits_{e \in p} d_e \le dl. \tag{6.3}$$
- P_R is **fulfilling**, iff
$$\sum\nolimits_{p \in P_R} v_p \ge a. \tag{6.4}$$

Based on the above definition, a payment plan that is *feasible, available, timely and fulfilling* is able to transfer a amount of value from sender s to recipient t within the deadline dl, and we call it a *realizing* payment plan for request R. In this study, we are interested in finding a realizing payment plan for each payment request, while satisfying as many design goals as possible.

6.4 Dynamic Routing Design

Most existing bandwidth-aware routing algorithms cannot be applied in our scenario, due to the frequently changing link balances and delays. A naive approach is to empirically estimate the channel statuses and route accordingly, yet such an approach is subject to poor accuracy and can lead to low goodput. We propose *CoinExpress*, a novel probing-based dynamic routing mechanism. In CoinExpress, the sender probes for channel statuses before payment, and reserves balances in advance. CoinExpress is fully distributed, adaptive to network dynamics and concurrent, and achieves high goodput. Unfortunately it does not provide privacy guarantee, which we aim to address in future work.

It should be noted that due to the availability and timeliness constraints, the problem of finding a realizing payment plan is NP-hard, even when the sender has full knowledge of the network. With only the timeliness constraint, the problem is equivalent to the MultiPath routing with Bandwidth and Delay constraints problem (MPBD), which has been proved to be NP-hard in [85], and the best centralized algorithm one can expect is an expensive *fully polynomial-time approximation scheme* (FPTAS) [85]. Instead, our approach is a distributed algorithm derived from the Ford-Fulkerson algorithm for maximum flow [36], and applies a locking technique to resolve concurrency among multiple simultaneous requests. Before diving into our algorithm, we first define some basic concepts on network flow.

6.4.1 Network Flow Preliminaries

To avoid ambiguity, we use *balance* to replace the term *capacity* that is commonly used in flow networks. For simplicity, for $u, v \in V$ such that $(u, v) \notin E$, we assume $b_{(u,v)} = 0$.

Definition 6.1 (**Network flow**). *Given network G with balances $\{b_e\}$, source s and destination t, an (s,t)-**flow** is defined as a mapping $f : V \times V \mapsto \mathbb{R}^*$, with the following properties:*
1) Flow conservation: for $\forall v \in V, v \neq s, t$,

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u);$$

2) Balance constraint: $f(u,v) \leq b_{(u,v)}$.
*We define $b(f) = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$ as the **flow value** of f.* □

Definition 6.2 (**Concurrent network flows**). *Given network G and a set of flows $F = \{f\}$, we say that the flows are **concurrent** iff they satisfy the joint balance constraint: $\sum_{f\in F} f(u,v) \leq b_{(u,v)}$.* □

Definition 6.3 (**Residual network**). *Given network G with balances $\{b_{(u,v)}\}$, and a flow f, the **residual balance** $b^f_{(u,v)}$ of each pair of $u, v \in V$ is defined as follows:*

$$b^f_{(u,v)} = \begin{cases} b_{(u,v)} - f(u,v) + f(v,u), & (u,v) \text{ or } (v,u) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

*The **residual network** G^f regarding f is the network with the residual balances.* □

Note that the residual network may contain more links than the original one, due to the addition of backward links (v,u) when only the forward link (u,v) exists in the original network. For simplicity, we assume that each flow contains no loop with positive flow value, hence $f(u, v)$ and $f(v, u)$ cannot both be positive for $\forall u, v \in V$. For completeness, we show the Ford-Fulkerson algorithm in Algorithm 6.1.

Algorithm 6.1: Ford-Fulkerson max-flow algorithm [36]

```
   Input: network $G = (V, E)$, source $s$, destination $t$
   Initialize: start with an empty flow $f$ and $G^f = G$
1  repeat
2      Find $(s, t)$-path $p$ in $G^f$ with positive balance $f_p$;
3      Add $p$ to $f$, and update $G^f$;
4  until no augmenting $(s, t)$-path can be found;
5  return $f$.
```

6.4.2 Dynamic Routing Design

To apply the Ford-Fulkerson algorithm, we need to address several challenges. The first one is to transform Algorithm 6.1 into a distributed algorithm where each node only has local knowledge. Second, the timeliness and availability constraints need to be taken into account. Third, the concurrency issues needs to be tackled to ensure no harmful racing between concurrent requests. We address these in the following.

Our CoinExpress algorithm is jointly shown in Algorithms 6.2, 6.3 and 6.4. Note that when we say "send a message along a link e", we essentially mean *one party sending message to the other through a secure communication channel*, rather than actually sending fund through the payment channel.

Our algorithm employs a distributed *breadth-first-search* (BFS) to search for augmenting paths. For the request, each node maintains a local view of the residual network, with residual balances initialized to the initial link balances. In each round, the sender sends a forward probe $\rho = (R, \beta, \delta, \Pi)$ to each neighbor who has a link e with

```
Algorithm 6.2: Sender algorithm
Input: local links of $G$, request $R = (s, t, a, st, dl)$
Initialize: empty flow $f$, residual balances $\{b^f_e\}$
1  while $b(f) < a$ do
2      for any out-going link $e$ of $s$ such that $b^f_e > 0$ do
3          Construct probe $\rho = (R, \min\{a - b(f), b^f_e\}, d^1_e, [(e, \eta_e, d^2_e)])$ ;
4          Send $\rho$ along $e$;
5      end
6      Wait for current-round confirmation for time $T_{\text{conf}}$;
7      if confirmation $\gamma = (R, p, \beta)$ is received from $e$ then
8          if $b^f_e \geq \beta$ then
9              Add path $p$ to $f$ with value $\beta$;
10             Update residual $b^f_e$;
11         else
12             Send cancellation $\kappa = (R, p, \beta, \text{cancel})$ via $p$;
13         end
14     else
15         Send cancellation along all paths;
16         Retry within time $T$;
17     end
18 end
19 return flow $f$ with $b(f) = a$.
```

positive residual balance from the sender, with starting balance $\beta = \min\{a - b(f), b^f_e\}$ and delay $\delta = d^1_e$; the parameter Π is an ordered list of all links, their expiration times and their backward delays along the path, and is initialized as $\Pi = [(e, \eta_e, d^2_e)]$. Each node, upon reception of the probe, also sends out a probe along each out-going link e with positive residual balance (except to the receiving neighbor), with updated probe $\rho' = (R, \beta', \delta', \Pi')$ where $\beta' = \min\{\beta, b^f_e\}$, $\delta' = \delta' + d^1_e$ and $\Pi' = \Pi \,\|\, (e, \eta_e, d^2_e)$ ($\|$ means appending). When the recipient receives a probe, first it checks for the timeliness constraint, *i.e.*, whether $st + \delta + \sum_{e \in p} d^2_e \leq dl$; second, it checks for availability of each link, *i.e.*, whether $st + \delta + \sum_{e \in p^+_e} d^2_e \leq \eta_e$. If either check fails, the recipient drops the probe and waits for the next one. Otherwise, the recipient returns a confirmation

$\gamma = (R, p, \beta)$ backward along p, to confirm the new augmenting path p and flow value β. Each node then updates its residual balances, and waits for the next round. The algorithm stops when the sender has collected sufficient flow value for the request. If the recipient cannot find an augmenting path in the current round, it informs the sender, who cancels all reserved flows and retry later. If the sender does not receive confirmation within T_{conf} of sending the probe, it also cancels all flows and retry later.

There are two things worth mentioning. First, the asynchrony of the network can affect the algorithm's performance. Ideally, we want each node to forward the probe that has the lowest cumulative processing time, in order to better meet the timeliness and availability constraints. However, such a probe may not be the first to reach an intermediate node or the recipient; probes from paths with longer processing times but shorter transmission delays may arrive first. To account for this, each intermediate node can wait for a certain period τ after receiving the first probe of the current round, before forwarding the probe with the lowest cumulative delay ever seen; the recipient can also wait before making a decision on path selection. The choice of τ is subject to each node's estimation of the network status, as well as the urgency of the request. On the other hand, probes may arrive out-of-order (w.r.t. rounds) in asynchronous networks. The sender can attach a round number *rnd* in each probe. Each node should discard probes in earlier rounds after a probe with higher *rnd* is received, because the recipient has already made a decision on the augmenting paths in the previous rounds.

The second thing is about concurrency. When multiple requests are jointly probing in the residual network, they may steal each other's flow by pushing flow through backward links that have reversed flow of other requests, a problem defined as *capacity stealing* by Rohrer *et al.* [107]. To address this, we apply their locking technique.

Algorithm 6.3: Intermediate node algorithm

```
– Forward direction operation:
  Input: probe $\rho = (R, \beta, \delta, \Pi)$, incoming link $e_{\text{in}}$
  Initialize: request list $\mathbf{R}$, flow $f_R$, residuals $\{b_e^f(R)\}$, current BFS round last
              hop $e_{\text{last}}(R)$
1  if $R \notin \mathbf{R}$ then
2      Add $R$ to $\mathbf{R}$, and create empty flow $f_R$ and $b_e^f(R)$;
3  end
4  for any out-going link $e \neq e_{in}$ such that $b_e^f(R) > 0$ do
5      Update probe $\rho = (R, \beta, \delta, \Pi)$ such that
           $\beta = \min\{\beta, b_e^f(R)\}$,
           $\delta = \delta + d_e^1$, and
           $\Pi = \Pi \,\|\, (e, \eta_e, d_e^2)$ ($\|$ means appending);
6      Send $\rho$ to neighbor along $e$;
7      Store last hop: $e_{\text{last}}(R) \leftarrow e_{\text{in}}$;
8  end

– Backward direction operation:
  Input: confirmation $\gamma = (R, p, \beta)$, incoming link $e$
9  if residual balance $b_e^f(R) \geq \beta$ then
10     Add path $p$ to $f_R$ with value $\beta$;
11     Update residual $b_e^f(R)$ based on Eq. (6.5);
12     Send $\gamma$ backward along link $e_{\text{last}}(R)$;
13 else
14     Send cancellation $\kappa = (R, p, \beta, \text{cancel})$ to $t$ along $p$;
15 end

– Cancellation operation:
  Input: cancellation $\kappa = (R, p, \beta, \text{cancel})$
16 Cancel previous update of $\beta$ of residual balance;
17 Send $\kappa$ to next hop along $p$;
```

Algorithm 6.4: Recipient algorithm

```
– Upon receiving probe:
  Input: probe $\rho = (R, \beta, \delta, \Pi)$
1 Form path $p$ from list $\Pi$;
2 if $\delta + \sum_{e \in p} d_e^2 > dl - st$ then drop $\rho$ and wait;
3 for $e \in p$ do
4     if $\delta + \sum_{e \in p_e^+} d_e^2 > \eta_e - st$ then drop $\rho$ and wait;
5 end
6 Construct confirmation $\gamma = (R, p, \beta)$;
7 Send $\gamma$ backward along $p$;
8 For subsequent probes of the same round who also pass Lines 2–5, save them
    into $\boldsymbol{\rho}_{\text{list}}$ until a next-round probe is received;

– Upon receiving cancellation:
9 if there is subsequent probe in $\boldsymbol{\rho}_{list}$ then
10    Pop the next probe $\rho$ with path $p$ and value $\beta$;
11    Construct confirmation $\gamma = (R, p, \beta)$;
12    Send $\gamma$ backward along $p$;
13 else
14    Inform sender $s$ of the failure.
15 end
```

Specifically, each node maintains a residual network for each request R, with flow f_R and residual balances $b^f_{(u,v)}(R)$ for $u, v \in V$. For each (u, v), the amount $f_R(v, u)$ is locked for request R only, while the rest $b^f_{(u,v)}(R) - f_R(v, u)$ can be shared by other requests. Therefore the residual balance in the concurrent case is defined as

$$b^f_{(u,v)}(R) = \begin{cases} b_{(u,v)} - \sum_{R' \in \mathbf{R}} f_{R'}(u, v) + f_R(v, u), & \\ \qquad\qquad \text{if } (u, v) \text{ or } (v, u) \in E; \\ 0, \qquad\qquad \text{otherwise.} \end{cases} \tag{6.5}$$

When a new augmenting path goes through a link with locked balance, it first consumes the locked balance before consuming any remaining link balance, thus allowing other requests to come through. Another race condition that may happen is when two or more requests find augmenting paths that share the same link simultaneously, in which case their joint augmented flow may exceed the link balance. In this case, each node will serve confirmations in a *first-come first-serve* (FCFS) manner; when a subsequent confirmation arrives that cannot be served, the node will send a cancellation to reverse all updated residual capacities at intermediate nodes and inform the recipient. The recipient can then select another path to augment, or if no path can be found, inform the sender to either initiate a new round of BFS or abort and retry later.

