Internet of Computing Things: Design, Optimization, and Implementation

BY

YUXUAN XING
B.Eng. North China Electric Power University

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2019

Chicago, Illinois

ACKNOWLEDGMENTS

My Ph.D. study is a long journey.

I would like to thank my Ph.D. advisor, Prof. Hulya Seferoglu, for guiding me through the entire journey and offering tremendous help, academically and financially.

I would like to thank my parents as well. They are always caring and supporting.

I would also like to thank my uncle, aunt, grandparents, and my cousins. They are the reason I choose Chicago in the first place and they make my life in a foreign country much easier.

I would like to thank my Ph.D. defense committee members, Prof. Rashid Ansari, Prof. Ahmet Enis Cetin, Prof. K. K. Ramakrishnan, and Prof. Besma Smida for their valuable time and suggestions.

Last but not least, I would like to thank all my friends along this journey, Haochen, Mingze, Chenjie, Yannan, Xinghao, Shanyu, Haoyu, Yang, and Chang. I cannot complete this journey without your companion.

I will never forget this journey. This great journey will always be my treasure in future chapters of my life.

YX

CONTRIBUTION OF AUTHORS

A version of Chapter 2 has been published in IEEE CloudNet Conference (Seferoglu, H. et al., 2014). I was mainly responsible for the simulation section to evaluate the effectiveness of the algorithm. Prof. Hulya Seferoglu was the primary author and developed the main ideas and algorithms.

A version of Chapter 3 has been published in IEEE IFIP Networking Conference (Singh, A. et al., 2016) and under journal submission. I was the primary author for the journal version. I developed the main communication architecture and video application for the Android implementation. Ajita Singh is the primary author and major driver of research for the conference version. Prof. Hulya Seferoglu helped me with the development of ideas and improvement of writings.

A version of Chapter 4 has been published in IEEE ITA Workshop (Xing, Y. et al., 2017). I was the primary author and major driver of research. Prof. Hulya Seferoglu helped me with the development of ideas and improvement of writings.

A version of Chapter 5 has been published in IEEE LANMAN Conference (Xing, Y. et al., 2018). I was the primary author and major driver of research. Dr. Shanyu Zhou provided the proof of optimality of the parallel task allocation scheme. Prof. Hulya Seferoglu helped me with development of ideas and improvement of writings.

A version of Chapter 6 has been published in IEEE LANMAN Conference (Jahanian, M. et al., 2018). I was responsible for developing a web crawler downloading tweets posted within areas affected by hurricane Harvey and Irma. Mohammad Jahanian was the primary author and major driver of research. Dr. Jiachen Chen, Prof. K. K. Ramakrishnan, Prof. Hulya Seferoglu, and Prof. Murat Yuksel helped with development of ideas and improvement of writings.

A version of Chapter 7 has been published in IEEE ICNP Conference (Keshtkarjahromi Y. et at., 2018). I was responsible for implementing the proposed and baseline algorithms on Android-based smartphones to evaluate the effectiveness. Dr. Yasaman Keshtkarjahromi was the primary author and major driver of research. Prof. Hulya Seferoglu helped with the development of ideas and improvement of writings.

A version of Chapter 8 has been published in SPIE Defense + Commercial Sensing (Bitar, R. et al., 2019). I was responsible for implementing the proposed and baseline algorithms on Android-based smartphones to evaluate the effectiveness. Rawad Bitar was the primary author and major driver of research. Dr. Yasaman Keshtkarjahromi, Dr. Venkat Dasari, Prof. Salim El Rouayheb, and Prof. Hulya Seferoglu helped with the development of ideas and improvement of writings.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

FIGURE **PAGE**

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

ACK	Acknowledgement
ALI	Automatic Location Identification
API	Application Programming Interface
ARC	AnyRun Computing
ARQ	Automatic Repeat reQuest
AVC	Advanced Video Coding
C3P	Coded Cooperative Computation Protocol
CPU	Central Processing Unit
D2D	Device to Device
DARS	Device-Aware Routing and Scheduling algorithm
DASH	Dynamic Adaptive Streaming over HTTP
DcC	Device-centric Cooperation
DIRS	Disaster Information Report System
DMS	Department of Management Services
DTN	Disruption-Tolerant Networking
EaCC	Energy-aware Cooperative Computation
FCC	Federal Communications Commission

LIST OF ABBREVIATIONS (Continued)

FEMA	Federal Emergency Management Agency
GC3P	Genie C3P
GO	Group Owner
GPU	Graphics Processing Unit
HCMM	Heterogeneous Coded Matrix Multiplication
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Thing
IP	Internet Protocol
JSON	JavaScript Object Notation
MDS	Maximum Distance Separable
MPEG	Moving Picture Experts Group
NUM	Network Utility Maximization
PRAC	Private and Rateless Adaptive Coded Computation
PrComp	Predictive Computation offloading mechanism
PSAP	Public Safety Answering Point
PSHSB	Public Safety and Homeland Security Bureau
PSSC	Public Safety Support Center
QoS	Quality of Service

LIST OF ABBREVIATIONS (Continued)

RAM	Random-Access Memory
RR	Round-Robin
RTT	Round Trip Time
SDK	Software Development Kit
ScC	Source-centric Cooperation
TCP	Transmission Control Protocol
TTI	Transmission Time Interval
UDP	User Datagram Protocol

SUMMARY

The Internet of Things (IoT) is emerging as a new paradigm that connects an exponentially increasing number of devices, including smartphones, wireless sensors, smart meters, health monitoring devices, etc. The number of these devices keeps increasing and is estimated to reach billions in the next five years. As a result, the data collected by these devices will grow at exponential rates. In many applications, unlocking the full power of these devices requires (i) communicating high volumes of data, and (ii) analyzing and processing this data through computationally intensive algorithms at unprecedentedly high rates. This thesis focuses on the design, optimization, and implementation of communication and computation algorithms by particularly focusing on computing devices; i.e., Internet of Computing Things.

The first chapter of this thesis focuses on exploiting local area connections for delivering the same video content to mobile devices within proximity of each other. More particularly, each can download a portion of the content and cooperatively share it over Wi-Fi Direct links with each other. Unlike traditional source-centric network control algorithms, which introduce high overhead and delay, we develop a device-centric stochastic cooperation scheme instead. The proposed scheme reduces overhead and improves the quality of service. We show the benefit of our scheme by a set of simulation.

The second chapter of this thesis focuses on energy-aware cooperative computation framework at the edge. In this setup, a group of cooperative devices, within proximity of each other, (i) use their cellular or Wi-Fi (802.11) links as their primary networking interfaces, and (ii) exploit their device-to-device connections (e.g., Wi-Fi Direct) to overcome processing power and energy bottlenecks. We evaluate our energy-aware cooperative computation framework on a real test-bed consisting of smartphones and tablets, and we show that it brings significant performance benefits.

The third chapter focuses on device-to-device networks at the edge, where devices with heterogeneous capabilities including computing power, energy limitations, and incentives participate in D2D activities heterogeneously. We develop (i) a device-aware routing and scheduling algorithm (DARS) by taking into account device capabilities, and (ii) a multi-hop D2D testbed using Android-based smartphones and tablets by exploiting Wi-Fi Direct and legacy Wi-Fi connections. We show that DARS significantly improves throughput in our testbed as compared to the state-of-the-art.

The fourth chapter focuses on predictive edge computing with hard deadlines. We design an algorithm; PrComp, which (i) predicts the uncertain dynamics of resources of devices at the edge including energy, computing power, and mobility, and (ii) makes sub-task offloading decisions by taking into account the predicted available resources. We evaluate PrComp on a testbed consisting of real Android-based smartphones and show that it significantly improves the energy consumption of edge devices and task completion delay as compared to the baselines.

The fifth chapter focuses on seeking the possibility of utilizing social media to obtain and disseminate information during natural disasters. We find that there are areas where people can still access the Internet even when 911 services are down. Furthermore, our analysis indicates that social media can potentially help in disaster management and improve outcomes.

The sixth chapter focuses on dynamic heterogeneity-aware coded cooperative computation at the edge. To exploit the potential of edge computing, we proposed Coded Cooperative Computation Protocol (C3P) to overcome the challenge caused by the heterogeneous and time-varying nature of edge devices. We show that C3P can improve task completion delay significantly as compared to baselines via both simulations and in a testbed consisting of real Android-based smartphones.

The seventh chapter focuses on private and rateless adaptive coded computation at the edge. To take into account the privacy requirement of IoT applications and the heterogeneous, time-varying resources of edge devices, we develop a private and rateless adaptive coded computation (PRAC) algorithm. We demonstrate the benefit of PRAC via a set of simulations and experiments in a testbed consisting of real Android-based smartphones.

CHAPTER 1

INTRODUCTION

The contents of this chapters are based on our work that is published in [2–8].©2014 IEEE. Reprinted, with permission, from [2]. ©2016 IEEE. Reprinted, with permission, from [3].©2017 IEEE. Reprinted, with permission, from [4]. ©2018 IEEE. Reprinted, with permission, from [5–7]. ©2019 SPIE. Reprinted, with permission, from [8].

1.1 Motivation

New data-intensive applications are continuously emerging in the daily routines of mobile devices. Furthermore, the number of edge devices, *e.g.,* Internet of Things (IoT) keeps increasing and is estimated to reach billions in the next five years [9]. Thus, the dramatic increase in throughput and connectivity demand [10], [11], in addition to heterogeneous device capabilities, poses a challenge for current and future wireless networks. One of the promising solutions is Device-to-Device (D2D) networking. D2D networking, advocating the idea of connecting two or more devices directly without traversing the core network, can potentially improve spectral efficiency, throughput, energy efficiency, delay, and fairness [12]. Therefore, D2D networking is promising to address increasing data and connectivity demand. Also, edge computing advocates that computationally intensive tasks in a device (master) could be offloaded to other edge or end devices (workers) in close proximity via D2D networks. In this thesis, several applications for such networks are proposed: (i) Wireless devices within the approximate among each other could help each other processing computation tasks to increase throughput. This application is introduced in work *Device-Centric Cooperation in Mobile Networks* and *Energy-Aware Cooperative Computation in Mobile Devices*; (ii) Cooperative wireless devices self-organize into a multi-hop network for communication. This application is introduced in work *Device-Aware Routing and Scheduling in Multi-Hop Device-to-Device Networks*; (iii) Wireless devices exploit computation task offloading opportunities to nearby edge devices to reduce energy consumption while satisfying a hard deadline. This application is introduced in work *Predictive Edge Computing with Hard Deadlines*; (iv) Wireless devices exploits computation task offloading opportunities to nearby edge devices to reduce delay considering heterogeneous, time-varying resources and privacy requirement. This application is introduced in work *Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge* and *PRAC: Private and Rateless Adaptive Coded Computation at the Edge*.

1.2 Device-Centric Cooperation in Mobile Networks

Cooperation among mobile devices by exploiting both cellular and local area connections is a promising approach to meet the increasing demand for network throughput. More particularly, we consider a group of cooperative mobile devices that are interested in the same content. Each mobile device can download a portion and share it with its peers via D2D networks such as Wi-Fi Direct and Bluetooth. Traditionally, network control algorithms such as backpressure make control decisions by servers in the cloud. This approach introduces a significant amount of overhead over the cellular links. Our approach is grounded on a network utility maximization (NUM) formulation of the problem and its solution. The solution decomposes into several parts with an intuitive interpretation, such as flow control, scheduling over cellular links, and cooperation and scheduling over local area links. Based on the structure of the decomposed solution, we develop a stochastic algorithm: Device-Centric Cooperation.

We further evaluate our proposed scheme via a set of simulation to show it indeed reduces overhead and delay.

1.3 Energy-Aware Cooperative Computation in Mobile Devices

D2D networking is a promising solution to address the ever-growing demand for throughput. In D2D networks, due to close proximity among devices and the development of communication theory, sometimes the processing power and energy become main scarce resources instead of bandwidth. We carried out a pilot study to demonstrate that processing power can be the bottleneck of throughput. When a mobile device downloading contents while performing computational intensive processing on previously downloaded data, there is a significant decrease in throughput. In order to fully exploit the potential of D2D networking, a new network mechanism is proposed. In this work, we develop an energy-aware cooperative computation framework for mobile devices. In this setup, a group of device that helps each other cooperatively by utilizing the high rate of D2D connections. In particular, mobile devices are classified into two categories, receivers and helpers. At first, all mobile devices use their cellular or Wi-Fi (802.11) links as their primary networking interfaces to download unprocessed data from the source. The data then is processed and helpers forward processed data to receivers by exploiting their D2D connections (Wi-Fi Direct) among them. Our framework is based on a Network Utility Maximization (NUM) formulation of the problem and its solutions. The solution decomposes into several parts with an intuitive interpretation, such as flow control, computation control, energy control, and cooperation & scheduling. Based on the structure of the decomposed solution, we develop a stochastic algorithm. Finally, we tested our energy-aware cooperative computation framework on a testbed consists of several Android-based smart devices. Results showed that our framework brings significant performance benefits in terms of throughput and energy.

1.4 Device-Aware Routing and Scheduling in Multi-Hop Device-to-Device Networks

D2D networking also has the potential to address connectivity issues and data demand for IoT devices. In particular, mobile devices and IoT devices can seek Internet connectivity via other devices through D2D networks rather than base stations or access points. However, due to heterogeneous device capabilities, energy constraint and incentives, how to design routing and scheduling in multi-hop D2D networks is still an open problem. We propose a pilot study to illustrate that among different multi-hop paths, it is crucial to determine which device should forward packets and provide higher throughput. We develop NUM functions to formulate the optimization problem. Its solution is also provided and decomposed into two parts, rate control and routing & scheduling. From the solutions, we develop our Device-aware Routing and Scheduling (DARS) framework considering devices' capabilities, energy, and willingness. We then proposed a way to enable multi-group intercommunication for Wi-Fi direct by utilizing both legacy Wi-Fi interfaces and Wi-Fi direct interfaces on real mobile devices. Finally, DARS is implemented on such multi-hop networks with different topologies. We compared DARS with the state-of-the-art and experimental results demonstrate the benefits of our algorithm.

1.5 Predictive Edge Computing with Hard Deadlines

IoT devices are often required performing computation-intensive tasks to analyze the data they collected promptly. In many scenarios, those tasks have to be out-sourced to clouds since IoT devices usually have very limited computing powers. Yet, offloading to remote clouds is costly in terms of delay and availability. Edge computing is a promising approach for localized data processing for many edge applications and systems including Internet of Things (IoT), where computationally intensive tasks in IoT devices could be divided into sub-tasks and offloaded to other IoT devices, mobile devices, and / or servers at the edge. However, existing solutions on edge computing do not address the full range of challenges, specifically heterogeneity; edge devices are highly heterogeneous and dynamic in nature. In this work, we develop a predictive edge computing framework with hard deadlines. Our algorithm; `PrComp`(i) predicts the uncertain dynamics of resources of devices at the edge including energy, computing power, and mobility, and (ii) makes sub-task offloading decisions by taking into account the predicted available resources, as well as the hard deadline constraints of tasks. We evaluate PrComp on a testbed consisting of real Android-based smartphones and show that it significantly improves the energy consumption of edge devices and task completion delay as compared to baselines.

1.6 The Evolving Nature of Disaster Management in the Internet and Social Media Era

Traditional means for contacting emergency responders depend critically on the availability of the 911 service to request help. Large-scale natural disasters such as hurricanes and earthquakes often result in overloading and sometimes failure of communication facilities. Affected citizens are increasingly using social media to obtain and disseminate information. We firstly crawled millions of tweets sent in affected areas during hurricane Harvey and Irma in 2017. We investigate tweets based on keyword association and categorizing for disaster management. Furthermore, the status of the civilian communication infrastructure is monitored and we find that there exists a sizable number of people with access to the Internet even in areas where 911 services were down. Also, they tweet disaster-related information including requests for help. Our analysis indicates that social media can potentially help in disaster management and improve outcomes.

1.7 Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge

We consider a master / worker cluster where there is one master requires to complete a computationally intensive task. We focus on the computation of linear functions. In particular, we assume that the collector's data is represented by a large matrix A and it wishes to compute the product $\mathbf{y} = A\mathbf{x}$, for a given vector $\mathbf{x}$. The master can divide the task into subtasks and offload subtasks to workers within proximity via D2D networks. Coded computation, which advocates mixing data in sub-tasks by employing erasure codes and offloading these sub-tasks to other devices for computation, is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this work, we utilize coded computation in edge devices to overcome the challenge brought by the heterogeneous and time-varying nature of edge devices and mitigate the affection of possible stragglers. We formulate the coded cooperative computation problem as an optimization problem. We investigate the non-ergodic and static solutions to this problem. As a dynamic solution to the optimization problem, we develop a coded cooperative computation protocol (`C3P`), which is based on Automatic Repeat reQuest (ARQ) mechanism. We evaluate `C3P` via simulations as well as in a testbed consisting of real Android-based smartphones and show that (i) `C3P` improves task completion delay significantly as compared to baselines, and (ii) the efficiency of `C3P` in terms of resource utilization is higher than 99%.

1.8 PRAC: Private and Rateless Adaptive Coded Computation at the Edge

The very nature of task offloading from a master to worker devices makes the computation framework vulnerable to attacks. One of the attacks, which is also the focus of this work, is eavesdropper adversary, where one or more of the workers can behave as an eavesdropper and can spy on the coded data sent to these devices for computations. In this work, we consider a similar model as previous work where a master device requires to compute a large matrix multiplication $\mathbf{y} = A\mathbf{x}$ with the cooperation of nearby helpers. In addition to heterogeneous and time-varying nature of edge devices, we also consider the privacy requirement of IoT applications. We propose Private and Rateless Adaptive Coded Computation (PRAC) to satisfy privacy conditions, which is any collection of a particular number of colluding workers will not be able to obtain any information about matrix A, in an information-theoretic sense. Each information matrix is padded with a linear combination of random keys to create a secure matrix. We show that PRAC outperforms known secure coded computing methods when resources are heterogeneous. We provide theoretical guarantees on the performance of PRAC and its comparison to baselines. Moreover, we confirm our theoretical results through simulations and experiments in a testbed consisting of real Android-based smartphones.

1.9 Thesis Contributions

The key contributions of this thesis are as follows:

1.9.1 Device-Centric Cooperation in Mobile Networks

- We consider a scenario where a group of cooperative mobile devices, exploiting both cellular and local area links, are within proximity of each other, and are interested in the same content. We propose a novel "device-centric cooperation" scheme for this scenario.
- We develop network utility maximization (NUM) formulation of the device-centric problem, and provide its decomposed solution. Based on the structure of the decomposed solution, we develop a stochastic device-centric algorithm; DcC. We show that DcC moves the functionality required for cooperation to mobile devices without loss of optimality.

- We evaluate our scheme via simulations for multiple mobile devices. The simulation results confirm that DcC reduces; (i) overhead; *i.e.,* the number of control packets that should be transmitted over cellular links, and (ii) the amount of delay that each packet experiences.

1.9.2 Energy-Aware Cooperative Computation in Mobile Devices

- We consider a group of cooperative mobile devices within proximity of each other. In this scenario, we first investigate the impact of processing power to transmission rate. Then, we develop an energy-aware cooperative computation model, where devices depending on their energy constraints could cooperate to get benefit of aggregate processing power in a group of cooperative devices.
- We characterize our problem in a NUM framework by taking into account processing power, energy, and bandwidth constraints. We solve the NUM problem, and use the solution to develop our stochastic algorithm; *energy-aware cooperative computation (EaCC)*. We show that EaCC provides stability and optimality guarantees.
- An integral part of our work is to understand the performance of EaCC in practice. Towards this goal, we develop a testbed consisting of Nexus 5 smartphones and Nexus 7 tablets. All devices use Android 5.1.1 as their operation systems. We implement *EaCC* in this testbed, and evaluate it for two applications; *Byte Counting* and *Video Streaming*. The experimental results show that our algorithm brings significant performance benefits.

1.9.3 Device-Aware Routing and Scheduling in Multi-Hop Device-to-Device Networks

- We consider a group of devices that form a multi-hop D2D network. We develop a network utility maximization (NUM) formulation of the device-aware framework, which provides a systematic approach to take into account device capabilities. We provide a decomposed solution of the NUM formulation, and based on the structure of the solution, we develop a stochastic *device-aware routing and scheduling algorithm* (DARS).
- An integral part of our work is to understand the performance of DARS in practice. Towards this goal, we develop a testbed consisting of Nexus 5 smartphones, and Nexus 7 tablets. In this testbed, mobile devices can be configured in a multi-hop topology using Wi-Fi Direct interfaces. To the best of our knowledge, our implementation is the first that enables and supports real time multi-hop forwarding (instead of store and forward mechanism [13] or using broadcast [14]) over Android-based mobile devices with Wi-Fi Direct.
- We implemented DARS as well as the backpressure algorithm [15], which is a state-of-the-art baseline on the testbed we developed. The experimental results show that DARS brings significant performance benefits as compared to backpressure.

1.9.4 Predictive Edge Computing with Hard Deadlines

- We develop a resource prediction module for Android-based mobile devices. Our prediction module determines the amount of delay and energy consumption when a task is processed locally or remotely in an Android device. This module also predicts the mobility of devices.

- We develop online task scheduling algorithms `PrComp` for serial and parallel tasks by using the predicted available resources. Our algorithms are based on the structure of the solution of an optimal task scheduling problem.
- We evaluate our `PrComp` framework on a testbed consisting of real smartphones, and we show that it brings significant performance benefits in terms of energy consumption and delay.

1.9.5 The Evolving Nature of Disaster Management in the Internet and Social Media Era

- We investigate how people call for help via social media, what kinds of help do they require and where.
- We analyze what is the network status during the disaster and how that affects people when they need to call for help.
- We process the tweets based on keyword association and categorized tweets for disaster management.
- Our analysis indicates that social media can potentially help in disaster management and improve outcomes.

1.9.6 Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge

- We formulate the coded cooperative computation problem as an optimization problem. We investigate the non-ergodic and static solutions of this problem. As a dynamic solution to the optimization problem, we develop a coded cooperative computation protocol (`C3P`), which is based on Automatic Repeat reQuest (ARQ) mechanism. In particular, a collector device offloads coded sub-tasks to helpers gradually, and receives Acknowledgment (ACK) after each sub-task is computed. Depending on the time difference between offloading a sub-task to a helper and its ACK, the collector estimates the runtime of the helpers, and offloads more/less tasks accordingly. This makes `C3P` dynamic and adaptive to heterogeneous and time-varying resources at helpers.

- We characterize the performance of `C3P` as compared to the non-ergodic and static solutions, and show that (i) the gap between the task completion delays of `C3P` and the non-ergodic solution is finite even for large number of sub-tasks, *i.e.,* $R \to \infty$, and (ii) the task completion delay of `C3P` is approximately equal to the static solution for large numbers of sub-tasks. We also analyze the efficiency of `C3P` in each helper in closed form, where the efficiency metric represents the effective utilization of resources at each helper.
- We evaluate `C3P` via simulations as well as in a testbed consisting of real Android-based smartphones and show that (i) `C3P` improves task completion delay significantly as compared to baselines, and (ii) the efficiency of `C3P` in terms of resource utilization is higher than 99%.

1.9.7 PRAC: Private and Rateless Adaptive Coded Computation at the Edge

- We design PRAC for heterogeneous and time-varying private coded computing with colluding workers. In particular, PRAC codes sub-tasks using Fountain codes, and determines how many coded packets and keys each worker should compute dynamically over time.
- We provide theoretical analysis of PRAC and show that it (i) guarantees privacy conditions, and (ii) uses minimum number of keys to satisfy privacy requirements. Furthermore, we provide a closed form task completion delay analysis of PRAC.

- We evaluate the performance of PRAC via simulations as well as in a testbed consisting of real Android-based smartphones as compared to baselines.

1.10 Thesis Organization

The rest of the dissertation is organized as follows. In Chapter 2, we present our work on device-centric cooperation in mobile networks. In Chapter 3, we present the work on energy-aware cooperative computation in mobile devices. In Chapter 4, we present our work on device-aware routing and scheduling in multi-hop device-to-device networks. In Chapter 5, we present our work on predictive edge computing with hard deadlines. In Chapter 6, we present our work on the evolving nature of disaster management in the Internet and social media era. In Chapter 7, we present our work on dynamic heterogeneity-aware coded cooperative computation at the edge. In Chapter 8, we present our work on PRAC: private and rateless adaptive coded computation at the edge. And in Chapter 9, we conclude the thesis.

CHAPTER 2

DEVICE-CENTRIC COOPERATION IN MOBILE NETWORKS

The contents of this chapter are based on our works that are published in the proceedings of the 2014 IEEE CloudNet conference [2]. ©2014 IEEE. Reprinted, with permission, from [2].

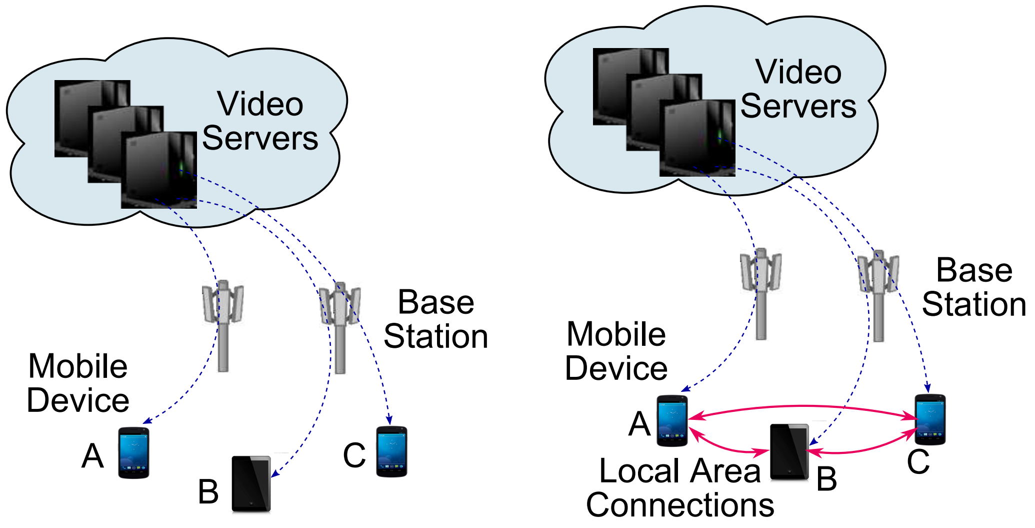
The increasing popularity of applications such as video streaming in today's mobile devices introduces higher demand for throughput, and puts a strain especially on cellular links. Cooperation among mobile devices by exploiting both cellular and local area connections is a promising approach to meet the increasing demand. In this chapter, we consider that a group of cooperative mobile devices, exploiting both cellular and local area links and within proximity of each other, are interested in the same video content. Traditional network control algorithms introduce high overhead and delay in this setup as the network control and cooperation decisions are made in a source-centric manner. Instead, we develop a device-centric stochastic cooperation scheme. Our device-centric scheme; DcC allows mobile devices to make control decisions such as flow control, scheduling, and cooperation without loss of optimality. Thanks to being device-centric, DcC reduces; (i) overhead; *i.e.,* the number of control packets that should be transmitted over cellular links, so cellular links are used more efficiently, and (ii) the amount of delay that each packet experiences, which improves quality of service. The simulation results demonstrate the benefits of DcC.

2.1 Background

The increasing popularity of applications such as video streaming in today's mobile devices introduces higher demand for throughput, and puts a strain especially on cellular links. In fact, cellular traffic is growing exponentially and it is expected to remain so for the foreseeable future [16], [11].

Cooperation among mobile devices is a promising approach to meet the increasing throughput demand over cellular links. In particular, when mobile devices are in the close proximity of each other and are interested in the same content, device-to-device connections such as WiFi or Bluetooth can be opportunistically used to construct a cooperative system [17], [18]. Indeed, this scenario is getting increasing interest [17]. *E.g.,* a group of friends may be interested in watching the same video on YouTube, or a number of students may participate in an online education class [17]. More details about the practicality of this scenario is provided in [17]. To better illustrate this setup, we provide the following example.

Example 1. *Let us consider Figure 1, where mobile device users in close proximity are interested in the same video content. Figure 1(a) shows no-cooperation where each mobile device uses only its cellular link to stream video. For example, if the cellular link rates are 100kbps, each user's streaming rate will be 100kbps. Figure 1(b) shows cooperation, where each mobile device uses cellular and local area links simultaneously (these links operate simultaneously thanks to using different parts of the spectrum) to stream video. Each user downloads 100kbps of video through their cellular connection, and 200kbps from their neighbors. Thus, the streaming rate increases to 300kbps from 100kbps, which is a significant improvement [17], [18]. One important problem, and is the focus of this chapter, is the design of a stochastic control algorithm that is efficient in practice in terms of overhead and delay.* □



(a) No-cooperation (b) Cooperation

Figure 1. Mobile device users; A, B, and C are in close proximity, and interested in the same video content. (a) No-cooperation. Each mobile device uses its own cellular link to stream video. (b) Cooperation. Each mobile device uses cellular and local area links simultaneously to stream video.

Traditional network control algorithms such as backpressure [19], [20], [21] make control decisions such as routing and scheduling (and cooperation decision in our problem setup) in a “source-centric” manner. In our problem, this corresponds to the case that the servers in the cloud make decisions about (i) the number of video packets that should be pushed to each mobile device, and (ii) the amount of cooperation among mobile devices; *i.e.,* the number of packets that each mobile device should transmit to other mobile devices in its neighborhood. In order to make these decisions, video servers should keep track of the states of the mobile devices, which includes queue sizes in mobile devices as well as cellular link qualities towards each mobile device. This puts significant amount of overhead over the cellular links. Furthermore, when there is congestion over the cellular links, the state information, *i.e.,* control packets can be delayed significantly, and the video servers may not make timely decisions such as reducing or increasing the rates towards each mobile device. This increases end-to-end delay, which may not fulfill quality of service (QoS) requirements of video streaming applications.

In this chapter, we develop a device-centric cooperation scheme to determine the number of video packets each mobile device should receive via cellular links as well as from its neighbors. Our approach is grounded on a network utility maximization (NUM) formulation of the problem and its solution [22]. The solution decomposes into several parts with an intuitive interpretation, such as flow control, scheduling over cellular links, and cooperation and scheduling over local area links. Based on the structure of the decomposed solution, we develop a stochastic algorithm; Device-Centric Cooperation; DcC.

The structure of the rest of the chapter is as follows. Section 2.2 gives an overview of the system model. Section 2.3 presents the NUM formulation of our device-centric scheme. Section 2.4 presents the stochastic device-centric cooperation algorithm; DcC. Section 2.5 evaluates DcC. Section 2.6 presents related work.

2.2 System Model

In this section, we provide an overview of the device- and source-centric cooperation models demonstrated in Figure 2.[1] First, we provide a cooperative system setup that are common to both device- and source-centric models.

[1]Note that we provide the source-centric model in addition to our device-centric model so that we can make a connection and comparison between device- and source-centric schemes in the rest of the chapter.

2.2.1 Cooperative System

Setup: We consider a cooperative system shown in Figure 2(a), where each mobile device is able to connect to the Internet via cellular links[1], and forward packets to other mobile devices through the local links, *e.g.,* Bluetooth or WiFi.

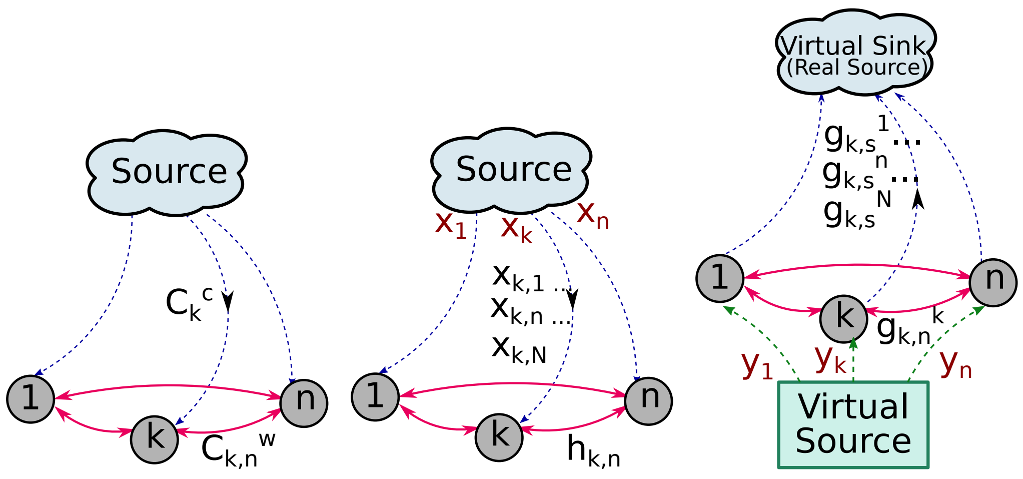
The cooperative system consist of N mobile devices and a source node. Note that the source node represents video servers, proxies, and base stations. This representation allows us to focus on the bottlenecks of the system, namely cellular links from the base station to the mobile devices and the local area links [18]. $\mathcal{N}$ is the set of the mobile devices, where $N = |\mathcal{N}|$. The mobile devices are interested in the same content and they construct a cooperating group.[2] We consider that time is slotted and t refers to the beginning of slot t.

Cellular Links: Each mobile device $k \in \mathcal{N}$ is connected to the Internet via its cellular link. At slot t, $\boldsymbol{C}^c(t)$ is the channel state vector of the cellular links, where $\boldsymbol{C}^c(t) = \{C_1^c(t), ..., C_k^c(t), ..., C_N^c(t)\}$. We assume that $C_k^c(t)$ is the state of the cellular links to mobile node k. We consider that cellular links towards different mobile devices are interference free as interference could be handled by base stations. Let $\Gamma_{\boldsymbol{C}^c(t)}$ denote the set of the link transmission rates feasible at time slot t for channel state $\boldsymbol{C}^c(t)$.

Local Area Links: In our setup, we consider that mobile devices are in close proximity and they hear each other. Therefore, in the local area, each mobile device can connect to another device directly. This

[1]Note that our device-centric scheme is generic enough to include Internet connections via WiFi, but we only focus on cellular links for Internet connection in this chapter to make the presentation and analysis simple.

[2]We consider that all mobile devices volunteer to cooperate without any malicious activity. This is possible in our setup due to existing social ties as the mobile device users are in close proximity to each other.



(a) Cooperative System (b) Source-Centric (c) Device-Centric

Figure 2. (a) Cooperative system. (b) Source-centric cooperation. (c) Device-centric cooperation.

gives us a fully connected topology. Depending on the underlying technology, local area transmissions can be unicast (*e.g.,* Bluetooth, or WiFi) or broadcast (can be achieved by extending WiFi [17]). In our formulations, we consider both unicast and broadcast transmissions in the local area. We consider protocol model in our formulations [23], where each mobile device can either transmit or receive at the same time. Since our local area network is fully connected, only one mobile device can transmit in a slot.