Hence, the only race condition that may happen is when multiple requests are simultaneously confirming paths that block each other, in which case each path's confirmation is cancelled. The algorithm resolves this by two methods. First, the recipient can select another path to confirm, until all received paths are simultaneously blocked by others, which is very rare under normal network conditions. Second, in the rare event of all paths being blocked, the recipient will inform the sender, who will cancel its reserved balances on all confirmed paths to let pass other requests, meanwhile employing *random back-off* to retry in a later time (before its start time). In summary, through flow locking, each request will enjoy dedicated balance once enough confirmations are received. Therefore no single request will block the network from progressing at any stage of the actual payments.

6.4.3 Discussions

Here we highlight a few things that should be considered when implementing the above mechanism.

Criterion for path selection: In the proposed mechanism, the sender initiates each probing round, while the recipient is responsible for selecting paths to confirm in each round. One thing that may affect the algorithm's performance is the criterion of selecting paths to confirm. In general, selecting short paths over long paths can reduce payment settlement time. On the other hand, selecting paths with larger balance can reduce the number of paths per payment, hence reducing the number of routing rounds, as well as the overhead and transaction fees during actual payment. Choosing the right criterion depends on the specific quality-of-service requirement of the actual use cases [128]. For example, requests with larger amounts may prefer widest paths to avoid long waiting time for routing and large fees, while smaller and more time-sensitive requests may prefer shortest paths to reduce settlement time.

Flooding avoidance in large networks: In large-scale networks, using BFS can lead to large flooding overhead. A common alleviating technique is to limit the hop count of each probe, *i.e.*, encoding a time-to-live (TTL) field in the message. The sender specifics the maximum TTL in the initial probe. Each node, upon reception of a forward probe, deducts the TTL by 1. If the TTL of a probe becomes 0, it will be dropped by the node that receives it. Advanced techniques can be employed to find the best TTL value to use in practice [16].

6.5 Performance Evaluation

6.5.1 Experiment Settings

To realistically evaluate our distributed mechanism, we developed a simulation tool based on network simulator `ns-3` [92]. The tool was developed as an application module in `ns-3`, which agrees with the actual standing of PCN protocol in real-world networks.

We used mesh topologies between users to model a PCN overlay network, where each node is connected to its PCN neighbors via direct communication links. Users were deployed in randomly generated Watts-Strogatz graphs [129]. The number of nodes varied from 50 to 250. Each node had a degree of 10, and each link had a re-wiring probability of 0.2 in the Watts-Strogatz model. We generated bi-directional payment channels with a uniform capacity of 100. However, each channel's initial balances on both directions were uniformly divided. Each channel's settlement times were uniformly generated in $[10, 50]$ seconds. Since our approach guarantees path availability, we assumed an arbitrarily large expiration time for all channels, to simulate the PCN in a static period of time; we focused on the timeliness constraint in our evaluation. We considered the transmission delays between nodes, which were uniformly generated in $[50, 200]$ ms. The data rate of the communication links were 100 Mbps.

In each simulation, we generated 1000 Poisson arriving requests between random nodes with mean arrival of 30 seconds. Requests had amounts uniformly generated in $[25, 75]$. We allowed 5 minutes for routing before the payment start time, and also a 5 minute payment deadline after start. Requests not routed before start time were cancelled and aborted.

6.5.2 Comparison Algorithms

CoinExpress has two versions: *CnExp-W* with widest path selection, and *CnExp-S* with shortest path selection.

We compared our algorithm to a state-of-the-art routing algorithm proposed by Rohrer *et al.* [107], which is based on the push-relabel algorithm for network flow. Before heading to the results, we first elaborate on a few issues of their algorithm.

First, their algorithm is *delay-agnostic*. Unlike our augmenting path algorithm where path lengths can be easily bounded, the push-relabel algorithm cannot encode delay information during flow updates. Second, their algorithm requires an additional step of *flow decomposition* after obtaining a flow, which incurs extra overhead and routing time. Our algorithm directly derives payment paths during probing. Later on, we also show that their algorithm results in an excessive number of paths using a standard flow decomposition (Edmonds-Karp algorithm). This greatly increases system overhead during the payment process, and leads to high transaction fees in pay-for-use PCNs.

We denote their algorithm as *PR-A*, which stands for *Push-Relabel in delay-Agnostic mode*. For reference, we also implemented *PR-D* (*Push-Relabel in Delay-aware mode*), where we enforced strict delay bound to paths generated by the standard flow decomposition. In addition, two more baselines were implemented: *WP* for one-round widest path routing, and *SP* for one-round shortest path routing. Both baselines used confirmation after probing to assure non-blockingness.

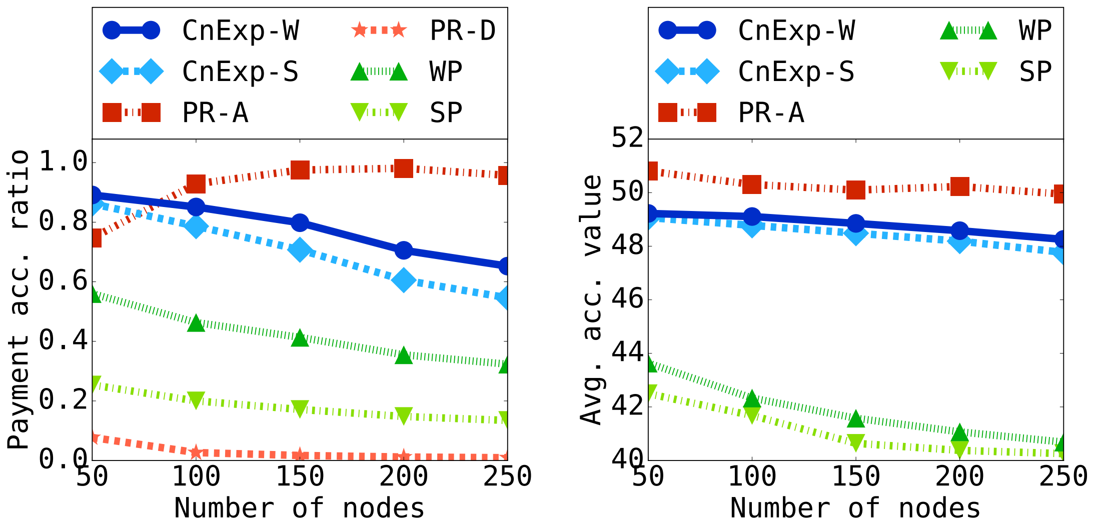
6.5.3 Performance Metrics

All algorithms achieve *timelock-compatibility* (except *PR-A*), *distributedness* and *concurrency*. We therefore mainly evaluated the *goodput* and *efficiency* of the algorithms. The following metrics were used:

- **Acceptance ratio:** number of accepted payments over all submitted requests.
- **Average accepted value:** the average amount of values of each accepted payment.
- **Payment delay:** the average payment delay of each accepted payment.

- **Routing time:** the average time consumed for routing for each accepted payment.
- **Number of messages:** the average number of network-wide messages for successfully routing a payment.
- **Number of paths:** the average number of payment paths for each accepted payment.

6.5.4 Evaluation Results



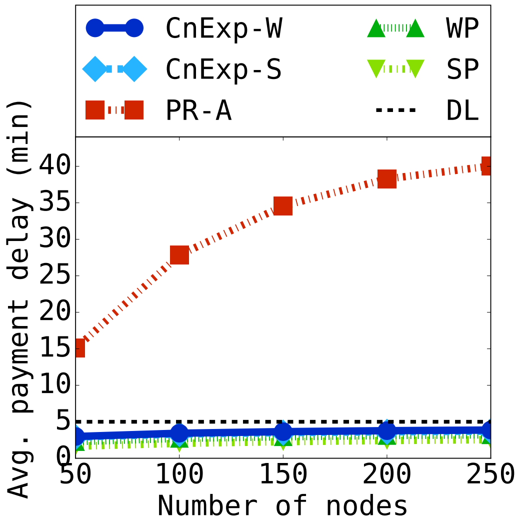
(a) Acceptance ratio (b) Average accepted value

Figure 6.2: Goodput (higher the better): acceptance ratio and average accepted value against number of nodes.

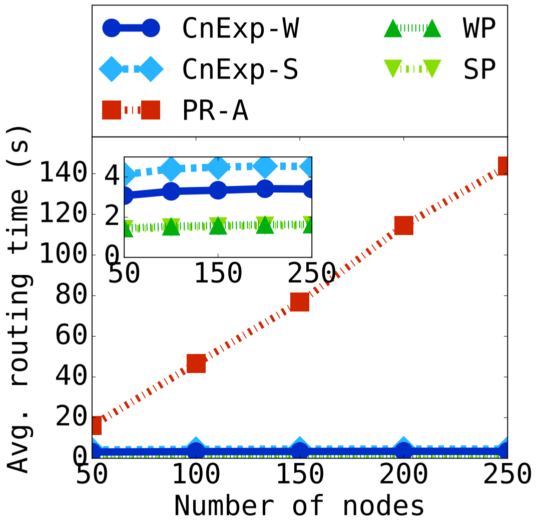
6.5.4.1 Goodput

Fig. 6.2 shows the acceptance ratios and average accepted values of the algorithms. From Fig. 6.2(a), CoinExpress algorithms achieve almost the highest goodput, except when compared to PR-A. This is because PR-A does not consider the timeliness constraints, hence although it achieves higher acceptance ratio, most accepted payments will fail due to deadline violation, as shown later. CnExp-W achieves slightly better goodput than CnExp-S, because the latter in general needs more paths due to each path carrying less value, where there may not be sufficient number of paths in some cases. All algorithms (except PR-A) has decreasing acceptance with increasing nodes. This is because although the nodes increase, the rewiring probability does not, which means longer paths between arbitrary node pairs. In other words, less paths that satisfy the timeliness constraints exist. PR-A has increasing acceptance because it neglects the deadlines. When enforcing timeliness on PR-A, we get the PR-D algorithm, which has extremely low acceptance. Less than 1% of the payments can be successfully settled using PR-D. Hence we neglect PR-D in the rest of our analysis due to insufficient samples for average analysis. The overhead of PR-D is very similar to PR-A, as their only difference is in the flow decomposition, which has a low overhead compared to the push-relabeling process.

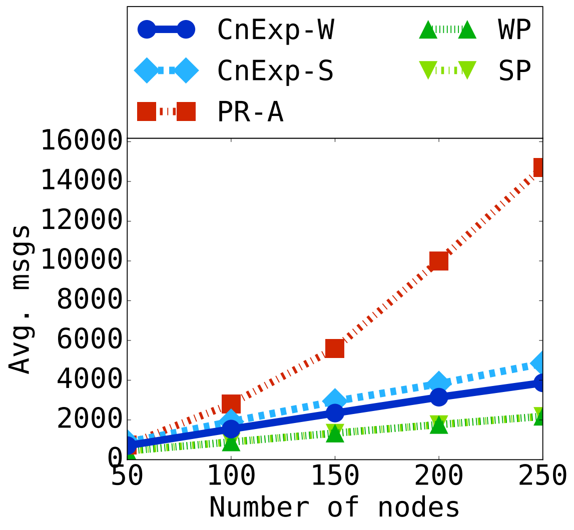
In Fig. 6.2(b), our algorithms not only accept more payments, but also accept payments with larger amounts, compared to WP and SP. CoinExpress achieve near-optimal average accepted values (the average is 50 based on our experiment setting), very close to the delay-agnostic PR-A algorithm.



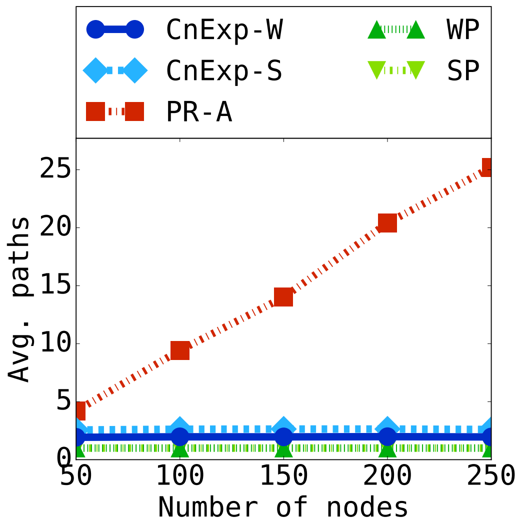
(a) Average payment delay



(b) Average routing time



(c) Average number of messages



(d) Average number of paths

Figure 6.3: Timeliness and efficiency (lower the better): average payment delay, routing time, number of messages, and number of paths. In (a), dotted line DL shows the uniform 5-minute deadline for all payment requests.

6.5.4.2 Timeliness and Efficiency

Fig. 6.3 further shows the timeliness status and efficiency metrics of the compared algorithms. First, in Fig. 6.3(a), we can observe the important timeliness measure of the algorithms. We can see that all algorithms except PR-A respect timeliness constraints (below DL). PR-A can result in payment delays of $8\times$ the deadline, severely violating users' fast payment requirements.

Fig. 6.3(b) shows the routing time of the algorithms. PR-A has extremely long routing times compared to the other algorithms. This may be a little counter-intuitive: Push-Relabel is regarded as a faster algorithm for maximum flow than Ford-Fulkerson. The reason is that here we are interested in finding a fulfilling flow rather than a max-flow, in which case Push-Relabel is slower due to its non-greedy pushing of flow in each step. Meanwhile, all others algorithms employed the flooding avoidance technique by setting a fixed TTL, hence their efficiency grows very slowly with increasing network size, demonstrating the great scalability of our algorithms.

Fig. 6.3(c) further shows the routing overhead in terms of the total number of routing messages per request. We can see that PR-A has a much larger overhead than CoinExpress, not to mention WP and SP. One reason is the flooding avoidance technique, which greatly restricts the overhead of our algorithms; however, Push-Relabel is hard to employ such techniques, resulting in large overhead for flow operations.

Fig. 6.3(d) shows the average number of payment paths output by each algorithm. As a baseline, both WP and SP have only one path. We can see that both CnExp-W and CnExp-S result in very few payment paths, typically around 2–3. Meanwhile, PR-A can result in as many as 25 different payment paths for a single payment. This reflects two things. First, this further explains why the routing overhead of

CoinExpress is much lower than PR-A: our algorithms require much fewer paths to be probed. Second, our algorithms have much lower payment overhead by employing only a few paths, which can result in much lower transaction fees in practice.

To summarize, our algorithm achieves very impressive acceptance goodput performance compared to all algorithms that consider user timeliness constraints, while achieving much lower overhead than the state-of-the-art Push-Relabel routing.

6.6 Related Work

6.6.1 Blockchain Scalability

Since the invention of blockchain in Bitcoin [88], extensive efforts have been devoted to improving the scalability of blockchain-based cryptocurrencies. Existing efforts can be divided into *on-chain solutions* and *off-chain solutions*. On-chain solutions focus on improving scalability by modifying existing blockchain design. A few promising techniques include increasing block size, using lightweight consensus algorithms, sharding [77], using Directed Acyclic Graph (DAG) instead of chain to store blocks [98], etc. Increasing block size directly increases a blockchain's capability to store and process more transactions, yet its direct threat is the fear of centralization. Lightweight consensus, such as Delegated Proof-of-Stake (DPoS) [65] or Practical Byzantine Fault-Tolerance (PBFT) [15], can greatly reduce overhead and increase scalability over the original Proof-of-Work (PoW) algorithm in Bitcoin and Ethereum. However, they either sacrifice decentralization (*e.g.*, DPoS) or require trust relationship between users (*e.g.*, PBFT). Sharding alleviates the scalability issue by dividing transactions into shards that are stored and processed at different nodes [77].

Block DAGs use weaker consensus where each transaction is only confirmed by a few instead of all up-coming blocks, which lowers block security.

Off-chain solutions seem more promising in solving blockchain scalability with limited compromise to its decentralization and security. One approach is to run multiple parallel blockchains that support cross-chain communications. Currently, the difficulty in this direction lies in the design of cross-chain communication protocols. Exchange-based protocols are the most popular at present, which uses one or multiple chains as cryptocurrency exchanges that bridge between all other chains; for example, see [117]. The problem is that the exchanges can be more vulnerable to attacks, which may endanger the entire exchanging system. Another proposal is a hierarchy of blockchains organized as a *blockchain tree*, where child chains are supervised and secured by parent chains [96]. It does not solve the attacks on chains, but instead constrains the loss due to attacks to the local chain only.

PCN is possibly the only mechanism that are totally *off-chain* for now. Here, most transactions are carried in the off-chain payment network, and does not involve the blockchain at all. The only involvement of the blockchain is either when opening or closing the channels, or when parties are non-cooperative in channel updates, when the blockchain is used as arbitration. Through protocols such as HTLC, PCN guarantees almost the same security as the original blockchain, while dramatically increasing its scalability. Moreover, PCN technology is among the most mature over all the above, since the leading two cryptocurrencies are already on the edge of deploying PCN for their global chains: the Lightning Network for Bitcoin [97], and the Raiden Network for Ethereum [102].

6.6.2 PCN and Routing

The PCN concept originates from the *credit/payment networks* in economics and finance [27]. Early credit networks do not have blockchains, hence they commonly rely on the trust relationship between peers to establish and maintain channel states [27]. Ripple [104] and Stellar [118] are among the first to employ blockchain technology in credit networks. Much like in PCN, routing in credit networks can only be done in a distributed manner due to decentralization. Malavolta *et al.* [79] studied privacy-preserving routing in credit networks, where they designed a landmark-based routing scheme for privacy-preserving distributed routing. This idea is extended by Roos *et al.* [108] to provide enhanced routing capability. However, landmark routing assumes a small set of *trusted* landmark users who controls the entire routing process, an assumption that is commonly not true, and if true can lead to centralization of the P2P network. For PCN, Prehodko *et al.* [100] first proposed a beacon-based routing scheme in the Lightning Network, borrowing from existing ideas in mobile ad hoc networks. Their proposal is a path-based routing scheme, and does not guarantee the fulfillment of the payment. Rohrer *et al.* [107] proposed a distributed push-relabel algorithm for PCN routing with guaranteed concurrency.

Aside from the routing problem, some related efforts in PCN include automatic channel re-balancing [64], privacy-preserving contracts [45], [80], etc. In general, PCN is a promising area of research, where extensive efforts are in need to address its performance, security and privacy issues.

6.7 Conclusions

In this study, we studied the routing problem in PCN. We distinguished a number of important design goals for PCN routing, and proposed a mathematical model to capture these goals. As a first step, we designed a distributed routing mechanism, which achieved all but the privacy goals. We showed through extensive simulations that the proposed mechanism achieves outstanding goodput performance and very small overhead compared to the state-of-the-art routing design. In our future work, we will further address the privacy issue in PCN routing, as well as other possible issues.

Chapter 7

P^{4}PCN: PRIVACY-PRESERVING PATH PROBING FOR PAYMENT CHANNEL NETWORKS

7.1 Introduction

Blockchain is a cryptographic mechanism that achieves security through decentralization. Designed to implement a distributed ledger, blockchain ensures data security through the consensus of distributed blockchain maintainers, so that no one can manipulate the ledger data without breaking a significant portion of the maintainers. Security of the blockchain largely depends on the size of the maintainer set. For this reason, cryptocurrency has been introduced both as a killer application and as the incentive mechanism, driving the general crowd to participate in maintaining the blockchain. Since the invention of the blockchain, thousands of cryptocurrencies have been developed, supporting numerous novel applications such as smart contract, supply chain, etc. The total capitalization of the cryptocurrency market tops at over \$800B, with even far more implicit economic impact in all business sectors.

As a basic functionality of cryptocurrency, however, digital payments are encountering a seemingly conflicting situation. On one hand, our conventional centralized financial infrastructure is efficient enough to handle billions of transactions every day, but suffers from the intrinsic financial risks and the lack of security and transparency. On the other hand, blockchain enhances security and eliminates the financial risk, at the cost of severely degraded efficiency and scalability due to the need for global consensus. This issue has drawn significant interests from both the academia and the industry.

The *payment channel network* (PCN) has emerged as a very promising solution to this problem, which combines the blockchain technology with the conventional credit network in economics [79], [87]. Specifically, users establish peer-to-peer channels with deposits, and transfer funds by adjusting the deposit allocation on the channels. Honest transactions are thus stacked in each channel, while only the final results are published onto the blockchain when the channels are closed. For security, each channel is protected by an on-chain smart contract, such that a dishonest off-chain behavior will be punished through on-chain arbitration. Therefore, expensive blockchain operations are limited to the establishment, close-out, and rare dispute arbitration for each off-chain channel. A well-connected network of payment channels can enable off-chain transactions for most payment scenarios, drastically improving the efficiency and scalability of the blockchain itself. Bitcoin and Ethereum, the two leading cryptocurrencies, are both deploying PCNs to scale their main blockchains [97], [102].

At its core, a PCN relies on routing to find payment paths with sufficient fund balances, and employs a multi-hop payment contract to secure indirect payments through the network [97]. The biggest challenge for PCN routing is the distributed and dynamic nature of the PCN, where channel balances are constantly changing with each on-going transaction. To improve routing success, many algorithms have employed probing-based techniques, which actively gathers up-to-date information from the network before making routing decisions [116], [127], [149]. It has been shown that probing-based solutions lead to significantly improved routing success rate compared to algorithms based on static or periodically updated information [116], [127].

A practical concern of digital payment users is the privacy of their transactions. Due to the transparency of the blockchain, it is intrinsically difficult to ensure strong anonymity for on-chain transactions. As a result, existing privacy-preserving blockchains only ensure pseudonymity of the transactions but not unlinkability [112]. PCN has a natural advantage over the blockchain for privacy, as most transactions are stacked within channels without being published. With the newly developed privacy-preserving payment contracts, information such as the identity and location of transacting users, as well as the transaction value, can be hidden from external adversaries and curious intermediate nodes [81], [93], [160]. Yet this does not solve the whole problem, as such information may also be leaked in the routing process. For example, many routing algorithms, such as the Spider network [116], Flash [127] and CoinExpress [149], rely on probing to improve routing performance over algorithms based on static information [79], [108], but does not have the anonymity properties of the latter ones.

In this study, we aim to resolve this one last piece of puzzle for privacy-preserving PCN. We propose P^4PCN, an anonymous path probing protocol for probing-based routing algorithms, which can be combined with the privacy-preserving payment contracts to construct a full protocol stack for privacy-preserving payments through the PCN. Compared to existing anonymous communication protocols such as onion routing [42], the biggest challenge for anonymous path probing is that *the sender may not know the path(s) that the probe will traverse in advance*. This violates the conventional requirement of knowing all the public keys of nodes on the path in anonymous communications, rendering all such protocols inapplicable, including but not limited to [18], [26], [42]. We address this by designing a novel cryptographic protocol. The key idea is to allow each intermediate node to both derive a symmetric key with the sender, and encrypt the queried data as well as necessary information for later decryption at the recipient, at the same time. A core technique in our construction is the Universal Re-encryption protocol used to re-encrypt the probe at each hop [43], while the symmetric key derivation is inspired by Sphinx [26] and HORNET [18]. Our protocol is both lightweight and scalable. We validated its efficiency and scalability via implementations and contrast experiments against a naive construction based on the *hybrid universal mixing* (HUM) protocol in [43].

Our main contributions are summarized as follows:

- To our best knowledge, no existing work has studied or addressed the anonymous probing problem in networking. We are the first to study and address this problem.
- We design a cryptographic protocol for anonymous probing that preserves sender and recipient anonymity, and ensures the integrity and confidentiality of queried data.
- We thoroughly analyze the security of our protocol, and validate its efficiency and scalability through implementation and contrast experiments.

The rest of this study is organized as follows. Section 7.2 presents our system model and security goals. Section 7.3 presents the detailed design of our protocol. Section 7.4 presents the security analysis of our protocol. Section 7.5 presents our performance evaluation results. Section 7.6 discusses how this protocol affects the routing algorithm design, as well as other potential applications of this protocol. Section 7.7 concludes this study.