At slot t, $\boldsymbol{C}^w(t)$ is the channel state vector of the local area links, where $\boldsymbol{C}^w(t) = \{C^w_{1,2}(t), ..., C^w_{k,n}(t), ..., C^w_{N-1,N}(t)\}$. We assume that $C^w_{k,n}(t)$ is the state of the wireless link between node k and n. Let $\Gamma_{\boldsymbol{C}^w_u(t)}$ denote the set of the link transmission rates feasible at time slot t for channel state $\boldsymbol{C}^w(t)$ for unicast transmission. Similarly, $\Gamma_{\boldsymbol{C}^w_b(t)}$ denote the set of the link transmission rates feasible at time slot t for channel state $\boldsymbol{C}^w(t)$ for broadcast transmission.

2.2.2 Source-Centric Model

The source-centric cooperation model is shown in Figure 2(b), where the source node transmits a video flow to a set of mobile devices $\mathcal{N}$. The flow generation rate at the source for mobile device k is $x_k(t)$, $k \in \mathcal{N}$. $x_k(t)$ is i.i.d. over the slots and their expected values; $A_k = E[x_k(t)]$, $E[x_k(t)^2]$ are finite. Note that even if all mobile devices are interested in the same content, they may receive the content at different rates. In video streaming applications, this corresponds to different levels of video quality. Flow rate $x_k(t)$ is associated with a utility function $U_k(x_k(t))$, which we assume to be strictly concave function of $x_k(t)$.

Flow rate over the cellular link towards node k is $\max_{n \in \mathcal{N}}\{x_{k,n}(t)\}$, where $x_{k,k}(t)$ is the rate towards node k to help node k, while $x_{k,n}(t)$, $k \neq n$ is the rate towards node k to help node n. The flow rate over the cellular link is maximum of the rates, *i.e.,* $\max_{n \in \mathcal{N}}\{x_{k,n}(t)\}$ as all mobile devices are interested in the same content. Note that $x_{k,k}(t)$ is the rate over the cellular link towards node k, while $x_k(t)$ is the flow generation rate for device k. Flow rate over the local area link from node k to node n is $h_{k,n}(t)$, $k \neq n$. Note that $h_{k,n}(t)$ is to help node n using node k as a relay.

In the source-centric model, at time slot t, queue $\mu_k(t)$ is constructed at the source, and it queues packets that will be transmitted to node k, and changes according to following dynamics at every time slot t.

$$\mu_k(t+1) \leq \max[\mu_k(t) - \sum_{n \in \mathcal{N}} x_{n,k}(t), 0] + x_k(t) \tag{2.1}$$

At time slot t, queue $\nu_{n,k}(t)$ is the queue size at mobile device n, and it queues the packets that should be transmitted to node k. $\nu_{n,k}(t)$ changes according to following dynamics at every time slot t.

$$\nu_{n,k}(t+1) \leq \max[\nu_{n,k}(t) - h_{n,k}(t), 0] + x_{n,k}(t) \tag{2.2}$$

2.2.3 Device-Centric Model

In the device-centric model shown in Figure 2(c), a virtual source is added to the system and the real source becomes a virtual sink. Node k receives packets with rate $y_k(t)$ from the virtual source and forwards these packets to the virtual sink and other mobile devices. The transmission rate over the cellular link from node k to the virtual sink is $\max_{n \in \mathcal{N}}\{g^s_{k,s}(t)\}$. The transmission rate from node k to n is $g^k_{k,n}(t)$.

Note that the flow rates; $y_k(t)$, $g^n_{k,s}(t)$, $g^k_{k,n}(t)$ are virtual flow rates. In our device-centric scheme, these virtual flow rates are used to determine the real flow values; $x_k(t)$, $x_{k,n}(t)$, $h_{k,n}(t)$ as explained in Section 2.4.

In the device-centric model, at time slot t, queue $\lambda_k(t)$ is a virtual queue size constructed at node k. $\lambda_k(t)$ changes according to following dynamics at every time slot t.

$$\lambda_k(t+1) \leq \max[\lambda_k(t) - g^k_{k,s}(t) - \sum_{n \in \mathcal{N} - \{k\}} g^k_{k,n}(t), 0] + y_k(t) \tag{2.3}$$

At time slot t, queue $\eta_{n,k}(t)$ is a virtual queue size constructed at node n. $\eta_{n,k}(t)$ changes according to following dynamics at every time slot t.

$$\eta_{n,k}(t+1) \leq \max[\eta_{n,k}(t) - g^{k}_{n,s}(t), 0] + g^{k}_{k,n}(t) \tag{2.4}$$

In addition to the virtual queues $\lambda_k(t)$ and $\eta_{n,k}(t)$, a real queue $Q_{n,k}(t)$ is constructed at node n and evolves according to the following dynamics at every time slot t.

$$Q_{n,k}(t+1) \leq \max[Q_{n,k}(t) - h_{n,k}(t), 0] + x_{n,k}(t) \tag{2.5}$$

Note that $h_{n,k}(t)$ is the amount of the real outgoing traffic from node n to k (*i.e.,* from queue $Q_{n,k}$), and $x_{n,k}(t)$ is the amount of the real incoming traffic to node n from the source (*i.e.,* to the queue $Q_{n,k}$). The relationship between the real and virtual queues as well as real and virtual flows are provided in Section 2.4.

2.3 Device-Centric NUM

In this section, we formulate the device-centric network utility maximization (NUM) framework. This approach sheds light into the structure of the our stochastic algorithm DcC, which we present in the next section.[1]

[1]Note that NUM optimizes the average values of the parameters that are defined in Section 2.2. By abuse of notation, we use a variable, *e.g.,* ϕ as the average value of $\phi(t)$ in our NUM formulation if both ϕ and $\phi(t)$ refers to the same parameter.

2.3.1 Formulation

We provide NUM formulations for (i) unicast and (ii) broadcast transmissions in the local area. For unicast setup, the NUM formulation is P-Unicast:

$$\begin{aligned}
&\max_{\boldsymbol{y},\boldsymbol{g}} \sum_{k\in\mathcal{N}} U_k(y_k) \\
&\text{s.t. } g_{k,s}^{k} + \sum_{n\in\mathcal{N}-\{k\}} g_{k,n}^{k} = y_k,\ \forall k\in\mathcal{N} \\
&g_{n,s}^{k} = g_{k,n}^{k},\ \forall k\in\mathcal{N}, n\in\mathcal{N}-\{k\} \\
&\{\max_{n\in\mathcal{N}}\{g_{k,s}^{n}\}\}_{\forall k\in\mathcal{N}} \in \Gamma_{C^c}, \\
&\{g_{k,n}^{k}\}_{\forall k\in\mathcal{N},n\in\mathcal{N}-\{k\}} \in \Gamma_{C_u^w}.
\end{aligned} \tag{2.6}$$

The objective of P-Unicast is to determine $\boldsymbol{y} = \{y_k\}_{k\in\mathcal{N}}$, $\boldsymbol{g} = \{g_{n,s}^{k}\}_{k\in\mathcal{N},n\in\mathcal{N}}$ which maximize the total utility function; $\sum_{k\in\mathcal{N}} U_k(y_k)$. The first constraint is the flow conservation constraint at node k; y_k is the incoming traffic rate from virtual source to node k, and $g_{k,s}^{k} + \sum_{n\in\mathcal{N}-\{k\}} g_{k,n}^{k}$ is the outgoing traffic rate from node k to the virtual sink and the neighbors. The second constraint is the flow conservation constraint at node n for node k's flow; $g_{k,n}^{k}$ is the incoming flow rate to node n from node k, and $g_{n,s}^{k}$ is the flow rate from node n towards virtual sink. The last two constraints are the capacity constraints over cellular and local links.

For broadcast setup, the NUM formulation is P-Broadcast. The objective function and the first three constraints of P-Broadcast is the same as P-Unicast in Equation 2.6. The rest of the constraints of P-Broadcast:

$$g^k_{k,n} \leq \sum_{\mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} f_{n,\mathcal{J}}, \; \forall k \in \mathcal{N}, n \in \mathcal{N} - \{k\}$$

$$\{f_{n,\mathcal{J}}\}_{\forall n \in \mathcal{N}, \mathcal{J} \in \mathcal{H} | n \notin \mathcal{J}} \in \Gamma_{C^w_b}. \tag{2.7}$$

The first constraint in Equation 2.7 relates the broadcast transmission rate to the link rate. Let $\mathcal{J}$ be a set of nodes, and $\mathcal{H}$ be the set of node combinations, *i.e.,* $\mathcal{J} \in \mathcal{H}$. If packets are broadcast from node n to node set $\mathcal{J}$, each node $k \in \mathcal{J}$ can receive the packets (depending on the loss probability). In the device-centric system, this corresponds to simultaneous transmission from nodes in $\mathcal{J}$ to node k. $f_{n,\mathcal{J}}$ is the broadcast rate in the source-centric system. Since there may be different $\mathcal{J}$ sets which contain node k, $f_{n,\mathcal{J}}$ is summed $\forall \mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}$ to determine $g^k_{k,n}$. The second constraint in Equation 2.7 is the broadcast capacity constraint.

2.3.2 Solution

Lagrangian relaxation of the first two constraints of both Equation 2.6 and Equation 2.7 gives the following Lagrange function:

$$L = \sum_{k \in \mathcal{N}} U_k(y_k) + \sum_{k \in \mathcal{N}} \lambda_k (g^k_{k,s} + \sum_{n \in \mathcal{N} - \{k\}} g^k_{k,n} - y_k) + \sum_{k \in \mathcal{N}} \sum_{n \in \mathcal{N} - \{k\}} \eta_{n,k} (g^k_{n,s} - g^k_{k,n}) \tag{2.8}$$

where λ_k and $\eta_{n,k}$ are the Lagrange multipliers. Note that λ_k and $\eta_{n,k}$ represent the virtual queue sizes defined by Equation 2.3 and Equation 2.4. The values of λ_k and $\eta_{n,k}$ are tracked at nodes k and n, respectively. Note that these values are virtual values, and a counter is sufficient to keep track of these values.

Equation 2.8 can be decomposed into several intuitive sub-problems such as rate control, and scheduling. First, we solve the Lagrangian function with respect to y_k:

$$y_k = (U_k')^{-1}(\lambda_k) \tag{2.9}$$

where $(U_k')^{-1}$ is the inverse of the derivative of U_k. Since U_k is strictly concave function of y_k, y_k is inversely proportional to λ_k. This means that when the queue size λ_k increases, y_k should reduce. In the system implementation, node k requests y_k packets from the real source (*e.g.,* video server).

Second, we solve the Lagrangian for $g_{k,s}^k$ and $g_{n,s}^k$:

$$\begin{aligned} \max_{\boldsymbol{g}} \quad & \sum_{k \in \mathcal{N}} [\lambda_k g_{k,s}^k + \sum_{n \in \mathcal{N}-\{k\}} \eta_{k,n} g_{k,s}^n] \\ \text{s.t.} \quad & \{\max_{n \in \mathcal{N}} \{g_{k,s}^n\}\}_{\forall k \in \mathcal{N}} \in \Gamma_{C^c}, \end{aligned} \tag{2.10}$$

After $g_{k,s}^k$ and $g_{k,s}^n$ are determined, node k requests $\max_{n \in \mathcal{N}}\{g_{k,s}^n\}$ packets from the source through its cellular link. Note that $g_{k,s}^k$ and $g_{k,s}^n$ are different from y_k as y_k is the total flow rate requested by node k and this rate can be transmitted through both its cellular link or from the neighboring nodes, while $g_{k,s}^k$ and $g_{k,s}^n$ are the rates over cellular links.

Finally, we solve the Lagrangian with respect to $g^k_{k,n}$. Note that the solutions in Equation 2.9 and Equation 2.10 holds for both P-Unicast and P-Broadcast. However, the solutions of P-Unicast and P-Broadcast with respect to $g^k_{k,n}$ differ as explained next. The solution of P-Unicast with respect to $g^k_{k,n}$ is: $\max_{\boldsymbol{g}} \sum_{k\in\mathcal{N}}\sum_{n\in\mathcal{N}-\{k\}}(\lambda_k - \eta_{n,k})g^k_{k,n}$ subject to the last two constraints of Equation 2.6. The solution of P-Broadcast with respect to $g^k_{k,n}$ is: $\max_{\boldsymbol{g}} \sum_{k\in\mathcal{N}}\sum_{n\in\mathcal{N}-\{k\}}(\lambda_k - \eta_{n,k})g^k_{k,n}$ subject to all the constraints in Equation 2.7.

Next, we design our stochastic algorithm; Device-Centric Cooperation (DcC) based on the structure of the decomposed NUM solutions, *i.e.,* Equation 2.9 and Equation 2.10 as well as the local area scheduling solution presented above.

2.4 Device-Centric Cooperation (DcC)

Now, we provide our Device-Centric Cooperation (DcC) algorithm which includes *rate control*, *cellular link scheduler* and *cooperation & local area link scheduler*. Note that both unicast and broadcast setups have the same rate control and cellular link scheduling parts. The only different part is the cooperation & local area link scheduling as explained later.

Device-Centric Cooperation (DcC):

- *Rate Control:* At every time slot t, the rate controller at node k determines the number of packets that should be requested from the source according to;

$$\max_{\boldsymbol{y}}[MU_k(y_k(t)) - \lambda_k(t)y_k(t)]$$
$$\text{s.t. } y_k(t) \leq R_k^{max} \tag{2.11}$$

where R_k^{max} is be a positive constant larger than the cellular rate from the actual source, and M is a large positive constant. The values of R_k^{max} and M are important for the stability of the DcC algorithm [24]. $y_k(t)$ is the number of packets that will be requested from the source.

- *Cellular Link Scheduler:* At every time slot t, the cellular link scheduler at node k determines the number of packets requested through the cellular links.

$$\begin{aligned} &\max_{\boldsymbol{g}} \lambda_k(t) g_{k,s}^k(t) + \sum_{n \in \mathcal{N} - \{k\}} (\eta_{k,n}(t) - Q_{k,n}(t)) g_{k,s}^n(t) \\ &\text{s.t. } \{g_{k,s}^n(t)\}_{\forall n \in \mathcal{N}} \in \Gamma_{\boldsymbol{C}^c(t)}. \end{aligned} \tag{2.12}$$

After $g_{k,s}^k(t)$ and $g_{k,s}^n(t)$ are determined, the real flow rates are determined as $x_{k,k}(t) = g_{k,s}^k(t)$ and $x_{k,n}(t) = g_{k,s}^n(t) - \beta$, where $\beta > 0$ can be chosen to be arbitrarily small, and $\max_{n \in \mathcal{N}}\{x_{k,n}(t)\}$ amount of video packets are requested from the source by node k.

- *Cooperation & Local-Area Link Scheduler for Unicast:* At time slot t, the link rate $g_{k,n}^k(t)$ is determined by;

$$\begin{aligned} &\max_{\boldsymbol{g}} \sum_{k \in \mathcal{N}} \sum_{n \in \mathcal{N} - \{k\}} [\lambda_k(t) - \eta_{n,k}(t) + Q_{n,k}(t)] g_{k,n}^k(t) \\ &\text{s.t. } \{g_{k,n}^k(t)\}_{\forall k \in \mathcal{N}, n \in \mathcal{N} - \{k\}} \in \Gamma_{\boldsymbol{C}_u^w(t)}. \end{aligned} \tag{2.13}$$

After $g_{k,n}^k(t)$ is determined, $h_{n,k}(t) = g_{k,n}^k(t)$ amount of video packets is requested from node n by node k.

- *Cooperation & Local-Area Link Scheduler for Broadcast:* At time slot t, the link broadcast rate is determined by;

$$\max_{\boldsymbol{f}} \sum_{k \in \mathcal{N}} \sum_{n \in \mathcal{N}-\{k\}} \sum_{\mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} [\lambda_k(t) - \eta_{n,k}(t) + Q_{n,k}(t)] f_{n,J}(t)$$
$$\text{s.t. } \{f_{n,\mathcal{J}}(t)\}_{\forall n \in \mathcal{N}, \mathcal{J} \in \mathcal{H} | k \notin \mathcal{J}} \in \Gamma_{\boldsymbol{C}_b^w(t)} \tag{2.14}$$

After $f_{n,\mathcal{J}}(t)$ is determined, $f_{n,\mathcal{J}}(t)$ amount of video packets are transmitted from node n to nodes in $\mathcal{J}$. The optimum value of $g_{k,n}^k(t)$ is $g_{k,n}^k(t) = \sum_{\mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} f_{n,\mathcal{J}}(t), \forall k \in \mathcal{N}, n \in \mathcal{N} - \{k\}$. Therefore, the real transmission rate of over each link is equal to $h_{n,k}(t) = g_{k,n}^k(t) = \sum_{\mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} f_{n,\mathcal{J}}(t), \forall k \in \mathcal{N}, n \in \mathcal{N} - \{k\}$.

Theorem 1. *If channel states are i.i.d. over time slots, and the arrival rates $E[y_t(t)] = A_k, \forall k \in \mathcal{N}$ are interior of the stability region of cellular and local area links, then DcC stabilizes the network and the total average queue sizes, including both virtual and real queues, are bounded for both unicast and broadcast setups.*

Proof: The proof is provided in [24]. ■

Theorem 2. *If the channel states are i.i.d. over time slots, and the traffic arrival rates are controlled by the rate control algorithm in Equation 2.11, then the admitted flow rates converge to the utility optimal operating point with increasing M.*

Proof: The proof is provided in [24]. ■

2.5 Evaluation of Device-Centric Cooperation

In this section, we evaluate our DcC algorithm as compared to Source-Centric Cooperation (ScC), and highlight the benefits of DcC over ScC. Therefore, we first provide a brief description of ScC algorithm in the following.

2.5.1 Source-Centric Cooperation (ScC)

- *Rate Control:* At every time slot t, the source node determines $x_k(t)$;

$$\max_{\boldsymbol{x}} \left[MU_k(x_k(t)) - \mu_k(t)x_k(t)\right]$$
$$\text{s.t. } x_k(t) \leq R_k^{max} \tag{2.15}$$

- *Cellular Link Scheduler:* At every time slot t, the source node determines $x_{k,k}(t)$ and $x_{n,k}(t)$;

$$\max_{\boldsymbol{x}} \mu_k(t)x_{k,k}(t) + \sum_{n \in \mathcal{N}-\{k\}} (\mu_k(t) - \nu_{n,k}(t))x_{n,k}(t)$$
$$\text{s.t. } \{x_{n,k}(t)\}_{\forall n \in \mathcal{N}} \in \Gamma_{\boldsymbol{C}^c(t)}. \tag{2.16}$$

- *Cooperation & Local-Area Link Scheduler for Unicast:* At time slot t, node n determines the link rate $h_{n,k}(t)$;

$$\max_{\boldsymbol{h}} \sum_{k \in \mathcal{N}} \sum_{n \in \mathcal{N}-\{k\}} \nu_{n,k}(t)h_{n,k}(t)$$
$$\text{s.t. } \{h_{n,k}(t)\}_{\forall k \in \mathcal{N}, n \in \mathcal{N}-\{k\}} \in \Gamma_{\boldsymbol{C}_u^w(t)}. \tag{2.17}$$

- *Cooperation & Local-Area Link Scheduler for Broadcast:* At time slot t, node n determines the broadcast rate;

$$\begin{aligned} &\max_{\boldsymbol{f}} \sum_{k \in \mathcal{N}} \sum_{n \in \mathcal{N}-\{k\}} \sum_{\mathcal{J} in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} \nu_{n,k}(t) f_{n,J}(t) \\ &\text{s.t. } \{f_{n,\mathcal{J}}(t)\}_{\forall n \in \mathcal{N}, \mathcal{J} \in \mathcal{H} | k \notin \mathcal{J}} \in \Gamma_{\boldsymbol{C}_b^w(t)} \end{aligned} \tag{2.18}$$

where $h_{n,k}(t) = \sum_{\mathcal{J} \in \mathcal{H} | k \in \mathcal{J}, n \notin \mathcal{J}} f_{n,\mathcal{J}}(t)$.

2.5.2 Benefits of DcC over ScC

In this section, we explain the benefits of DcC over ScC in terms of overhead, delay, and practical deployment.

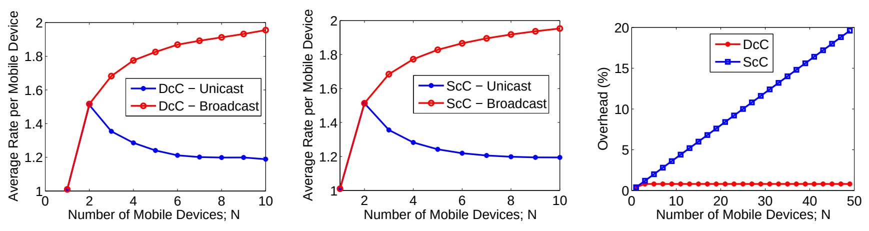
Overhead: ScC determines $x_k(t)$, $x_{k,k}(t)$, and $x_{n,k}(t)$ at the source node according to Equation 2.15, and Equation 2.16. Therefore, the source node should know the queue sizes; $\mu_k(t)$, $\nu_{n,k}(t)$, and cellular downlink properties $\Gamma_{\boldsymbol{C}^c(t)}$. Although $\mu_k(t)$ is constructed at the source node, $\nu_{n,k}(t)$ is constructed at mobile devices, and the cellular downlink properties $\Gamma_{\boldsymbol{C}^c(t)}$ are usually measured by mobile devices. Therefore, $\nu_{n,k}(t)$ and $\Gamma_{\boldsymbol{C}^c(t)}$ should be carried to the source node from each mobile device over a cellular uplink. These control messages introduce $O(N)$ overhead over each cellular uplink.

On the other hand, in DcC, mobile devices construct all the real and virtual queues and make all decisions. *E.g.,* mobile device k determines and requests $x_k(t)$ and $\max_{n \in \mathcal{N}}\{x_{k,n}(t)\}$ amount of video packets from the source. These request messages introduce $O(1)$ overhead over each cellular uplink. Thus, DcC reduces the overhead from $O(N)$ to $O(1)$, which is significant considering the fact that cellular link capacities are limited as the demand for cellular links is already high and keeps increasing

[16], [11]. Furthermore, since DcC introduces constant overhead over the cellular links, it provides scalability.

Delay: DcC improves packet delay over ScC thanks to employing virtual queues. Indeed, although the virtual queue sizes could be large in DcC, the real queue sizes could be significantly small as compared to the real queue sizes in ScC. Furthermore, the loss of control packets carrying queue size and cellular link quality information over cellular links increases real queue sizes in ScC. On the other hand, DcC makes all the decisions using local information in the mobile devices, so control packets are not carried over cellular links (only packet request messages are carried over the cellular links in DcC), so the loss of control packets does not affect DcC as much as ScC. The simulation results provided in the next section demonstrate the benefit of DcC in terms of delay as compared to ScC.

Practical Deployment: With the introduction of Dynamic Adaptive Streaming over HTTP (DASH) or MPEG-DASH [25], there is an increasing interest to client-based video streaming applications, *e.g.,* Netflix uses DASH [26]. According to DASH, the clients request video chunks at different rates using their connection level measurements. Our device-centric approach, since it operates at the client side, could be easily engaged with DASH to develop cooperative video streaming applications. Note that this could not be possible in ScC as it requires the video servers to be involved in the decision of which video chunks should be transmitted to the clients. We believe that our approach could be used to extend DASH for cooperative video streaming in mobile devices.



(a) DcC (b) ScC (c) Overhead

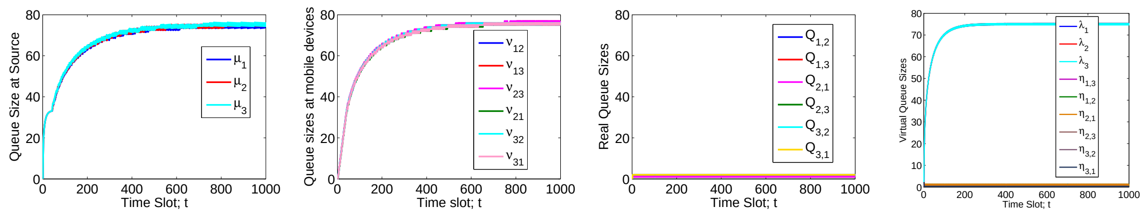
Figure 3. Average rate per mobile device in unicast and broadcast scenarios for (a) DcC and (b) ScC. (c) Percentage of overhead vs packet size.

2.5.3 Simulation Results

In this section, we demonstrate the benefits of DcC over ScC in terms of overhead and delay through simulations. We consider a cooperative video streaming system and topology shown in Figure 2 for different number of users.

Figure 3 presents the average rate per mobile device versus number of users for DcC and ScC. In this setup, the cellular and local area link rates are the same and 1 unit, and there is no loss over the links. As seen, in both DcC and ScC, broadcast improves over unicast as local area resources are used more efficiently. More importantly, DcC and ScC achieve the same rates for both unicast and broadcast, which is expected from Theorem 2. Note that we do not take into account the effect of overhead in this simulation, *i.e.,* the length of control packets are zero bytes.

Let us now consider overhead. We consider that queue size and channel state information are carried using 4 bytes from the mobile devices to the video servers in ScC, and the video rate request messages are carried from the mobile devices to the video servers using 4 bytes in DcC. The percentage of the



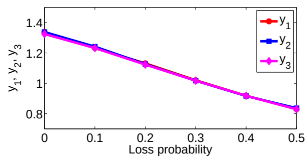
(a) ScC - $\mu_k(t)$ (b) ScC - $\nu_{n,k}(t)$ (c) DcC - Real Queues (d) DcC - Virtual Queues

Figure 4. Queue Sizes. (a) ScC. Queue sizes at the source. (b) ScC. Queue sizes at the mobile devices. (c) DcC. Real queue sizes at the mobile devices. (d) Virtual queue sizes at the mobile devices.

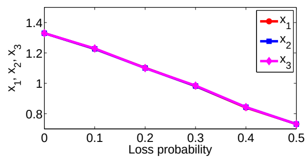
overhead as compared to packet size, which we assume to be 1000 bytes is presented in Figure 3(c). The overhead of ScC is increasing with the increasing number of users, while the overhead does not change with the increasing number of users for DcC. For example, the overhead is almost 20% when the number of mobile devices is 50. This means that 20% of the cellular link capacities should be allocated to carry the control messages in ScC. On the other hand, the overhead of DcC is small for any number of mobile devices.

Figure 4 presents queue size vs time for DcC and ScC. In this setup, both cellular and local area link rates are 1 units, and there is no loss over the links. As seen, the real queue sizes of ScC; *i.e.,* $\mu_k(t)$ and $\nu_k(t)$, could be very large, up to 75 packets. On the other hand, although virtual queue sizes could be also large in DcC, the real queue sizes; $Q_{n,k}(t)$ is very low. Thus, our scheme reduces queueing delay.

Figure 5 presents transmission rate towards each user versus the loss probability over the cellular links. In this setup, both cellular and local area link rates are 1 units, and there is loss only over the cellular links, *i.e.,* there is no loss over the local-area links. As expected, in both DcC and ScC, flow

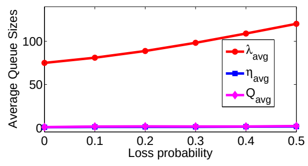


(a) DcC

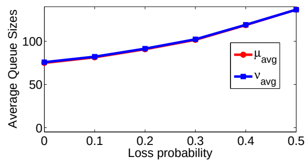


(b) ScC

Figure 5. Rate versus loss probability over the cellular links. (a) DcC. (b) ScC.



(a) DcC



(b) ScC

Figure 6. Rate versus loss probability over the cellular links. (a) DcC. (b) ScC.

rates decrease with increasing loss probability. However, DcC improves over ScC when the loss rate increases, because control packets are lost over the cellular links at high loss rates, and the source cannot make correct decisions in ScC. Figure 6 shows the average queue size versus the loss probability for the same setup. In particular, queue sizes are averaged over time and per-node queues. For example, λ_{avg} is the average queue size of λ_1, λ_2, and λ_3 which are time averages of $\lambda_1(t)$, $\lambda_2(t)$, and $\lambda_3(t)$, respectively. As seen, although the virtual queue sizes increase in DcC with the increasing loss probability, the real queue size Q_{avg} is very small and does not really increase with the increasing loss probability. On the other hand, the queue sizes in ScC, which are already very high as compared to DcC, increase significantly with increasing loss rate, which introduces significant delay.

2.6 Related Work

This work combines ideas from cooperation, network utility maximization, and stochastic network control.

When several users are interested in the same content, cooperative streaming is promising to improve throughput. For instance, [27], [28], [29] consider a scenario in which device-to-device and cellular connections are used to disseminate the content, considering the social ties and geographical proximity for cooperation. Cooperation between mobile devices for content dissemination taking into account social ties, has been studied extensively [30,31]. Cooperative video streaming systems are implemented over mobile devices in [32, 33]. As compared previous work, the goal of this chapter is to design device-centric cooperation scheme.

The NUM framework is promising to understand how different layers and/or algorithms, such as flow control, congestion control, and routing should be designed and optimized [22], [34]. We follow a similar approach, but we formulate the NUM framework considering the specific requirements such as device-centric design of the cooperative mobile devices.

The traditional source-centric, and backpressure-based stochastic network control algorithms have emerged from the pioneering work in [19], [20], which showed that in wireless networks where nodes route packets and make scheduling decisions based on queue backlog differences, one can stabilize queues for any feasible traffic. It has also been shown that backpressure can be combined with flow control to provide utility-optimal operation guarantee [21]. Recently, receiver-based flow control scheme is developed for overloaded networks [35]. As compared to previous work, our scheme is designed for cooperative mobile devices, and it creates virtual flows and queues to move control functionality to mobile devices, and reduces the overhead over cellular links and delay, which was not the focus of the previous work.

CHAPTER 3

ENERGY-AWARE COOPERATIVE COMPUTATION IN MOBILE DEVICES

The contents of this chapters are based on our work that is published in the proceedings of 2016 IFIP Networking conference [3] and a journal under submission. ©2016 IEEE. Reprinted, with permission, from [3].

New data intensive applications, which are continuously emerging in daily routines of mobile devices, significantly increase the demand for data, and pose a challenge for current wireless networks due to scarce resources. Although bandwidth is traditionally considered as the primary scarce resource in wireless networks, the developments in communication theory shifts the focus from bandwidth to other scarce resources including processing power and energy. Especially, in device-to-device networks, where data rates are increasing rapidly, processing power and energy are becoming the primary bottlenecks of the network. Thus, it is crucial to develop new networking mechanisms by taking into account the processing power and energy as bottlenecks. In this chapter, we develop an energy-aware cooperative computation framework for mobile devices. In this setup, a group of cooperative mobile devices, within proximity of each other, (i) use their cellular or Wi-Fi (802.11) links as their primary networking interfaces, and (ii) exploit their device-to-device connections (*e.g.,* Wi-Fi Direct) to overcome processing power and energy bottlenecks. We evaluate our energy-aware cooperative computation framework on a testbed consisting of smartphones and tablets, and we show that it brings significant performance benefits.

3.1 Background

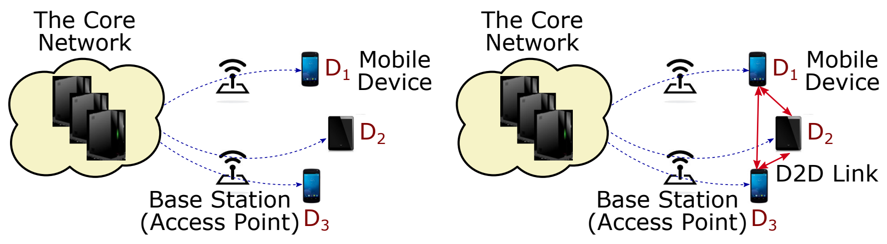
The dramatic increase in mobile applications and the number of devices demanding for wireless connectivity poses a challenge in today's wireless networks, and calls for new networking mechanisms.

One of the promising solutions to address the increasing data and connectivity demand is Device-to-Device (D2D) networking. As illustrated in Figure 7(a), the default operation in current wireless networks is to connect each device to the Internet via its cellular or Wi-Fi interface. The D2D connectivity idea, which is illustrated in Figure 7(b), breaks this assumption: it advocates that two or more devices in close proximity can be directly connected, *i.e.,* without traversing through auxiliary devices such as a base station or access point. D2D networking, that can be formed by exploiting D2D connections such as Wi-Fi Direct [36], is a promising solution to the ever increasing number and diversity of applications and devices. In this context, it is crucial to identify scarce resources and effectively utilize them to fully exploit the potential of D2D networking.

Although bandwidth is traditionally considered as the primary scarce resource in wireless networks, in D2D networks, thanks to close proximity among devices and the developments in communication theory, the main bottleneck shifts from bandwidth to other scarce resources including processing power and energy.

Next, we present our pilot study demonstrating that processing power can be more pronounced as a bottleneck than bandwidth in D2D networks.

Pilot Study: We developed a prototype for this pilot study as shown in Figure 8(a), where a mobile device D_2 receives data from another device D_1 over a Wi-Fi Direct link. We use Android operating system [37] based Nexus 7 tablets [38] as mobile devices. In this experiment, after receiving the packets,

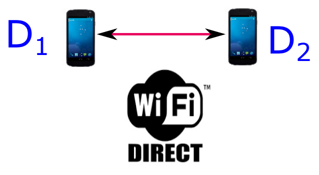


(a) The default operation (b) D2D connectivity

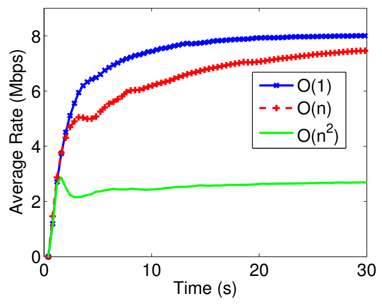
Figure 7. (a) The default operation for the Internet connection. (b) D2D connectivity: two or more mobile devices can be connected directly, *i.e.,* without traversing through the core network, if they are in close proximity by exploiting local area connections such as Wi-Fi Direct

the mobile device D_2 performs operations with complexities of $\mathcal{O}(1)$, $\mathcal{O}(n)$, and $\mathcal{O}(n^2)$ above the transport layer (TCP), where n is the packet size, and the operations we perform are counting the bytes in the packets. In particular, $\mathcal{O}(1)$, $\mathcal{O}(n)$, and $\mathcal{O}(n^2)$ correspond to (i) no counting, (ii) counting every byte in a packet once, and (iii) counting every byte in a packet n times, respectively. We demonstrate in Figure 8(b) the received rate at the mobile device D_2 (note that this is the rate we measure at the mobile device D_2 after performing computations) versus time. This figure demonstrates that the received rate decreases significantly when the complexity increases. $\square$

Our pilot study shows that even if actual bandwidth is high and not a bottleneck, processing power could become a bottleneck in D2D networks. Similar observations can be made for the energy bottleneck. Furthermore, with the advances in communication theory, *e.g.,* millimeter wave communication [39], it is expected that data rates among devices in close proximity will increase significantly, which will make processing power and energy more pronounced as bottlenecks. However, existing



(a) Setup



(b) Rate vs Time

Figure 8. Pilot Study: (a) Setup: Data is transmitted from mobile device D_1 to another mobile device D_2. In this setup, the mobile devices are Android operating system (OS) based Nexus 7 tablets. The specific version of the Anroid OS is Android Lollipop 5.1.1. The devices have 16GB storage, 2GB RAM, Qualcomm Snapdragon S4 Pro, 1.5GHz CPU, and Adreno 320, 400MHz GPU. Packet size is $500B$. (b) Transmission rate versus time for different computational complexities at the receiver side. Note that we present the rate that we measure at the mobile device after performing the computations. The presented rates are the averages over 10 seeds

applications, algorithms, and protocols are mainly designed by assuming that bandwidth is the main bottleneck. Thus, it is crucial to develop new networking mechanisms when bandwidth is not the primary bottleneck, but processing power and energy are.