7.2 System Model and Security Goals

7.2.1 System Model

We consider a fully distributed PCN denoted by $G = (V, E)$, where V is the set of user accounts that constitute the network, and E denotes the set of directional payment channels between nodes. Each channel $e \in E$ is associated with a set of channel status attributes, $\mathsf{attr}_e = \{\mathsf{balance}_e, \mathsf{delay}_e, \mathsf{expiration}_e, \mathsf{fee}_e, \dots\}$. Due to the dynamic nature of the network, some of the attributes, notably the balance of each channel, are constantly changing with on-going payment requests and transactions. As a result, each node only has up-to-date information regarding all the channels adjacent to it, while it has no knowledge of the instantaneous channel status of any remote channel.

A payment request is comprised as $(\mathsf{src}, \mathsf{dst}, \mathsf{val})$, where src and dst are the sender and recipient respectively, and val is the amount to be transferred. Some requests may have additional constraints, for example, a deadline dl, a fee budget cost, etc. Due to the primary constraint on val as well as these secondary constraints, the sender commonly needs to gather instantaneous information from the network to decide on its actual payment paths. Both information gathering and path selection are part of the *payment routing* process. If a guaranteed payment success is preferred, the sender can also reserve the balances on the path(s) until the payment is done, to avoid concurrency issues [149].

In this study, we primarily focus on the information gathering process, which we call *path probing*. In path probing, the sender sends out probing messages to gather information from network nodes. Each node attaches the queried information onto the probe, and then forwards the probe to one or multiple next hops, until each probe reaches the intended recipient. Note that since the sender has no knowledge of the remote channels, we assume that the actual choice of next hops is at the discretion of each forwarding node. For example, a node may either choose a simple broadcast-based method [149], or guide the selection of forwarding nodes with its local information, such as in imbalance-aware routing [116] or coordinate-based routing [108]. For generality, we do not rely on a specific probing algorithm, and assume that each node v independently decides the set of neighbors $\mathcal{N}_v$ to forward a received probe. We use data_v to denote the data that node v attaches to an on-going probe. Based on the request, data_v may contain balance, congestion, delay, fee, etc. Each node reports data of the same length l_{data}.

7.2.2 Threat Model

Existing probing-based routing algorithms do not consider the privacy of the sender and/or the recipient. For example, CoinExpress [149] explicitly involves the sender and recipient nodes in its probes. In this study, we focus on an adversary who tries to infer the payment patterns of senders and/or recipients in the network. For example, observing a sender sending out a probe that passes through several (corrupted) nodes, an adversary can link this action with a future anonymous payment transaction that goes through the same set of nodes. The popularity of a recipient may also be inferred by observing how many probes are targeting a recipient during a period.

We consider a local adversary that controls a subset of nodes by either inserting malicious nodes or compromising existing nodes. We assume the non-existence of a global adversary that can observe all network traffic, as all communications between peers are conducted via secure and anonymous channels. The adversary can access all the stored secrets and past communications on the compromised nodes, but cannot access such information on non-compromised nodes. For any privacy-concerning user, we assume that the adversary cannot compromise (or does not know if it has compromised) all the adjacent nodes of the user; otherwise, the sender/recipient's privacy can be trivially broken since the adversary can access all the user's incoming/out-going channels. Note that this assumption realistically holds in PCNs that support private channels [105]. The goal of the adversary is to undermine user privacy instead of launching denial-of-service attacks. Defense against denial-of-service attacks is mainly through detection and prevention, which is out of the scope of this study.

7.2.3 Security Goals

We expect our protocol (or any other anonymous probing protocol) to fulfill the following security goals.

- **Correctness:** Correctness means that a cryptographic protocol correctly implements all the functions of a normal non-cryptographic protocol. In our probing problem, this means that 1) each node (both intermediate node and recipient) can identify its role regarding a received probe, 2) each intermediate node is able to attach queried information onto the probe, and 3) the recipient can obtain all the attached information from the probe.
- **Data Integrity:** The adversary cannot break the integrity of the queried data attached by a non-compromised node without being detected by the sender/recipient.

- **Data Confidentiality:** The adversary cannot access the information attached by a non-compromised node, except for the knowledge that it already has access to, *e.g.*, statuses of channels adjacent to a compromised node.
- **Sender Privacy:** For any payment request between non-compromised users, the adversary cannot infer the identity or location of the sender. It also cannot decide if a non-compromised node is the sender of any request.
- **Recipient Privacy:** For any payment request between non-compromised users, the adversary cannot infer the identity or location of the recipient. It also cannot decide if a non-compromised node is the recipient of any request.
- **Sender-Recipient Privacy:** The adversary cannot decide whether there is any on-going request between any users.

7.3 Protocol Design

7.3.1 Preliminaries

Let $\mathcal{G}$ be a cyclic group of prime order q (with length l_{key}), satisfying the Decisional Diffie-Hellman (DDH) assumption. Let g be a published generator of $\mathcal{G}$. We omit writing the modulus operation for brevity. We use the symbol $\perp$ to denote an empty string, group element or identifier of an arbitrary size. The following cryptographic primitives are used in our protocol:

- $\mathsf{E}(pk, m)$: encryption of message m with public key pk.
- $\mathsf{D}(sk, \xi)$: decryption of ciphertext ξ with private key sk.
- $\mathsf{PRG}(s)$: a secure pseudorandom generator with key s.

- $\mathsf{MAC}(k, m)$: the Message Authentication Code (MAC) of message m under key k, with length l_{MAC}.
- $\mathsf{H_M}(\cdot)$, $\mathsf{H_E}(\cdot)$: two cryptographic hash functions.

7.3.2 Universal Re-encryption (URE)

Universal Re-encryption (URE) is a cryptographic protocol for mixnets proposed by Golle *et al.* [43]. In plain words, URE enables mix nodes to re-encrypt an encrypted message *without* knowing the public key used for encryption. The original construction of URE was built upon the ElGamal cryptosystem, utilizing the homomorphic property of ElGamal. Let $(x, y = g^x)$ be a private-public key pair for ElGamal encryption where x is the private key. The ciphertext of message $m \in [1 \dots q-1]$ is $\xi = \mathsf{E}(y, m) = (m \cdot y^k, g^k)$ where $k \in [1 \dots q-1]$ is a secret random number, and the decryption algorithm runs as $m = \mathsf{D}(x, \xi) = \xi[0] \cdot (\xi[1]^x)^{-1}$. Given two ciphertexts $\mathsf{E}(pk, a)$ and $\mathsf{E}(pk, b)$ encrypted under the same key pk, the ElGamal cryptosystem satisfies that $\mathsf{E}(pk, a) \times \mathsf{E}(pk, b) = \mathsf{E}(pk, a \times b)$ for a group operator $\times$. Based on this, a ciphertext in the URE protocol has two components, $\mathsf{E}(pk, m)$ and $\mathsf{E}(pk, 1)$, where the latter can be used to re-encrypt the former without knowing the public key initially used to encrypt it.

Note that the task of the original URE protocol is the opposite of ours. The original URE aims to anonymously transmit a message from sender to receiver through a sequence of known mix nodes, such that no mix node has the knowledge of the sender, the receiver, or any node other than itself and its neighbors on the route. The message itself stays fixed and encrypted through the entire transmission. In our task, not only the path that a probe will traverse is undetermined before probing, but each intermediate node also needs to attach data (local channel information) queried by the sender. Each node's data must be kept secret so that other nodes cannot infer whether this node is on the probed path or not. On the other hand, each node needs to provide enough information for the sender/recipient to decrypt the attached data. To address these, we must modify the original URE protocol to fit our needs.

7.3.3 Anonymous Probing with Unknown Paths

Our insight is that, besides using URE for re-anonymization as in mixnets, we can also use part of the URE protocol to carry out a Diffie-Hellman Key Exchange (DHKE) with each node that the probe traverses. The derived key can then be used to encrypt the attached data of the intermediate node, while the node can provide necessary DH-value to help the recipient decrypt the encrypted data. To ensure anonymity, decryption-related information and the data are encrypted within a *reversed onion*: each node wraps a layer of encryption over the received payload plus the newly attached DH-value and data. The recipient gradually derives all the symmetric keys, and uses the corresponding key to "peel off" each encryption layer, revealing all the attached data layer-by-layer.

7.3.3.1 Probing with a Single Path

For simplicity, let us first consider the case where probing is done through just one (unknown) path. Here, each node knows which exact neighbor it will forward a given probe to as next hop. But the sender does not know the public key of any node on this path. Assume this path is represented by $(\mathsf{src} = v_0, v_1, v_2, ..., v_{n-1}, v_n = \mathsf{dst})$. We also assume that the sender and the recipient securely share any information that either of them uses in the protocol. We thus do not distinguish between the cryptographic operations by the sender or the recipient. The probing protocol works as follows:

```
Algorithm 7.1: Create Probe (Sender)
  Input: Probe ID $I$.
  Output: Initial probe message $\rho_0$.
1 Generate random $x, \kappa, k_\beta, k_\gamma \in_U \mathcal{Z}_q$; let $y \leftarrow g^x$;
2 $s_0 \leftarrow g^{\kappa k_\beta}$;
3 $m \leftarrow (pk_0, \perp, \perp)$;
4 $\mathsf{pl}_0 \leftarrow (\xi_0, \mathsf{MAC}(\mathsf{H_M}(s_0), \xi_0), \zeta_0)$, where
      $\xi_0 = m \oplus \mathsf{PRG}(\mathsf{H_E}(s_0)), \zeta_0 = \perp \oplus \mathsf{PRG}(\mathsf{H_E}(s_0))$;
5 Generate probe $\rho_0 = (a_0, b_0, c_0, d_0, \mathsf{pl}_0)$, where
      $a_0 = y^{k_\beta k_\gamma}$, $b_0 = g^{k_\beta k_\gamma}$, $c_0 = g^{k_\gamma}$, $d_0 = g^{\kappa}$;
  return $\rho_0$.
```

Create Probe: The sender generates secrets $(x, \kappa, k_\beta, k_\gamma)$ that are shared with the recipient via a secure channel. The probe contains the ElGamal ciphertext $(a_0, b_0) = \mathsf{E}(y, 1)$, an element c_0 containing part of b_0, a random element d_0, and the payload pl_0. Besides being part of the ciphertext, b_0 acts as the DH-value of the sender, used to establish a DH symmetric key with each hop. c_0 and d_0 are both used for per-hop DHKE, so that the recipient can derive the symmetric keys. Specifically, c_0 is used to pass the common part of the per-hop DH-value to the next hop for constructing the next hop's own DH-value, while d_0 is used to pass the current node's DH-value to the next hop for encryption into the payload. The DHKE with v_i is based on secret keys k_β and x_i (and common randomizing terms), and for sender, though purely for formality, $x_0 = \kappa$. Both k_γ and κ are to make the initial probe indistinguishable from the ones that have already passed some hop(s). Note that both the symmetric key encryption using PRG and the encrypted empty string ζ_0 are to facilitate padding, discussed later on.

Algorithm 7.2: Process Probe (Intermediate Node)

```
  Input: Node $v_i$, probe $(a, b, c, d, \mathsf{pl})$, $\mathsf{data}$.
  Output: Next probe message $\rho_i$.
1 Decrypt $(a, b)$ using its own key pair to see if it is the recipient, and jump to
    Algorithm 7.3 if so;
2 Generate random $k_i, x_i \in_U \mathcal{Z}_q$;
3 $s_i \leftarrow b^{x_i}$;
4 $m \leftarrow (pk_i, d, \mathsf{data})$;
5 $\mathsf{pl}_i \leftarrow (\xi_i, \mathsf{MAC}(\mathsf{H_M}(s_i), \xi_i), \zeta_i)$, where
      $\xi_i = m \oplus \mathsf{PRG}(\mathsf{H_E}(s_i)), \zeta_i = \mathsf{pl} \oplus \mathsf{PRG}(\mathsf{H_E}(s_i))$;
6 Construct probe $\rho_i = (a_i, b_i, c_i, d_i, \mathsf{pl}_i)$, where
      $a_i = a^{k_i}$, $b_i = b^{k_i}$, $c_i = c^{k_i}$, $d_i = c^{x_i}$;
  return $\rho_i$.
```

Process Probe: Each intermediate node's procedure starts from checking if it is the recipient. This is done by trying to decrypt the ciphertext $(a, b) = \mathsf{E}(y, 1)$ using its own set of private keys whose corresponding requests are expecting in-coming probes. If the node is not the recipient, it proceeds to generate the next probe to be forwarded. The first step is to derive its shared DH key s_i using b as the DH value of the sender. s_i is then used to encrypt a new payload, which contains the previous payload, the old element d as the DH-value of the previous hop (which is essential for the recipient to decrypt the previous payload), and the queried data to be attached. Next is to update all the group elements. The ciphertext (a, b) is re-encrypted using new key k_i. The common part of the DH-value, c, is also updated by key k_i to reflect the update on b. d_i stores the DH-value of the current node, which will be either wrapped within the payload at the next hop if the next hop is still intermediate, or used for decryption if the next hop is the recipient.