In this chapter, our goal is to create group of devices that help each other cooperatively by exploiting high rate D2D connections to overcome the processing power and energy bottlenecks. The next example demonstrates our approach.

Example 2. *Let us consider Figure 7(a) again, where device D_1 would like to receive a file from a remote resource via its cellular or Wi-Fi connection. Assume that the cellular (or Wi-Fi) rates of all devices are 1Mbps, but device D_1 can receive data with 500kbps rate due to processing power bottleneck,* i.e., *device D_1 has limited processing power (similar to our pilot study we presented earlier). In a traditional system, D_1 will behave as a single end point, so its receiving rate will be limited to 500kbps. On the other hand, if devices D_1, D_2, and D_3 will behave as a group and cooperate, then devices D_2 and D_3 can also receive and process 500kbps portions of data, and transmit the processed data to device D_1 over D2D connections. This increases the receiving rate of device D_1 to 1.5Mbps from 500kbps, which is a significant improvement.*

This example could be extended for scenarios when energy (battery of mobile devices) is limited. For example, if device D_2's battery level is too low, its participation to the group activity should be limited. $\square$

Application Areas. The scenario in the above motivating example could arise in different practical applications from health, education, entertainment, and transportation systems. The following are some example applications. *Health:* A person may own a number of health monitoring devices (activity monitoring, hearth monitoring, etc.) which may need updates from the core network. These updates - potentially coded for error correction, compression, and security reasons - should be processed (decoded) by these devices. Processing takes time, which may lead to late reaction to the update (which may require timely response) and energy consumption. On the other hand, by grouping mobile devices, the person's smartphone or tablet could receive the update, process, and pass the processed data to the health monitoring devices via high rate D2D links. *Education & Entertainment:* A group of students may want to watch the video of a lecture from an online education system (or an entertainment video) while sitting together and using several mobile devices. In this setup, one of the devices can download a base layer of a video and decode, while the other devices could download enhancement layers and decode. The decoded video layers could be exchanged among these mobile devices via high rate D2D links. As in the motivating example, if a device's download and decoding rate is limited to 500kbps, it could be improved to 1.5Mbps with the help of other devices. □

Note that the processing overhead in these applications could be due to any computationally intensive task related to data transmission. For example, for video transmission applications, H.264/AVC decoders introduce higher computational complexity when higher quality guarantees are needed [40], [41]. Another example could be network coding; data could be network coded at the source to improve throughput, error correction, packet randomization potential of network coding [42]. However, most of the network coding schemes introduce high computational complexity at the receiver side; $\mathcal{O}(n^3)$, [43], [44], which limits the transmission rate. Encryption could be another example that introduces processing overhead [45]. As seen, there exist several applications and scenarios where bandwidth and energy could be bottlenecks, while bandwidth is not the bottleneck. This makes our approach demonstrated in Example 2 part promising.

In this work, we develop an *energy-aware cooperative computation* framework for mobile devices. In this setup, a group of cooperative mobile devices, within proximity of each other, (i) use their cellular or Wi-Fi (802.11) links as their primary networking interfaces, and (ii) exploit their D2D connections (Wi-Fi Direct) for cooperative computation. Our approach is grounded on a network utility maximization (NUM) formulation of the problem and its solution [22]. The solution decomposes into several parts with an intuitive interpretation, such as flow control, computation control, energy control, and cooperation & scheduling. Based on the structure of the decomposed solution, we develop a stochastic algorithm; *energy-aware cooperative computation*.[1]

The structure of the rest of the chapter is as follows. Section 3.2 presents related work. Section 3.3 gives an overview of the system model. Section 3.4 presents the NUM formulation of our cooperative computation scheme. Section 3.5 presents our stochastic algorithm; *EaCC*. Section 3.6 evaluates the performance of our scheme in a real testbed.

3.2 Related Work

This work combines ideas from D2D networking, network utility maximization, and stochastic network control.

The idea of D2D networking is very promising to efficiently utilize resources, so it has found several applications in the literature. In particular, D2D connections are often used to form cooperative groups for data streaming applications, and for the purpose of (i) content dissemination among mobile devices [30], [31], (ii) cooperative video streaming over mobile devices [17], [18], [32], [33], and (iii) creating multiple paths and providing better connectivity by using multiple interfaces simultaneously [48], [49]. As compared to this line of work, we investigate the impact of processing power and energy in D2D networks, and develop mechanisms to effectively utilize these scarce resources.

---

[1]Note that our work focuses on cooperative resource utilization in mobile devices. In this sense, our work is complementary to and synergistic with: (i) creating incentive mechanisms in D2D networks, and (ii) providing privacy and security for D2D users [46], [47]. Looking into the future, it is very likely that our proposed work on the design, analysis, and implementation of cooperative resource utilization is gracefully combined with the work on creating incentives and providing privacy and security.

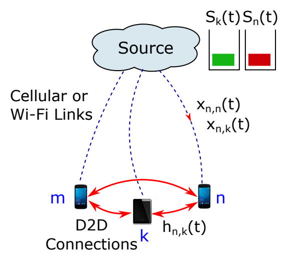
D2D networking is often used for the purpose of offloading cellular networks. For example, previous work [27], [28], [30] disseminates the content to mobile devices by taking advantage of D2D connections to relieve the load on cellular networks. Instead of offloading to cellular networks, our goal is to create energy-aware cooperation framework to overcome the processing power and energy bottlenecks of mobile devices.

There is an increasing interest in computing by using devices at the edge [50], [51], [52], [53] as a cheaper and delay-efficient alternative to remote clouds. This approach, sparking a lot of interest, led to some very interesting work in the area [54], [55], [56]. As compared to this line of work, we focus on processing power and energy bottlenecks in mobile devices and address the problem by (i) exploiting D2D connections, and (ii) developing energy-aware cooperative computation mechanism.

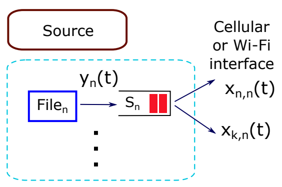
An integral part of our proposed work in this task is to develop efficient resource allocation mechanisms. In that sense, our approach is similar to the line of work emerged after the pioneering work in [19], [20], [21]. However, our focus is on energy-aware cooperative computation, which is not considered in previous work.

3.3 System Model

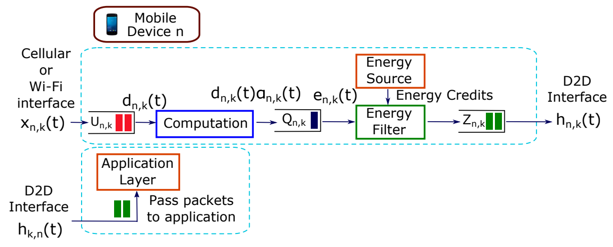
We consider a cooperative system setup with N mobile devices, where $\mathcal{N}$ is the set of the mobile devices. Our system model for three nodes is illustrated in Figure 9(a). The source in Figure 9(a) represents the core network and base stations (access points). This kind of abstraction helps us focus on the bottlenecks of the system; processing power, energy of mobile devices, and downlink/uplink data rates. In this setup, mobile devices communicate via D2D connections such as Wi-Fi Direct, while the



(a) System Model



(b) Building blocks of the source.



(c) Building blocks of mobile device n

Figure 9. (a) System model for the scenario of three devices; n, m, k. The source in this model represents the core network and base stations (access points). (b) Building blocks of the source. File$_n$, $\forall n$ is read and inserted in the buffer $S_n(t)$, and packets are transmitted from $S_n(t)$. $x_{n,k}(t)$ is the transmission rate of the packets from the source towards device n, and these packets will be processed by device n and forwarded to device k. (c) Building blocks of mobile device n. If packets are received from the source via cellular and Wi-Fi interfaces, then they go to the computation and energy control blocks. If packets are received from other mobile devices via D2D interface, they are directly passed to the application

source communicates with mobile devices via cellular or Wi-Fi links. We consider in our analysis that time is slotted and t refers to the beginning of slot t.

Connecting Devices Together: The total flow rate towards device n in Figure 9(a) (as also explained in Figure 9(b)) is $\sum_{k\in\mathcal{N}} x_{n,k}(t)$, where $x_{n,n}(t)$ is the transmission rate of the packets from the source towards device n, and these packets will be used by device n. Note that $x_{n,k}(t)$ is the transmission rate of the packets from the source towards device n, and these packets will be processed by device n and forwarded to device k. On the other hand, $y_n(t)$ is the total flow rates targeting device n as demonstrated in Figure 9(b). The source constructs a queue $S_n(t)$ for the packets that will be transmitted to the mobile device n. The evolution of $S_n(t)$ based on $y_n(t)$ and $x_{k,n}(t)$ is expressed as

$$S_n(t+1) \leq \max[S_n(t) - \sum_{k\in\mathcal{N}} x_{k,n}(t), 0] + y_n(t), \tag{3.1}$$

where the inequality comes from the fact that there may be less than $y_n(t)$ packets arriving into $S_n(t)$ at time t in practice (*e.g.,* in real time applications, the number of available packets for transmission could be limited).

The flow rate $y_n(t)$ is coupled with a utility function $g_n(y_n(t))$, which we assume to be a strictly concave function of $y_n(t)$. This requirement is necessary to ensure stability and utility optimality of our algorithms. The ultimate goal in our resource allocation problem is to determine the flow rates; $y_n(t)$ which maximize the sum utility $\sum_{n\in\mathcal{N}} g_n(y_n(t))$.

Finally, flow rate over D2D connection between device n and k is $h_{n,k}(t)$, $k \neq n$. Note that $h_{n,k}(t)$ is to help node k using node n as a processing device.

Inside a Mobile Device: In each device, we develop different modules depending on where data is arriving from (as shown in Figure 9(c)); *i.e.,* from the source via cellular or Wi-Fi interface, or other mobile devices via D2D interfaces.

When data is arriving from a D2D interface, it is directly passed to the application layer, as this data is already processed by another device.[1] On the other hand, when data is arriving from the source via cellular or Wi-Fi interfaces, packets go through multiple queues as shown in Figure 9(c), where $U_{n,k}$, $Q_{n,k}$, and $Z_{n,k}$ represent three different queues constructed at mobile device n for the purpose of helping node k. Incoming packets via cellular or Wi-Fi links are stored in $U_{n,k}$, which then forwards the packets to *computation* block with rate $d_{n,k}(t)$. The computation block processes the packets, and pass them to queue $Q_{n,k}$. Note that the output rate from computation block is $d_{n,k}(t)\alpha_{n,k}(t)$, where $\alpha_{n,k}(t)$ is a positive real value. This value captures any possible rate changes at the computation block, *i.e.,* $\alpha_{n,k}(t)$ is a rate shaper. For example, if the computation block is H.264/AVC decoder or transcoder, we expect that the rate at the output of the computation block should be higher than the input. Thus, $\alpha_{n,k}(t)$ captures this fact for any n, k, t. On the other hand, if there is no rate change after the processing, then $\alpha_{n,k}(t) = 1$.

The processed (and possibly rate shaped) packets are queued at $Q_{n,k}(t)$ and passed to *energy filter*. The energy filter is coupled to the energy source, which determines the amount of energy that can be spent to support the tasks at each slot. The amount of energy is determined according to *energy credits*. In particular, the energy source, depending on the battery level as well as the estimate on the

[1]Note that we make this assumption only for the theoretical modeling and analysis; this assumption will be relaxed in practice in Section 3.6.

expected battery consumption in the near future, calculates the number of packets that can be supported by the mobile device, and the same number of energy credits enter the energy filter. (Note that both energy filter, energy source, and energy credits are not real, but virtual entities, so they can be modeled by using a few counters in practice.) Thus, at each transmission slot, packets are transmitted from $Q_{n,k}(t)$ to $Z_{n,k}(t)$ with rate $e_{n,k}(t)$ if there exist energy credits in the energy filter. Finally, packets from $Z_{n,k}(t)$ are transmitted to application if device n is the destination of the data (*i.e.,* $n = k$), or they are transmitted to the original destination via D2D interface with rate $h_{n,k}(t)$.

The computation and energy filter blocks in Figure 9(c) model the processing and energy bottlenecks of the mobile device, respectively. If packets in $U_{n,k}$ increase too much, this means that the computation block, hence processing power, is the bottleneck, so node n should not receive much packets from the source. Similarly, if $Q_{n,k}$ increases too much, this means that energy filter is the bottleneck, so again node n should not receive much packets. Note that there could be also some buildup in $Z_{n,k}$ if the link between node n and k is the bottleneck of the system, and it should be taken into account when the energy-aware cooperative computation framework is developed.

Also, it is crucial in our system model to put energy filter after the computation block, because if device n will help device k, the actual amount of packets that are supposed to be transmitted are the processed packets, which will cause energy consumption (*i.e.,* not the packets before processing).

TABLE I

EVOLUTION OF QUEUES $U_{N,K}(T)$, $Q_{N,K}(T)$, AND $Z_{N,K}(T)$.

$U_{n,k}(t+1) \leq \max[U_{n,k}(t) - d_{n,k}(t), 0] + x_{n,k}(t)$
$Q_{n,k}(t+1) \leq \max[Q_{n,k}(t) - e_{n,k}(t), 0] + d_{n,k}(t)\alpha_{n,k}(t)$
$Z_{n,k}(t+1) \leq \max[Z_{n,k}(t) - h_{n,k}(t), 0] + e_{n,k}(t)$

Based on the above intuitions and observations, we will develop our resource allocation problem and algorithm in the next sections. The evolution of the queues $U_{n,k}(t)$, $Q_{n,k}(t)$, and $Z_{n,k}(t)$ are provided in Table I.[1]

Links: In our system model, we consider two scenarios: (i) cellular + Wi-Fi Direct, and (ii) Wi-Fi + Wi-Fi Direct. In both cases, the D2D links between mobile devices are Wi-Fi Direct. In the first case, *i.e.,* in cellular + Wi-Fi Direct, the links between the source and mobile devices are cellular, while they are Wi-Fi in the second case, *i.e.,* in Wi-Fi + Wi-Fi Direct. These two scenarios are different from each other, because in the first scenario, cellular and Wi-Fi Direct links could operate simultaneously as they use different parts of the spectrum. On the other hand, in the second scenario, both Wi-Fi and Wi-Fi Direct use the same spectrum, so they time share the available resources. Our model and energy-aware

[1]We note that the buffer sizes are finite and buffer overflows occur in practice and in our implementation in Section 3.6, although they are assumed to be very large for the sake of analysis in Sections 2.3 and 3.5.

cooperative computation framework are designed to operate in both scenarios. Next, we provide details about our link models.[1]

In the system model in Figure 9(a), each mobile device $n \in \mathcal{N}$ is connected to the Internet via its cellular or Wi-Fi link. At slot t, $\boldsymbol{C}^s(t)$ is the channel state vector of these links, where $\boldsymbol{C}^s(t) = \{C_1^s(t)$, ..., $C_n^s(t)$, ..., $C_N^s(t)\}$. We assume that $C_n^s(t)$ is the state of the link between the source and mobile device n, and it takes "ON" and "OFF" values depending on the state of the channel. Without loss of generality, if mobile device n does not have Internet connection, then $C_n^s(t)$ is always at "OFF" state, which means there is no cellular or Wi-Fi connection.

Since we consider that mobile devices are in close proximity and transmission range, they form a fully connected clique topology. At slot t, $\boldsymbol{C}^w(t)$ is the channel state vector of the D2D links, where $\boldsymbol{C}^w(t) = \{C_{1,2}^w(t)$, ..., $C_{n,k}^w(t)$, ..., $C_{N-1,N}^w(t)\}$. We assume that $C_{n,k}^w(t)$ is the state of the D2D link between node n and k.

We consider protocol model in our formulations [23], where each mobile device can either transmit or receive at the same time at the same frequency. Assuming that $\boldsymbol{C}(t) = \{\boldsymbol{C}^s(t), \boldsymbol{C}^w(t)\}$ is the channel state vector of the system including both the links between the source and mobile devices as well as among mobile devices, $\Gamma_{\boldsymbol{C}(t)}$ denotes the set of the link transmission rates feasible at time slot t depending on our protocol model. In particular, for cellular + Wi-Fi Direct setup, $\Gamma_{\boldsymbol{C}(t)}$ is the set that

[1]Note that the link models described in this section provide a guideline in our algorithm development and basis in our theoretical analysis. However, in Section 3.6, we relax the link model assumptions we made in this section, and evaluate our algorithms on real devices and using real links.

allows more links to operate at the same time, while for the Wi-Fi + Wi-Fi Direct setup, $\Gamma_{\boldsymbol{C}(t)}$ is a more limited set due to the interference among the links.

3.4 Problem Formulation

In this section, we characterize the stability region of the energy-aware cooperative computation problem, and formulate network utility maximization (NUM) framework. The solution of the NUM framework provides us insights for developing the stochastic control algorithms in the next section.[1]

3.4.1 Stability Region

We provide the stability region of the cooperative computation system for both cellular + Wi-Fi Direct and Wi-Fi + Wi-Fi Direct setups. First, the flow conservation constraint at the source should be $y_n \leq \sum_{k \in \mathcal{N}} x_{k,n}$ to stabilize the system. This constraint requires that the total outgoing rate from the source, *i.e.,* $\sum_{k \in \mathcal{N}} x_{k,n}$ should be larger than the generated rate y_n.

Furthermore, the following flow conservation constraints inside a mobile device should be satisfied for stability; $x_{n,k} \leq d_{n,k}$, $d_{n,k}\alpha_{n,k} \leq e_{n,k}$, and $e_{n,k} \leq h_{n,k}$. These constraints are necessary for the stability of queues $U_{n,k}$, $Q_{n,k}$, and $Z_{n,k}$, respectively. Finally, the transmission rates over the links should be feasible, *i.e.,* $\{x_{n,k}, h_{n,k}\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}} \in \Gamma_{\boldsymbol{C}}$.

Thus, we define the stability region as $\Lambda = \{\{y_n, x_{n,k}, d_{n,k}, e_{n,k}, h_{n,k}\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}} \mid y_n,\ x_{n,k},\ d_{n,k},\ e_{n,k},\ h_{n,k} \geq 0,\ \forall n \in \mathcal{N}, k \in \mathcal{N},\ y_n \leq \sum_{k \in \mathcal{N}} x_{k,n},\ x_{n,k} \leq d_{n,k},\ d_{n,k}\alpha_{n,k} \leq e_{n,k},\ e_{n,k} \leq h_{n,k},\ \{x_{n,k}, h_{n,k}\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}} \in \Gamma_{\boldsymbol{C}}\}$.

[1]Note that NUM optimizes the average values of the parameters that are defined in Section 8.2. By abuse of notation, we use a variable, *e.g.,* ϕ as the average value of $\phi(t)$ in our NUM formulation if both ϕ and $\phi(t)$ refers to the same parameter.

3.4.2 NUM Formulation

Now, we characterize our NUM problem.

$$\begin{aligned}&\max_{\boldsymbol{y}} \sum_{n\in\mathcal{N}} g_n(y_n)\\ &\text{s.t. } y_n, x_{n,k}, d_{n,k}, e_{n,k}, h_{n,k} \in \Lambda,\ \forall n \in \mathcal{N}, k \in \mathcal{N}\end{aligned} \tag{3.2}$$

The objective of the NUM problem in (Equation 3.2) is to determine $y_n, x_{n,k}, d_{n,k}, e_{n,k}, h_{n,k}$ for $\forall n \in \mathcal{N}, k \in \mathcal{N}$, which maximizes the total utility $\sum_{n\in\mathcal{N}} g_n(y_n)$.

3.4.3 NUM Solution

Lagrangian relaxation of the flow conservation constraints that characterize the stability region Λ gives the following Lagrange function:

$$\begin{aligned}L = &\sum_{n\in\mathcal{N}} g_n(y_n) - \sum_{n\in\mathcal{N}} s_n(y_n - \sum_{k\in\mathcal{N}} x_{k,n}) - \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} u_{n,k}(x_{n,k} - d_{n,k}) -\\ &\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} q_{n,k}(d_{n,k}\alpha_{n,k} - e_{n,k}) - \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} z_{n,k}(e_{n,k} - h_{n,k})\end{aligned} \tag{3.3}$$

where s_n, $u_{n,k}$, $q_{n,k}$, and $z_{n,k}$ are the Lagrange multipliers. Note that we will convert these Lagrange multipliers to queues S_n, $U_{n,k}$, $Q_{n,k}$, and $Z_{n,k}$ when we design our stochastic algorithm in the next section.

The Lagrange function in (Equation 3.3) is decomposed into sub-problems such as flow, computation, and energy controls as well as cooperation and scheduling. The solutions of (Equation 3.3) for y_n, $d_{n,k}$, $e_{n,k}$, $x_{n,k}$, and $h_{n,k}$ are expressed as:

- <u>Flow control:</u>

$$\max_{\boldsymbol{y}} \sum_{n\in\mathcal{N}} (g_n(y_n) - y_n s_n) \tag{3.4}$$

- <u>Computation control:</u>

$$\max_{\boldsymbol{d}} \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} d_{n,k}(u_{n,k} - q_{n,k}\alpha_{n,k}) \tag{3.5}$$

- <u>Energy control:</u>

$$\max_{\boldsymbol{e}} \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} e_{n,k}(q_{n,k} - z_{n,k}) \tag{3.6}$$

- <u>Cooperation & Scheduling:</u>

$$\begin{aligned} &\max_{\boldsymbol{x},h} \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} [x_{n,k}(s_k - u_{n,k}) + z_{n,k}h_{n,k}] \\ &\text{s.t. } \{x_{n,k}, h_{n,k}\}_{\forall n\in\mathcal{N}, k\in\mathcal{N}} \in \Gamma_{\boldsymbol{C}} \end{aligned} \tag{3.7}$$

Next, we design a stochastic algorithm; energy-aware cooperative computation inspired by the NUM solutions in Equation 3.4, Equation 3.5, Equation 3.6, and Equation 3.7.

3.5 Energy-Aware Cooperative Computation

Now, we provide our energy-aware cooperative computation algorithm which includes *flow control*, *computation control*, *energy control*, and *cooperation & scheduling*.

Energy-Aware Cooperative Computation (EaCC):

- *Flow Control:* At every time slot t, $y_n(t)$ is determined by maximizing $\max_{\boldsymbol{y}} [Mg_n(y_n(t)) - S_n(t)y_n(t)]$ subject to $y_n(t) \leq R_n^{\max}$, where $R_n^{\max}$ is a positive constant larger than the transmission rate from the source, and M is a large positive constant. Note that $S_n(t)$ is the queue size at the source of flow and stores packets that are supposed to be transmitted to mobile device n. After $y_n(t)$ is determined, $y_n(t)$ packets are inserted in queue $S_n(t)$ (as illustrated in Figure 9(a)).
- *Computation Control:* At every time slot t, the computation control algorithm at device n determines $d_{n,k}(t)$ by optimizing

$$\begin{aligned} \max_{\boldsymbol{d}} \; & \sum_{k \in \mathcal{N}} d_{n,k}(t)[U_{n,k}(t) - Q_{n,k}(t)\alpha_{n,k}(t)] \\ \text{s.t.} \; & \sum_{k \in \mathcal{N}} d_{n,k}(t) \leq D_n^{\max} \end{aligned} \tag{3.8}$$

where $D_n^{\max}$ is a positive constant larger than the processing rate of the computation block in device n dedicated to help device k. The interpretation of Equation 3.8 is that at every time slot t, $d_{n,k^*} = D_n^{\max}$ packets are passed to the computation block (in Figure 9(b)) if (i) $U_{n,k^*}(t) - Q_{n,k^*}(t) > 0$, where k^* is the mobile device that maximizes Equation 3.8, and (ii) the computation block is idle. Otherwise, no packets are sent to the computation block. The packets that are being processed by the computation block are passed to $Q_{n,k}(t)$. Note that some computation blocks may require to receive a group of packets to be able to process them. In that case, $D_n^{\max}$ is arranged accordingly (*i.e.,* it can be increased to transfer a group of packets).

- *Energy Control:* At every time slot t, the energy control algorithm at device n determines $e_{n,k}(t)$ by optimizing

$$\begin{aligned} &\max_{\boldsymbol{e}} \sum_{k \in \mathcal{N}} e_{n,k}(t)[Q_{n,k}(t) - Z_{n,k}(t)] \\ &\text{s.t.} \sum_{k \in \mathcal{N}} e_{n,k}(t) \leq E_{n,k}^{\max} \end{aligned} \tag{3.9}$$

where $E_n^{\max}$ is a positive constant larger than the energy capacity of device n dedicated to help device k. The interpretation of Equation 3.9 is that at every time slot t, $e_{n,k^*} = E_n^{\max}$ packets are passed to the energy filter (as illustrated in Figure 9(b)) if $Q_{n,k^*}(t) - Z_{n,k^*}(t) > 0$, where k^* is the mobile device that maximizes Equation 3.9. Otherwise, no packets are sent to the energy filter. The packets passing through the energy filter are inserted in $Z_{n,k}(t)$.

- *Scheduling & Cooperation:* At every time slot t, the scheduling and cooperation algorithm determines transmission rates over links, *i.e.,* $x_{n,k}(t)$ and $h_{n,k}(t)$ by maximizing

$$\begin{aligned} &\max_{\boldsymbol{x},\boldsymbol{h}} \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} [x_{n,k}(t)(S_k(t) - U_{n,k}(t)) + h_{n,k}(t) Z_{n,k}(t)] \\ &\text{s.t. } \boldsymbol{x}, \boldsymbol{h} \in \Gamma_{\boldsymbol{C}(t)} \end{aligned} \tag{3.10}$$

For cellular + Wi-Fi Direct system, Equation 3.10 is decomposed into two terms: maximizing $\sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} x_{n,k}(t)(S_k(t) - U_{n,k}(t))$ and $\sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} h_{n,k}(t) Z_{n,k}(t)$, because cellular and Wi-Fi Direct transmissions operate simultaneously and transmission over one link does not affect the other. On the other hand, for Wi-Fi + Wi-Fi Direct setup, the joint optimization in Equation 3.10 should be solved.

Note that transmissions over all links are unicast transmissions in our work, where unicast is dominantly used in practice over cellular, Wi-Fi, and Wi-Fi Direct links. Also, it is straightforward to extend our framework for broadcast transmissions.

Theorem 3. *If channel states are i.i.d. over time slots, and the arrival rates $E[y_n(t)] = A_n, \forall n \in \mathcal{N}$ are interior of the stability region Λ, then energy-aware cooperative computation stabilizes the network and the total average queue sizes are bounded.*

Furthermore, if the channel states are i.i.d. over time slots, and the traffic arrival rates are controlled by the flow control algorithm of energy-aware cooperative computation, then the admitted flow rates converge to the utility optimal operating point with increasing M.

Proof: The proof is provided in Appendix A. ■

Our energy-aware cooperative computation framework has several advantages: (i) distributed, (ii) takes into account scarce resources such as processing power and energy in addition to bandwidth to make control decisions, and (iii) utilizes available resources; processing power, energy, and bandwidth in a utility optimal manner. Theorem 3 shows the theoretical performance guarantees of our framework. Next, we will focus on its performance in a practical setup.

3.6 Performance Evaluation

In this section, we evaluate our *energy-aware cooperative computation* (EaCC) scheme using a testbed that consists of Android-based smartphones and tablets. The evaluation results show that our scheme significantly improves throughput as compared to (i) no-cooperation, where each device receives its content from the source without cooperating with other devices, and (ii) cooperation, where multiple mobile devices cooperate, but the cooperating devices do not do computation and energy control for other devices (mobile devices just receive packets from the source, and relay them to other mobile devices without processing and energy control). Next, we present testbed setup and results in detail.

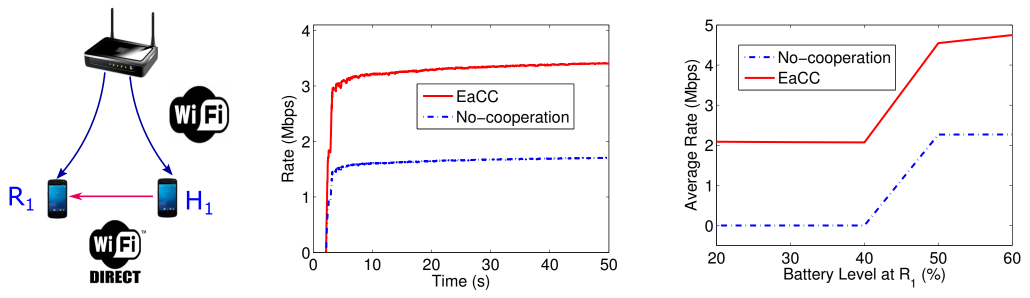
3.6.1 Setup & Implementation Details

Devices: We implemented a testbed of the setup shown in Figure 9(a) using real mobile devices, specifically Android 5.1.1 based Nexus 5 smartphones and Nexus 7 tablets.

We classify devices as (i) a source device, which acts as the source in Figure 9(a), (ii) helper devices, which receive data from the source, process it, and transmit to other devices (receivers) to help them, and (iii) receiver devices, which receive data from both the source device and the helpers. Note that a device could be both receiver and a helper device depending on the configuration.

Applications: We consider two types of computationally intensive applications in our evaluation; *Byte Counting* and *Video Streaming*.

Byte Counting. In byte counting application, the computation block counts the bytes in packets. In particular, similar to the pilot study in the introduction, $\mathcal{O}(1)$, $\mathcal{O}(n)$, and $\mathcal{O}(n^2)$ correspond to (i) no counting, (ii) counting every byte in a packet once, and (iii) counting every byte in a packet n times, respectively. In this application a receiver device processes data arriving from the source, but it does not



(a) System Model (b) Rate vs. Time (c) Rate vs. Energy

Figure 10. (a) System model consisting of a source device, one receiver, and one helper. Wi-Fi is used between the source and the receiver device (R_1) and the helper device (H_1), while Wi-Fi Direct is used to connect R_1 to H_1. (b) Average rate versus time for the setup shown in (a) for the case that all the devices are Android-based Nexus 7 tablets. (c) Average rate versus energy level at receiver device R_1. In this setup, all devices are Android-based Nexus 5 smartphones. In both (b) and (c), the average rate is calculated as the average over 10 trials (with different seeds). The computation under consideration in this experiment is $O(n^2)$, which counts the number of bytes in a packet for each byte in the packet (*i.e.,* recursive counting).

process the data arriving from helpers as the helpers send already processed data. Note that we relax this assumption in our video streaming application.

Video Streaming. In video streaming application, a source device sends video packets to receivers, while helper devices help decoding video. The source device determines the number of video packets that should be sent to the receiver and helpers by taking into account each device's computation and energy capabilities. In this setup, when a helper receives video chunks, it pre-processes the chunks by using an open source Android-based transcoder [57].

Note that the task of the transcoder in the helper is not to fully decode the video chunks (as decoded video size becomes so huge that it cannot be transmitted from the helper to the receiver), but to make the receiver devices' decoding task easier. The transcoder makes this possible by changing the format of the encoded video chunks, and extracting the video chunks slightly (but not to the full extent). Note that the receiver should still decode the transcoded video chunks after receiving them, *i.e.,* the receiver always processes video chunks even if they are directly received from the source node, or helper nodes.

Integration to the Protocol Stack: We implemented our energy-aware cooperative computation (EaCC) framework as a slim layer between transport and application layers. In other words, we implemented our framework on top of TCP. This kind of implementation has benefits, because (i) mobile devices do not need rooting, and (ii) our framework and codes could be easily transferred to mobile devices using other operating systems such as iOS.

Source Configuration and EaCC Implementation: We implemented the source node in Figure 9 using a Nexus 5 smartphone. Basically, multiple files; File$_n$, File$_k$ requested by devices n and k are categorized if they are text or video files. If they are text files that belong to the byte counting application, they are read by using the public java class *BufferedInputStream* according to the flow control algorithm described in Section 3.5 and shown in Figure 9(b). The bytestream is packetized by setting each packet to $500B$, and packets are inserted into source buffers; $S_n(t)$, $S_k(t)$.

If the files are video files that belong to video streaming application, then a large video file is divided into small video chunks using a third party application; Boilsoft Video Splitter [58]. Each video chunk is divided into $500B$ video packets. These video packets are inserted into the source buffers $S_n(t)$, $S_k(t)$ according to the flow control algorithm as described in Section 3.5.

For both text and video files, we set the flow control parameters as; $M = 500$, $R_n^{\max} = 100$, and slot duration is $20msec$. We used $\log$ function as our utility function. In this setup, reading files, converting them into packets, and inserting packets into the input queues are done by multiple threads, *i.e.,* a thread runs for each file; File$_n$ in Figure 9(b).

The other set of threads at the source device make packet transmission decisions from the source device to receiver and helper devices. In particular, the source node collects $U_{n,k}(t)$ information from all mobile devices. At each time slot, the source node checks $S_k(t) - U_{n,k}(t)$, and if $S_k(t) - U_{n,k}(t) > 0$, then 100 packets are transmitted from $S_k(t)$ to the TCP socket at the source device for transmission to mobile device n.

EaCC Operation on Mobile Devices: All mobile devices (including helper or receiver+helper devices) implement all the building blocks illustrated Figure 9(c). Multiple threads are used to make these blocks operating simultaneously.

The first thread at mobile device n receives packets that are transmitted by the source node, and inserts these packets in $U_{n,k}$.

The second thread has two tasks. First, it transfers packets from $U_{n,k}$ to $Q_{n,k}$ according to the computation control algorithm in Equation 3.8, where $D_n^{\max} = 100$ packets and the slot duration is $20msec$. We set $\alpha_{n,k}(t) = 1$ in our experiments for the byte counting application as this application does not change the rate as explained later in this section. On the other hand, for video streaming application, the size of the video may change after it is processed by the transcoder. The value of $\alpha_{n,k}(t)$ is decided depending on the transcoding rate that we measure periodically. The second task of this thread is to actually do the computation tasks related to the application. In our experiments, the computation block either (i) counts the bytes in the packets, or (ii) processes video chunks by employing a video transcoder.

The third thread transfers packets from $Q_{n,k}$ to $Z_{n,k}$ using the energy control algorithm in Equation 3.9, where we set $E_{n,k}^{\max}$ depending on the battery level of the device. For example, if the battery level is below some threshold, $E_{n,k}^{\max}$ is limited. We evaluated different configurations in our experiments as we explain later. The slot duration is again set to $20msec$.

The final thread transfers packets from $Z_{n,k}$ to application layer if $n = k$, or transmits to node k if $n \neq k$. In the second case, *i.e.,* if $n \neq k$, the number of packets in TCP socket is checked at every time slot, where the time slot duration is $20msec$. If it is below a threshold of 500 packets, then 100 packets are removed from $Z_{n,k}$ and inserted to the TCP socket to be transmitted to node k.

When node n receives packets from node k, it directly passes the packets to the application layer for the byte counting application, because these packets are the ones that are already processed by node k. On the other hand, node n decodes video packets even if they are received from node k, because the helpers do not fully decode video packets as explained earlier. If node n is both a helper and a receiver device, it runs all the threads explained above in addition to the receiving thread from node k.

Information Exchange: Our implementation is lightweight in the sense that it limits control information exchange among mobile devices. The only control information that is transmitted in the system is $U_{n,k}$ from each mobile device to the source node. Each mobile device n collects $U_{n,k}$, $\forall k \in \mathcal{N}$, and transmits this information to the source node periodically, where we set the periods to $100msec$.

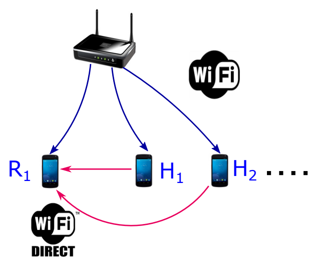
Connections: All the devices in the system including the source device, helpers, receivers, and helper+receiver devices are connected to each other using Wi-Fi Direct connections in our testbed. The source node is configured as the group owner of the Wi-Fi Direct group. We note that cooperation in this setup does not bring any benefit in terms of bandwidth utilization as all the links use the same transmission channel in a Wi-Fi Direct group. However, as we demonstrate later in this section, it brings benefit due to cooperative processing power and energy utilization, which is our main focus in this chapter. Therefore, this setup (where all the devices are connected to each other using Wi-Fi Direct links) well suits to our evaluation purposes.

Test Environment: We conducted our experiments using our testbed in a lab environment where several other Wi-Fi networks were operating in the background. We located all the devices in close proximity of each other, and we evaluated EaCC for varying levels of computational complexity, number of receivers, and number of helpers. Next, we present our evaluation results.

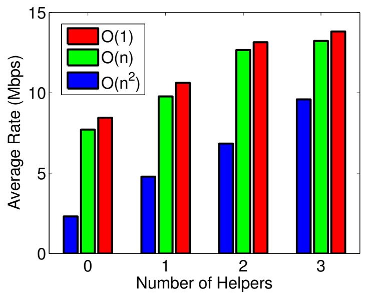
3.6.2 Results

3.6.2.1 Byte Counting Application

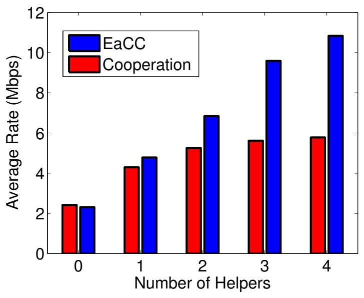
We first consider a setup as shown in Figure 10(a) which consists of a source device, one receiver (R_1), and one helper (H_1). Figure 10(b) shows the average rate versus time graph for the setup shown in Figure 10(a) when all three devices are Android-based Nexus 7 tablets. The average rate is calculated as the average over 10 trials (with different seeds). The computation under consideration in this experiment is $O(n^2)$, which counts the number of bytes in a packet n times, where n is the packet size. As can be seen, if there is no cooperation, the rate measured at R_1 is on the order of 1.5Mbps. On the other hand, EaCC increases the rate to almost 3Mbps. This means that helper device H_1 helps the receiver device R_1 process the packets in EaCC. In this setup, EaCC doubles the rate as compared to no-cooperation, which is a significant improvement.



(a) System Model



(b) Rate vs. Number of Helpers

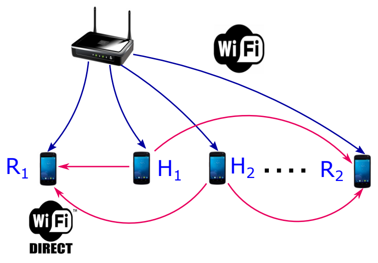


(c) Rate vs. Number of Helpers

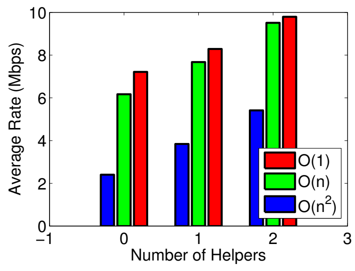
Figure 11. (a) System model consisting of a source device, one receiver, and multiple helpers. Wi-Fi is used between the source and the receiver device (R_1) and the helper devices ($H_1, \ldots$), while Wi-Fi Direct is used to connect the receiver devices with the helper devices. (b) EaCC: Average rate measured at receiver R_1 versus the number of helpers. (c) Average rate measured at receiver R_1 versus the number of helpers for EaCC and cooperation, when the complexity is $O(n^2)$.

For the same setup in Figure 10(a), we also evaluate the impact of energy control part of EaCC on the average rate performance. In particular, Figure 10(c) shows the average rate versus battery level at the receiver device R_1. In these results, we used Android-based Nexus 5 smartphones. The average rate is calculated as the average over 10 trials (with different seeds). The computation under consideration in this experiment is $O(n^2)$, which counts the number of bytes in a packet for each byte in the packet. We consider that if the battery level of a device reduces below 40% threshold, then energy credits are not generated for the processing of the received packets. This makes $Q_{n,k}$ large over time, and after some point no packets are transmitted to that device for the processing task. In Figure 10(c), when the battery level of R_1 reduces below 40%, then it stops receiving packets for processing. If there is no cooperation, then the rate towards R_1 reduces to 0. On the other hand, with EaCC, the rate is still higher than 0 thanks to having a helper. The helper device with larger energy level (for the sake of this experiment), receives packets from the source, processes them, and forwards them to R_1, which receives already processed data. After 40% threshold, both EaCC and no-cooperation improve, because R_1 starts processing packets. This result shows the importance of energy-awareness in our cooperative computation setup.

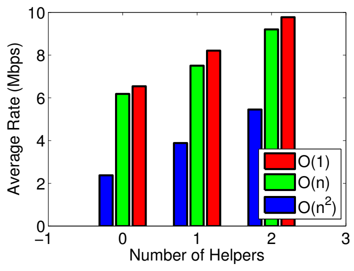
Now, we consider the impact of the number of helpers to overall rate performance. In particular, we develop a setup shown in Figure 11(a), where there is one source, one receiver, and a varying number of helpers. In this setup, the source device, receiver, and the first two helper devices are Nexus 5 smartphones, while the other helpers are Nexus 7 tablets. Figure 11(b) shows the average rate (averaged over 10 seeds) when EaCC is employed versus the number of helpers for different computational complexities such as $O(1)$, $O(n)$, and $O(n^2)$, where the processing task is counting the number of bytes in a packet. As expected, when complexity increases, the rate decreases. More interestingly, the increasing number of helpers increases the rates of all complexity levels. There are two reasons for this behavior. First, even if complexity level is low, *e.g.,* $O(1)$, processing power is still a bottleneck, and it can be solved by increasing the number of helpers. Note that after the number of helpers exceeds a value, the achievable rates saturate, which means that processing power is not a bottleneck anymore, but bandwidth is. The second reason is that receiving data over multiple interfaces increases diversity. In other words, when the channel condition over one interface (*e.g.,* between source and the mobile device) degrades, the other interface (*e.g.,* between two mobile devices) can still have a better channel condition.



(a) System Model



(b) Rate at R_1 vs. Number of Helpers



(c) Rate at R_2 vs. Number of Helpers

Figure 12. (a) System model consisting of a source device, two receivers, and multiple helpers. Wi-Fi is used between the source and the receiver devices (R_1, R_2) and the helper devices ($H_1, \ldots$), while Wi-Fi Direct is used to connect the receiver devices with the helper devices. (b) EaCC: Average rate measured at receiver R_1 versus the number of helpers. (c) EaCC: Average rate measured at receiver R_2 versus the number of helpers.

In order to understand the real impact of processing power in a cooperative system, we tested both EaCC and cooperation (without computation and energy control) in the setup shown in Figure 11(a). The results are provided in Figure 11(c) when the complexity is $O(n^2)$. As can be seen, while EaCC significantly increases the rate with increasing number of helpers, cooperation slightly increases the rate (due to diversity). The improvement of EaCC over cooperation is as high as 83%, which is significant.

Finally, we consider a scenario that there are multiple receivers interested in different files. Figure 12(a) shows the system model with one source, two receivers, and multiple helpers. In this setup, the source, two receivers, and the first helper is Android-based Nexus 5 smartphone, while the rest of the helpers are Nexus 7 tablets. Figure 12(b) and (c) show the average rate (averaged over 10 seeds) measured at R_1 and R_2 when EaCC is employed with respect to the increasing number of helpers, respectively. Similar to previous setups, $O(1)$, $O(n)$, and $O(n^2)$ correspond to different computational complexities, where the processing task is counting the number of bytes in a packet. As can be seen, the measured rate at both R_1 and R_2 increases with increasing number of helpers. This shows that our EaCC algorithm successfully accommodates multiple flows and receivers.

3.6.2.2 Video Streaming Application

In this section, we test the video streaming application over a setup shown in Figure 11(a), where there is one source, one receiver, and a varying number of helpers. The video file that we use in this experiment was recorded by using our own devices, and captures a scene of a street with moving pedestrians and vehicles. The size of the video file is 374.97 MB, and its duration is 144 sec. We divided this video into 72 chunks (with 2 sec duration each). As we noted earlier, each chunk is divided into 500 B packets.

Figure 13 shows the completion time of video at the receiver versus the number of helpers. The completion time in this scenario corresponds to the total time of receiving and decoding the entire video file at the receiver device. The helper devices in EaCC run a transcoder [57], while the helpers in Cooperation only receive and transmit packets without transcoding. As seen, EaCC significantly reduces the completion time as compared to Cooperation with increasing number of helpers, because the transcoder in each helper in EaCC changes the format of video chunks and extract them, which reduces the computational load at the receiver device. On the other hand, the completion time increases in Cooperation with increasing number of helpers, because Cooperation unnecessarily increases the number of packet exchanges over D2D links, which increases the completion time.

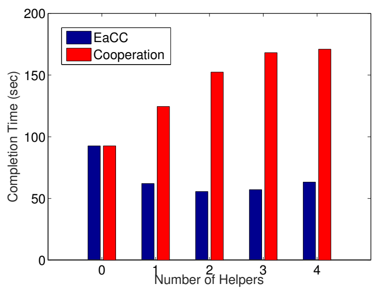


Figure 13. Completion time of a video file (*i.e.,* total time of receiving and decoding the entire video file) at the receiver device versus the number of helpers for the setup shown in Figure 11(a).

CHAPTER 4

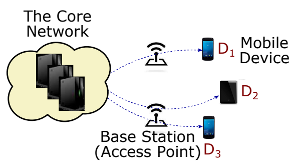
DEVICE-AWARE ROUTING AND SCHEDULING IN MULTI-HOP DEVICE-TO-DEVICE NETWORKS

The contents of this chapters are based on our work that is published in the proceedings of 2017 IEEE ITA workshop [4]. ©2019 IEEE. Reprinted, with permission, from [4].

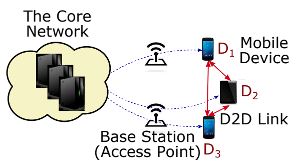
The dramatic increase in data and connectivity demand, in addition to heterogeneous device capabilities, poses a challenge for future wireless networks. One of the promising solutions is Device-to-Device (D2D) networking. D2D networking, advocating the idea of connecting two or more devices directly without traversing the core network, is promising to address the increasing data and connectivity demand. In this chapter, we consider D2D networks, where devices with heterogeneous capabilities including computing power, energy limitations, and incentives participate in D2D activities heterogeneously. We develop (i) a *device-aware routing and scheduling algorithm* (DARS) by taking into account device capabilities, and (ii) a multi-hop D2D testbed using Android-based smartphones and tablets by exploiting Wi-Fi Direct and legacy Wi-Fi connections. We show that DARS significantly improves throughput in our testbed as compared to state-of-the-art.

4.1 Background

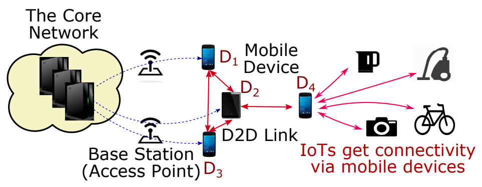
The default operation in current wireless networks is to connect each device to the Internet via its cellular or Wi-Fi interface, Figure 14(a). The D2D connectivity breaks this assumption: it advocates that two or more devices can be connected directly, *i.e.,* without traversing through an auxiliary device



(a) The default operation



(b) D2D connectivity


The Core Network
Mobile Device
D_1
D_2
D_4
D2D Link
Base Station (Access Point)
D_3
IoTs get connectivity via mobile devices

(c) IoT get connectivity via D2D

Figure 14. (a) The default operation for the Internet connection. (b) D2D connectivity: two or more mobile devices can be connected directly, *i.e.,* without traversing through the core network, if they are in close proximity by exploiting D2D connections such as Wi-Fi Direct or Bluetooth. (c) A number of devices (*e.g.,* IoT) seek connectivity via other devices (*e.g.,* mobile devices) using D2D connections.

such as a base station if they are in close proximity [47], Figure 14(b). D2D networks can be formed by exploiting D2D connections such as Wi-Fi Direct [36] or Bluetooth. D2D networks are promising to address the ever-increasing number of devices as well as the demand for data and connectivity.

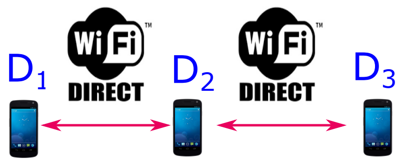
Although D2D networking looks very promising to address the increasing data demand and the number of devices, and is expected to play a crucial role for the next generation networks, the following question is still open: How to design device-aware networking algorithms and protocols?

In this chapter, we consider a scenario where a number of devices, *e.g.,* mobile devices or Internet of Things (IoT), seek Internet connectivity via other devices using D2D connections as shown in

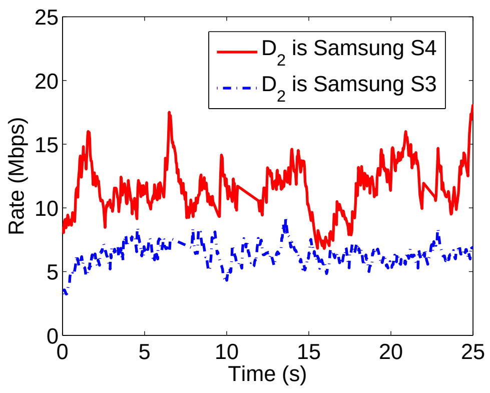
Figure 14(c). In this context, it is possible to connect a device to the Internet via multiple hops, so it is crucial to determine which devices should forward packets, and how to make scheduling decisions. *E.g.,* in Figure 14(c), there are two paths; $D_1 - D_2 - D_4$ and $D_3 - D_2 - D_4$, and it is crucial to determine the path that provides better connectivity. However, these decisions should be made by taking into account device capabilities.

Pilot Study: In order to show the importance of taking account of device capability in D2D routing and scheduling, we developed a prototype for this pilot study as shown in Figure 15(a), where three devices D_1, D_2, and D_3 are connected as a line topology by exploiting the Wi-Fi Direct connections. In our pilot study, we used two Nexus 7 tablets, one Samsung S4 smartphone, and one Samsung S3 smartphone. Nexus 7 tablets are used as D_1 and D_3, and either Samsung S4 or Samsung S3 smartphone is used as D_2. In this setup, the capabilities of the intermediate device D_2, have direct impact on the transmission rate from D_1 to D_3. Our experimental results in Figure 15(b) show that when D_2 is Samsung S4 (a more powerful device as compared to Samsung S3), the transmission rate between $D_1 - D_3$ is higher as compared to the case that D_2 is Samsung S3. As seen, the intermediate device with less computing power (Samsung S3) limits the transmission rate. $\square$

Our pilot study shows that it is crucial to take into account device capabilities while designing D2D networking algorithms. Although our pilot study only focuses on the computing power, other parameters such as limited energy, human participation (or incentives), and bandwidth should be taken into account. For example, it could be possible that the owner of D_2 may limit its participation in a D2D activity, which would eventually reduce the rate between $D_1 - D_3$.



(a) 3-Node Line Topology



(b) Rate vs Time

Figure 15. Pilot study. (a) The line topology, where D_1, D_2, and D_3 are connected via Wi-Fi Direct links. In our experiments, we used two Nexus 7 tablets, one Samsung S4 smartphone, and one Samsung S3 smartphone. All devices use Android as their operating systems, and Nexus 7 tablets are used as D_1 and D_3. (b) The rate between $D_1 - D_3$ versus time when (i) D_2 is Samsung S4, and (ii) D_2 is Samsung S3.

In this chapter, we consider D2D networks, where devices with heterogeneous capabilities including computing power, energy limitations, and incentives participate in D2D activities heterogeneously. We first develop network utility maximization problem, and provide its solution. Then, based on the structure of the solution, we develop a *device-aware routing and scheduling algorithm* (DARS) that takes into account device capabilities. Furthermore, we design a multi-hop D2D testbed using Android-based smartphones and tablets by exploiting Wi-Fi Direct and legacy Wi-Fi connections. We evaluate DARS on this testbed.

The structure of the rest of the chapter is as follows. Section 4.2 presents related work. Section 4.3 gives an overview of the system model and the problem formulation. Section 4.4 presents DARS algorithm. Section 4.5 presents the implementation and evaluation of DARS.

4.2 Related Work

The idea of exploiting D2D connectivity is very promising to improve throughput and reduce delay, so it has found several applications in the literature. For example, opportunistic D2D connections is often used for the purpose of (i) offloading cellular networks [27], [28], [29], (ii) content dissemination among mobile devices [30], [31], and (iii) cooperative video streaming over mobile devices [32], [33]. As compared to this line of work, we focus on developing *device-aware routing and scheduling* algorithm over multi-hop D2D networks by taking into account device capabilities such as computing power, energy, and incentives.

Our approach in this work involves using network utility maximization to characterize the system as it is promising to understand how different layers and/or algorithms, such as flow control, routing, and scheduling should be designed and optimized [22], [34]. However, we formulate the NUM framework considering device capabilities to develop device-aware framework. Second, we develop DARS. In that sense, our approach is similar to the line of work emerged after the pioneering work in [19], [20], [21]. However, our focus is on incorporating device capabilities in the framework. Furthermore, we develop a testbed of our algorithm using real devices, which was not the focus of the previous work.

Multi-hop data transmission testbed using Wi-Fi Direct over mobile devices has been considered in [13]. In this work, intermediate devices receive a whole file first, and then transmits it to other devices. As compared to this work, our implementation makes simultaneous transmission and reception possible, so there is no need to wait to receive a complete file before starting to transmit it to a next hop. More similar work to ours is [14], where both legacy and Wi-Fi interfaces are exploited at group owners (not at clients as in our approach). As compared to this work, our approach (i) uses unicast transmissions (rather than broadcast like [14]), so our testbed can operate at higher rates, (ii) supports bidirectional IP communication supporting both TCP and UDP, (iii) requires minimal changes to existing Wi-Fi Direct and legacy Wi-Fi operations. Furthermore, we implement DARS over this testbed by taking into account device capabilities.

4.3 System Overview and Problem Formulation

4.3.1 System Overview

We consider a multi-hop D2D network with mobile devices, where devices are connected to each other via D2D connections. In this setup, packets from a source device traverse potentially multiple devices before arriving to the destination device. Devices in this setup are capable of performing various tasks including routing, scheduling, and rate control. However, depending on device capabilities and configurations, the transmission rates vary. Our system model, and algorithm design capture this heterogeneity. In this section, we provide an overview of this setup and highlight some of its key characteristics.[1]

[1]We note that this section introduces our setup and assumptions needed for the theoretical development of our device-aware framework. We will revise some of these assumptions in Section 4.5 when we discuss implementation details of our algorithm in a testbed.

Setup: We consider a multi-hop D2D network, which consists of N devices and L edges, where $\mathcal{N}$ and $\mathcal{L}$ are the set of nodes and edges, respectively. We consider in our formulation and analysis that time is slotted, and t refers to the beginning of slot t.

Sources and Flows: Let $\mathcal{S}$ be the set of unicast flows between source and destination device pairs. Each flow $s \in \mathcal{S}$ generates $A_s(t)$ packets at the application layer at time t.

The packet arrivals are i.i.d. over the slots and the first and second moments of the arrival distribution is finite; *i.e.,* $\lambda_s = E[A_s(t)]$, and $E[A_s(t)^2]$. Packets are stored at the source device in an *initial buffer* in the application layer. Each flow s is associated with rate x_s and a utility function $g_s(x_s)$, which we assume to be a strictly concave function of x_s for our analysis purposes. Packets from the initial buffer are passed to the *main buffer* with rate $x_s(t)$ at time t and depending on the utility function $g_s(x_s(t))$.

At time t, $f_{i,j}^s(t)$ packets from flow s are passed from node i to node j. The number of packets, *i.e.,* $f_{i,j}^s(t)$, are determined by device-aware framework by taking into account device capabilities.

4.3.2 Problem Formulation

Now, we formulate our device-aware framework. Our objective is to determine $\boldsymbol{x}, \boldsymbol{f}$, where $\boldsymbol{x} = \{x_s\}_{s\in\mathcal{S}}$, and $\boldsymbol{f} = \{f_{i,j}^s\}_{s\in\mathcal{S},(i,j)\in\mathcal{L}}$, by maximizing the total utility function; $\sum_{s\in\mathcal{S}} g_s(x_s)$ subject to the constraints[1]

$$\begin{aligned} &\sum_{j\in\mathcal{N}} f_{i,j}^s - \sum_{j\in\mathcal{N}} f_{j,i}^s = x_s 1_{[i=o(s)]},\ \forall s\in\mathcal{S}, i\in\mathcal{N} \\ &\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} f_{i,j}^s \le \min\{R_P^j, R_E^j, R_W^j\}, \forall j\in\mathcal{N} \\ &\boldsymbol{f}\in\boldsymbol{\Gamma} \end{aligned} \tag{4.1}$$

The first constraint in Equation 4.1 is the flow conservation at device i and for flow s, where $\sum_{j\in\mathcal{N}} f_{j,i}^s + x_s 1_{[i=o(s)]}$ is the arrival rate of flow s to node i, while $\sum_{j\in\mathcal{N}} f_{i,j}^s$ is the departure rate.

The second constraint captures device capabilities. The arrival rate to device j, *i.e.,* $\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} f_{i,j}^s$ should be supported by the device, where R_P^j is the maximum rate that device j can support (receive and transmit) with its computing power, while R_E^j and R_W^j are the rates that device j can support with its energy and incentives, respectively.

The last constraint in Equation 4.1 is the feasibility constraint, where $\boldsymbol{\Gamma}$ is the set of all feasible rates that can be in the network. Thus, $\boldsymbol{f}$ should be an element of $\boldsymbol{\Gamma}$.

[1]Note that, in this section, we optimize the average values of the parameters defined in Section 4.3.1. Thus, by abuse of notation, we use a variable, *e.g.,* ϕ as the average value of $\phi(t)$ if both ϕ and $\phi(t)$ refers to the same parameter.

Although the solution of Equation 4.1 provides a device-aware routing and scheduling, the solution is not practical, because it requires an active involvement of all devices in D2D network even if a device does not prefer any involvement. For example, even if a node j has very small $\min\{R_P^j, R_E^j, R_W^j\}$, it needs to periodically update the other devices in the network about its status (whether it can relay packets or not), which is not practical and introduces overhead. Thus, we modify the problem in Equation 4.1 so that the solution can be more practical.

Our first step is to explicitly involve link rates in the formulation. Assume that $R_{i,j}$ is the transmission rate between nodes i, j and $\tau_{i,j}$ is the percentage of time that the link $i - j$ is used. Then, we can express $f_{i,j}^s = R_{i,j}\tau_{i,j}^s$, $\forall i \in \mathcal{N}, j \in \mathcal{N}, s \in \mathcal{S}$. This translates the constraints in Equation 4.1 to $\sum_{j\in\mathcal{N}} R_{i,j}\tau_{i,j}^s - \sum_{j\in\mathcal{N}} R_{j,i}\tau_{j,i}^s = x_s 1_{[i=o(s)]}$, $\forall s \in \mathcal{S}, i \in \mathcal{N}$, $\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} R_{i,j}\tau_{i,j}^s \leq \min\{R_P^j, R_E^j, R_W^j\}, \forall j \in \mathcal{N}$, and $\boldsymbol{\tau} \in \boldsymbol{\Gamma}_\tau$, where $\boldsymbol{\tau} = \{\tau_{i,j}^s\}_{s\in\mathcal{S},(i,j)\in\mathcal{L}}$, and $\boldsymbol{\Gamma}_\tau$ is the set of all feasible link schedules, so $\boldsymbol{\tau} \in \boldsymbol{\Gamma}_\tau$ should hold.

The next step is to create a new variable $\gamma_{i,j}^s$ as $\gamma_{i,j}^s = \tau_{i,j}^s \frac{R_{i,j}}{\min\{R_E^j, R_W^j, R_P^j\}}$ assuming that $\min\{R_P^j$, R_E^j, $R_W^j\}$ is positive, at least slightly larger than 0. Then, the constraints become

$$\sum_{j\in\mathcal{N}} \gamma_{i,j}^s \min\{R_P^j, R_E^j, R_W^j\} - \sum_{j\in\mathcal{N}} \gamma_{j,i}^s \min\{R_E^i, R_W^i, R_P^i\} = x_s 1_{[i=o(s)]}, \ \forall s \in \mathcal{S}, i \in \mathcal{N}$$

$$\sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} \gamma_{i,j}^s \leq 1, \forall j \in \mathcal{N} \text{ and } \boldsymbol{\gamma} \in \boldsymbol{\Gamma}_\gamma, \tag{4.2}$$

where $\boldsymbol{\gamma} = \{\gamma_{i,j}^s\}_{s\in\mathcal{S},(i,j)\in\mathcal{L}}$, and $\boldsymbol{\Gamma}_\gamma$ is the set of feasible $\boldsymbol{\gamma}$'s. Next, we provide a solution to the problem of maximizing the total utility; $\sum_{s\in\mathcal{S}} g_s(x_s)$ subject to the constraints in Equation 4.2.

4.3.3 Solution

Lagrangian relaxation of the first constraint of Equation 4.2 gives the following Lagrange function:

$$L = \sum_{s\in\mathcal{S}} g_s(x_s) - \sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} u_i^s \Big(\sum_{j\in\mathcal{N}} \gamma_{i,j}^s \min\{R_E^j, R_W^j, R_P^j\} - \sum_{j\in\mathcal{N}} \gamma_{j,i}^s \min\{R_E^i, R_W^i, R_P^i\} - x_s 1_{[i=o(s)]}\Big) \tag{4.3}$$

where u_i^s is the Lagrange multiplier. The Lagrange function is expressed as

$$L = \sum_{s\in\mathcal{S}} [g_s(x_s) - u_{o(s)}^s x_s] + \sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} \gamma_{i,j}^s \min\{R_E^j, R_W^j, R_P^j\}(u_i^s - u_j^s). \tag{4.4}$$

This Lagrange function is decomposed into two sub-problems: (i) rate control, and (ii) routing and scheduling. If we solve the Lagrangian function with respect to x_s, we have an optimization problem: $\max_{\boldsymbol{x}} \sum_{s\in\mathcal{S}}[g_s(x_s) - u_{o(s)}^s x_s]$, which is the rate control part. On the other hand, the routing and scheduling part solves the following optimization problem

$$\begin{aligned} \max_{\boldsymbol{\gamma}} \ & \sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} \gamma_{i,j}^s \min\{R_E^j, R_W^j, R_P^j\}(u_i^s - u_j^s) \\ \text{s.t.} \ & \sum_{s\in\mathcal{S}}\sum_{i\in\mathcal{N}} \gamma_{i,j}^s \le 1, \forall j \in \mathcal{N} \text{ and } \boldsymbol{\gamma} \in \boldsymbol{\Gamma}_\gamma. \end{aligned} \tag{4.5}$$

Note that the solution of Equation 4.5 is easier as compared to the solution of Equation 4.1, because it reduces to selecting the link $i - j$, which maximizes $\min\{R_E^j, R_W^j, R_P^j\}(u_i^s - u_j^s)$ among all feasible schedules of links. Based on this idea, we will develop our device-aware stochastic routing and scheduling algorithm in the next section.

4.4 DARS: Device-Aware Routing and Scheduling

Now, we design DARS, which has (i) rate control, (ii) routing and scheduling, and (iii) queue evolution parts, based on the solutions developed in Section 4.3.3.

Device-Aware Routing and Scheduling Algorithm (DARS):

- *Rate Control:* At slot t, the rate controller at node $o(s)$ determines the number of packets that should be passed from the initial buffer to the main buffer according to

$$\begin{aligned} &\max_{\boldsymbol{x}} \sum_{s \in \mathcal{S}} [M g_s(x_s(t)) - U_{o(s)}^s x_s(t)] \\ &\text{s.t. } x_s(t) \leq R_{\max}, \end{aligned} \tag{4.6}$$

  where $U_{o(s)}^s$ is the queue that stores packets from flow s at node $o(s)$, $R_{\max}$ is a positive constant larger than the transmission rate from device $o(s)$, and M is a large positive constant. Note that flow control algorithm in Equation 4.6 is designed based on the structure of the rate control solution in Section 4.3.3.

- *Routing and Scheduling:* At slot t, device j determines the number of packets that it can receive, process, and forward according to

$$\begin{aligned} &\max_{\gamma} \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{N}} \gamma_{i,j}^{s}(t) \min\{R_P^j, R_E^j, R_W^j\}[U_i^s(t) - U_j^s(t)] \\ &\text{s.t. } \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{N}} \gamma_{i,j}^{s}(t) \leq 1, \text{ and } \boldsymbol{\gamma}(t) \in \boldsymbol{\Gamma}_{\gamma}(t) \end{aligned} \tag{4.7}$$

  where $U_i^s(t)$ and $U_j^s(t)$ are queue sizes at nodes i and j, respectively. After the value of $\gamma_{i,j}^s(t)$ is determined, if $\gamma_{i,j}^s(t) = 1$, $f_{i,j}^s(t)$ is set to $f_{i,j}^s(t) = F_{\max}$, where $F_{\max}$ is a positive constant larger than the transmission rate from device i to device j, as well as larger than $\min\{R_P^j, R_E^j, R_W^j\}$. Otherwise, *i.e.,* if $\gamma_{i,j}^s(t) = 0$, then $f_{i,j}^s(t) = 0$. The solution in Equation 4.7 has two strengths: (i) it takes into account device capabilities, *i.e.,* $\min\{R_P^j, R_E^j, R_W^j\}$, and (ii) each device j makes its own decision on how much data it can handle (route & schedule), which is fundamentally different than the classical backpressure [19], [21], [15], where each device j should do its best to route and schedule any amount of data it receives. Also, note that the solution of Equation 4.7 is both a routing decision as it determines the next hops, and a scheduling decision as it determines which links to activate (the ones $\gamma_{i,j}^s(t) = 1$ are activated).

- *Queue Evolution:* The evolution of the queue $U_i^s(t)$ at time t is as follows;

$$U_i^s(t+1) \leq \max[U_i^s(t) - \sum_{j \in \mathcal{N}} f_{i,j}^s(t), 0] + \sum_{j \in \mathcal{N}} f_{j,i}^s(t) + x_s(t) 1_{[i=o(s)]} \tag{4.8}$$

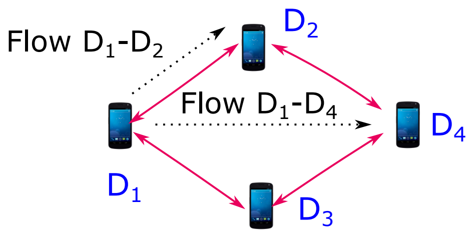
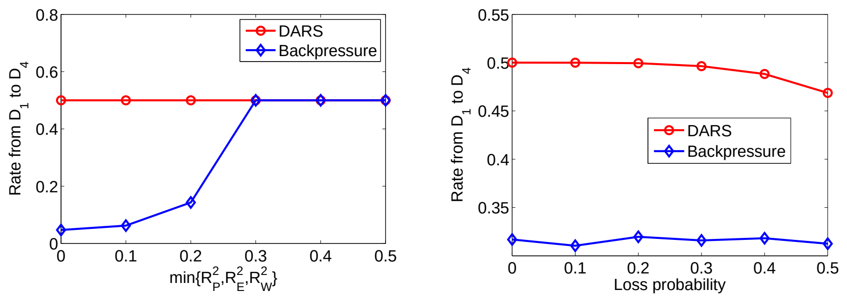


Figure 16. Diamond topology for simulations.

where $o(s)$ is the source node of flow s and $1_{[i=o(s)]}$ is an indicator function, which is 1 if $i = o(s)$, and 0, otherwise. Note that Equation 4.8 is an inequality, because the actual amount of data arriving to the queue may be smaller than $\sum_{j \in \mathcal{N}} f_{j,i}^{s}(t) + x_s(t)1_{[i=o(s)]}$.

In Section 4.5, we will provide the implementation details and evaluation of DARS in a real testbed. Yet, before delving into that, we present simulation results of DARS as compared to backpressure [21] in an idealized setup.

We first consider a diamond topology shown in Figure 16, where there is a flow from D_1 to D_4. For this scenario, Figure 17(a) shows the flow rate versus min $\{R_P^2,\, R_E^2,\, R_W^2\}$, when min $\{R_P^i,\, R_E^i,\, R_W^i\} = 1$, for $i = 1, 3, 4$, and links are not lossy. Figure 17(b) shows the flow rate versus loss probability (all links are lossy), when min $\{R_P^2,\, R_E^2,\, R_W^2\} = 0.1$ and min $\{R_P^i,\, R_E^i,\, R_W^i\} = 1$, for $i = 1, 3, 4$. In both simulations, $M = 200$, $R_{\max} = 1$, $F_{\max} = 1$. The results show that DARS significantly improves over backpressure (implemented according to [21]), because DARS takes into account device capabilities while backpressure does not. *I.e.,* even if packets and links are scheduled by backpressure, they may not be realized due to the device (D_2 in this simulation) bottleneck.