Decrypt Probe: Once a recipient receives a probe targeting itself, it starts decrypting the onion-encrypted payload to obtain all the data. For each layer, the secret key is derived by combining the corresponding node's DH-value, either directly transmitted

Algorithm 7.3: Decrypt Probe (Recipient)

```
   Input: Probe $(a, b, c, d, \mathsf{pl})$, shared keys $x, k_\beta$.
   Output: Path $p$, per-hop data $\mathbf{data} = \{\mathsf{data}\}$.
1  while $\mathsf{sizeof}(\mathsf{pl}) \geq$ valid payload segment length do
2      $s \leftarrow d^{k_\beta}$;
3      $(\xi, \nu, \zeta) \leftarrow \mathsf{pl}$;
4      Check if $\nu$ is a valid MAC of $\xi$ under $s$; abort if not;
5      $(pk, d', \mathsf{data}, \mathsf{pl}') \leftarrow (\xi, \zeta) \oplus \mathsf{PRG}(\mathsf{H_E}(s))$;
6      Add $pk$ to $p$, and add $\mathsf{data}$ to $\mathbf{data}$;
7      $d \leftarrow d'$, $\mathsf{pl} \leftarrow \mathsf{pl}'$;
8  end
9  return $(p, \mathbf{data})$.
```

from the last hop or obtained from the last layer (for previous hops), with the secret k_β of the sender/recipient. The secret key is then used to decrypt the payload, revealing the next layer of payload and DH-value, and the attached data of this layer. Eventually, the path is reconstructed from the attached public keys, and all data are decrypted layer-by-layer.

7.3.3.2 Probing with Multiple Next Hops

Now, let us consider the case where each node can forward a probe to multiple neighbors to increase the probing success probability and path diversity. Forwarding the same probe message to multiple neighbors would enable linkable attacks, if two colluding nodes both receive the same message. For this reason, a new pair of (k_i, x_i) should be generated for every neighbor that the probe is sent to. Due to the re-encryption and the onion encryption at every hop, even if colluding nodes receive the probe sent by the same sender, they cannot relate them or distinguish them from any other probe in the network.

7.3.3.3 Length-based Inference Attacks and Padding

One issue with the original onion routing protocol is that an intermediate node can infer its relative location along the path by observing the length of the encrypted message, if it knows or can estimate the length of the original message and/or the size of each layer. To prevent such an attack, padding is commonly used to keep all encrypted messages of constant length. In our scenario, padding can serve an additional purpose. In payment routing, commonly the sender/recipient has a limit on the length of an acceptable payment path. If a path is too long, both the risk of a failed payment is high, and it may incur a high transaction fee at the sender. For this reason, the sender can pad the onion-encrypted payload up to a pre-defined limit. Each node attaches information by dropping the last π bits of the previous onion, where π is the size of a layer of the payload and is known based on the message format. The recipient can then detect paths exceeding the pre-defined length, if the last layer of encryption does not contain the initial payload sent by the sender, and discard such paths accordingly.

7.4 Security Analysis

7.4.1 Correctness

First, each node v_i can identify whether it is the recipient of a given probe by trying to decrypt the URE ciphertext (a, b) against its own set of key pairs that are awaiting probes. If the decrypted value for any of the key pairs is 1, the probe belongs to the corresponding probing request. During probing, each node attaches its local data onto the probe by encrypting it with the secret key s_i derived using DHKE with the sender's provided DH-value (element b in the URE ciphertext). Finally, the recipient opens each layer of the onion by deriving s_i. Note that for the i-th hop ($i > 0$), it satisfies that $b_i = c_i^{k_\beta}$ as well as $d_i = c_{i-1}^{x_i}$. Therefore, we have $s_i = b_{i-1}^{x_i} = c_{i-1}^{k_\beta x_i} = d_i^{k_\beta}$. Since d_i is always encrypted in the onion of the next hop (or directly transmitted to recipient at last hop), the recipient (knowing the shared secret k_β) can gradually derive the secret keys to decrypt all layers and obtain all the data. Note that the recipient does not need to decrypt the innermost layer, as any information there can be directly shared between the sender and the recipient, and hence s_0 is purely used to achieve indistinguishability.

7.4.2 Data Integrity and Confidentiality

Since each piece of data attached to the probe is MACed and onion-encrypted, it suffices to show that the secret key s_i of a non-compromised node cannot be obtained by an adversary if both the sender and the recipient are non-compromised. Note that the secret key s_i contains two secret components, the x_i held by the non-compromised node v_i, and the $k_\beta k_\gamma$ initially embedded by the sender. Without either of these two pieces or the key itself (which is discarded after use), no adversary can derive s_i based on the DDH assumption.

7.4.3 Sender, Recipient, and Sender-Recipient Privacy

Regarding sender and recipient privacy, we consider several cases. First, a compromised node not adjacent to either the sender or the recipient can only see some probe passing through, but cannot link it to either the sender or the recipient due to the encryption. A node adjacent to the sender may try to infer if the previous hop is the sender, but cannot tell since some other node that connects to the last hop directly or via a path may have originated the probe. Similarly, a node adjacent to the recipient cannot tell whether the probe is targeted to the next hop or some other node afterwards. Furthermore, a node cannot know its own location with regard to either the sender or the recipient in the network, since all probes are of the same length and are re-randomized at every hop. Finally, since the adversary cannot tell whether any probe can be linked to any sender or recipient, he also cannot tell whether a pair of sender and recipient is communicating or not.

7.5 Performance Evaluation

In this section, we evaluate the performance of our protocol. Since we are the first to propose a protocol for the anonymous probing problem in networks, our only reference solution is a modified version of the *hybrid universal mixing* (HUM) protocol described in the original URE paper [43], which uses a different way for establishing symmetric keys used for encryption. Below, we first describe the modified HUM protocol, and then analyze the performance of these two protocols.

7.5.1 Modified Hybrid Universal Mixing [43]

In the HUM protocol, the authors proposed to use symmetric key encryption to encrypt and re-encrypt the initial message, while using the URE protocol to deliver the symmetric keys used by each hop for final decryption. This protocol can be modified to additionally use the symmetric keys to encrypt and re-encrypt the data each node attaches to the probe. Assume that the maximum path length is L. The sender sends out the following message as the initial probe:

$$\rho_0 = (\mathsf{pl}, \mathsf{E}(pk, 1), \mathsf{vec}),$$

where $\mathsf{pl} = (\xi, \mathsf{MAC}(\mathsf{H}_\mathsf{M}(s_0), \xi), m_\mathsf{rand})$, $\xi = (pk_0, \perp) \oplus \mathsf{PRG}(\mathsf{H}_\mathsf{E}(s_0))$, m_rand is a random filler string of length $(L-1) \cdot l_\mathsf{seg}$ with $l_\mathsf{seg} = l_\mathsf{data} + l_\mathsf{key} + l_\mathsf{MAC}$, and

$$\mathsf{vec} = (\mathsf{E}(pk, 1), \ldots, \mathsf{E}(pk, 1), \mathsf{E}(pk, s_0))$$

contains $(L-1)$ copies of the encrypted value 1 plus an encrypted copy of the initial secret key s_0.

When a node v_i receives a probe, it attaches its pk_i and data to the head of the payload while dropping the last l_seg bits, picks a new key s_i, and then encrypts the new payload with s_i; the MAC is also computed and attached. It then encrypts s_i with the help of the value $\mathsf{E}(pk, 1)$ at the head of vec (if the path does not exceed length L), and then left-rotates all the components in vec. This can be done by multiplying the first element of $\mathsf{E}(pk, 1)$ with s_i in ElGamal encryption. Finally, it re-encrypts the middle component $\mathsf{E}(pk, 1)$ in the probe, as well as all the other components in vec, using a new random key k_i. If there are more than one next hop, then as in our protocol, new keys need to be generated, and the entire message re-randomized, for each of the next hops of the current node.

Identification of the recipient is also done by decrypting the middle component $\mathsf{E}(pk, 1)$. After that, the recipient decrypts all the secret keys stored in vec from the back, and uses all the keys to gradually decrypt the payload and obtain all the data.

7.5.2 Analytical Comparison

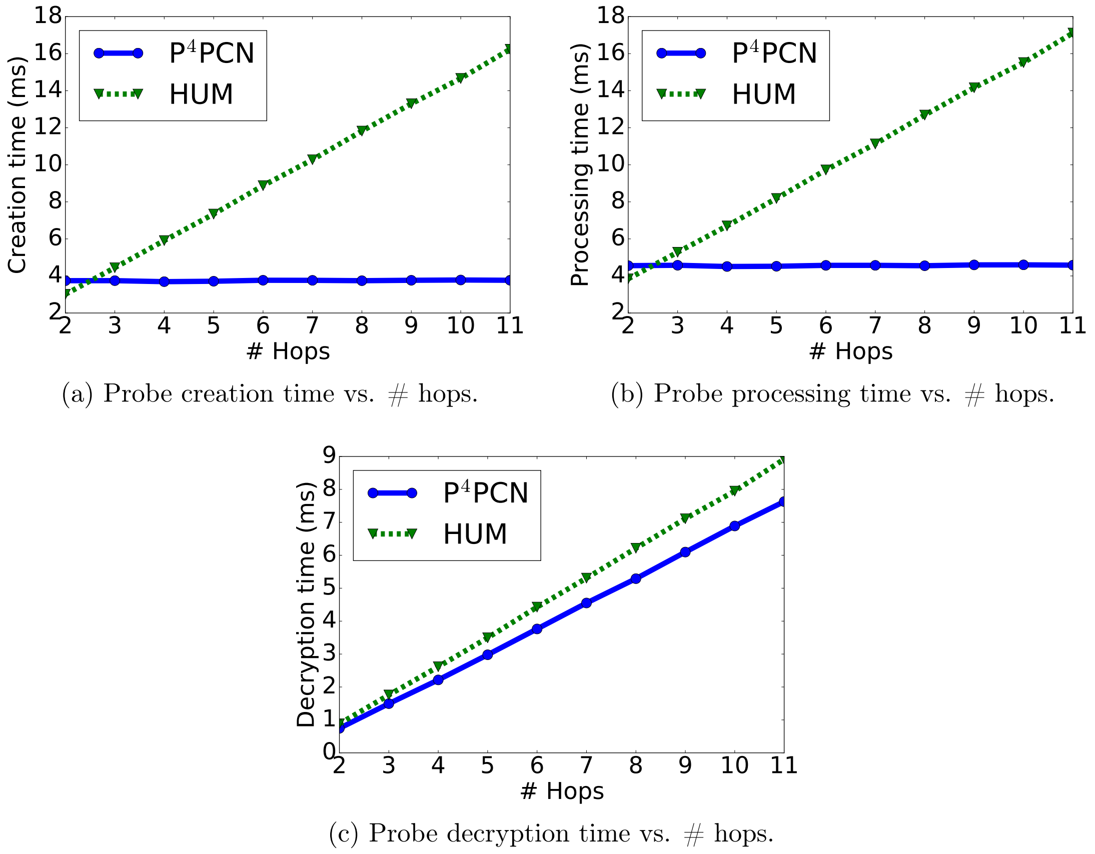
Here we analytically compare the overheads of our protocol and the modified HUM protocol. We assume that the network has a path length limit of L intermediate nodes, and each probe has traversed a path of length $P \leq L$ on average. Size of a group element including public/private keys is $l_{\mathsf{key}} = K$ bytes. We focus on the overhead for processing 1 probe at each node, as the number of probes a node receives/forwards is determined by the actual probing algorithm employed, which is out of the scope of this study. We omit the generation of all public/private key pairs. Table 7.1 shows an analytical comparison between the two protocols.