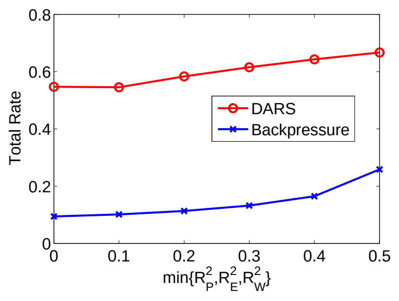
(a) Rate vs. min $\{R_P^2, R_E^2, R_W^2\}$ (b) Rate vs. loss probability

Figure 17. Diamond topology with one flow from D_1 to D_4. (a) Rate of flow $D_1 - D_4$ versus min $\{R_P^2, R_E^2, R_W^2\}$, where links are not lossy. (b) Rate of flow $D_1 - D_4$ versus loss probability, where all links are lossy. In both simulations, $M = 200$, $R_{\max} = 1$, $F_{\max} = 1$.

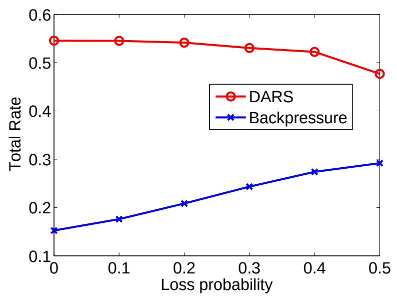
Next, we consider the diamond topology shown in Figure 16 for two flows; one from D_1 to D_2, and another from D_1 to D_4. Considering the same parameters of the one-flow scenario above, we have results as shown in Figure 18. As seen, DARS dramatically improves over backpressure as in the one-flow case thanks to taking into account device capabilities.

4.5 Implementation Details and Evaluation

In this section, we present the implementation details of our testbed. First, we start with how we create a multi-hop topology with Android-based devices using Wi-Fi Direct. Then, we present our DARS implementation on our testbed.



(a) Rate vs. min $\{R_P^2, R_E^2, R_W^2\}$



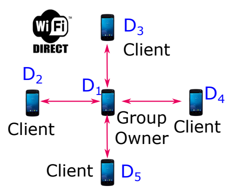
(b) Rate vs. loss probability

Figure 18. Diamond topology with two flows; one from D_1 to D_4, and another from D_1 to D_2. (a) Total rate (of both flows) versus $\min\{R_P^2, R_E^2, R_W^2\}$. (b) Total rate (of both flows) versus loss probability, where all links are lossy. In both simulations, $M = 200$, $R_{\max} = 1$, $F_{\max} = 1$.

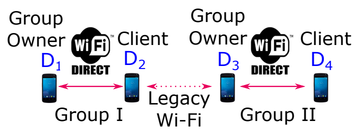
4.5.1 Creating Multi-Hop Topology with Android Devices

Wi-Fi Direct: Our approach to create multi-hop topology using Android-based devices is to employ Wi-Fi Direct connections [36]. However, existing Wi-Fi Direct implementation in Android-based devices only supports a star topology as shown in Figure 19(a), but not any other multi-hop topology.

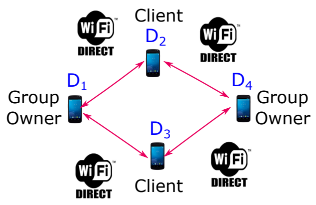
Our Approach: We use the star topology of Wi-Fi Direct shown in Figure 19(a) as our basic constructing unit for creating multi-hop topologies. In particular, multiple groups are constructed using the star topology (*i.e.,* each group is a star topology), and these groups are connected to each other. Connecting multiple groups (star topologies) is quite challenging, because the star topology of Wi-Fi Direct is constructed in a way that one device (center of the topology) is a group owner (GO), and the other devices are clients. In this setup, a device cannot act as both a group owner and a client simultaneously, which makes connecting multiple groups to each other prohibitively difficult.



(a) Star topology with Wi-Fi Direct



(b) Line topology in our testbed



(c) Diamond topology in our testbed

Figure 19. (a) Star topology. A device that receives a connection request first becomes a group owner and center of the star topology, *i.e.,* D_1, and all other devices, *i.e.,* D_2 to D_5, connect to the group owner. In this setup, there can be only one group owner, and a device cannot act as both group owner and a client simultaneously. (b) Line topology with two groups. Group I and Group II are created using Wi-Fi Direct. D_1 and D_3 are group owners, and D_2 and D_4 are clients of Group I and Group II, respectively. D_2 is connected to D_3 using legacy Wi-Fi interface. This creates a 4-node line topology. (c) Diamond topology. D_1 and D_4 are group owners, and D_2 and D_3 are clients in Wi-Fi Direct groups. D_2 and D_3 are connected to D_1 via Wi-Fi Direct interface, and connect to D_4 using their legacy Wi-Fi interfaces.

In our testbed, we use legacy Wi-Fi interface to connect multiple groups. Let us consider Figure 19(b), where there are two groups; Group I: D_1, D_2 and Group II: D_3, D_4. In this example, D_1 and D_3 are group owners, and D_2 and D_4 are clients of Group I and Group II, respectively. Let us assume that our goal is to connect D_2 and D_3. In existing Wi-Fi Direct, D_2 cannot connect to D_3 as D_2 can only connect to its group owner (which is D_1) via Wi-Fi Direct interface. On the other hand, D_3 cannot connect to D_2 as a client, because D_3 is already a group owner of Group II, so it cannot be a client of

D_2. Therefore, our approach is to use legacy Wi-Fi interface of D_2 to connect to D_3. This connection is possible as D_2 will see D_3 as an access point of the legacy Wi-Fi connection. Thus, D_2 can connect to D_3, which provides a line topology consisting of 4 nodes, which was not possible by using only existing Wi-Fi Direct setup.

Similarly, we can create other multi-hop topologies. For example, we can create a diamond topology as shown in Figure 19(c), where D_1 and D_4 are group owners, and D_2 and D_3 are clients in Wi-Fi Direct groups. D_2 and D_3 are connected to D_1 via Wi-Fi Direct interface, and connect to D_4 using their legacy Wi-Fi interfaces.

Our approach of using legacy Wi-Fi interfaces simultaneously with Wi-Fi Direct interfaces is challenging, because both legacy Wi-Fi interface and Wi-Fi Direct interfaces are actually using the same means of communication interface in the MAC layer, which is 802.11. Thus, if we naively open both legacy Wi-Fi and Wi-Fi Direct interfaces, only one of them will operate due to IP addressing conflicts. For example, D_2 would transmit data to D_1 even if it means to transmit to D_3 in Figure 19(b). We provide a solution to this problem in a simple way (*i.e.,* without rooting mobile devices). In particular, we use a class called ConnectivityManager in Android API, which provides instances (objects of the class) of all active network interfaces on each device. Thus, we access the instance of legacy Wi-Fi interface, and bind it with transmission sockets TCP or UDP. This approach eliminates addressing issues and conflicts between legacy Wi-Fi and Wi-Fi Direct interfaces.

4.5.2 DARS Implementation

In this section, we present how DARS is implemented over our multi-hop testbed described in Section 4.5.1.

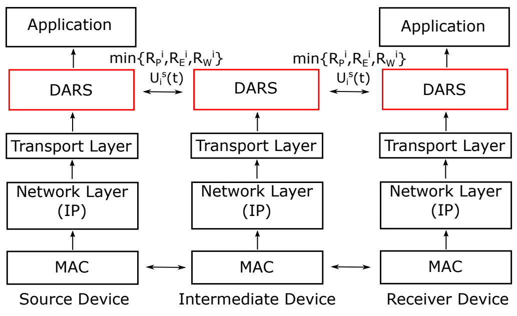


Figure 20. DARS operations at end-points and intermediate nodes.

Devices: We implemented a testbed of the different topologies including line topology, diamond topology using real mobile devices, specifically Android 5.1.1 based Nexus 5 smartphones and Nexus 7 tablets.

Integration to the Protocol Stack: We implemented DARS as a slim layer between transport and application layers as demonstrated in Figure 20. In other words, we implemented DARS on top of TCP. This kind of implementation has benefits as (i) mobile devices do not require rooting, and (ii) our DARS codes could be easily transferred to mobile devices using other operating systems such as iOS.

Virtual Slots: As mentioned in Section 8.2, DARS uses slots to make transmission decisions. By following the theory, in our implementation, we divided the time into virtual slots. Each decision is made at the start of the slot. We set slot durations to 50msec.

Multiple Threads: Three sets of threads operate at each device simultaneously to perform the tasks of rate control, routing and scheduling, and actual data transmission.

The first set of threads are implemented for the rate control, so we call them rate control threads. In particular, the rate control thread at the source device $o(s)$ reads data bytes from a file, packetizes them and inserts them into the transmission queue $U^s_{o(s)}$. The rate of reading packets from the file is determined according to the rate control algorithm in Equation 4.6. Note that if a device is the source of two flows, a rate control thread is created for each flow.

The second set of threads determine how many packets should be transmitted from U^s_i to other devices. Thus, these threads are called routing and scheduling threads. This part implements Equation 4.7. For example, in the diamond topology in Figure 19(d), at each slot, D_1 determines whether it should transmit packets to D_2, or D_3, or none of them.

The final set of threads make actual packet transmissions possible, so we call them transmission threads. A transmission thread is constructed for each neighboring node. For example, in the diamond topology Figure 19(c), D_1 constructs two transmission threads for D_2 and D_3. Packets, whose number is determined by the routing and scheduling thread, are received by this thread and inserted into queues that we call socket queues. The transmission threads will dequeue packets from the socket queues and pass them to TCP sockets. Another task of transmission threads is to receive queue size information from neighboring nodes.

Information Exchange: Our implementation is lightweight in the sense that it limits control information exchange among mobile devices. The control information that is transmitted by node i is $U^s_i(t)$ and $\min\{R^i_P, R^i_E, R^i_W\}$, and this information is transmitted to only i's neighbors. These control packets are transmitted periodically at every 50msec.

Calculating $\min\{R^i_P, R^i_E, R^i_W\}$*:* Each node i, based on its computing power, energy level, and incentives (willingness), calculates its rate. For example, if node i has limited energy, then it limits R^i_E

to 1Mbps even if it can support up to 20Mbps. Thus, in our implementation, every device calculates its own rate, and exchange this information with its neighbors.

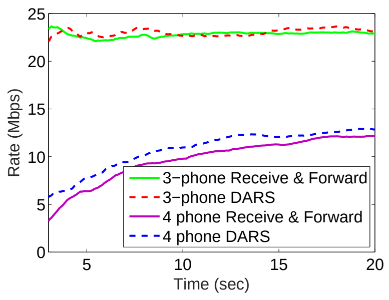
Test Environment: We conducted our experiments in a lab environment where several other Wi-Fi networks were operating in the background. We located all the devices at varying distances, and we have evaluated device-centric routing and scheduling. Next, we present our evaluation results.

4.5.3 Evaluation Results

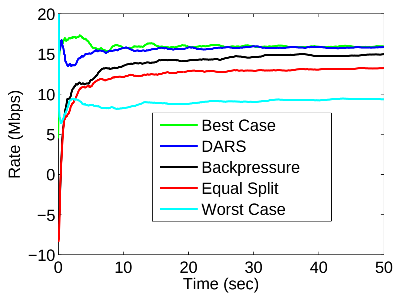
Now, we focus on evaluating the performance of DARS. Figure 21(a) shows data rate versus time graph for three-node and four-node line topologies shown in Figure 19(b), where a a flow is transmitted from D_1 to D_4. The receive & forward algorithm is the baseline in this scenario, where the intermediate nodes just receive packets and forward. As seen, the performance of DARS is close to receive & forward. Note that since there is no routing and scheduling diversity in the line topology, the receive & forward provides the best performance. The results prove that DARS does not introduce too much overhead into the system.

Figure 21(b) shows the transmission rate for the diamond topology shown in Figure 19(c), where a flow is transmitted from D_1 to D_4. In this setup, D_1, D_2, and D_4 are Nexus 5 smartphones, and D_3 is a Nexus 7 tablet. The computing power of Nexus 5 smartphones is better than Nexus 7 tablet. In particular, Nexus 5 supports two times faster rate as compared to Nexus 7. Thus, DARS should prefer $D_1 - D_2 - D_4$ route instead of $D_1 - D_3 - D_4$ when D_1 is the source and D_4 is the receiver. The simulation results support this as further explained next.

The best case scenario in Figure 21(b) is the case that $D_1 - D_2 - D_4$ route is selected a priori. As seen, DARS performs very close to the best case scenario, which shows the efficiency of our algorithm.



(a) Line topology results



(b) Diamond topology results

Figure 21. (a) Line topology (as shown in Figure 19(c)) results with three and four devices. (b) Diamond topology Figure 19(d) results.

Backpressure is the implementation of the scheme proposed in [21]. As seen, DARS improves over backpressure (up to $\%15$), because it takes into account device specific properties, while backpressure does not. Equal Split is another baseline, which allows transmitting data over both paths in the diamond topology. As long as TCP supports transmissions, packets are simultaneously transmitted over both links. As seen, DARS significantly improves as compared to this baseline. Finally, the worst case scenario is the case that $D_1 - D_3 - D_4$ route is selected a priori, which is included in the results for completeness.

CHAPTER 5

PREDICTIVE EDGE COMPUTING WITH HARD DEADLINES

The contents of this chapters are based on our work that is published in the proceedings of 2018 IEEE LANMAN conference [5] and a journal under submission. ©2019 IEEE. Reprinted, with permission, from [5].

Edge computing is a promising approach for localized data processing for many edge applications and systems including Internet of Things (IoT), where computationally intensive tasks in IoT devices could be divided into sub-tasks and offloaded to other IoT devices, mobile devices, and / or servers at the edge. However, existing solutions on edge computing do not address the full range of challenges, specifically heterogeneity; edge devices are highly heterogeneous and dynamic in nature. In this chapter, we develop a predictive edge computing framework with hard deadlines. Our algorithm; `PrComp` (i) predicts the uncertain dynamics of resources of devices at the edge including energy, computing power, and mobility, and (ii) makes sub-task offloading decisions by taking into account the predicted available resources, as well as the hard deadline constraints of tasks. We evaluate `PrComp` on a testbed consisting of real Android-based smartphones, and show that it significantly improves energy consumption of edge devices and task completion delay as compared to baselines.

5.1 Background

The number of edge devices, *e.g.,* Internet of Things (IoT) keeps increasing and is estimated to reach billions in the next five years [59]. As a result, the data collected by edge devices will grow at exponential rates. In many applications, unlocking the full power of edge devices requires analyzing and processing this data through computationally intensive algorithms with stringent latency constraints.

In many scenarios, these algorithms cannot be run locally on the computationally-limited edge devices (*e.g.,* IoT devices) and are rather outsourced to the cloud [60]. Yet, offloading tasks to remote could be costly, inefficient in terms of delay, or may not be feasible (*e.g.,* when there is no cellular or Wi-Fi infrastructure support). An alternative is edge computing, where tasks in an edge device could be offloaded to other edge devices including IoT devices, mobile devices, and / or servers in close proximity.

However, existing solutions on edge computing do not address the full range of challenges, specifically heterogeneity; edge devices are highly heterogeneous and dynamic in nature. For example, if master device M offloads some tasks to helper device W_4 in Figure 22, but if device W_4 is running another computationally intensive application (either originated from itself or offloaded from another master device - M' in this example), delay increases. Similarly, if device M offloads some tasks to device W_4, but before completing processing these tasks, device W_4 moves away, D2D connection between M and W_4 is broken. In this case, device M should offload its tasks again to other devices in close proximity (*e.g.,* device W_3). This re-offloading process increases delay and energy consumption. Thus, we should develop an edge computing mechanism, which is aware of the heterogeneity and time varying nature of resources as well as mobility of devices.

In this chapter, we develop a predictive computation offloading mechanism (`PrComp`) by taking into account heterogeneous and time varying resources as well as mobility. In particular, we consider a master / helper setup as seen in Figure 22, where a master device predicts the resources of helpers

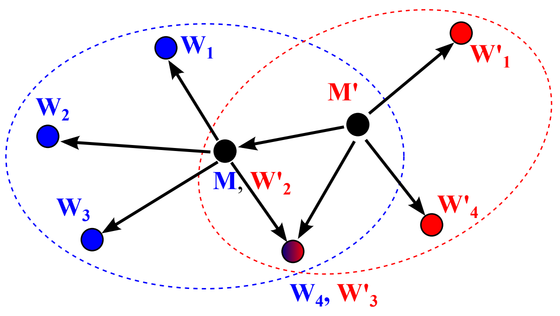


Figure 22. Example computation at the edge. Two master devices M and M' offload their tasks to helpers $W_1, \ldots, W_4$ and $W'_1, \ldots, W'_4$, respectively. As seen, a device could be (i) both a master and a helper at the same time, and (ii) helpers of multiple masters simultaneously.

using periodic probes. Then, the master device makes computation offloading decisions to minimize the energy consumption of master and helper devices while satisfying deadline constraints of tasks.

We evaluate `PrComp` on a testbed consisting of real smartphones, and show that it significantly improves energy consumption and task completion time as compared to baselines.

The structure of the rest of the chapter is as follows. Section 5.2 presents related work. Section 5.3 gives an overview of the system model. Section 5.4 presents our delay, energy, and mobility prediction module. Section 5.5 presents our `PrComp` algorithms. Section 5.6 evaluates the performance of our algorithms in a real testbed.

5.2 Related Work

Mobile cloud computing is a rapidly growing field with the goal of providing extensive computational resources to mobile devices as well as higher quality of experience [61], [62], [63], [64], [65]. The initial approach to mobile cloud computing has been to offload resource intensive tasks to remote clouds by exploiting Internet connectivity of mobile devices. This approach has received a lot of attention which led to extensive literature in the area [66], [67], [68], [69], [70]. The feasibility of computation offloading to remote cloud by mobile devices [71] as well as energy efficient computation offloading [72, 73] have been considered in the previous work. As compared to this line of work, our focus is on edge computing rather than remote clouds.

There is an increasing interest in edge computing by exploiting connectivity among mobile devices [50]. This approach suggests that if devices in close proximity are capable of processing tasks cooperatively, then local area computation groups could be formed and exploited for computation. This approach, sparking a lot of interest, led to some very interesting work in the area [54], [55], [56]. The performance of computing at the edge including the computation group size of IoT devices, lifetime, and reachable time is characterized in [74] by taking into account the mobility of devices. As compared to this line of work, we focus on offloading tasks from one edge device to others by predicting delay, energy, and mobility.

Edge computing is investigated in [75] to deal with mobility by exploiting both cellular and Wi-Fi connections. As compared to this work, our work makes offloading decisions by predicting the mobility patterns of helper devices. Task offloading to minimize energy consumption of master devices is considered in [76]. As compared to [76], our goal is to minimize the energy consumption of both master and helper devices by taking into account hard deadlines.

Mobile cloud computing with hard deadlines are investigated in [77], [78], [79]. As compared to this line of work, we consider heterogeneous and time-varying resources as well as mobility. Available resources and mobility is predicted in our work. Furthermore, we investigate both serial and parallel tasks.

5.3 Setup and Problem Statement

Topology. We consider a setup illustrated in Figure 22 with multiple master and helper devices. We particularly focus on a master / helper cluster to make presentation simple. We assume that there is a *master* device and N *helper* devices in a cluster. We define a set $\mathcal{C} = \{M, W_1, \ldots W_N\}$ as the set of all devices in a cluster. The device $M \in \mathcal{C}$ is the *master* device that creates tasks and offloads them to other devices. The set $\{W_1, W_2 \ldots W_N\} \in \mathcal{C}$ represents *helper* devices that are in close proximity of the master device.

Offloading Scenario & Applications. We consider a scenario that a master device M runs multiple applications, and offloads computationally intensive applications to helper devices. In this chapter, we focus on face detection and matrix multiplication applications.

Face detection application processes a number of images, and detects human faces in each image. We employ Android SDK's *FaceDetector* and *FaceDetector.Face* classes to implement a face detection application, which puts a circle on the faces it detects. The processed image is stored in the external memories of devices.

Matrix multiplication application computes $Y = AX$ where $A = (a_{i,j}) \in \mathbb{R}^{R_1 \times R_2}$, $X = (x_{i,j}) \in \mathbb{R}^{R_2 \times R_3}$, and $Y = (y_{i,j}) \in \mathbb{R}^{R_1 \times R_3}$. Our application uses simple matrix multiplication, *i.e.,* computes $y_{i,j} = \sum_{r=1}^{R_2} a_{i,r} x_{r,j}, \forall i, j$.

Task Model. We consider a set of tasks; $\mathcal{K} = \{\text{Task } 1, \dots, \text{Task } K\}$. We consider two types of tasks: (i) serial, and (ii) parallel. When the tasks are serial, kth task can only be processed after the $k-1$th task is processed. Parallel tasks could be processed simultaneously at multiple helpers.

Offloading Policy. We assign each task to a device in the set $\mathcal{C}$. However, due to mobility, a master or a helper device may move after a task is offloaded, so they may be out of transmission range of each other. Thus, task offloading becomes unsuccessful (helper cannot send the processed task back to the master). In this case, the task should be rescheduled again. Assume that $t_{k,l}$ is the time that Task k is scheduled for the lth time, and $\pi_{k,l}$ is the policy that shows at which device that Task k is scheduled at the lth trial. Thus, $\pi_{k,l} \in \mathcal{C}$. For example, if the kth task is scheduled to be processed at the master device M (*i.e.,* if the task is not offloaded to any helper) at the lth trial, then $\pi_{k,l} = M$.

Let us assume that the set of policies $\boldsymbol{\pi}_k = \{\pi_{k,1}, \dots, \pi_{k,L_k}\}$ corresponds to the policy for Task k, where L_k is the last scheduling trial of Task k. Note that L_k depends on the optimal policy as well as the randomness due to mobility. The set $\boldsymbol{\pi} = \{\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_K\}$ corresponds to the policy for scheduling all the tasks.

Problem Statement. Our goal is to determine a policy $\boldsymbol{\pi}$ that minimizes the total energy consumption at all devices (master and helpers) subject to hard deadline constraints by estimating per-task energy consumption and delay at the master and helper devices as well as predicting the mobility of helpers.

5.4 Delay, Energy, and Mobility Prediction

In this section, we present our approach for predicting energy, delay, and mobility of master and helper devices using Android-based mobile devices. The results of this section will be used in our algorithm design in the next section.

We implemented a testbed of a master and multiple helper cluster using real mobile devices, specifically Android 6.0.1 based Nexus 6P smartphones. All the helpers are connected to the master device using Wi-Fi Direct connections in our testbed. (In other words, the master device is configured as the group owner of the Wi-Fi Direct group, which is a star topology.) We conducted our experiments using our testbed in a realistic lab environment, where several other Wi-Fi networks were operating in the background. We located all the devices in close proximity of each other (within a few meters distance). We use face detection application as a demonstrating example for our prediction.

5.4.1 Delay

We determine task delay as the amount of time for offloading, processing, and receiving tasks back from the helper devices. Each master device in our predictive edge computing framework determines per task delay by periodically probing itself as well as its helpers. In particular, the master device puts a timestamp to each task before processing or offloading it to a helper. For example, the time stamp of the ith task such that $i = kl$ is t_i.[1].

Assume that $\tilde{t}_{i,0}$ is the time that the master device completes the task by processing locally (no offloading), and $\tilde{t}_{i,n}$ is the time that it receives the completed task from helper W_n. Let us define $\mathcal{I}_{\mathcal{C}} = \{0, 1, \ldots, N\}$ as an index set of $\mathcal{C}$, where $0 \in \mathcal{I}_{\mathcal{C}}$ corresponds to the master device, and $n \in \mathcal{I}_{\mathcal{C}}$ corresponds to helper W_n. Thus, the delay for each device $j \in \mathcal{I}_{\mathcal{C}}$ becomes $\theta_{i,j} = \tilde{t}_{i,j} - t_i$. Note that $\theta_{i,j}$ includes only processing delay at the master device when $j = 0$, and it includes offloading and processing times as well as the time to receive tasks back from the helper devices when $j \neq 0$.

[1]Note that if device (master or helper) is not assigned any tasks due to our scheduling algorithm, we still offload tasks to this device periodically to predict its resources.

Figure 23 presents delay versus number of images for a master / helper setup. The amount of delay is the average of 10 trials. `Local - Wi-Fi Direct On` and `Local - Wi-Fi Direct Off` correspond to the scenarios that master device is locally processing the images when (i) it is connected to another device via Wi-Fi Direct and (ii) Wi-Fi Direct connection is closed, respectively. `Offloading` is the scenario that the master device offloads the images to its helper. As seen, the delay of `Local - Wi-Fi Direct Off` is less than both `Local - Wi-Fi Direct On` and `Offloading`, because it locally processes the packet, and no time is wasted for transmitting packets. On the other hand, `Offloading` is better than `Local - Wi-Fi Direct On`, because when a master device opens Wi-Fi Direct connection on and becomes a group owner (*i.e.,* behaves as an access point), it has more computational load, which increases delay. This figure shows that (i) delay characteristics of devices can be measured by probing these devices, and (ii) delay performance of a device is heterogeneous and can be time-varying depending on its configuration.

5.4.2 Energy

The main source of energy consumption in edge computing applications comes from computing and offloading tasks. This section deals with predicting energy consumption due to CPU usage and packet transmission (and reception) using Wi-Fi interfaces in Android-based devices. However, Android APIs do not provide granular (per application and hardware) energy consumption.

Next, we present our approach to predict energy consumption without using any external devices.

*Energy consumption due to computation (*i.e., *CPU).* The modern CPUs of Android devices consist of multiple clusters, and each cluster can operate at different speeds. Let $\iota_{c,s}$ be the amount of electrical

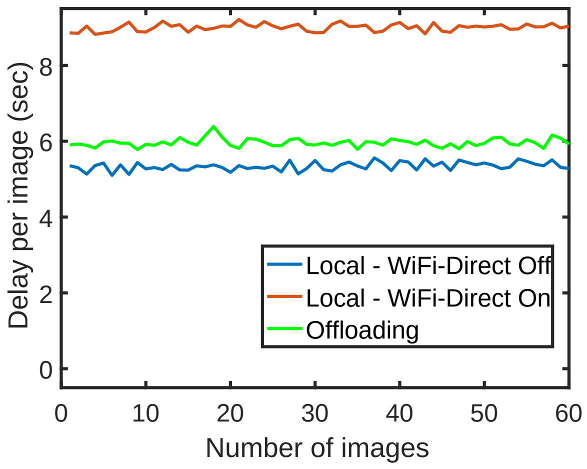


Figure 23. Average delay per image for a master / helper setup.

current (in mA) that cluster c uses when operating at speed s, which is not time-varying, and can be found on power profile of every Android device.

When a computationally intensive application is run on a device, multiple clusters at different speeds could be used. If we can predict the amount of time that each cluster - speed pair is used, we can characterize the amount of battery power (in mAh) used per application. Although Android API does not provide this information directly, the following information can be gathered.

Let (i) $T_a(t)$ be the total amount of time that application a has used CPU (across all clusters and speeds) since the device is plugged off from a power supply (let us denote this time t_0) until time t, and (ii) $\tau_{c,s}(t)$ be the amount of time cluster c is used at speed s between t_0 and t. Both $T_a(t)$ and $\tau_{c,s}(t)$ information can be acquired using the class *BatteryStatsHelper* of Android in the form of a list of “battery sippers”.

Each battery sipper represents an application associated with a unique application ID. The application ID of a desired application can be found in the *Process* Android class. The battery sipper has the battery related information including both $T_a(t)$ and $\tau_{c,s}(t)$. Next, we define per application battery consumption using these parameters.

The battery consumption due to application a between time interval $t-\delta$ and t, where δ is a small time interval, is expressed as

$$e_a^{\text{CPU}}(t, t-\delta) = \frac{T_a(t) - T_a(t-\delta)}{T_{\text{all}}(t) - T_{\text{all}}(t-\delta)} \sum_{\forall c} \sum_{\forall s} (\tau_{c,s}(t) - \tau_{c,s}(t-\delta)) \iota_{c,s}, \tag{5.1}$$

where $T_{\text{all}}(t)$ is the total amount of time that CPU is used for all applications. Although $T_{\text{all}}(t)$ cannot be directly gathered from Android APK, we can characterize it as $T_{\text{all}}(t) = \sum_{\forall c} \sum_{\forall s} \tau_{c,s}(t)$.

Note that the term $\frac{T_a(t)-T_a(t-\delta)}{T_{\text{all}}(t)-T_{\text{all}}(t-\delta)}$ in Equation 5.1 represents the percentage of time that application a uses available CPU resources as compared to all other applications. On the other hand, $\sum_{\forall c} \sum_{\forall s} (\tau_{c,s}(t) - \tau_{c,s}(t-\delta)) \iota_{c,s}$ represents the total energy consumption between time $t-\delta$ and t for all applications. The multiplication of these two terms is a good predictor of the energy consumption by application a between time interval $t-\delta$ and t.

Now, let us assume that ith task of application a is processed at device j at time $t-\delta$, and the total processing time is δ. Thus, we can characterize the amount of battery consumption due to CPU for processing task i at device j as $\epsilon_{i,j}^{\text{CPU}} = e_a^{\text{CPU}}(t, t-\delta)$. Figure 24 shows the real and calculated energy consumption for face detection application at the master device. In particular, the master device (Nexus 6P) detects faces in multiple images one by one. There is no other user-level applications running on the

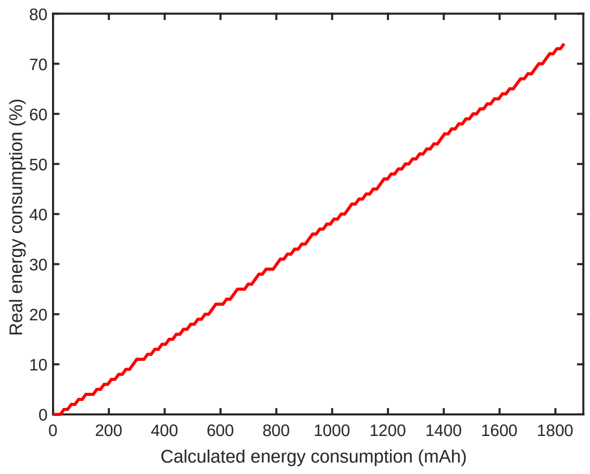


Figure 24. Real versus calculated energy consumption for face detection application at the master device.

device, and it operates in the *airplane mode*, so all network interfaces are closed (*i.e.,* no other energy consumption). The x-axis shows the cumulative energy consumption, *i.e.,* $\sum_{\forall\alpha}^{i} \epsilon_{\alpha,j}^{\text{CPU}}$, while the y-axis shows the battery drop percentage directly read from the device. Considering that the battery capacity of Nexus 6P devices is 3450 mAh, our energy calculation is a good predictor of per application energy consumption.

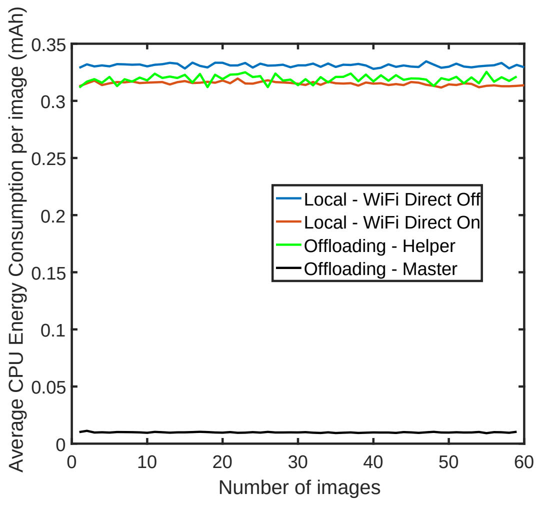
Energy consumption due to Wi-Fi. We measure this type of energy consumption using the energy consumption profile of Wi-Fi interface. (Note that Wi-Fi and Wi-Fi Direct share the same interface.) Unlike CPU energy consumption, it is straightforward to measure energy consumption in Android for using Wi-Fi interface. In particular, *wifiPowerMah* value is stored in the battery sipper discussed above, and it represents the total energy consumption to keep Wi-Fi interface open, transmitting, and receiving packets. Thus, we can directly obtain the battery consumption $e_a^{\text{Wi-Fi}}(t, t-\delta)$ due to application a between time interval $t-\delta$ and t, where δ is a small time interval. If ith task from application a is processed at device j during $t-\delta$ and t, the battery consumption due to Wi-Fi interface becomes $\epsilon_{i,j}^{\text{Wi-Fi}} = e_a^{\text{Wi-Fi}}(t, t-\delta)$.

The energy consumption per image due to CPU and Wi-Fi is presented in Figure 25. Each graph is an average of 10 experiments. `Offloading - Master` and `Offloading - helper` are the energy consumption at master and helper devices when tasks are offloaded to a helper device. `Local - Wi-Fi Direct On` and `Local - Wi-Fi Direct Off` are the same as in Figure 23.

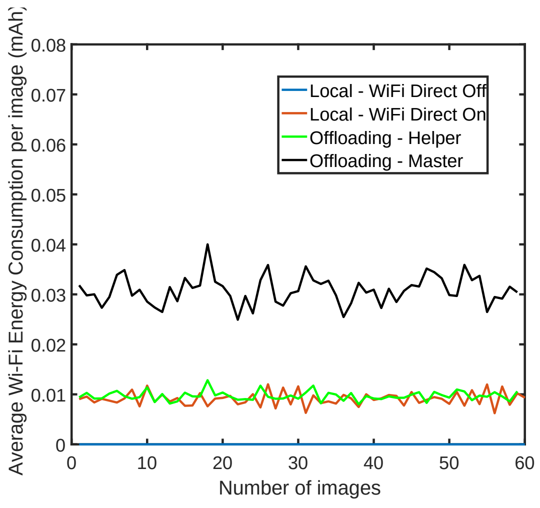
As seen, energy consumption due to CPU is higher than Wi-Fi and changes depending on whether a device is processing or offloading a task.

5.4.3 Mobility

We consider three types of mobility models: (i) *Statistical:* The probability that a master and helper devices are in the same transmission range is known a priori. (ii) *Predicted:* A master device predicts that a helper device is in its transmission range with some error margin. (iii) *Majority voting:* A master device uses history to predict the mobility of itself and helpers. In particular, a master device divides the time into slots and checks the most recent encounters (*i.e.,* being in the same transmission range) with a helper. If during most of the recent slots, there is encounter between a master and helper, the master decides that they will be in the same transmission range in the next slot.



(a) CPU



(b) Wi-Fi

Figure 25. Average energy consumption due to (a) computation (*i.e.,* CPU), (b) transmission and reception (*i.e.,* Wi-Fi).

5.5 `PrComp` Algorithms

In this section, we develop `PrComp` algorithms for serial and parallel tasks. Our `PrComp` algorithms are based on the solution to the optimal task allocation, and use the predicted delay, energy consumption, and mobility in Section 5.4.

5.5.1 Serial Tasks

Our first step is to solve the following optimization problem.

$$\begin{aligned} \min\nolimits_{\{k_n\}_{n\in\mathcal{I}_C}} & \sum_{n\in\mathcal{I}_C} E(n)k_n \\ \text{subject to} & \sum_{n\in\mathcal{I}_C} \Delta(n)k_n \le \Delta_{\text{thr}} \\ & \sum_{n\in\mathcal{I}_C} k_n = K, \end{aligned} \tag{5.2}$$

where $E(n)$ and $\Delta(n)$ are the average energy consumption and delay for processing one task at device n, k_n is the number of tasks assigned to device n, and Δ_{thr} is the hard deadline constraint for processing tasks. (We will describe how $E(n)$ and $\Delta(n)$ are calculated later in this section.) The objective function of Equation 5.2 is to minimize the total energy consumption at master and helper devices. The first constraint is the deadline constraint, and the last constraint makes sure that all K tasks are scheduled. We propose an online solution `PrComp` to Equation 5.2, which determines device $n^* = \pi^*_{k,l}$ at time $t_{k,l}$ for the lth trial of the kth task according to the following rule: Determine policy $\pi^*_{k,l} = \arg\min E(\pi_{k,l})$

that satisfies $\Delta(\pi_{k,l})(K-k+1) \leq \Delta_{\text{thr}} - t_{k,l}$. Next, we describe how to determine $\Delta(\pi_{k,l})$ and $E(\pi_{k,l})$ using our predicted values in Section 5.4.

The average delay $\Delta(\pi_{k,l})$ at device $n = \pi_{k,l}$ for the kth task at lth trial is expressed by taking into account mobility as

$$\Delta(\pi_{k,l}) = \Big(\sum_{i=1}^{kl-1} \frac{\theta_{i,\pi_{k,l}}}{kl-1}\Big)\frac{1}{1-P_{\pi_{k,l}}}, \tag{5.3}$$

where $\theta_{i,\pi_{k,l}}$ is the delay that is measured as described in Section 5.4.1 and $P_{\pi_{k,l}}$ is the probability that device $\pi_{k,l}$ will not be in the transmission range of the master device. In this formulation, $\sum_{i=1}^{kl-1} \frac{\theta_{i,\pi_{k,l}}}{kl-1}$ is the average delay of all per-task delays until $i = kl$th task, and $\frac{1}{1-P_{\pi_{k,l}}}$ reflects the contribution of the mobility on average delay.

The average energy consumption is formulated as

$$E(\pi_{k,l}) = \frac{\tilde{\epsilon}^{\text{Proc}}_{kl-1,\pi_{k,l}} + \tilde{\epsilon}^{\text{Off}}_{kl-1,\pi_{k,l}}}{1-P_{\pi_{k,l}}} \tag{5.4}$$

where $\tilde{\epsilon}^{\text{Proc}}_{kl-1,\pi_{k,l}}$ and $\tilde{\epsilon}^{\text{Off}}_{kl-1,\pi_{k,l}}$ are processing and offloading energy consumptions, respectively. The processing energy consumption at device j for the $i = kl$th task is formulated as

$$\tilde{\epsilon}^{\text{Proc}}_{i+1,j} = \Big(\tilde{\epsilon}^{\text{Proc}}_{i,j}\beta + (\epsilon^{\text{CPU}}_{i,j} + \epsilon^{\text{Wi-Fi}}_{i,j}1_{[j\neq 0]})\tilde{\beta}\Big)1_{[i\to j]} + \tilde{\epsilon}^{\text{Proc}}_{i,j}(1 - 1_{[i\to j]}) \tag{5.5}$$

where β is a small constant, $\tilde{\beta} = 1-\beta$, $1_{[x]}$ is an indicator function and takes value 1 if x is true, and 0 otherwise. $i \to j$ represents (a mapping) that task i is offloaded to device j. Note that $\epsilon^{\text{CPU}}_{i,j}$ and $\epsilon^{\text{Wi-Fi}}_{i,j}$

are measured as described in Section 5.4.2. The term $(\epsilon_{i,j}^{\text{CPU}} + \epsilon_{i,j}^{\text{Wi-Fi}} 1_{[j \neq 0]})$ in Equation 5.5 states that there is always energy consumption due to CPU, but there is energy consumption due to Wi-Fi only when the task is offloaded from the master device to helper devices (*i.e.,* when $j \neq 0$). $\tilde{\epsilon}_{i,j}^{\text{Proc}}(1 - 1_{[i \to j]})$ term shows that processing energy consumption is updated only if task i is offloaded to device j, *i.e.,* when $i \to j$ mapping is true. Similarly, the energy consumption due to offloading is expressed as

$$\tilde{\epsilon}_{i+1,j}^{\text{Off}} = \Big((\tilde{\epsilon}_{i,j}^{\text{Off}} \beta + \epsilon_{i,j}^{\text{Wi-Fi}} \tilde{\beta})(1 - 1_{[i \to j]}) + \tilde{\epsilon}_{i,j}^{\text{Off}} 1_{[i \to j]} \Big) 1_{[j=0]} \tag{5.6}$$

where $\tilde{\epsilon}_{i+1,j}^{\text{Off}} = 0$ when $j \neq 0$, because only the master device (*i.e.,* when $j = 0$) offloads tasks to helper devices. Note that both Equation 5.5 and Equation 5.6 assumes that $\tilde{\epsilon}_{0,j}^{\text{Proc}} = 0$, $\tilde{\epsilon}_{0,j}^{\text{Off}} = 0$, $\forall j$.

5.5.2 Parallel Tasks

Our first step is to solve the following optimization problem

$$\begin{aligned} \min_{\{k_n\}_{n \in \mathcal{I}_\mathcal{C}}} & \sum_{n \in \mathcal{I}_\mathcal{C}} E(n) k_n \\ \text{subject to } & \max_{n \in \mathcal{I}_\mathcal{C}} \{\Delta(n) k_n\} \leq \Delta_{\text{thr}} \\ & \sum_{n \in \mathcal{I}_\mathcal{C}} k_n = K, \end{aligned} \tag{5.7}$$

where the delay constraint is $\max_{n \in \mathcal{I}_\mathcal{C}} \{\Delta(n) k_n\} \leq \Delta_{\text{thr}}$ instead of $\sum_{n \in \mathcal{I}_\mathcal{C}} \Delta(n) k_n \leq \Delta_{\text{thr}}$ in Equation 5.2 thanks to parallel processing.

The optimal solution to Equation 5.7 orders devices depending on their average energy consumption $E(n)$ (in increasing order). The vector of ordered devices is $\boldsymbol{d}_e$, where $[\boldsymbol{d}_e]_r$ is the rth element of the vector $\boldsymbol{d}_e$. The optimal solution assigns $k_{[\boldsymbol{d}_e]_1} = \left\lfloor \frac{\Delta_{\text{thr}}}{\Delta([\boldsymbol{d}_e]_1)} \right\rfloor$ to device $[\boldsymbol{d}_e]_1$. If there still exist tasks waiting to be scheduled, it continues assigning tasks to $[\boldsymbol{d}_e]_2, \ldots, [\boldsymbol{d}_e]_{N+1}$ one by one using the same rule and stops when all the tasks are scheduled as summarized in Algorithm 1.

Algorithm 1 The optimal solution for parallel task allocation

```
1: $K_{\text{sch}} = K$. $r = 1$. $k_{[\boldsymbol{d}_e]_r} = 0, \forall r \in \{1, \ldots, N+1\}$.
2: while $K_{\text{sch}} > 0$ AND $r \leq N+1$ do
3:     Assign $k_{[\boldsymbol{d}_e]_r} = \left\lfloor \frac{\Delta_{\text{thr}}}{\Delta([\boldsymbol{d}_e]_r)} \right\rfloor$ tasks to device $[\boldsymbol{d}_e]_r$
4:     $K_{\text{sch}} = \max\{0, K_{\text{sch}} - k_{[\boldsymbol{d}_e]_r}\}$. $r = r + 1$
5: end while
```

Theorem 4. *Suppose there exists a feasible solution to Equation 5.7, then Algorithm 1 is the optimal solution to Equation 5.7.*

Proof: Without loss of generality, let N denote the total number of helpers in the set $\mathcal{C}$ and each helper is placed in index 1 through N according to their average energy consumption where $E(1) \leq E(2) \leq \cdots \leq E(N)$. Let $sol = \{k_1, k_2, \cdots, k_{z-1}, k_z, k_{z+1}, \cdots, k_N\}$ denote the solution obtained from Algorithm 1 where $\sum_{i=1}^{z} k_i = K$ and $k_{z+1} = \cdots = k_N = 0$ $(1 \leq z \leq N)$. That is to say helper z is the last helper in the task assignment order that completes the K tasks scheduling requirement. It can be noted that $k_i = \left\lfloor \frac{\Delta_{\text{thr}}}{\Delta(i)} \right\rfloor$ $(i < z)$ and $k_z = \min(\left\lfloor \frac{\Delta_{\text{thr}}}{\Delta(i)} \right\rfloor, K - \sum_{i=1}^{z-1} k_i)$.

Due to the deadline constraint in Equation 5.7, for any other solution $sol^* = \{k_1^*, k_2^*, \cdots, k_z^*, k_{z+1}^*, \cdots, k_N^*\}$ we have

$$k_i^* \leq \left\lfloor \frac{\Delta_{\text{thr}}}{\Delta(i)} \right\rfloor, \forall i \in \{1, \cdots, N\} \tag{5.8}$$

Thus,

$$k_i^* \leq k_i, \forall i \in \{1, \cdots, z-1\} \tag{5.9}$$

Furthermore, since $k_{z+1} = \cdots = k_N = 0$, we have

$$k_i^* \geq k_i, \forall i \in \{z+1, \cdots, N\} \tag{5.10}$$

As for the relationship between k_z^* and k_z, let us consider two scenarios, namely, $k_z^* \leq k_z$ and $k_z^* > k_z$.

Scenario 1 $k_z^* \leq k_z$: Let D denote the total task completion difference among the first z helpers using solution sol and sol^*.

$$D = \sum_{i=1}^{z} k_i - \sum_{i=1}^{z} k_i^* \geq 0 \tag{5.11}$$

Due to the total K tasks completion constraint in Equation 5.7, we have $\sum_{i=1}^{N} k_i = \sum_{i=1}^{N} k_i^* = K$. Thus,

$$\sum_{i=z+1}^{N} k_i^* - \sum_{i=z+1}^{N} k_i = \sum_{i=z+1}^{N} k_i^* = D \geq 0 \tag{5.12}$$

The difference of total average energy consumption using sol and sol^* is

$$\begin{aligned}
& \sum_{i=i}^{N} E(i)k_i^* - \sum_{i=1}^{N} E(i)k_i \\
= & \ (\sum_{i=1}^{z} E(i)k_i^* + \sum_{i=z+1}^{N} E(i)k_i^*) - \sum_{i=1}^{z} E(i)k_i \\
= & \ \sum_{i=1}^{z} E(i)(k_i^* - k_i) + \sum_{i=z+1}^{N} E(i)k_i^* \\
\geq & \ E(z)\sum_{i=1}^{z}(k_i^* - k_i) + E(z+1)\sum_{i=z+1}^{N} k_i^* \\
= & \ -E(z)D + E(z+1)D \\
= & \ D(E(z+1) - E(z)) \\
\geq & \ 0 \qquad (5.13)
\end{aligned}$$

Scenario 2 $k_z^* > k_z$: Let D' denote the total task completion difference among the first $z-1$ helpers using solution sol and sol^*.

$$D' = \sum_{i=1}^{z-1} k_i - \sum_{i=1}^{z-1} k_i^* \geq 0 \qquad (5.14)$$

Due to the total K tasks completion constraint in Equation 5.7, we have $\sum_{i=1}^{N} k_i = \sum_{i=1}^{N} k_i^* = K$. Thus,

$$\sum_{i=z}^{N} k_i^* - \sum_{i=z}^{N} k_i = D' \geq 0 \qquad (5.15)$$

Hence, the difference of total average energy consumption using sol and sol^* is

$$\begin{aligned}
& \sum_{i=1}^{N} E(i)k_i^* - \sum_{i=1}^{N} E(i)k_i \\
= \ & (\sum_{i=1}^{z-1} E(i)k_i^* + \sum_{i=z}^{N} E(i)k_i^*) - (\sum_{i=1}^{z-1} E(i)k_i + \sum_{i=z}^{N} E(i)k_i) \\
= \ & \sum_{i=1}^{z-1} E(i)(k_i^* - k_i) + \sum_{i=z}^{N} E(i)(k_i^* - k_i) \\
\geq \ & E(z-1)\sum_{i=1}^{z-1}(k_i^* - k_i) + E(z)\sum_{i=z}^{N}(k_i^* - k_i) \\
= \ & -E(z-1)D' + E(z)D' \\
= \ & D'(E(z) - E(z-1)) \\
\geq \ & 0
\end{aligned} \tag{5.16}$$

Therefore, any solution sol^* that is different from sol would achieve higher total average energy consumption than sol. This concludes the proof. ■

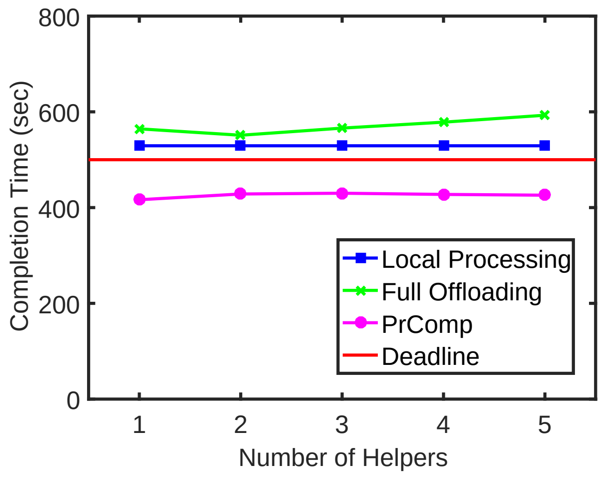
Our online algorithm mimics the offline solution in Algorithm 1. At the start (when scheduling starts), our algorithm runs Algorithm 1. If $k_{[\boldsymbol{d}_e]_r} > 0$, one task is assigned to device $[\boldsymbol{d}_e]_r$. Then, periodically or when a device finishes processing a task, Algorithm 1 is run again and if $k_{[\boldsymbol{d}_e]_r} > 0$, a task is assigned to device $[\boldsymbol{d}_e]_r$. This procedure continues until all tasks are successfully scheduled or hard deadline constraint is reached. As compared to Algorithm 1, our online algorithm assigns tasks to devices one by one, which better adapts to the time-varying resources at edge devices.

5.6 Performance Evaluation

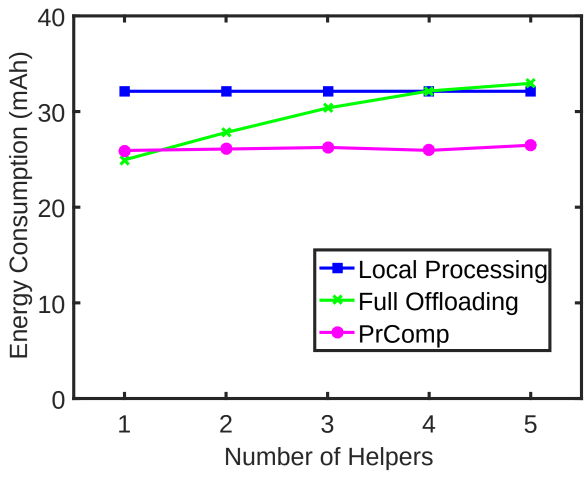
In this section, we evaluate the performance of our algorithm; `PrComp` for serial and parallel setups using Android-based smartphones. We implemented a testbed of a master and multiple helpers using real mobile devices, specifically Android 6.0.1 based Nexus 6P and Nexus 5 smartphones. Nexus 6P has higher energy efficiency than Nexus 5. All the helpers are connected to the master device using Wi-Fi Direct connections. We conducted our experiments using our testbed in a lab environment where several other Wi-Fi networks were operating in the background. We located all the devices in close proximity of each other (within a few meters distance).

Figure 26 shows the performance of `PrComp` for serial tasks, where we used the face detection application, similar to the setup in Section 5.4, as a serial task. The master device is Nexus 5, and the helpers are Nexus 6P. The performance of `PrComp` is evaluated as compared to baselines: (i) `Full Offloading`, which offloads each task to a helper device that has the least energy consumption, but does not allow local processing. This baseline is similar to the algorithm developed in [76], but updated for serial tasks setup, (ii) `Local Processing`, where the master device processes all the tasks (*i.e.,* it does not offload tasks). We assume *statistical* mobility model described in Section 5.4, where time is divided into 10 sec slots. At each slot, one of the helpers moves out of transmission range of the master device with probability 0.3, and comes back to the transmission range with probability 0.5. For the other helpers, these (both moving out and in) probabilities are 0.9. Given these values, it is straightforward to calculate $P_{\pi_{k,l}}$. Figure 26 (a) shows the task completion time versus number of helpers, and Figure 26 (b) shows the total energy consumption (at all masters and helpers). As seen `PrComp` satisfies the hard deadline constraint and significantly reduces task completion time, while `Local Processing` fails to satisfy the deadline constraint on the average and `Full Offloading` fails to satisfy the deadline constraint in some instances (confidence interval exceeds deadline). `Full Offloading` performs worse in terms of both delay and energy consumption, because the master device is Nexus 5 which is a weaker device as compared to helpers (Nexus 6P). The energy consumption of `Full Offloading` increases with increasing number of helpers, because more helpers cause more energy consumption. `PrComp` performs the best as it (i) takes advantage of local resources at the master device as well as helpers, and (ii) is adaptive to time-varying resources.

Figure 27 shows the performance of `PrComp` for serial tasks when we use matrix multiplication $Y = AX$ as the application. A is a 10K $\times$ 10K matrix, X is a 10K $\times$ 1 vector. Matrix A is divided into 500 sub-matrices, each of which is a 20 $\times$ 10K matrix. Again, the master device is a Nexus 5 and the helpers are Nexus 6P. The performance of `PrComp` is compared to the same baselines, `Local Processing` and `Full Offloading`. We assume that for each task, the first helper has 0.1 probability of failing (low mobility helper) and the other helpers have 0.6 probability of failing (high mobility helper). Figure 27 (a) and Figure 27 (b) show the task completion time and energy consumption against number of helpers. As seen, `Local Processing` performs better in terms of task completion time, while the energy consumption performance of `PrComp` is better than `Local Processing` and the task completion time of `PrComp` still meets the hard deadline constraint. On the other hand, `Full Offloading` cannot meet the deadline as it does not use local (at master) resources. Its energy consumption is better than both `PrComp` and `Local Processing` in the one-helper scenario, as `Full Offloading` offloads all the tasks from the master device (Nexus 5) to the helper device (Nexus 6P), and Nexus 6P is more energy efficient than Nexus 5. Both the task completion time and energy



(a) Task completion time



(b) Energy consumption

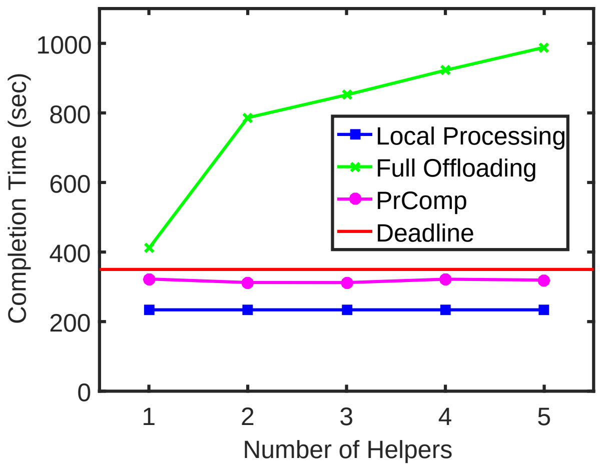
Figure 26. The task completion time and energy consumption performance of `PrComp` for serial face detection tasks. The figures are generated by averaging 16 trials. 60 images are processed and the deadline threshold is 500 sec.

consumption of `Full Offloading` increases with increasing number of helpers as more time and energy are wasted on helpers with high mobility. To summarize, `PrComp` takes advantage of using both local (at master) and remote (at helper) resources to minimize the total energy consumption while satisfying hard deadline constraints.

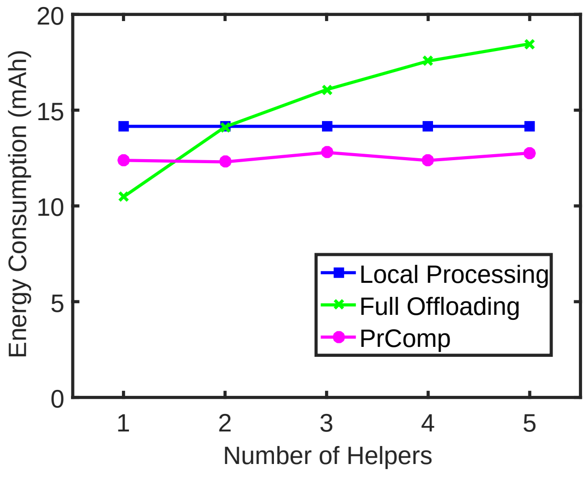
Figure 28 demonstrates the performance of `PrComp` for face detection application in a serial-task setup. The master device is a Nexus 5 and the helpers are Nexus 6P. There are five helpers in this setup. One of the helpers is a low mobility helper, which can have connection to the master device 70% of the time. The connection (or fail) probability of other helpers varies from 0.1 to 0.9.

Figure 28 (a) and Figure 28 (b) show the task completion time and the average energy consumption in the system versus fail probability of the four helpers. As can be seen, when the fail probability is small `PrComp` and `Full Offloading` meet the deadline requirement (while `Local Processing` misses the deadline) and their energy consumption levels are close to each other. The reason is that offloading is a better decision in terms of both delay and energy when fail probability is small as helper devices are more delay and energy efficient than the master device. When the fail probability increases, more and more offloaded tasks will fail and more packets will be rescheduled which increases completion time and energy consumption. For example, the completion time and energy consumption of `Full Offloading` is exponentially increasing with increasing fail probability. On the other hand, `PrComp` wisely uses local (*i.e.,* at master) and remote (*i.e.,* at helper) resources efficiently. and its completion time is always below the hard deadline and it is efficient in terms of energy consumption.

Figure 29 shows the delay and energy performance of `PrComp` for parallel tasks, where we used matrix multiplication $Y = AX$ as a parallel task. The sizes of A and X are the same as the serial case.

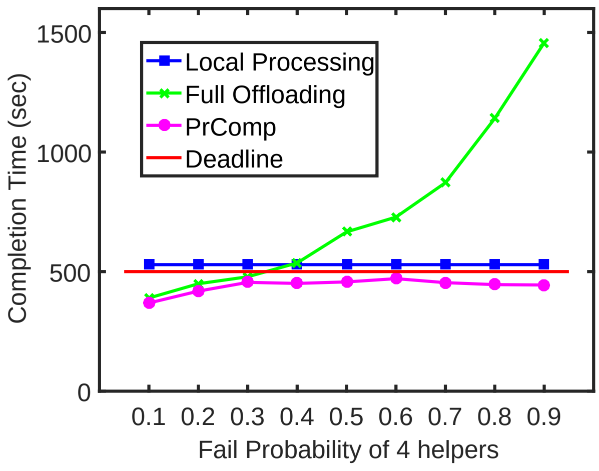


(a) Task completion time

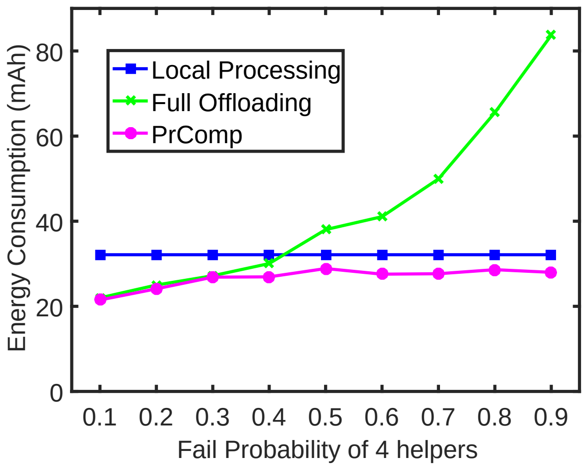


(b) Energy consumption

Figure 27. The task completion time and energy consumption performance of `PrComp` for serial matrix multiplication tasks. The figures are generated by averaging 10 trials. 500 matrices are processed and the deadline threshold is 350 sec.



(a) Task completion time



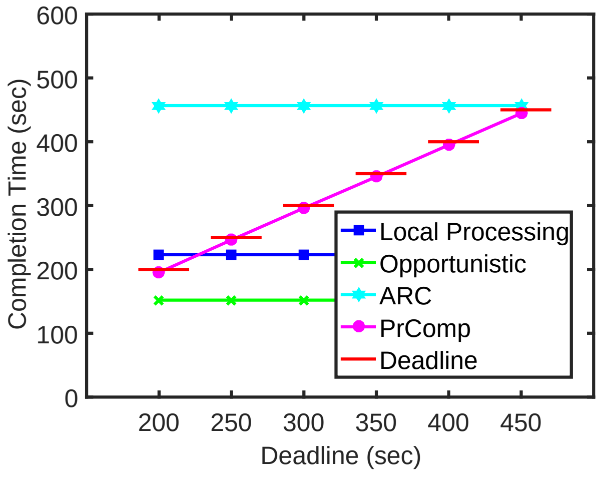
(b) Energy consumption

Figure 28. The task completion time and energy consumption performance of `PrComp` for serial face detection tasks. The figures are generated by averaging 5 trials. 60 images are processed and the deadline threshold is 500 sec.

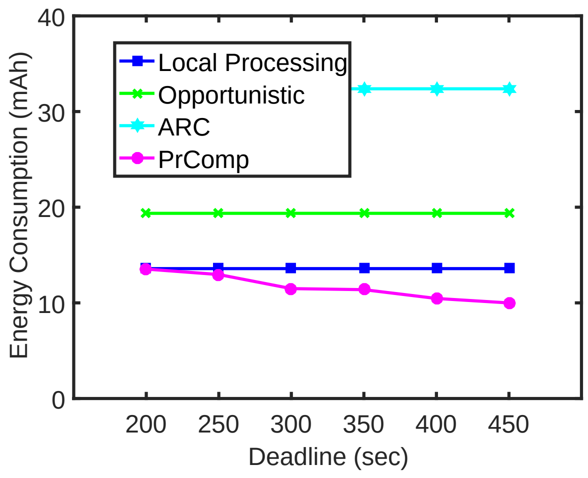
There is a master device (Nexus 5) and two helpers (Nexus 6P). The probability of not being in the same transmission range of helpers are $P_1 = 0.1$ and $P_2 = 0.8$. PrComp is compared with baselines: Local Processing, which is the same algorithm described above; Opportunistic, which uses master and helper devices simultaneously; and ARC, which is an algorithm developed in [76] to reduce the energy consumption at local devices (*i.e.,* the master device). As seen in Figure 29(a), PrComp, although it has larger task completion time as compared to baselines, it always satisfies hard deadline constraints. Furthermore, PrComp reduces total energy consumption as compared to baselines, and its energy efficiency increases when the hard deadline threshold increases, because PrComp has a larger set of task scheduling policies that it can exploit when deadline threshold increases.

Figure 30 and Figure 31 demonstrate the delay and energy performance of PrComp versus the number of helpers for the same setup described above, but we use face detection as a parallel task. The hard deadline is 500 seconds. The probability of not being in the same transmission range of helpers are $P_1 = 0.3$ and $P_2 = \ldots = P_5 = 0.6$ and $P_1 = 0.3$ and $P_2 = \ldots P_5 = 0.9$ for Figure 30 and Figure 31, respectively. As seen, PrComp always satisfies the hard deadline constraints and performs better in terms of energy consumption when mobility of helpers increases thanks to making task offloading decisions by taking into account the mobility of devices.

Figure 32 shows the task completion time and energy consumption performance of PrComp versus error margin. Master device is Nexus 5 and we have three helper devices; all of them are Nexus 6P. The hard deadline constraint is 400 seconds. The helper devices follow mobility pattern from the dateset in [1], which collects data on if the master and a helper device are in the same transmission range or



(a) Task completion time



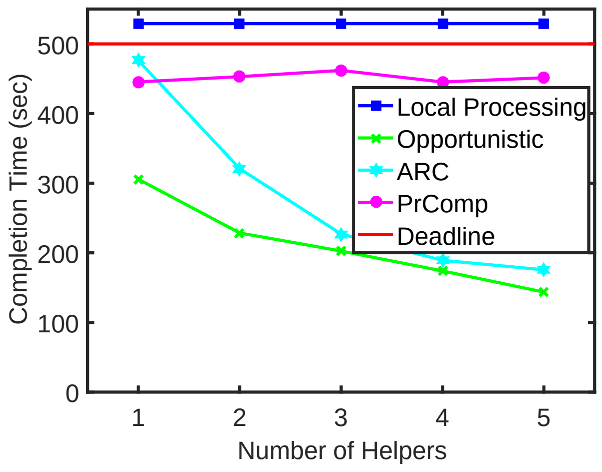
(b) Energy consumption

Figure 29. The task completion time and energy consumption performance of `PrComp` for parallel tasks. The figures are generated by averaging 16 trials.

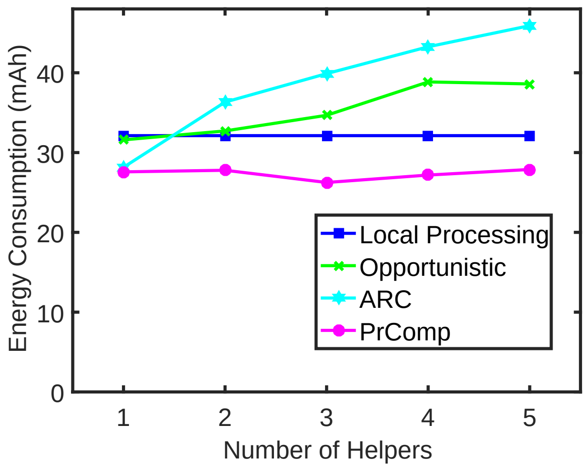
not. The master device estimates whether a helper device is in its transmission range with some error probability, which corresponds to the error margin.

`PrComp` is compared with baselines: `Opportunistic` and `ARC`. As seen in Figure 32 (a), both `Opportunistic` and `PrComp` satisfies the hard deadline while the average completion time of `ARC` exceeds the deadline. The energy consumption of `PrComp` is less than `Opportunistic` as seen in Figure 32 (b), but higher than `ARC` as `ARC` is optimized for energy, but `ARC` misses the hard deadline constraint as seen in Figure 32 (a).

Figure 33 shows the delay and energy performance of `PrComp` when majority voting is used to predict the mobility of devices. These results are for the parallel-task setup, where we used the face detection application as a parallel task. The master device is Nexus 5 and helpers are Nexus 6P smartphones. There are three helpers. All helpers move according to the mobility pattern from the dataset in [1]. In order to predict the mobility, we divide the time into slots as described in Section 5.4.3, and the master device counts the number of encounters in the last 5 slots with each helper device. If there are encounters during most of the slots with a helper, the master concludes that it will encounter with this helper at the next time slot. Figure 33 shows that `PrComp` significantly improves task completion time as well as energy consumption as compared to `Local Processing` thanks to effectively using resources at master and helpers by predicting mobility, while `Local Processing` is limited with the resources at the master device. Note that the average completion time of `ARC` exceeds the deadline, while `Opportunistic` can achieve minimum delay by utilizing all available resources. However, `PrComp` reduces the energy consumption as compared to `Opportunistic` while meeting the deadline requirement by offloading more tasks to stable helpers.

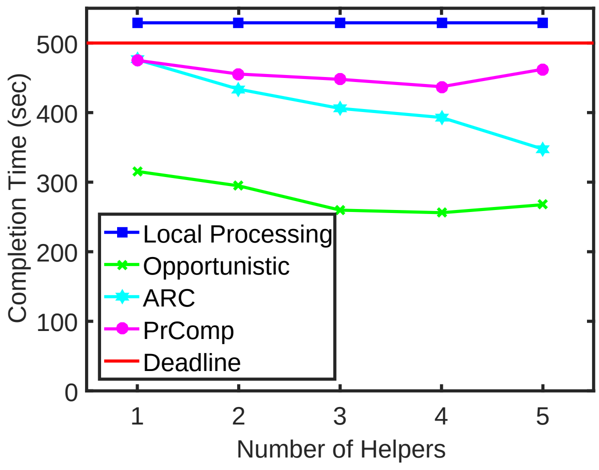


(a) Task completion time

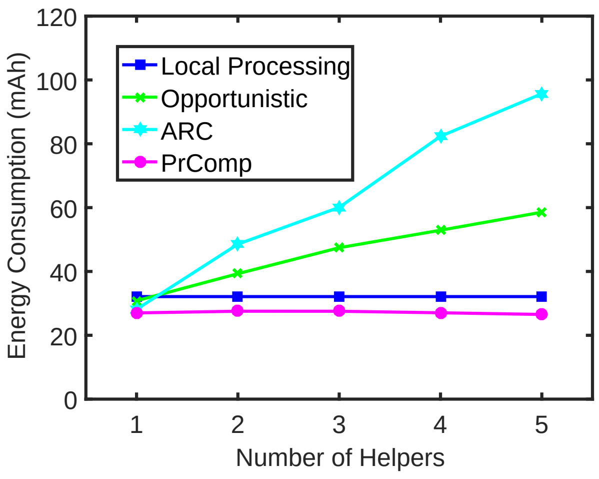


(b) Energy consumption

Figure 30. The task completion time and energy consumption performance of `PrComp` versus number of helpers for parallel tasks (face detection). The figures are generated by averaging 5 trials. The probability of not being in the same transmission range of helpers are $P_1 = 0.3$ and $P_2 = \ldots = P_5 = 0.6$.

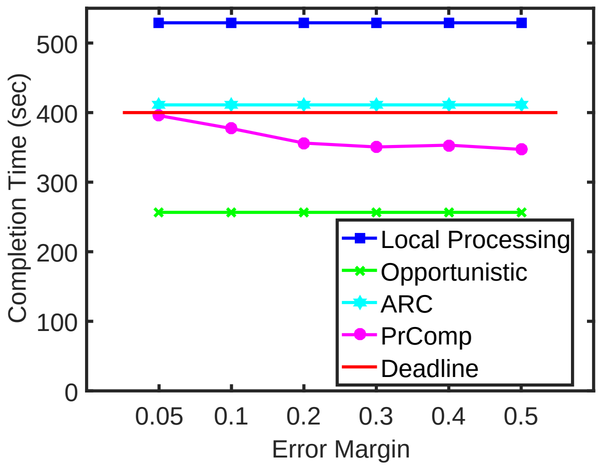


(a) Task completion time

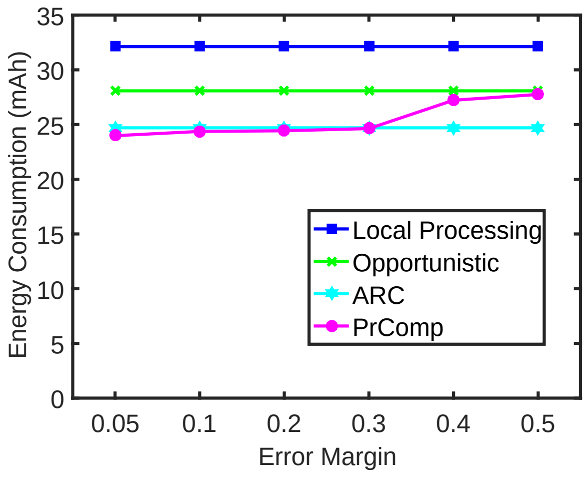


(b) Energy consumption

Figure 31. The task completion time and energy consumption performance of `PrComp` versus number of helpers for parallel tasks (face detection). The figures are generated by averaging 5 trials. The probability of not being in the same transmission range of helpers are $P_1 = 0.3$ and $P_2 = \dots P_5 = 0.9$.