Scheme		Our Protocol	Modified HUM
Comp.	Sender	$(5, 0)$	$(2L+2, 0)$
	Interm.	$(6, 1)$	$(2L+3, 1)$
	Recipient	$(P+1, 1)$	$(P+1, P)$
Communications		$(2L+4)K$	$(3L+2)K$

Table 7.1: Overhead comparison for one probe. Computation is displayed as $(\mathsf{mod\_exp}, \mathsf{mod\_inv})$, denoting the numbers of modular exponentiation and modular inverse operations respectively; communication overhead is measured by the length of the probe header excluding data payloads and MACs but including the path information (public keys of each hop).

7.5.3 Simulation Experiments

We implemented both protocols in Java. The ElGamal key size was 1024 bits (128 bytes). A data segment contained 4 bytes of data and a 128-byte public key identifying this hop. The MAC of a data query was 20 bytes using the HMAC-SHA-1 implementation of Java. The SHA1PRNG implementation of Java is used for the keyed PRG in order for reproducibility of the experiments. Each probe traversed the maximum allowed hops before reaching the recipient. We ran our experiments on a Linux PC with Quad-Core 3.4GHz CPU and 16GB of memory, and repeated each experiment for $10,000$ times.



(a) Probe creation time vs. # hops.

(b) Probe processing time vs. # hops.

(c) Probe decryption time vs. # hops.

Figure 7.1: Probe creation (sender), processing (intermediate node) and decryption (recipient) execution times per probe.

Fig. 7.1 shows the average execution times for the creation, processing and decryption of a probe, respectively. Clearly, the creation and processing times of our protocol remain constant with increasing number of hops the probe can explore, while those of HUM increase linearly. For decryption, both protocols have an increasing execution time, but ours is faster than HUM due to less number of modular inverse operations.

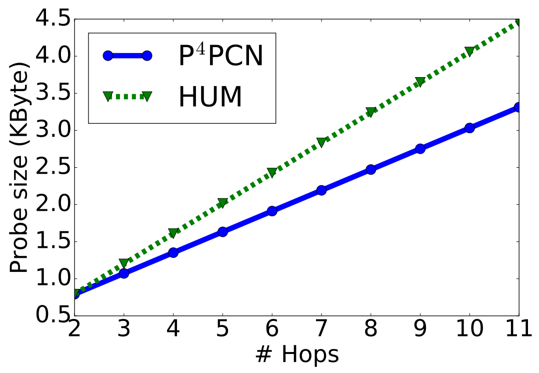


Figure 7.2: Probe size vs. # hops.

Fig. 7.2 shows the probe size versus the number of hops. HUM has a larger probe size over ours due to the more number of ElGamal ciphertexts used to store all the encryption keys, resulting in higher communication overhead in practice.

7.6 Discussions

Probing with anonymity: With our protocol, now the intermediate nodes will have no way of knowing the sender and/or the recipient. Unfortunately, this complicates the probing algorithm design, because each node now has limited information in deciding where to forward a probe. For instance, simple broadcasting can lead to the case where every copy of a probe will wander in the network until reaching the recipient, and all the probes generated by a request will most likely overwhelm the entire network. Two promising solutions are probabilistic forwarding and coordinate-based routing. The former naturally preserves privacy, while the latter can be implemented using a privacy-preserving coordinate system to avoid breaking user anonymity [108]. We plan to investigate these in our future work.

Other applications: Beyond PCN, our protocol may also find applications in a wide range of other scenarios. For example, this protocol can be used to anonymously construct a communication path towards a remote location through a dynamic sensor network or a vehicular network, or to find a trust path in a trust-based social network.

7.7 Conclusions

In this study, we studied the problem of anonymous probing in PCN routing, the last piece in building a fully privacy-preserving PCN payment protocol stack. We proposed a cryptographic protocol to ensure both data security and user privacy during probing-based network information gathering. Combined with existing probing-based dynamic routing algorithms and privacy-preserving payment protocols, this can achieve efficient payments with high success probability and full privacy preservation for PCN. Our protocol is both lightweight and scalable. Comparing our protocol with another possible protocol derived from hybrid universal mixing, our protocol has constant probe creation and processing overheads, lower (although linear) probe decryption overhead, and smaller communication overhead. Though we specifically targeted the PCN, our protocol can be used in a broad range of other domains, such as for anonymous data querying in vehicular networks, sensor networks, social networks, and beyond.

Chapter 8

CONCLUSIONS

There is no doubt that IoT will transform the world. However, challenges exist towards such a goal. At this time, the resource mismatch between IoT infrastructure and applications is greatly hindering the applicability and acceptance of this entire technology. In this dissertation, we studied smart resource allocation as an immediate remedy to the resource issue in IoT, enabling best utilization of the existing resources to provide performance improvements and guarantees for the IoT operator, applications and users. In general, we explored three different dimensions of smart resource allocation: 1) network resource allocation, 2) security deployment, and 3) micropayment transactions.

Specifically, for the network resource allocation problem, we studied different problems involving different layers in the network stack. The first problem considered routing and engineering IoT data traffic in the network layer. We addressed the challenge of using existing resources to meet all the traffic flows' bandwidth, delay, reliability and policy routing requirements. Since the problem was NP-hard, we proposed an FPTAS which enforces all above requirements, while approximating the minimum load factor on the links. The second problem considered both application hosting and traffic routing and engineering. We considered one or multiple IoT applications receiving data streams from distributed data sources. Each application could be parallelizable or non-parallelizable, and we considered either one or multiple applications. We still considered the QoS (bandwidth, delay and reliability) requirements of the applications. This problem was a generalization over the previous one, and we again proved its NP-hardness. Since there were four variants, we proposed FPTASs for three of the variants, while for the forth one we proposed a randomized algorithm based on one of the FPTASs. The third problem considered a distributed application scenario with edge-deployed IoT microservices and an overlay network interconnecting these microservices. We studied the microservice load balancing problem within this overlay network, featuring a general model that abstracts the complex interdependencies among microservices using a DAG. Of course, this problem was still NP-hard, and hence we proposed another FPTAS for this problem, this time enforcing the load balancing goal while approximating the worst-case QoS of the distributed application. We used extensive simulation experiments to validate the superior performance of all our proposed algorithms, compared to other heuristic solutions for each problem.

For the security problem, we studied how the IoT provider can optimally deploy security functions given a limited cost budget. We used a stochastic model to capture the dynamics in the IoT edge network, and used the CVaR abstraction to capture the worst-case security risk of the system in face of the dynamics. An optimization framework was proposed for the robust security deployment problem. We further proposed optimization techniques to efficiently find the optimal deployment plan. While we focused on optimizing the security risk under a simple static risk model and based on shortest path routing, our optimization framework in general can incorporate many other risk models and routing algorithms, and hence has much wider applications than this specific context.

For the micropayment problem, we made two major contributions. The first one was a distributed routing protocol that efficiently finds payment paths in a blockchain-based PCN. This could enable fast and cost efficient transactions that could support machine-to-machine payment transactions in IoT. We further proposed an anonymous protocol for path probing in PCN. This protocol novelly combined Universal Re-encryption with the HORNET packet format, in order for a probe message to anonymously carry sufficient information submitted by intermediate nodes to establish a payment path. These two contributions, combined with the anonymous payment processing protocol in PCN, can yield a high-performance and privacy-preserving payment stack for machine-to-machine IoT micropayment transactions, which is crucial in constructing a fully automated IoT ecosystem.

While the above contributions tackle different aspects of IoT, jointly they show that smart resource allocation is a genuine and practical approach to addressing the resource issue within the entire IoT ecosystem, specifically those network-related ones. Continuous investment into the IoT infrastructure is the eventual solution to the resource issue. However, these algorithms will still be crucially useful in addressing the specific performance demands of future applications. The goal of this dissertation, in addition to providing algorithmic solutions to specific problems in certain contexts, is also to propose general and extensible models and frameworks that are able to handle changes in the infrastructure, the applications, and the user patterns. On the other hand, more work also needs to be done along the line, as more complicated infrastructural technologies and applications are constantly being developed in IoT. In the future work, part of our focus will be on extending our existing solutions to new infrastructures and applications, including but not limited to hierarchical edge networks, mobile augmented/virtual reality, and machine learning-based IoT applications.

REFERENCES

[1] S. Acharya, B. Gupta, P. Risbood, and A. Srivastava, "PESO: Low Overhead Protection for Ethernet over SONET Transport," in *Proc. IEEE INFOCOM*, 2004, pp. 165–175.

[2] S. Akkermans, N. Small, W. Joosen, and D. Hughes, "Demo: Niflheim: End-to-End Middleware for Applications Across All Tiers of the IoT," in *Proc. ACM SenSys*, 2017, pp. 1–2.

[3] M. Alizadeh, N. Yadav, G. Varghese, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, and R. Pan, "CONGA: Distributed Congestion-Aware Load Balancing for Datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.

[4] H. M. Almohr, L. T. Watson, D. Yao, and X. Ou, "Security Optimization of Dynamic Networks with Probabilistic Graph Modeling and Linear Programming," Tech. Rep., 2014.

[5] G. Attiya and Y. Hamam, "Two Phase Algorithm for Load Balancing in Heterogeneous Distributed Systems," in *Proc. IEEE EMPDP*, 2004, pp. 434–439.

[6] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions," in *Proc. IEEE CNSM*, 2015, pp. 50–56.

[7] S. Basudan, X. Lin, and K. Sankaranarayanan, "A Privacy-Preserving Vehicular Crowdsensing-Based Road Surface Condition Monitoring System Using Fog Computing," *IEEE Internet Things J.*, vol. 4, no. 3, pp. 772–782, Jun. 2017.

[8] L. Belli, S. Cirani, G. Ferrari, L. Melegari, and M. Picone, "A Graph-Based Cloud Architecture for Big Stream Real-Time Applications in the Internet of Things," in *Proc. ESOCC*, 2014, pp. 91–105.

[9] J. F. Benders, "Partitioning Procedures for Solving Mixed-Variables Programming Problems," *Numer. Math.*, vol. 4, no. 1, pp. 238–252, Dec. 1962.

[10] E. Bin, O. Biran, O. Boni, E. Hadad, E. K. Kolodner, Y. Moatti, and D. H. Lorenz, "Guaranteeing High Availability Goals for Virtual Machine Placement," in *Proc. IEEE ICDCS*, 2011, pp. 700–709.

[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proc. ACM MCC*, 2012, pp. 13–16.

[12] Z. Cao, P. Claisse, R. J. Essiambre, M. Kodialam, and T. V. Lakshman, "Optimizing Throughput in Optical Networks: The Joint Routing and Power Control Problem," in *Proc. IEEE INFOCOM*, 2015, pp. 1921–1929.

[13] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic Steering in Software Defined Networks: Planning and Online Routing," in *Proc. ACM DCC*, 2014, pp. 65–70.

[14] V. Cardellini, M. Colajanni, and P. Yu, "Dynamic Load Balancing on Web-server Systems," *IEEE Internet Comput.*, vol. 3, no. 3, pp. 28–39, 1999.

[15] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[16] N. Chang and M. Liu, "Revisiting the TTL-based Controlled Flooding Search," in *Proc. ACM MobiCom*, 2004, pp. 85–99.

[17] Y. Chang, S. Rao, and M. Tawarmalani, "Robust Validation of Network Designs under Uncertain Demands and Failures," in *Proc. USENIX NSDI*, 2017, pp. 347–362.

[18] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: High-speed Onion Routing at the Network Layer," in *Proc. ACM CCS*, 2015, pp. 1441–1454.

[19] H. Chen, G. Xue, and Z. Wang, "Efficient and Reliable Missing Tag Identification for Large-Scale RFID Systems With Unknown Tags," *IEEE Internet Things J.*, vol. 4, no. 3, pp. 736–748, Jun. 2017.

[20] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New Algorithms for Secure Outsourcing of Modular Exponentiations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2386–2396, Sep. 2014.

[21] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in *Proc. IEEE INFOCOM*, 2009, pp. 783–791.