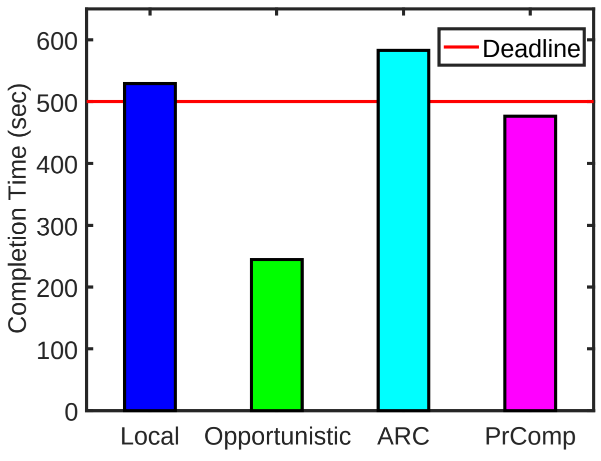


(a) Task completion time

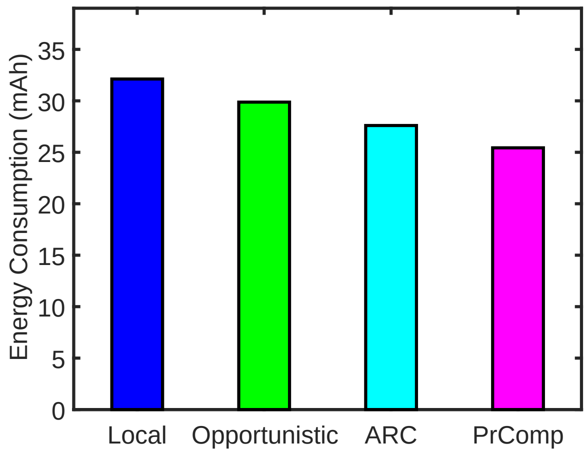


(b) Energy consumption

Figure 32. The task completion time and energy consumption performance of `PrComp` versus error margin for parallel tasks (face detecton) when helpers follow real mobility data in [1]. The figures are generated by averaging 5 trials.



(a) Task completion time



(b) Energy consumption

Figure 33. Completion time and energy consumption performance of `PrComp` when mobility is predicted via majority voting. The figures are generated by averaging 5 trials.

CHAPTER 6

THE EVOLVING NATURE OF DISASTER MANAGEMENT IN THE INTERNET AND SOCIAL MEDIA ERA

The contents of this chapters are based on our work that is published in the proceedings of 2018 IEEE LANMAN conference [6]. ©2019 IEEE. Reprinted, with permission, from [6].

Traditional means for contacting emergency responders depend critically on the availability of the 911 service to request help. Large-scale natural disasters such as hurricanes and earthquakes often result in overloading and sometimes failure of communication facilities. Affected citizens are increasingly using social media to obtain and disseminate information. Social media is not only being used to communicate with first responders but also for people to organically volunteer and seek help from each other, complementing the role of first responders. In this chapter, we examine the use of Twitter during two major hurricanes in the U.S. in 2017. We find that there exists a sizable number of people with access to the Internet even in areas where 911 services were down, and they tweet disaster-related information including requests for help. Our analysis indicates that social media can potentially help in disaster management and improve outcomes.

6.1 Background

Effective communication among citizens in need of help, first responders and others who are able to help during and in the aftermath of a disaster can affect outcomes dramatically. The ability to provide timely and relevant information to the right person can help manage disasters better and save lives.

The past several years have brought significant changes on how our society communicates, increasingly dominated by social media. The integration of social media in daily lives has also dramatically changed how victims, volunteers, and first responders exchange information, seek and provide help during and after disasters.

As an example, during Hurricane Harvey [80], hundreds of stranded Texas residents sought help by posting on Twitter, Facebook, Instagram, *etc.* [81]. Social media was also used in the aftermath of the Great Eastern Japanese earthquake and tsunami in 2011 [82]. In the case of Hurricanes Harvey and Irma [83], they tweeted their addresses to emergency officials and posted pictures to clarify or emphasize their situation. Especially, when people felt that traditional aid-seeking methods such as 911 was not adequate, due to the overloaded demand and also infrastructure being down, and mainstream news media was not real-time enough [81, 84–86], many of them posted their address, location with information and pictures about their situation online, hoping to get help. Another concern when traditional approaches such as “911” were overloaded as that cell phone batteries ran down while on hold [81], and without power, social media became important.

Thus, people sought help by using their everyday communication mechanisms on social media. The asynchronous nature of data communication and the ability to spread the information widely with reasonable efforts may also be likely driving factors. Further, informal ad-hoc volunteer groups are increasingly becoming an integral part of rescue efforts. Such volunteer groups used social media extensively to organize effective rescue missions.

We believe it is crucial to better harness the potential of social media information and the Internet connectivity to individuals during time-critical situations. Techniques that can automatically process social media posts and Internet connectivity to identify potential victims or areas needing help should be developed to improve the disaster management. Further, again, it is imperative to devise social media data processing methods to understand the trust and security challenges. Identifying if a tweet calling for help is malicious or establishing a framework for civilians and communities to communicate securely via social media without being risked of looters or individuals with malicious intent are examples of such challenges.

Communications infrastructures fail as a result of disasters (especially the last mile, with telephone poles, cable and fiber nodes in the neighborhood or cell phone base stations being impacted). A critical lifeline for disaster management has been the ability to contact emergency services through a telephone number (such as “911” in the United States). It primarily provides ability for citizens to request help from first responders. Also in modern day systems, it provides automatic location identification, a feature that is very helpful in speeding up the delivery of emergency services. It is therefore very helpful to understand the current capabilities of the infrastructure to sustain disasters. The United States Federal Communications Commission (FCC) collect and report data on the availability of these emergency services as well as of cellular communication on a relatively fine grain (both spatial and temporal) data, which we examine in this chapter. When emergency services (*i.e.,* 911) are impacted, it is also useful to examine if other forms of communication (*e.g.,* Internet connection via cellular) are impacted. Moreover, since emergency services are synchronous communication between people seeking help and a human facilitator at the other end, the limited availability of a large enough number of 911 operators in the disaster stricken area can severely limit service. Moreover, when a person calls such a service and is put on hold, then consumes power on the hand-held device (often a cellular phone because land-line, wired access may have failed), and the battery drain can be a serious concern at times when power is a precious, scarce commodity. As such, the ability to utilize alternative form of communication, such as the Internet via social media (broadcast, group multicast) or email can be valuable. However, even this requires access to communication facilities (*e.g.,* cellular networks, which may be the "last facility standing"). It is important therefore to understand the availability of the cellular infrastructure as well. We examine the failure and repair times for these facilities as well as the finer-grained service of providing automatic location identification (ALI) during and in the aftermath of recent disasters in the United States.

6.2 Related Work

Social media has been increasingly used for communication and information dissemination for crisis response purposes. Work in [87, 88], among others, describe the effect of online social media for emergency relief. Gao et al. [89] describe the benefits, issues and research topics of using online social media for disaster relief, confirming our motivation. In this chapter, we focus on two major recent disasters in the U.S.: Hurricanes Harvey and Irma, during which numerous civilians in the affected areas, chose to or had to go online on Social media, such as Facebook or Twitter, in order to get information and also ask for help.

Social media can be very beneficial, as a supplement to traditional communication methods, to help requests get distributed and go viral, catching the attention of more people, especially volunteers who can help. One example of this benefit was a case of the residents of a nursing home being rescued after it was posted online during Harvey [84]. Additionally, social media can bring more comfort to disaster victims, as they feel that they have a better chance of being noticed and possibly rescued [85], as it can be a more reliable source when infrastructure is down [86].

The latest Google crisis response tools provide several disaster management related features such as Google person finder, Google crisis maps and Google public alerts [90]. Google person finder is a web application serves like a message board for survivors and their friends and families to find each other during a natural disaster [91]. Google crisis maps publish the geo-spatial disaster information such as updated satellite images, flood zones, evacuation routes and shelters [92]. Furthermore, users that are close to the impacted areas will get notifications about the disasters pushed to their mobile devices via Google public alerts available in the search engine and maps [93]. Facebook's disaster response on the other hand, is a service that allows people in the affected areas to find or offer help. The types of help are categorized such as food, clothes, shelters, fuels, *etc.* Also, people in the affected area can share information that they are safe quickly to their friends on Facebook via the "Safety check" [94]. Safety check function will be activated automatically when a lot of people in the affected area are posting about the incident or disaster. Finally, Facebook utilizes their app along with a location service to deliver disaster maps [95]. Disaster maps contain the information about the density, movement and Safety checks of the population in the affected areas.

Using the social media content in an efficient architecture can be helpful for disaster information dissemination. Work in [96] proposes a location-independent information-centric approach [97] for this purpose; it studies the 2010 Haiti earthquake dataset [98,99] for the communication trace for evaluation. The authors observed that a social media-like system with push capability can dramatically improve the performance of message dissemination in such disasters.

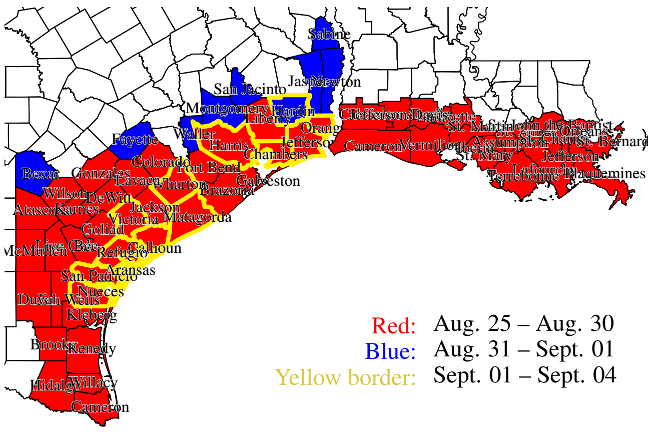


Figure 34. Disaster area of Hurricane Harvey.

6.3 Data Set Collection

We focus on a pair of hurricanes, which were significant natural disasters that occurred in the U.S. in August and September 2017. Hurricane Harvey hit the state of Texas on August 25th and its effects (*e.g.,* rain and flooding) lasted until the end of the month. Hurricane Irma hit the state of Florida on September 9th and its effects continued beyond September 14th. We collected data about Harvey and Irma from two sources: 1) FCC communication status report on these hurricanes, to get an understanding about the infrastructure failure including that for emergency services, and 2) social media data, in particular Twitter, to explore where and when users tweeted information about the disasters.

6.3.1 FCC Communication Status Report

The U.S. Federal Communications Commission (FCC) activates the Disaster Information Report System (DIRS) in disaster areas when requested by Federal Emergency Management Agency (FEMA). The DIRS collects data from the following three sources:

Figure 35. Tweets crawled during Harvey.

Date	Tweet #	Start (CDT)	End (CDT)
Aug.24	117,971	6:33:34 pm	6:59:59 pm
Aug.25	432,968	5:10:12 pm	6:59:59 pm
Aug.26	96,664	6:39:12 pm	6:59:59 pm
Aug.27	561,187	5:22:52 pm	6:59:59 pm
Aug.28	288,618	6:09:12 pm	6:59:59 pm
Aug.29	308,723	6:10:22 pm	6:59:59 pm
Aug.30	182,117	6:23:10 pm	6:59:59 pm

- *Communication providers* for civilians (including wireless, wireline and cable) submit their network outage. FCC reports county-based outage of cellular communications, and overall wireline and cable outage.
- *The Public Safety and Homeland Security Bureau (PSHSB)* learns about the status of each Public Safety Answering Point (PSAP) through the filings of 911 Service Providers in the DIRS, through reporting to the FCC's Public Safety Support Center (PSSC), coordination with state 911 Administrators and, if necessary, individual PSAPs.
- *Broadcast media* outages including TV and radio stations.

Of the FCC data, we mainly focus on cellular outage and PSAP (911 service) status.

For Harvey and Irma, FCC published the communication status report around 11am EDT each day [100, 101]. For Harvey, 55 counties in Texas and Louisiana (marked red in Figure 34) are listed as disaster areas from August 25 to 30. Nine more counties in Texas (marked in blue in Figure 34) were

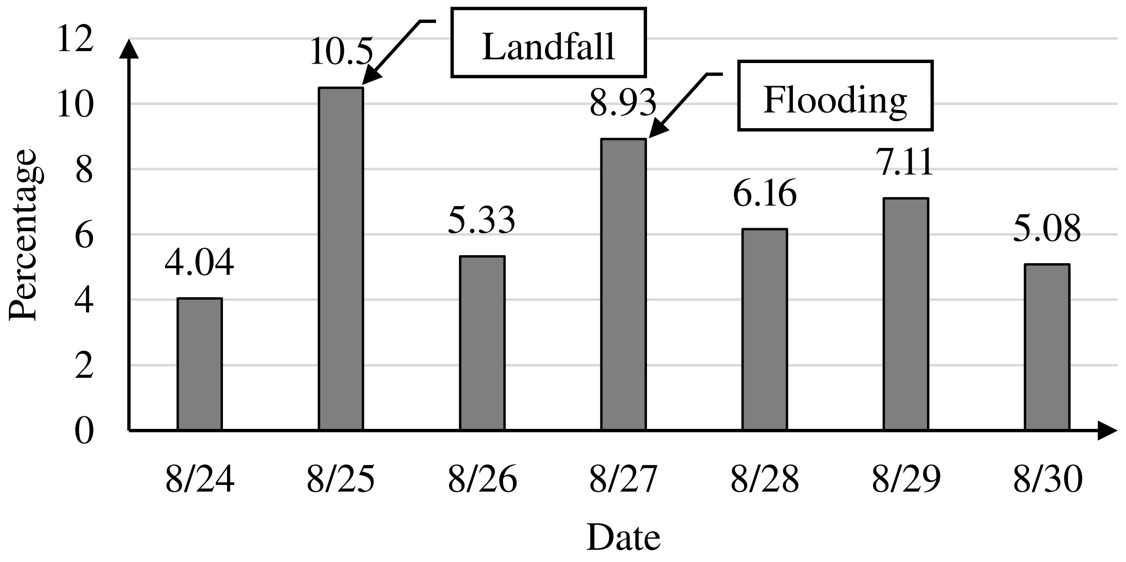


Figure 36. Percentage of Harvey-related Tweets over total collected tweets.

added for the period from August 31 to September 1. DIRS remained active in 13 counties (with yellow border in Figure 34) between September 2nd and 4th. For Irma, all the counties in Florida were marked as disaster areas from September 10 to 17.

6.3.2 Twitter Data Crawling

To obtain an idea of the extent to which social media was being used during and in the aftermath of disasters, we developed a Twitter crawling application. Twitter allows querying for tweets using developer accounts, with a limit on the rate of queries. The query builder and sender part of our application sends queries with

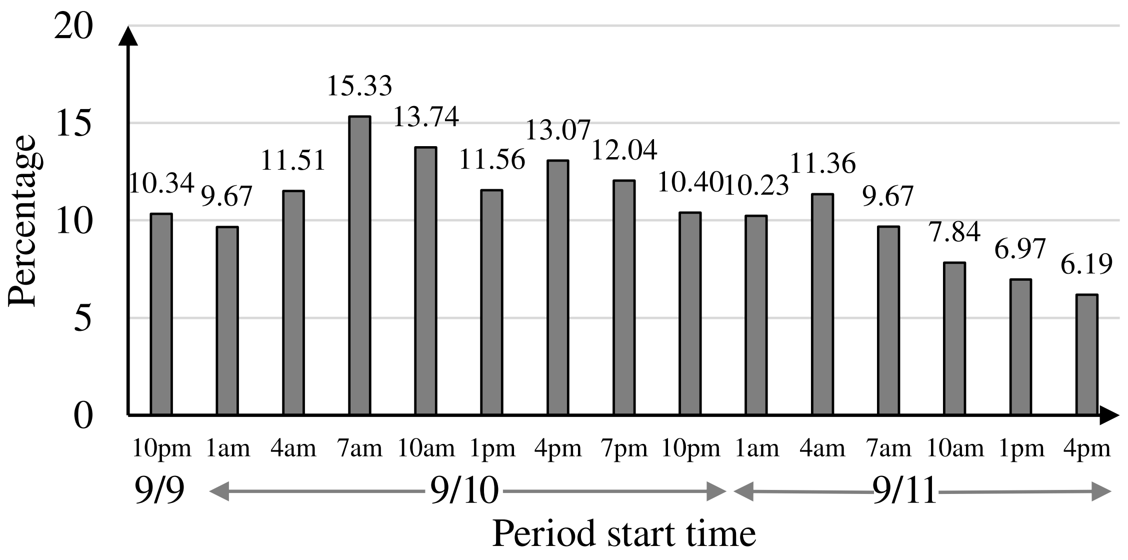


Figure 37. Percentage of Irma-related tweets during Sept. 9-11 over 3-hour periods.

- *geocode* that specifies the location of the queried tweets;
- *time-interval* of interest for the tweets to constrain what was collected;
- *count*: the number of tweets returned per query, and
- *maxId*: limits the tweets according to their IDs which is assigned and sorted by Twitter.

Our application receives and saves the responses to queries. We collected data sets for both Harvey and Irma:

- For Harvey, we crawled 1,988,248 tweets within a 500-mile radius of Houston. Figure 35 has timing and number of tweets.

- For Irma, a total of 14,416,118 tweets were collected within 500 miles of a central point in Florida (lat, long: (29.6875, –82.4150)). We observed 11,038,342 (46.7 GB of) tweets were sent between 9:27:02pm EDT Sept. 9th and 7:59:59pm EDT Sept. 11th (about 2 days), and 3,377,776 tweets were sent between 6:25:24am and 7:59:59pm EDT Sept. 12th.

6.4 Social Media Usage in Disasters

Data collected from social media (in particular, Twitter) can be very informative about disaster-related issues as it has been widely used for asking and offering help, by government, volunteers, civilians, *etc.* This was observed anecdotally in several news articles soon after Harvey. We crawled the Tweets sent during and just after the hurricanes Harvey and Irma and see if they can potentially answer many useful questions such as *what*, *when* and *where* the need/offer for help occurs. Analyzing this data both qualitatively and quantitatively, we can get insights for the design of communication capabilities to complement traditional emergency service communication.

6.4.1 Keyword-Based Association of Tweets

We implemented an early-stage tweet processing algorithm that:

- parses large collections of raw crawled tweets, and
- identifies keywords and performs a phrase-based classification of tweets.

For the first phase, we use Java JSON Parser to extract those attributes of a tweet that we are most interested in, *i.e.,* mainly `createdAt` showing the time of the tweet, `text` showing the content of the tweet, and `geolocation` field of the tweet as a (latitude, longitude) pair. For the second phase, we use the Lucene [102] library, an open-source text mining engine to determine whether or not a tweet is

TABLE II

TWEET COUNTS PER CATEGORY FOR HARVEY ON AUG. 27TH BETWEEN 5:22:52 PM AND 6:59:59 PM.

Category	**Query phrase**	**#**
Total		561,187
Harvey-related	`harvey* hurricane*`	50,140
Deaths	`death* dead`	6,012
Shelter	`shelter*`	3,552
Damage	`damage*`	1,067
Search & Rescue	`(search)AND(rescue)`	852
Fire	`Fire`	736
Missing Persons	`Missing`	522
Collapsed Infrastructure	`collaps*`	876
Trapped	`Trapped`	382
Forward This Message	`(please this)AND(forward retweet)`	337
Outage	`((electricity power) AND (no out without outage* blackout*)) outage* blackout*`	301
Shortage	`shortage* suffic* insuffic* ((run* ran are)AND(short low out))`	248
Distribution	`distribut*`	181
Earthquake & Aftershocks	`aftershocks AND earthquake* aftershock*`	157
Need Medical Equipment & Supplies	`(need*)AND(medic* suppl*)`	146
Human Remains	`remains bodies`	114

associated to the disaster. We mine the `text` field of the tweets to get an understanding of what a tweet is about. Additionally, a dictionary construction program on the tweet pool gives us the frequency of each word and also the top k most frequent words, thus allowing us to learn what words are most popular in a tweet collection.

We analyze temporal and spatial distribution of incident-related tweets. For the tweets we crawled from the approximate one-week duration of Harvey, we identified Harvey-related tweets by counting the results of the `harvey* hurricane*` query (similarly for Irma, with the keyword `irma*`). We used the `createdAt` and `geoLocation` fields to plot the temporal and spatial distribution of the hurricane-related tweets. In this section we focus on the temporal analysis and leave the spatial analysis to Section 6.5.

6.3 shows the percentage of the Harvey-related tweets during the crawling periods on Twitter, showing how the ratio of tweets related to Harvey was higher during the peak of the incident. Aug. 25th was the day Harvey made landfall while Aug. 27th was the day of considerable flooding. Figure 37 shows the percentage of Irma-related tweets for continuous 3-hour periods between the night of Sept. 9 and the evening of Sept. 11. We observe that Irma-related tweets tracked the progress of the hurricane. It is interesting to note the correlation between these results and real events: According to [103], "... Irma was upgraded to a Category 4 ..." on Sept. 10 and "... downgraded to a Category 1 ..." on Sept. 11. As for Florida (where most crawled tweets are from), "Hurricane Irma pummeled the Florida Keys late Saturday (Sept. 9) into Sunday (Sept. 10) as a Category 4 and hit the Florida mainland as a Category 3 storm around 1pm eastern time Sunday ..." [104]. While the frequency of tweets on a topic may rely on many different factors, we observed that Harvey-related tweets are more frequent during the peak of the hurricane.

TABLE III

CATEGORIES FOR IRMA TWEETS DURING SEPT. 9TH - SEPT. 12TH.

Period start	9/09 9:27pm		9/10 9:00am		9/10 9:00pm		9/11 9:00am		9/12 6:25pm		9/12 9:00am	
Period end	**9/10 8:59pm**		**9/10 8:59pm**		**9/11 8:59am**		**9/11 7:59pm**		**9/12 8:59am**		**9/12 7:59pm**	
Category	**#**	**per Hour**	**#**	**per Hour**	**#**	**per Hour**	**#**	**per Hour**	**#**	**per Hour**	**#**	**per Hour**
Total	1939504	167922.42	3640540	303378.33	2350414	195867.83	3107882	282534.73	381807	147987.21	2960088	269098.91
Irma-related	214245	18549.35	464809	38734.08	252559	21046.58	217026	19729.64	15616	6052.71	88055	8005
Outage	6721	581.90	15031	1252.58	13937	1161.42	21097	1917.91	2439	945.35	13933	1266.64
Deaths	6407	554.72	10858	904.83	11817	984.75	19357	1759.73	1879	728.29	15641	1421.91
Shelter	11931	1032.99	26958	2246.5	7384	615.33	7673	697.55	359	139.15	2314	210.36
Damage	2188	189.44	8610	717.5	8017	668.08	15356	1396	1268	491.47	8287	753.36
Looting	200	17.32	10383	865.25	13782	1148.5	12172	1106.55	434	168.22	2142	194.73
Fire	3068	265.63	6377	531.42	4700	391.67	5251	477.36	678	262.79	6605	600.45
Forward This Message	186	16.10	489	40.75	307	25.58	403	36.64	320	124.03	15919	1447.18
Missing Persons	2022	175.06	2986	248.83	1851	154.25	3155	286.82	290	112.40	3086	280.55
Shortage	1363	118.01	2375	197.92	1116	93	2171	197.36	238	92.25	2169	197.18
Human Remains	2031	175.84	1997	166.42	1446	120.5	1838	167.09	190	73.64	1349	122.64
Collapsed Infrastructure	211	18.27	3784	315.33	705	58.75	752	68.36	28	10.85	547	49.73
Earthquake & Aftershocks	803	69.52	1745	145.42	481	40.08	430	39.09	63	24.42	384	34.91
Distribution	750	64.94	854	71.17	283	23.58	456	41.45	65	25.19	609	55.36
Trapped	230	19.91	473	39.42	391	32.58	319	29	40	15.50	273	24.82
Need Medical Equip.&Supplies	193	16.71	383	31.92	150	12.5	310	28.18	53	20.54	434	39.45
Search & Rescue	77	6.67	89	7.42	253	21.08	627	57	43	16.67	146	13.27
Road Blocked	16	1.39	127	10.58	141	11.75	345	31.36	93	36.05	177	16.09
Contaminated Water	122	10.56	157	13.08	151	12.58	215	19.55	6	2.33	136	12.36
Unstable	42	3.64	78	6.5	48	4	88	8	17	6.59	88	8
Rubble	26	2.25	62	5.17	38	3.17	66	6	6	2.33	39	3.55
Medical Emergency	22	1.90	36	3	18	1.5	99	9	4	1.55	51	4.64
Water Sanitation&Hygiene	7	0.61	50	4.17	18	1.5	92	8.36	2	0.78	36	3.27
Security Concern	5	0.43	20	1.67	23	1.92	25	2.27	5	1.94	12	1.09

6.4.2 Categorizing Tweets for Disaster Management

Once disaster-related tweets are identified, we classify the tweets according to what the tweeter is requesting/offering regarding the disaster, *e.g.,* requesting or offering aid, volunteering, reporting, or complaining. We identified a set of disaster-related categories and show their frequency in Table II and Table III for Harvey and Irma, respectively. We picked the query phrase associated with each category after some trial and error to get a reasonable accuracy rate.

For Harvey, the tweet count for each category is shown in Table II. We found the most frequent topics in Harvey tweets to be on "deaths", "shelter", and "damage". For Irma, we did a more comprehensive classification. Table III shows the tweet count for each disaster-related category for three days (Sept. 10, 11 and 12) in 12-hour intervals (starting from 9pm). We used the same category list and search keywords as we did for Harvey. As time values for different periods differ, we also show the *tweets per hour* which is the tweet count divided by the number of hours. This is very helpful for fair comparisons. Most of the trends observed in the table correlate with real progression of events. For example, It is interesting to note that the most frequent Irma tweet category was "outage" which was not a frequent category in Harvey. For "outage", we observe that the related tweets increase from Sept. 10 (1,252.58) to Sept. 11 (1,917.91) and decrease on Sept. 12 (1,266.64) during the daylight hours 9am–9pm (a similar pattern, with lower numbers, is seen during the night time, 9pm–9am). According to [105], power outage increased till Sept. 11, peaked, and then decreased after that. That seems to be similar to the numbers in our results. The aforementioned article also states ".. power outages peaked at 3pm on Sept. 11, affecting 64% of customers .." which also correlates with the results; as we see an increase-then-decrease pattern, peaking on Sept. 11. Looking at other categories, "Looting" reports go up first (immediately after event) and goes down afterwards – probably because of law enforcement. Likewise, "Shelter" requests go down over time. Similar trends can be observed for "Deaths" and

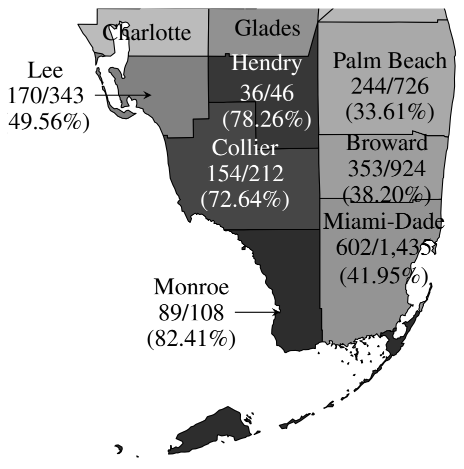


Figure 38. # of cell sites down *vs.* total (%age down) in the most affected southern counties of Florida on Sept. 11 (Darker implies a higher %age of sites down).

"Damage" categories. There may be anomalies too; *e.g.,* "Forwarding of message" goes strangely up on Sept. 12. This may be due to excessive retweets of one tweet.

6.5 Monitoring Communication Infrastructure

Knowing that people use social media for communicating disaster-related information, we seek to understand the relationship between the availability of traditional 911 service and cellular communication infrastructure, and the use of social media (Twitter) in the disaster areas. This can help answer questions like "*what is the potential of using social medial when 911 services are not running properly?*", and "*can social media help during a disaster?*"

6.5.1 Status of Civilian Communication Infrastructure

Focusing on Irma, we first look at the cell site outages in the counties hit by the hurricane. Figure 38 shows the outage-percentage by county on Sept. 11, the day after Irma made landfall. Since Irma made

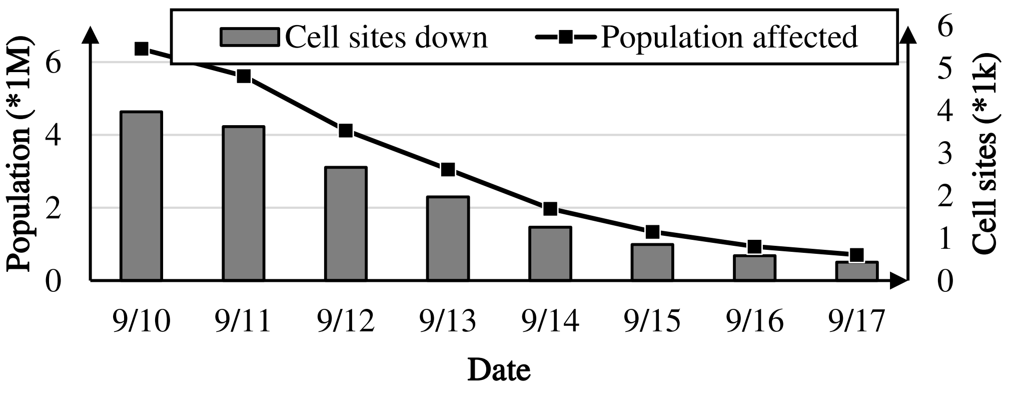


Figure 39. # of cell sites down *vs.* population affected in Florida during Hurricane Irma.

the first landfall at Cudjoe Key (southwest Monroe county) as a category 4 storm, and the second landfall on Marco Island (southwest Collier county) as a category 3 storm [103], we can see that the western side of the southern Florida (Monroe, Collier, Hendry and Lee counties) is affected more severely – up to 80% cell site outage.In comparison, the east side (Miami-Dade, Broward and Palm Beach counties) suffers less, *i.e.,* 30%–40% cell site outage.

To get an idea of the population affected,we correlated the FCC data with the county population data reported by World Population Review [106]. The population and # cell sites serving a county appear quite correlated, with $\approx$6 cell sites per 10K people according to our calculation. To get a rough estimation on the population affected in a county, we multiply the county's population with the percentage of cell sites down in that county. While this estimation may be over-simplified, assuming each person is served by exactly one cell site, we are limited by the granularity of the data reported. Even if this isn't the case, when a subset of cell sites are down, the impact is felt by users, possibly with lower throughput.

Figure 39 shows the # of cell sites down and the population affected during the period reported by FCC. We can see that in the first 2 days, around 1/3 cell sites are damaged by the hurricane (about 4K down out of a total of 14,730 cell sites serving Florida) and the population affected is over 5.5 million on those 2 days. Five days after (9/16) the hurricane made its first landfall, 95% of the cell sites were back to normal and the population affected reduced to 1.3 million, reflecting reasonably rapid failure recovery.

6.5.2 Status of PSAPs (911 Services)

We also look at the status of the PSAPs served in Florida during Irma. According to the FCC reports, each PSAP can be in one of the following 5 states:

- down (no service at all),
- reroute without Automatic Location Information (ALI),
- up but without ALI,
- reroute with ALI, and
- not affected.

We mainly focus on the first 3 states (either down or without ALI) and we categorize them as "abnormal". ALI is important in emergencies to help first responders provide help quickly, rather than depending on the caller to provide the exact location. Comparatively, "reroute w/ ALI" is less severe since all the functions of 911 are available, but there may be fewer answering positions, *i.e.,* operators answering the calls.

Since FCC only reports the PSAPs affected by Irma (the first 4 categories), we correlated the data with E911 plan in Florida from Florida Department of Management Services (DMS) [107]. In the

DMS documents, each county reports the detail of each PSAP served, including the location, the # of answering positions, total staff, *etc.* Table IV shows the total number and the affected number of PSAPs (and answer positions) in the 7 southern counties in Florida. Based on these, we make two observations. Firstly, there is a correspondence between the percentage of PSAPs affected and the percentage of the cellular infrastructure that failed. Counties with higher cell site outage (*i.e.,* Monroe, Collier, Hendry and Lee) also suffer from higher PSAP outage. This is understandable since the hurricane causes damage to both 911 and cellular service infrastructure and resources.

More interestingly, we observe that even in the counties with poor 911 service availability, there were still cell sites available. For example, in the first 3 days (reported in Table IV), the two PSAPs in Collier county are either down or w/o ALI. However, there are still 52–75 (24.5%–35.4%) cell sites available in the county. Similarly, in Lee county, we observe that around 95% of PSAPs were down or w/o ALI whilst 118–214 (34.4%–62.4%) cell sites were still functioning during that period. That means a proportion of citizens could still get access to social media (*e.g.,* to call for help) even when the 911 service is not functioning properly. With Device-to-Device (D2D) [108] and Disruption-Tolerant Networking (DTN) [109] techniques, we expect the coverage of the cell sites could be extended to even more people, enabling them to seek and possibly receive help through social media.

6.5.3 Infrastructure Failure *vs.* Geo-Tagged Tweets

Availability of geo-location by the smartphones and their social media use could be a significant help when civilians call for help – especially when 911 service or ALI in the county is not functioning. We inspected the `geoLocation` field of the Irma tweets and tried to observe if geo-tagged tweets can be useful in disaster management. A small percentage of tweets have `geoLocation` in them, and our

TABLE IV

STATUS OF PUBLIC SAFETY ANSWERING POINTS (PSAPS).