[22] Cisco, *Cisco IOx*. [Online]. Available: http://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html (Last accessed on 05/03/2019).

[23] CoinDesk, *How Will Ethereum Scale?* [Online]. Available: https://www.coindesk.com/information/will-ethereum-scale/ (Last accessed on 05/03/2019).

[24] J.-F. Cordeau, G. Stojković, F. Soumis, and J. Desrosiers, "Benders Decomposition for Simultaneous Aircraft Routing and Crew Scheduling," *Transp. Sci.*, vol. 35, no. 4, pp. 375–388, Nov. 2001.

[25] Y. Cui, L. Wang, X. Wang, H. Wang, and Y. Wang, "FMTCP: A Fountain Code-Based Multipath Transmission Control Protocol," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 465–478, Apr. 2015.

[26] G. Danezis and I. Goldberg, "Sphinx: A Compact and Provably Secure Mix Format," in *Proc. IEEE S&P*, 2009, pp. 269–282.

[27] D. Delli Gatti, M. Gallegati, B. Greenwald, A. Russo, and J. E. Stiglitz, "The Financial Accelerator in an Evolving Credit Network," *J. Econ. Dyn. Control*, vol. 34, no. 9, pp. 1627–1650, Sep. 2010.

[28] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal Workload Allocation in Fog-Cloud Computing Towards Balanced Delay and Power Consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, 2016.

[29] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley, "Optimal Security Hardening Using Multi-Objective Optimization on Attack Tree Models of Networks," in *Proc. ACM CCS*, 2007, pp. 204–213.

[30] W. Elmenreich, "Fusion of Continuous-valued Sensor Measurements Using Confidence-weighted Averaging," *J. Vib. Control*, vol. 13, no. 9-10, pp. 1303–1312, Sep. 2007.

[31] B. Erb, D. Meißner, J. Pietron, and F. Kargl, "Chronograph–A Distributed Processing Platform for Online and Batch Computations on Event-sourced Graphs," in *Proc. ACM DEBS*, 2017, pp. 78–87.

[32] M. Faloutsos, C. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," in *Proc. ACM SIGCOMM*, 1999, pp. 251–262.

[33] B. Familiar, *Microservices, IoT and Azure: Leveraging DevOps and Microservice*. 2015.

[34] J. Fan, Z. Ye, C. Guan, X. Gao, K. Ren, and C. Qiao, "GREP: Guaranteeing Reliability with Enhanced Protection in NFV," in *Proc. ACM HotMiddlebox*, 2015, pp. 13–18.

[35] L. K. Fleischer, "Approximating Fractional Multicommodity Flow Independent of the Number of Commodities," *SIAM J. Discret. Math.*, vol. 13, no. 4, pp. 505–520, 2000.

[36] L. R. Ford and D. R. Fulkerson, "Maximal Flow Through a Network," *Can. J. Math.*, vol. 8, pp. 399–404, Jan. 1956.

[37] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* 1990.

[38] N. Garg and J. Konemann, "Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems," in *Proc. ACM FOCS*, 1998, pp. 300–309.

[39] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling Innovation in Network Function Control," in *Proc. ACM SIGCOMM*, 2014, pp. 163–174.

[40] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella, "Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds," *arXiv: 1305.0209*, 2013.

[41] A. Giordano, G. Spezzano, and A. Vinci, "Smart Agents and Fog Computing for Smart City Applications," in *Smart-CT*, 2016, pp. 137–146.

[42] D. Goldschlag, M. Reed, and P. Syverson, "Onion Routing for Anonymous and Private Internet Connections," *Commun. ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[43] P. Golle, M. Jakobsson, A. Juels, and P. Syverson, "Universal Re-encryption for Mixnets," in *Proc. CT-RSA*, 2004, pp. 163–178.

[44] M. Gowda, A. Dhekne, S. Shen, R. R. Choudhury, L. Yang, S. Golwalkar, and A. Essanian, "Bringing IoT to Sports Analytics," in *Proc. USENIX NSDI*, 2017, pp. 499–513.

[45] M. Green and I. Miers, "Bolt: Anonymous Payment Channels for Decentralized Currencies," in *Proc. ACM CCS*, 2017, pp. 473–489.

[46] A. Gudipati, D. Perry, L. E. Li, S. Katti, and B. Labs, "SoftRAN: Software Defined Radio Access Network," in *Proc. ACM HotSDN*, 2013, pp. 25–30.

[47] L. Guo, J. Pang, and A. Walid, "Dynamic Service Function Chaining in SDN-enabled Networks with Middleboxes," in *Proc. IEEE ICNP*, 2016, pp. 1–10.

[48] Gurobi, *Gurobi Optimizer*. [Online]. Available: http://www.gurobi.com/products/gurobi-optimizer (Last accessed on 05/03/2019).

[49] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, *Service Function Chaining Use Cases in Mobile Networks*, 2017. [Online]. Available: https://tools.ietf.org/pdf/draft-ietf-sfc-use-case-mobility-07.pdf (Last accessed on 05/03/2019).

[50] T. Hagras and J. Janecek, "A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," in *Proc. IEEE IPDPS*, 2004.

[51] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the Art, Challenges, and Implementation in Next Generation Mobile Networks (vEPC)," *IEEE Netw.*, vol. 28, no. 6, pp. 18–26, Nov. 2014.

[52] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," in *Proc. ACM SIGCOMM*, 2015, pp. 465–478.

[53] D. S. Hochbaum, "Heuristics for the Fixed Cost Median Problem," *Math. Program.*, vol. 22, no. 1, pp. 148–162, Dec. 1982.

[54] J. Horn and *et. al*, *Meltdown and Spectre*. [Online]. Available: https://spectreattack.com/ (Last accessed on 05/03/2019).

[55] IDC, *IDC Forecasts Worldwide Spending on the Internet of Things to Reach $745 Billion in 2019, Led by the Manufacturing, Consumer, Transportation, and Utilities Sectors*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS44596319 (Last accessed on 05/03/2019).

[56] *IoT Market Forecasts*. [Online]. Available: https://www.postscapes.com/internet-of-things-market-size/ (Last accessed on 05/03/2019).

[57] Y. Jiang, L. R. Sivalingam, S. Nath, and R. Govindan, "WebPerf: Evaluating What-If Scenarios for Cloud-hosted Web Applications," in *Proc. ACM SIGCOMM*, 2016, pp. 258–271.

[58] Jiaqi Zheng, Hong Xu, Xiaojun Zhu, Guihai Chen, and Yanhui Geng, "We've Got You Covered: Failure Recovery with Backup Tunnels in Traffic Engineering," in *Proc. IEEE ICNP*, 2016, pp. 1–10.

[59] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, “SoftCell: Scalable and Flexible Cellular Core Network Architecture,” in *Proc. ACM CoNEXT*, 2013, pp. 163–174.

[60] F. K. Jondral, “Software-Defined Radio: Basics and Evolution to Cognitive Radio,” *EURASIP J. Wirel. Commun. Netw.*, vol. 2005, no. 3, pp. 275–283, 2005.

[61] W. Jung, S. Hong, M. Ha, Y.-J. Kim, and D. Kim, “SSL-Based Lightweight Security of IP-Based Wireless Sensor Networks,” in *Proc. IEEE WAINA*, 2009, pp. 1112–1117.

[62] M. Kablan, B. Caldwell, R. Han, H. Jamjoom, and E. Keller, “Stateless Network Functions,” in *Proc. ACM HotMiddlebox*, 2015, pp. 49–54.

[63] Y. Kanizo, O. Rottenstreich, I. Segall, and J. Yallouz, “Optimizing Virtual Backup Allocation for Middleboxes,” in *Proc. IEEE ICNP*, 2016, pp. 1–10.

[64] R. Khalil and A. Gervais, “Revive: Rebalancing Off-Blockchain Payment Networks,” in *Proc. ACM CCS*, 2017, pp. 439–453.

[65] A. Kiayias, A. Russell, B. David, and R. Oliynykov, “Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol,” in *Proc. CRYPTO*, 2017, pp. 357–388.

[66] M. S. Kiraz and O. Uzunkol, “Efficient and Verifiable Algorithms for Secure Outsourcing of Cryptographic Computations,” *Int. J. Inf. Secur.*, vol. 15, no. 5, pp. 519–537, Oct. 2016.

[67] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, *CRAWDAD Dataset Dartmouth/Campus (V. 2009-09-09)*. [Online]. Available: https://crawdad.org/dartmouth/campus/20090909 (Last accessed on 05/03/2019).

[68] A. Krylovskiy, M. Jahn, and E. Patti, “Designing a Smart City Internet of Things Platform with Microservice Architecture,” in *Proc. IEEE FiCloud*, 2015, pp. 25–30.

[69] K. Kumar and Y.-H. Lu, “Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?” *Computer (Long. Beach. Calif).*, vol. 43, no. 4, pp. 51–56, Apr. 2010.

[70] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, “Service Chain Embedding with Maximum Flow in Software Defined Network

and Application to the Next-Generation Cellular Network Architecture," in *Proc. IEEE INFOCOM*, 2017.

[71] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven Bandwidth Guarantees in Datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 467–478.

[72] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. Wilfong, Y. R. Yang, and C. Guo, "PACE: Policy-Aware Application Cloud Embedding," in *Proc. IEEE INFOCOM*, 2013, pp. 638–646.

[73] S. Li, L. D. Xu, and S. Zhao, "The Internet of Things: A Survey," *Inf. Syst. Front.*, vol. 17, no. 2, pp. 243–259, Apr. 2015.

[74] A. Lioy, A. Pastor, F. Risso, R. Sassu, and A. L. Shaw, "Offloading Security Applications into the Network," *Proc. IEEE eChallenges*, 2014.

[75] D. H. Lorenz and D. Raz, "A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem," *Oper. Res. Lett.*, vol. 28, no. 5, pp. 213–219, Jun. 2001.

[76] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A Secure Microservice Framework for IoT," in *Proc. IEEE SOSE*, 2017, pp. 9–18.

[77] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in *Proc. ACM CCS*, 2016, pp. 17–30.

[78] S. J. Maher, G. Desaulniers, and F. Soumis, "Recoverable Robust Single Day Aircraft Maintenance Routing Problem," *Comput. Oper. Res.*, vol. 51, pp. 130–145, Nov. 2014.

[79] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks," in *Proc. ISOC NDSS*, 2017.

[80] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and Privacy with Payment-Channel Networks," in *Proc. ACM CCS*, 2017, pp. 455–471.

[81] G. Malavolta, P. Moreno-sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability," in *Proc. ISOC NDSS*, 2019.

[82] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight Cryptography for Embedded Systems – A Comparative Analysis," in *Data Priv. Manag. Auton. Spontaneous Secur.* 2014, pp. 333–349.

[83] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the Art of Network Function Virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.

[84] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[85] S. Misra, G. Xue, and D. Yang, "Polynomial Time Approximations for Multi-Path Routing with Bandwidth and Delay Constraints," in *Proc. IEEE INFOCOM*, 2009, pp. 558–566.

[86] M. Moradi, W. Wu, L. E. Li, and Z. M. Mao, "SoftMoW: Recursive and Reconfigurable Cellular WAN Architecture," in *Proc. ACM CoNEXT*, 2014, pp. 377–390.

[87] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy Preserving Payments in Credit Networks: Enabling Trust with Privacy in Online Marketplaces," in *Proc. ISOC NDSS*, 2015, pp. 8–11.

[88] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf (Last accessed on 05/03/2019).

[89] H. Ning and Z. Wang, "Future Internet of Things Architecture: Like Mankind Neural System or Social Organization Framework?" *IEEE Commun. Lett.*, vol. 15, no. 4, pp. 461–463, Apr. 2011.

[90] Y. Niu, F. Liu, and Z. Li, "Load Balancing Across Microservices," in *Proc. IEEE INFOCOM*, 2018, pp. 1–9.

[91] S. Noel and S. Jajodia, "Optimal IDS Sensor Placement and Alert Prioritization Using Attack Graphs," *J. Netw. Syst. Manag.*, vol. 16, no. 3, pp. 259–275, Sep. 2008.

[92] *NS-3 Network Simulator*. [Online]. Available: https://www.nsnam.org/ (Last accessed on 05/03/2019).

[93] O. Osuntokun, *AMP: Atomic Multi-Path Payments over Lightning*, 2018. [Online]. Available: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html (Last accessed on 05/03/2019).

[94] A. Paya and D. C. Marinescu, "Energy-Aware Load Balancing and Application Scaling for the Cloud Ecosystem," *IEEE Trans. Cloud Comput.*, vol. 5, no. 1, pp. 15–27, Jan. 2017.

[95] N. Poolsappasit, R. Dewri, and I. Ray, "Dynamic Security Risk Management Using Bayesian Attack Graphs," *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 1, pp. 61–74, Jan. 2012.

[96] J. Poon and V. Buterin, *Plasma: Scalable Autonomous Smart Contracts*, 2017. [Online]. Available: http://plasma.io/plasma.pdf (Last accessed on 05/03/2019).

[97] J. Poon and T. Dryja, *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*, 2016. [Online]. Available: https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf (Last accessed on 05/03/2019).

[98] S. Popov, *The Tangle*, 2017. [Online]. Available: https://www.iota.org/IOTA_Whitepaper.pdf (Last accessed on 05/03/2019).

[99] R. Potharaju and N. Jain, "Demystifying the Dark Side of the Middle: A Field Study of Middlebox Failures in Datacenters," in *Proc. ACM IMC*, 2013, pp. 9–22.

[100] P. Prihodko, S. Zhigulin, M. Sahno, and A. Ostrovskiy, *Flare: An Approach to Routing in Lightning Network*, 2016. [Online]. Available: http://bitfury.com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf (Last accessed on 05/03/2019).

[101] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *IEEE Trans. Netw. Serv. Manag.*, vol. 10, no. 2, pp. 105–118, Jun. 2013.

[102] *Raiden Network*. [Online]. Available: https://raiden.network/ (Last accessed on 05/03/2019).

[103] S. Rajagopalan, D. Williams, and H. Jamjoom, "Pico Replication: A High Availability Framework for Middleboxes," in *Proc. ACM SOCC*, 2013.

[104] *Ripple.* [Online]. Available: https://www.ripple.com/ (Last accessed on 05/03/2019).

[105] P. Rochard, *Lightning Routing Node Starter Pack*, 2019. [Online]. Available: https://medium.com/lightning-power-users/lightning-routing-node-starter-pack-704c0e7d79cb (Last accessed on 05/03/2019).

[106] R. T. Rockafellar and S. Uryasev, "Optimization of Conditional Value-at-Risk," *J. Risk*, vol. 2, pp. 21–41, 2000.

[107] E. Rohrer, J.-F. Laß, and F. Tschorsch, "Towards a Concurrent and Distributed Route Selection for Payment Channel Networks," in *Proc. ESORICS Workshop–CBT*, 2017, pp. 411–419.

[108] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions," in *Proc. ISOC NDSS*, 2018.

[109] M. Rost and S. Schmid, "Service Chain and Virtual Network Embeddings: Approximations Using Randomized Rounding," *arXiv:1604.02180*, 2016.

[110] A. Rullo, E. Serra, E. Bertino, and J. Lobo, "Shortfall-Based Optimal Placement of Security Resources for Mobile IoT Scenarios," in *Proc. ESORICS*, 2017, pp. 419–436.

[111] A. Sajid, H. Abbas, and K. Saleem, "Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges," *IEEE Access*, vol. 4, pp. 1375–1384, 2016.

[112] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments From Bitcoin," in *Proc. IEEE S&P*, 2014, pp. 459–474.

[113] R. E. Sawilla and X. Ou, "Identifying Critical Attack Assets in Dependency Attack Graphs," in *Proc. ESORICS*, 2008, pp. 18–34.

[114] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker, "Rollback-Recovery for Middleboxes," in *Proc. ACM SIGCOMM*, 2015, pp. 227–240.

[115] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, A. Texas, C. Station, and M. Park, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proc. USENIX NDSS*, 2013.

[116] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. Fanti, and P. Viswanath, "Routing Cryptocurrency with the Spider Network," *arXiv:1809.05088*, 2018.

[117] M. Spoke and Nuco Engineering Team, *Aion: The Third-Generation Blockchain Network*, 2017. [Online]. Available: https://aion.network/downloads/aion.network_technical-introduction_en.pdf (Last accessed on 05/03/2019).

[118] *Stellar*. [Online]. Available: https://www.stellar.org/ (Last accessed on 05/03/2019).

[119] S. Subashini and V. Kavitha, "A Survey on Security Issues in Service Delivery Models of Cloud Computing," *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1–11, Jan. 2011.

[120] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online Job Dispatching and Scheduling in Edge-Clouds," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[121] O. Tilmans, S. Vissicchio, L. Vanbever, and J. Rexford, "Fibbing in Action: On-demand Load-Balancing for Better Video Delivery," in *Proc. ACM SIGCOMM*, 2016, pp. 619–620.

[122] L. Toka, B. Lajtha, E. Hosszu, B. Formanek, D. Gehberger, and J. Tapolcai, "A Resource-Aware and Time-Critical IoT Framework," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[123] L. Tong, Y. Li, and W. Gao, "A Hierarchical Edge Cloud Architecture for Mobile Computing," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[124] W. Trappe, R. Howard, and R. S. Moore, "Low-Energy Security: Limits and Opportunities in the Internet of Things," *IEEE Secur. Priv.*, vol. 13, no. 1, pp. 14–21, Jan. 2015.

[125] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic Computing: A New Paradigm for Edge/Cloud Integration," *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 76–83, Nov. 2016.

[126] J. Wallen, *Five Nightmarish Attacks That Show the Risks of IoT Security*. [Online]. Available: http://www.zdnet.com/article/5-nightmarish-attacks-that-show-the-risks-of-iot-security/ (Last accessed on 05/03/2019).

[127] P. Wang, H. Xu, X. Jin, and T. Wang, "Flash: Efficient Dynamic Routing for Offchain Networks," *arXiv:1902.05260v1*, 2019.

[128] Z. Wang and J. Crowcroft, “Quality-of-Service Routing for Supporting Multimedia Applications,” *IEEE J. Sel. Areas Commun.*, vol. 14, no. 7, pp. 1228–1234, 1996.

[129] D. J. Watts and S. H. Strogatz, “Collective Dynamics of ‘Small-World’ Networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.

[130] B. M. Waxman, “Routing of Multipoint Connections,” *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, 1988.

[131] *What Led Amazon to Its Own Microservices Architecture.* [Online]. Available: https://thenewstack.io/led-amazon-microservices-architecture/ (Last accessed on 05/03/2019).

[132] *Why You Can't Talk About Microservices Without Mentioning Netflix.* [Online]. Available: https://smartbear.com/blog/develop/why-you-cant-talk-about-microservices-without-ment/ (Last accessed on 05/03/2019).

[133] M. Willebeek-LeMair and A. Reeves, “Strategies for Dynamic Load Balancing on Highly Parallel Computers,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979–993, 1993.

[134] L. Wu, M. Shahidehpour, and T. Li, “Stochastic Security-Constrained Unit Commitment,” *IEEE Trans. Power Syst.*, vol. 22, no. 2, pp. 800–811, May 2007.

[135] Y. Xia, Y. Liu, C. Tan, M. Ma, H. Guan, B. Zang, and H. Chen, “TinMan: Eliminating Confidential Mobile Data Exposure with Security Oriented Offloading,” in *Proc. ACM EuroSys*, 2015, pp. 1–16.

[136] Y. Xiao and M. Krunz, “QoE and Power Efficiency Tradeoff for Fog Computing Networks with Fog Node Cooperation,” in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[137] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, “Enhancing Survivability in Virtualized Data Centers: A Service-Aware Approach,” *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2610–2619, Dec. 2013.

[138] G. Xue, L. Chen, and K. Thulasiraman, “Quality-of-service and Quality-Of-Protection Issues in Preplanned Recovery Schemes Using Redundant Trees,” *IEEE J. Sel. Areas Commun.*, vol. 21, no. 8, pp. 1332–1345, Oct. 2003.

[139] G. Xue, W. Zhang, J. Tang, and K. Thulasiraman, “Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 656–669, Jun. 2008.

[140] J. Yallouz and A. Orda, “Tunable QoS-Aware Network Survivability,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 139–149, Feb. 2017.

[141] J. Yallouz, O. Rottenstreich, and A. Orda, “Tunable Survivable Spanning Trees,” *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1853–1866, Jun. 2016.

[142] H. Yanagisawa, T. Osogami, and R. Raymond, “Dependable Virtual Machine Allocation,” in *Proc. IEEE INFOCOM*, 2013, pp. 629–637.

[143] Y. Ye, “An O(n3L) Potential Reduction Algorithm for Linear Programming,” *Math. Program.*, vol. 50, no. 1-3, pp. 239–258, Mar. 1991.

[144] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, “Joint Topology Design and Mapping of Service Function Chains for Efficient, Scalable, and Reliable Network Functions Virtualization,” *IEEE Netw.*, vol. 30, no. 3, pp. 81–87, May 2016.

[145] L. Ying, R. Srikant, and X. Kang, “The Power of Slightly More Than One Sample in Randomized Load Balancing,” in *Proc. IEEE INFOCOM*, 2015, pp. 1131–1139.

[146] H. Yu, C. Qiao, V. Anand, X. Liu, H. Di, and G. Sun, “Survivable Virtual Infrastructure Mapping in a Federated Computing and Networking System under Single Regional Failures,” in *Proc. IEEE GLOBECOM*, 2010, pp. 1–6.

[147] R. Yu, V. T. Kilari, G. Xue, and J. Tang, “Load Balancing for Interdependent IoT Microservices,” in *Proc. IEEE INFOCOM*, 2019.

[148] R. Yu, Y. Wan, V. T. Kilari, G. Xue, J. Tang, and D. Yang, “P4PCN : Privacy-Preserving Path Probing for Payment Channel Networks,” submitted to *IEEE GLOBECOM*, 2019.

[149] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, “CoinExpress: A Fast Payment Routing Mechanism in Blockchain-based Payment Channel Networks,” in *Proc. IEEE ICCCN*, 2018.

[150] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, “Deploying Robust Security in Internet of Things,” in *Proc. IEEE CNS*, 2018.

[151] R. Yu, G. Xue, and X. Zhang, "Application Provisioning in Fog Computing-enabled Internet-of-Things: A Network Perspective," in *Proc. IEEE INFOCOM*, 2018, pp. 1–9.

[152] ——, "Provisioning QoS-aware and Robust Applications in Internet-of-Things: A Network Perspective," submitted to *IEEE/ACM Trans. Netw.*

[153] ——, "QoS-Aware and Reliable Traffic Steering for Service Function Chaining in Mobile Networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2522–2531, Nov. 2017.

[154] R. Yu, G. Xue, X. Zhang, and D. Li, "Survivable and Bandwidth-Guaranteed Embedding of Virtual Clusters in Cloud Data Centers," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[155] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[156] N. Zhang, S. Demetriou, X. Mi, W. Diao, K. Yuan, P. Zong, F. Qian, X. Wang, K. Chen, Y. Tian, C. A. Gunter, K. Zhang, P. Tague, and Y.-H. Lin, "Understanding IoT Security Through the Data Crystal Ball: Where We Are Now and Where We Are Going to Be," *arXiv:1703.09809*, 2017.

[157] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable Virtual Data Center Embedding in Clouds," in *Proc. IEEE INFOCOM*, 2014, pp. 289–297.

[158] W. Zhang, J. Tang, C. Wang, and S. de Soysa, "Reliable Adaptive Multipath Provisioning with Bandwidth and Differential Delay Constraints," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[159] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "StEERING: A Software-Defined Networking for Inline Service Chaining," in *Proc. IEEE ICNP*, 2013, pp. 1–10.

[160] Y. Zhang, Y. Long, Z. Liu, Z. Liu, and D. Gu, "Z-Channel: Scalable and Efficient Scheme in Zerocash," in *Proc. ACISP*, 2018, pp. 687–705.

[161] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, “Security and Privacy for Cloud-Based IoT: Challenges,” *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 26–33, Jan. 2017.

[162] J. Zhu, D. Li, J. Wu, H. Liu, Y. Zhang, and J. Zhang, “Towards Bandwidth Guarantee in Multi-Tenancy Cloud Computing Networks,” in *Proc. IEEE ICNP*, 2012, pp. 1–10.