(D: down, U: up w/o ALI, R: reroute w/o ALI, A: reroute w/ ALI,

Abnormal %: % of answer positions down or w/o ALI)

		PSAPs (Answer Positions)					Abnor-	Cell sites
Date	County	Total	D	U	R	A	mal(%)	down (%)
9/10	Monroe	3 (11)	2 (7)				63.64	87 (80.56)
	Collier	2 (39)	2 (39)				100.00	160 (75.47)
	Hendry	4 (8)	2 (3)		1 (2)		62.50	31 (67.39)
	Lee	5 (41)	2 (15)	1 (14)	1 (2)		75.61	186 (54.23)
	Miami-Dade	7 (212)				1 (19)	0.00	739 (51.50)
	Broward	6 (126)					0.00	443 (47.94)
	Palm Beach	19 (142)				2 (13)	0.00	311 (42.84)
9/11	Monroe	3 (11)	2 (7)				63.64	89 (82.41)
	Collier	2 (39)		1 (33)	1 (6)		100.00	154 (72.64)
	Hendry	4 (8)		3 (5)			62.50	36 (78.26)
	Lee	5 (41)		4 (39)		1 (2)	95.12	170 (49.56)
	Miami-Dade	7 (212)				1 (19)	0.00	602 (41.95)
	Broward	6 (126)				1 (18)	0.00	353 (38.20)
	Palm Beach	19 (142)				2 (13)	0.00	244 (33.61)
9/12	Monroe	3 (11)	1 (5)				45.45	89 (82.41)
	Collier	2 (39)		1 (33)	1 (6)		100.00	137 (64.62)
	Hendry	4 (8)			3 (5)		62.50	35 (76.09)
	Lee	5 (41)		4 (39)		1 (2)	95.12	129 (37.61)
	Miami-Dade	7 (212)				1 (19)	0.00	457 (31.85)
	Broward	6 (126)				1 (18)	0.00	254 (27.49)
	Palm Beach	19 (142)				2 (13)	0.00	178 (24.52)

geo-analysis is based on this small set of tweets. We hope to increase their proportion with better geo-location techniques in future.

Figure 40 shows the origin locations of 7,806 geo-tagged Irma tweets. The size of the circles is proportional to the number of tweets from that exact coordinate. The figure shows in which areas the density of hurricane-related tweets is higher. The map in The map for Harvey looks similar. For Irma, there are several locations of different densities, showing the generation of tweets as Irma progressed north through the Florida panhandle relatively quickly.

According to the map in Figure 40, out of 7,806 geo-tagged tweets, 2,370 (30.4%) are in the 7 southern counties. To compensate for the difference in the population and the period we crawled the Twitter feed to get our dataset, we normalize the tweet count as the number of tweets per million people per hour. The exact number of tweets and the normalized value for the 7 southern counties are in Table V. The table shows that people tweeted more on the first days of the disaster. Monroe county had 17.92 tweets per million people per hour on Sept. 10 even with 80% cell sites down. Collier and Lee counties also had a significant value for the normalized tweet count. These three counties were the most affected, and did not have 911 services functioning properly during that period. Finally, there is a significant reduction of geo-tagged tweets after Sept. 11, as the hurricane had moved on. This is indication that the geo-tagged social media data can be an important tool in disaster management and recovery.

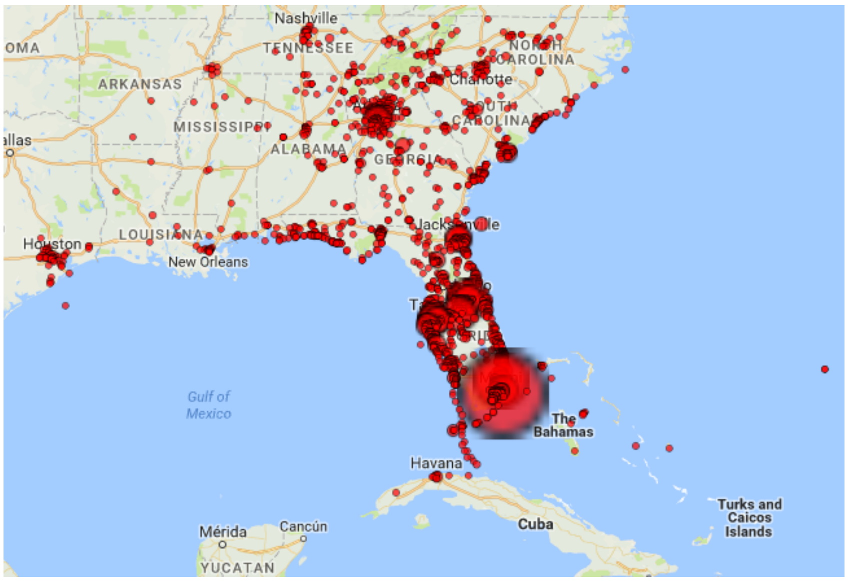


Figure 40. Geo-distribution of Irma-related tweets (circle size: # of tweets at the same location).

TABLE V

TWEET FREQUENCY IN DIFFERENT COUNTIES DURING HURRICANE IRMA.

(Normalized Tweet #: # of Tweets per million people per hour)

County	Population	# of Tweets (Normalized)			
		Sept. 9	Sept. 10	Sept. 11	Sept. 12
Monroe	79,077	3 (14.88)	34 (17.92)	10 (6.32)	7 (6.52)
Collier	365,136	2 (2.15)	58 (6.62)	13 (1.78)	11 (2.22)
Hendry	39,290	0 (0.00)	1 (1.06)	0 (0.00)	0 (0.00)
Lee	722,336	4 (2.17)	71 (4.10)	29 (2.01)	17 (1.73)
Miami-Dade	2,712,945	79 (11.42)	635 (9.75)	294 (5.42)	190 (5.16)
Broward	1,909,632	30 (6.16)	348 (7.59)	143 (3.74)	76 (2.93)
Palm Beach	1,443,810	15 (4.08)	176 (5.08)	88 (3.05)	36 (1.84)

CHAPTER 7

DYNAMIC HETEROGENEITY-AWARE CODED COOPERATIVE COMPUTATION AT THE EDGE

The contents of this chapters are based on our work that is published in the proceedings of 2018 IEEE ICNP conference [7]. ©2019 IEEE. Reprinted, with permission, from [7].

Cooperative computation is a promising approach for localized data processing at the edge, *e.g.,* for Internet of Things (IoT). Cooperative computation advocates that computationally intensive tasks in a device could be divided into sub-tasks, and offloaded to other devices or servers in close proximity. However, exploiting the potential of cooperative computation is challenging mainly due to the heterogeneous and time-varying nature of edge devices. Coded computation, which advocates mixing data in sub-tasks by employing erasure codes and offloading these sub-tasks to other devices for computation, is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this chapter, we develop a coded cooperative computation framework, which we name Coded Cooperative Computation Protocol (`C3P`), by taking into account the heterogeneous resources of edge devices. `C3P` dynamically offloads coded sub-tasks to helpers and is adaptive to time-varying resources. We show that (i) task completion delay of `C3P` is very close to optimal coded cooperative computation solutions, (ii) the efficiency of `C3P` in terms of resource utilization is higher than 99%, and (iii) `C3P` improves task completion delay significantly as compared to baselines via both simulations and in a testbed consisting of real Android-based smartphones.

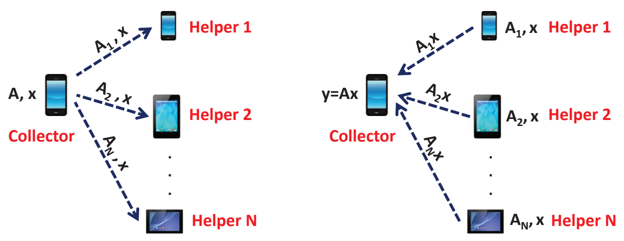
7.1 Background

Data processing is crucial for many applications at the edge including Internet of Things (IoT), but it could be computationally intensive and not doable if devices operate individually. One of the promising solutions to handle computationally intensive tasks is computation offloading, which advocates offloading tasks to remote servers or cloud. Yet, offloading tasks to remote servers or cloud could be luxury that cannot be afforded by most of the edge applications, where connectivity to remote servers can be lost or compromised, which makes localized processing crucial.

Cooperative computation is a promising approach for edge computing, where computationally intensive tasks in a device (collector device) could be offloaded to other devices (helpers) in close proximity as illustrated in Figure 41.

These devices could be other IoT or mobile devices, local servers, or fog at the edge of the network [74], [75]. However, exploiting the potential of cooperative computation is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Indeed, these devices may have different and time-varying computing power and energy resources, and could be mobile. Thus, our goal is to develop a dynamic, adaptive, and heterogeneity-aware cooperative computation framework by taking into account the heterogeneity and time-varying nature of devices at the edge.

We focus on the computation of linear functions. In particular, we assume that the collector's data is represented by a large matrix A and it wishes to compute the product $\mathbf{y} = A\mathbf{x}$, for a given vector $\mathbf{x}$, as seen in Figure 41. In fact, matrix multiplication forms the atomic function computed over many iterations of several signal processing, machine learning, and optimization algorithms, such as gradient descent based algorithms, classification algorithms, etc. [110–113].



(a) Offloading sub-tasks from a collector to helpers

(b) Helpers send computed sub-tasks back to the collector

Figure 41. Cooperative computation to compute $\mathbf{y} = A\mathbf{x}$. (a) Matrix A is divided into sub-matrices $A_1, A_2, ..., A_N$. Each sub-matrix along with the vector $\mathbf{x}$ is transmitted from the collector to one of the helpers. (b) Each helper computes the multiplication of its received sub-matrix with vector $\mathbf{x}$ and sends the computed value back to the collector.

In cooperative computation setup, matrix A is divided into sub-matrices $A_1, A_2, ..., A_N$ and each sub-matrix along with the vector $\mathbf{x}$ is transmitted from the collector to one of the helpers, as shown in Figure 41(a). Helper n computes $A_n\mathbf{x}$, and transmits the computed result back to the collector, illustrated in Figure 41(b), who can process all returned computations to obtain the result of its original task; *i.e.,* the calculation of $\mathbf{y} = A\mathbf{x}$.

Coding in computation systems is recently gaining interest in large scale computing environments, and it advocates higher reliability and smaller delay [110]. In particular, coded computation (*e.g.,* by employing erasure codes) mixes data in sub-tasks and offloads these coded sub-tasks for computation, which improves delay and reliability. The following canonical example inspired from [110] demonstrates the effectiveness of coded computation.

Example 3. *Let us consider that a collector device would like to calculate* $\mathbf{y} = A\mathbf{x}$ *with the help of three helper devices (helper* 1*, helper* 2*, and helper* 3*), where the number of rows in* A *is* 6*. Let us assume that each helper has a different runtime; helper* 1 *computes each row in* 1 *unit time, while the second and the third helpers require* 2 *and* 10 *units of time for computing one row, respectively. Assuming that these runtimes are random and not known a priori, one may divide* A *to three sub-matrices;* A_1*,* A_2*, and* A_3*; each with* 2 *rows. Thus, the completion time of these sub-matrices becomes* 2*,* 4*, and* 20 *at helpers* 1*,* 2*, and* 3*, respectively. Since the collector should receive all the calculated sub-matrices to compute its original task;* i.e., $\mathbf{y} = A\mathbf{x}$*, the total task completion delay becomes* $\max(2, 4, 20) = 20$.

As seen, helper 3 *becomes a bottleneck in this scenario, which can be addressed using coding. In particular,* A *could be divided into two sub-matrices* A_1 *and* A_2*; each with* 3 *rows. Then,* A_1 *and* A_2 *could be offloaded to helpers* 1 *and* 2*, and* $A_1 + A_2$ *could be offloaded to helper* 3*. In this setup, runtimes become* 3*,* 6*, and* 30 *at helpers* 1*,* 2*, and* 3*, respectively. However, since the collector requires reply from only two helpers to compute* $\mathbf{y} = A\mathbf{x}$ *thanks to coding, the total task completion delay becomes* $\max(3, 6) = 6$*. As seen, the task completion delay reduces to* 6 *from* 20 *with the help of coding.* $\square$

The above example demonstrates the benefit of coding for cooperative computation. However, offloading sub-tasks with equal sizes to all helpers, without considering their heterogeneous resources is inefficient. Let us consider the same setup in Example 3. If A_1 with 4 rows and A_2 with 2 rows are offloaded to helper 1 and helper 2, respectively, and helper 3 is not used, the task completion delay becomes $\max(4, 4) = 4$, which is the smallest possible delay in this example. Furthermore, the resources of helper 3 are not wasted, which is another advantage of taking into account the heterogeneity as compared to the above example. As seen, it is crucial to divide and offload matrix A to helpers by taking into account the heterogeneity of resources.

Indeed, a code design mechanism under such a heterogeneous setup is developed in [114], where matrix A is divided, coded, and offloaded to helpers by taking into account heterogeneity of resources. However, available resources at helpers are generally not known by the collector a priori and may vary over time, which is not taken into account in [114]. For example, the runtime of helper 1 in Example 3 may increase from 1 to 20 while computing (*e.g.,* it may start running another computationally intensive task), which would increase the total task completion delay. Thus, it is crucial to design a coded cooperation framework, which is dynamic and adaptive to heterogeneous and time-varying resources, which is the goal of this chapter.

In this chapter, we design a coded cooperative computation framework for edge computing. In particular, we design a Coded Cooperative Computation Protocol (`C3P`), which packetizes rows of matrix A into packets, codes these packets using Fountain codes, and determines how many coded packets each helper should compute dynamically over time. We provide theoretical analysis of `C3P`'s task completion delay and efficiency, and evaluate its performance via simulations as well as in a testbed consisting of real Android-based smartphones as compared to baselines.

The structure of the rest of this chapter is as follows. Section 7.2 presents the coded cooperative computation problem formulation. Section 7.3 presents the ergodic and static solutions to coded cooperative computation problem and the design of C3P. Section 7.4 provides the performance analysis of C3P. Section 7.5 presents the performance evaluation of C3P. Section 7.6 presents related work.

7.2 Problem Formulation

Setup. We consider a setup shown in Figure 41, where the collector device offloads its task to helpers in the set $\mathcal{N}$ (where $N = |\mathcal{N}|$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. In this setup, all devices could potentially be mobile, so the encounter time of the collector with helpers varies over time. *I.e.,* the collector can connect to less than N helpers at a time.

Application. As we described in Section 7.1, we focus on computation of linear functions; *i.e.,* the collector wishes to compute $\mathbf{y} = A\mathbf{x}$ where $A = (a_{i,j}) \in \mathbb{R}^{R\times R}$, and $\mathbf{x} = (x_{i,j}) \in \mathbb{R}^{R\times 1}$. Our goal is to determine sub-matrix $A_n = (a_{i,j}) \in \mathbb{R}^{r_n\times R}$ that will be offloaded to helper n, where r_n is an integer.

Coding Approach. We use Fountain codes [115], [116], which are ideal in our dynamic coded cooperation framework thanks to their rateless property, low encoding and decoding complexity, and low overhead. In particular, the encoding and decoding complexity of Fountain codes could be as low as $O(R\log(R))$ for LT codes and $O(R)$ for Raptor codes and the coding overhead could be as low as 5% [117]. We note that Fountain codes perform better than (i) repetition codes thanks to randomization of sub-tasks by mixing them, (ii) maximum distance separable (MDS) codes as MDS codes require a priori task allocation (due to their block coding nature) and are not suitable for the dynamic and adaptive framework that we would like to develop, and (iii) network coding as the decoding complexity of network coding is too high [118], which introduces too much computation overhead at the collector which obsoletes the computation offloading benefit.

Packetization. In particular, we packetize each row of A into a packet and create R packets; $\Gamma = \{\rho_1, \rho_2, \dots, \rho_R\}$. These packets are used to create Fountain coded packets, where ν_i is the ith coded packet. The coded packet ν_i is transmitted to a helper, where the helper computes the multiplication of $\nu_i \mathbf{x}$ and sends the result back to the collector. $R + K$ coded computed packets are required at the collector to decode the coded packets, where K is the coding overhead. Let $p_{n,i}$ be the jth coded packet generated by the collector and the ith coded packet transmitted to helper n; $p_{n,i} = \nu_j, j \geq i$.

Delay Model. Each transmitted packet $p_{n,i}$ experiences transmission delay between the collector and helper n as well as computing delay at helper n. Also, the computed packet $p_{n,i}\mathbf{x}$ experiences transmission delay while transmitted from helper n to the collector. The average round trip time (RTT) of sending a packet to helper n and receiving the computed packet, is characterized as RTT_n^{data}. The runtime of packet $p_{n,i}$ at helper n is a random variable denoted by $\beta_{n,i}$.[1] Assuming that r_n packets are offloaded to helper n, the total task completion delay for helper n to receive r_n coded packets, compute them, and send the results back to the collector becomes D_n, which is expressed as $D_n = RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i}$. Note that RTT_n^{data} in this formulation is due to transmitting the first packet $p_{n,1}$ and receiving the last computed packet $p_{n,r_n}\mathbf{x}$. The other packets can be transmitted while helpers are busy with processing packets; it is why we do not sum RTT_n^{data} across packets.

---

[1]Our framework is compatible with any delay distribution, but for the sake of characterizing the efficiency of our algorithm, and simulating its task completion delay, we use shifted exponential distribution in Sections 7.4.4 and 7.5.

Problem Formulation. Our goal is to determine the task offloading set $\mathcal{R} = \{r_1, \ldots, r_N\}$ that minimizes the total task completion delay, *i.e.,* we would like to dynamically determine $\mathcal{R}$ that solves the following optimization problem:

$$\begin{aligned} &\min_{\mathcal{R}} \max_{n \in \mathcal{N}} D_n \\ &\text{subject to} \sum_{n=1}^{N} r_n = R, r_n \in \mathbb{N}, \forall n \in \mathcal{N}. \end{aligned} \tag{7.1}$$

The objective of the optimization problem in Equation 7.1 is to minimize the maximum of per helper task completion delays, which is equal to $\max_{n \in \mathcal{N}} D_n$, as helpers compute their tasks in parallel. The constraint in Equation 7.1 is a task conservation constraint that guarantees that resources of helpers are not wasted, *i.e.,* the sum of the received computed tasks from all helpers is equal to the number of rows of matrix A. Note that this constraint is possible thanks to coding.[1] As we mentioned earlier, $R + K$ coded computed packets are required at the collector to decode the coded packets when we use Fountain codes. The constraint in Equation 7.1 guarantees this requirement in an idealized scenario assuming that $K = 0$. The constraint $r_n \in \mathbb{N}$ makes sure that the number of tasks r_n is an integer. The solution of Equation 7.1 is challenging as (i) $D_n = RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i}$ is a random variable and not known a priori, and (ii) it is an integer programming problem.

[1]We note that the optimal computation offloading problem, when coding is not employed, is formulated as $\min_{\Gamma_n} \max_{n \in \mathcal{N}} (RTT_n^{\text{data}} + \sum_{i=1}^{|\Gamma_n|} \beta_{n,i})$ subject to $\cup_{n=1}^{N} \Gamma_n = \Gamma$ where $\Gamma_n \subset \Gamma$ is the set of packets offloaded to helper n. As seen, the optimization problem in Equation 7.1 is more tractable as compared to this problem thanks to employing Fountain codes.

7.3 Problem Solution & C3P Design

In this section, we investigate the solution of Equation 7.1 for non-ergodic, static, and dynamic setups.

7.3.1 Non-Ergodic Solution

Let us assume that the solution of Equation 7.1 is

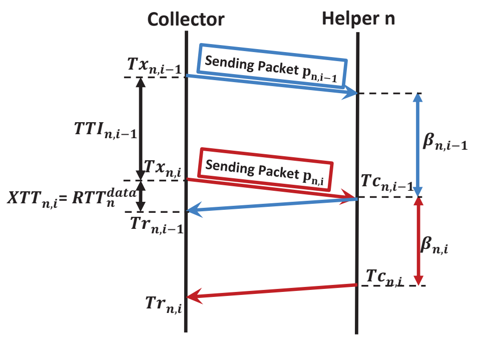
$$T^{\text{best}} = \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i} \Big), \tag{7.2}$$

where $r_n^{\text{best}} = \operatorname{argmin}_{r_n \in \mathbb{N}} \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n} \beta_{n,i} \Big)$. We note that Equation 7.2 is a non-ergodic solution as it requires the perfect knowledge of $\beta_{n,i}$ a priori. Although we do not have a compact solution of T^{best}, the solution in Equation 7.2 will behave as a performance benchmark for our dynamic and adaptive coded cooperative computation framework in Section 7.4.1.

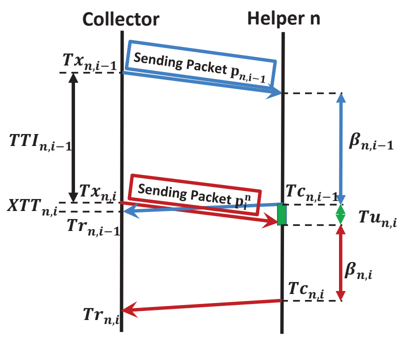
7.3.2 Static Solution

We assume that RTT_n^{data} becomes negligible as compared to $\sum_{i=1}^{r_n} \beta_{n,i}$. This assumption holds in practical scenarios with large R, and/or when transmission delay is smaller than processing delay. Then, D_n can be approximated as $\sum_{i=1}^{r_n} \beta_{n,i}$, and the optimization problem in Equation 7.1 becomes

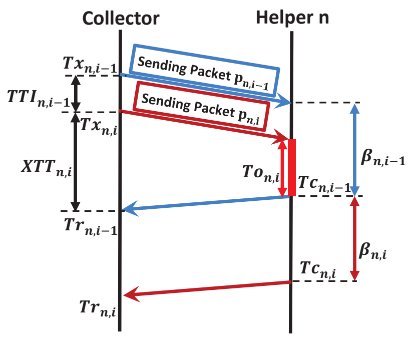
$$\begin{aligned} &\min_{\mathcal{R}} \max_{n \in \mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i} \\ &\text{subject to} \sum_{n=1}^{N} r_n = R, r_n \in \mathbb{N}, \forall n \in \mathcal{N}. \end{aligned} \tag{7.3}$$



(a) Ideal case



(b) Underutilized case



(c) Congested case

Figure 42. Different states of the system: (a) ideal case, (b) underutilized case, and (c) congested case.

As a static solution, we solve the expected value of the objective function in Equation 7.3 by relaxing the integer constraint, *i.e.*, $r_n \in \mathbb{N}$. The expected value of the objective function of Equation 7.3 is expressed as $E[\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i}]$, which is greater than or equal to $\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} E[\beta_{n,i}] = \max_{n\in\mathcal{N}} r_n E[\beta_{n,i}]$ (noting that $\max(.)$ is a convex function, so $E[\max(.)] \geq \max(E[.])$), where expectation is across the packets. Assuming that the average task completion delay is $T = E[\max_{n\in\mathcal{N}} \sum_{i=1}^{r_n} \beta_{n,i}] \geq \max_{n\in\mathcal{N}} r_n\ E[\beta_{n,i}]$, Equation 7.3 is converted to

$$\begin{aligned} &\min_{\mathcal{R}} \quad T \\ &\text{subject to } r_n E[\beta_{n,i}] \leq T, \forall n \in \mathcal{N} \\ &\sum_{n=1}^{N} r_n = R. \end{aligned} \tag{7.4}$$

We solve Equation 7.4 using Lagrange relaxation (we omit the steps of the solution as it is straightforward); the optimal task offloading policy becomes

$$r_n^{\text{static}} = \frac{R}{E[\beta_{n,i}] \sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}, \tag{7.5}$$

and the optimal task completion delay becomes $T^{\text{static}} = \frac{R}{\sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}$. Although the solution in Equation 7.5 is an optimal solution of Equation 7.4, the algorithm that offloads r_n^{static} sub-tasks to helper n a priori (static allocation) loses optimality as it is not adaptive to the time-varying nature of resources (*i.e.,* $\beta_{n,i}$). Next, we introduce our Coded Cooperative Computation Protocol (C3P) that is dynamic and adaptive to time-varying resources and approaches to the optimal solution in Equation 7.5 with increasing R.

7.3.3 Dynamic Solution: C3P

We consider the system setup in Figure 41, where the collector connects to N helpers. In this setup, the collector device offloads coded packets gradually to helpers, and receives two ACKs for each packet;

one confirming the receipt of the packet by the helper, and the second one (piggybacked to the computed packet $p_{n,i}\mathbf{x}$) showing that the packet is computed by the helper. Inspired by ARQ mechanisms [119], the collector transmits more/less coded packets based on the frequency of the received ACKs.

In particular, we define the transmission time interval $TTI_{n,i}$ as the time interval between sending two consecutive packets, $p_{n,i}$ and $p_{n,i+1}$, to helper n by the collector. The goal of our mechanism is to determine the best $TTI_{n,i}$ that reduces the task completion delay and increases helper efficiency (*i.e.,* exploiting the full potential of the helpers while not overloading them).

$TTI_{n,i}$ in an ideal scenario. Let $Tx_{n,i}$ be the time that $p_{n,i}$ is transmitted from the collector to helper n, $Tc_{n,i}$ be the time that helper n finishes computing $p_{n,i}$, and $Tr_{n,i}$ be the time that the computed packet (*i.e.,* by abusing the notation $p_{n,i}\mathbf{x}$) is received by the collector from helper n. We assume that the time of transmitting the first packet to each helper, *i.e.,* $p_{n,1}, \forall n \in \mathcal{N}$, is zero; *i.e.,* $Tx_{n,1} = 0, \forall n \in \mathcal{N}$.

Let us first consider the ideal scenario, Figure 42(a), where $TTI_{n,i}$ is equal to $\beta_{n,i}$ for all packets that are transmitted to helper n. Indeed, if $TTI_{n,i} > \beta_{n,i}$, Figure 42(b), helper n stays idle, which reduces the efficient utilization of resources and increases the task completion delay. On the other hand, if $TTI_{n,i} < \beta_{n,i}$, Figure 42(c), packets are queued at helper n. This congested (overloaded) scenario is not ideal, because the collector can receive enough number of packets before all queued packets in helpers are processed, which wastes resources.

Determining $TTI_{n,i}$ in practice. Now that we know that $TTI_{n,i} = \beta_{n,i}$ should be satisfied for the best system efficiency and smallest task completion delay, the collector can set $TTI_{n,i}$ to $\beta_{n,i}$. However, the collector does not know $\beta_{n,i}$ a priori as it is the computation runtime of packet $p_{n,i}$ at helper n. Thus, we should determine $TTI_{n,i}$ without explicit knowledge of $\beta_{n,i}$.

Our approach in `C3P` is to estimate $\beta_{n,i}$ as $E[\beta_{n,i}]$, where expectation is taken over packets. We will explain how to calculate $E[\beta_{n,i}]$ later in this section, but before that let us explain how to use estimated $E[\beta_{n,i}]$ for setting $TTI_{n,i}$. It is obvious that if the computed packet $p_{n,i}\mathbf{x}$ is received at the collector before packet $p_{n,i+1}$ is transmitted from the collector to helper n, the helper will be idle until it receives packet $p_{n,i+1}$. Therefore, to better utilize resources at helper n, the collector should offload a new packet before or immediately after receiving the computed value of the previous packet, *i.e.,* $TTI_{n,i} \leq Tr_{n,i} - Tx_{n,i}$ should be satisfied as in Figure 42. Therefore, if the calculated $E[\beta_{n,i}]$ is larger than $Tr_{n,i} - Tx_{n,i}$, then we set $TTI_{n,i}$ as $Tr_{n,i} - Tx_{n,i}$ to satisfy this condition. In other words, $TTI_{n,i}$ is set to

$$TTI_{n,i} = \min(Tr_{n,i} - Tx_{n,i}, E[\beta_{n,i}]). \tag{7.6}$$

Calculation of $E[\beta_{n,i}]$. In `C3P`, $E[\beta_{n,i}]$ is estimated using runtimes of previous packets:

$$E[\beta_{n,i}] \approx \frac{\sum_{j=1}^{m_n} \beta_{n,i}}{m_n}, \tag{7.7}$$

where m_n is the number of computed packets received at the collector from helper n before sending packet $p_{n,i+1}$. In order to calculate Equation 7.7, the collector device should have $\beta_{n,i}$ values from the previous offloaded packets. A straightforward approach would be putting timestamps on sub-tasks to directly access the runtimes $\beta_{n,i}$ at the collector. However, this approach introduces overhead on sub-tasks. Thus, we also developed a mechanism, where the collector device infers $\beta_{n,i}$ by taking into account transmission and ACK times of sub-tasks.

C3P in a nutshell. The main goal of C3P is to determine packet transmission intervals, $TTI_{n,i}$, according to Equation 7.6, which is summarized in Algorithm 2. Note that Algorithm 2 has also a timeout value defined in line 8, which is needed for unresponsive helpers. If helper n is not responsive, $TTI_{n,i}$ is quickly increased as shown in line 6 so that fewer and fewer packets could be offloaded to that helper. In particular, C3P doubles $TTI_{n,i}$ when the timeout for receiving ACK occurs. This is inspired by additive increase multiplicative decrease strategy of TCP, where the number of transmitted packets are halved to backoff quickly when the system is not responding.

After $TTI_{n,i}$ is updated when a transmitted packet is ACKed or timeout occurs, this interval is used to determine the transmission times of the next coded packets. In particular, coded packets are generated and transmitted one by one to all helpers with intervals $TTI_{n,i}$ until (i) $TTI_{n,i}$ is updated with a new ACK packet or when timeout occurs, or (ii) the collector collects $R+K$ computed packets. Next, we characterize the performance of C3P.

7.4 Performance Analysis of C3P

7.4.1 Performance of C3P w.r.t. the Non-Ergodic Solution

In this section, we analyze the gap between C3P and the non-ergodic solution characterized in Section 7.3.1. Let us first characterize the task completion delay of C3P as

$$T^{\text{C3P}} = \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{C3P}}} (\beta_{n,i} + Tu_{n,i}) \Big), \tag{7.8}$$

where $r_n^{\text{C3P}} = \text{argmin}_{r_n} \max_{n \in \mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n} (\beta_{n,i} + Tu_{n,i}) \Big)$, and $Tu_{n,i}$ is per packet under-utilization time at helper n, which occurs as C3P does not have a priori knowledge of $\beta_{n,i}$, but it

Algorithm 2 `C3P` algorithm at the collector.

```
1: Initialize: $TO_n = \infty, \forall n \in \mathcal{N}$.
2: while $R + K$ calculated packets have not been received do
3:     if Calculated packet $p_{n,i}\mathbf{x}$ is received before timeout expires then
4:         Calculate $TTI_{n,i}$ according to Equation 7.7 and Equation 7.6.
5:     else
6:         $TTI_{n,i} = 2 \times TTI_{n,i}$.
7:     end if
8:     Update timeout as $TO_n = 2TTI_{n,i}$.
9: end while
```

estimates $\beta_{n,i}$ and accordingly determines packet transmission times $TTI_{n,i}$ according to Equation 7.6. The gap between $T^{\texttt{C3P}}$ and T^{best} in Equation 7.2 is upper bounded by:

$$\begin{aligned}
T^{\texttt{C3P}} - T^{\text{best}} &= \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\texttt{C3P}}} (\beta_{n,i} + Tu_{n,i})\Big) - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&\le \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} (\beta_{n,i} + Tu_{n,i})\Big) - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&\le \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) + \max_{n\in\mathcal{N}} \sum_{i=1}^{r_n^{\text{best}}} Tu_{n,i} - \max_{n\in\mathcal{N}} \Big(RTT_n^{\text{data}} + \sum_{i=1}^{r_n^{\text{best}}} \beta_{n,i}\Big) \\
&= \max_{n\in\mathcal{N}} \sum_{i=1}^{r_n^{\text{best}}} Tu_{n,i},
\end{aligned} \tag{7.9}$$

where the first inequality comes from $r_n^{\texttt{C3P}} = \text{argmin}_{r_n} \max_{n \in \mathcal{N}} \left(RTT_n^{\text{data}} + \sum_{i=1}^{r_n} (\beta_{n,i} + Tu_{n,i}) \right)$ and the second inequality comes from the fact that $\max(f(x) + g(x)) \leq (\max(f(x)) + \max(g(x)))$.[1] As seen, the gap between `C3P` and the non-ergodic solution is bounded with the sum of $Tu_{n,i}$. The next theorem characterizes $Tu_{n,i}$.

Theorem 5. *$Tu_{n,i}$ is monotonically decreasing with increasing number of sub-tasks, and* $\lim_{i \to \infty}$ $Pr(Tu_{n,i} > 0) \to 0$.

Proof: Let us first consider the following lemma that determines the conditions for having a positive $Tu_{n,i+1}$.

Lemma 6. *The necessary and sufficient conditions to satisfy $Tu_{n,i+1} > 0$ are*

$$\sum_{j=i+1-k}^{i} \beta_{n,j} < kE[\beta_{n,i}], \forall k = 1, 2, \ldots, i \tag{7.10}$$

□

According to the conditions given in Lemma 6, the probability of $Tu_{n,i} > 0$ is calculated as:

$$Pr(Tu_{n,i} > 0) = \int_0^{E[\beta_{n,i}]} \int_0^{2E[\beta_{n,i}]-x_i} \cdots \int_0^{iE[\beta_{n,i}]-\sum_{j=2}^{i} \beta_{n,j}} \tag{7.11}$$
$$f_{\beta_{n,1},\ldots,\beta_{n,i}}(x_1, \ldots, x_i) dx_1 \ldots dx_i,$$

[1]Note that in Equation 7.9, we assume that the runtime of packet i at helper n is the same in both the non-ergodic solution and `C3P`, which is necessary for fair comparison.

where $f_{\beta_{n,1},\ldots,\beta_{n,i}}(x_1,\ldots,x_i)$ is the joint probability density function of $(\beta_{n,1},\ldots,\beta_{n,i})$. With the assumption that $\beta_{n,j}, j = 1,2,...,i$ is from an i.i.d distribution, the joint probability distribution function of $\beta_{n,1},\ldots,\beta_{n,i}$ is the product of i probability distribution functions:

$$\begin{aligned}
Pr(Tu_{n,i} > 0) = &\int_0^{E[\beta_{n,i}]} \int_0^{2E[\beta_{n,i}]-x_i} \ldots \int_0^{iE[\beta_{n,i}]-\sum_{j=2}^{i} x_j} \\
&f_{\beta_{n,i}}(x_1)f_{\beta_{n,i}}(x_2)\ldots f_{\beta_{n,i}}(x_i)dx_1dx_2\ldots dx_i \qquad (7.12)\\
= &\int_0^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_i) \int_0^{2E[\beta_{n,i}]-x_i} f_{\beta_{n,i}}(x_{i-1}) \\
&\ldots \int_0^{(i-1)E[\beta_{n,i}]-\sum_{j=3}^{i} x_j} f_{\beta_{n,i}}(x_2) \\
&\int_0^{iE[\beta_{n,i}]-\sum_{j=2}^{i} x_j} f_{\beta_{n,i}}(x_1)dx_1dx_2\ldots dx_i \qquad (7.13)\\
< &\int_0^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_i) \int_0^{2E[\beta_{n,i}]-x_i} f_{\beta_{n,i}}(x_{i-1}) \\
&\ldots \int_0^{(i-1)E[\beta_{n,i}]-\sum_{j=3}^{i} x_j} f_{\beta_{n,i}}(x_2)dx_2\ldots dx_i \qquad (7.14)\\
= &\int_0^{E[\beta_{n,i}]} f_{\beta_{n,i}}(x_{i-1}) \\
&\int_0^{2E[\beta_{n,i}]-x_{i-1}} f_{\beta_{n,i}}(x_{i-2})\ldots \\
&\int_0^{(i-1)E[\beta_{n,i}]-\sum_{j=2}^{i-1} x_j} f_{\beta_{n,i}}(x_1)dx_1\ldots dx_{i-1}, \qquad (7.15)
\end{aligned}$$

where the last inequality comes from the fact that $\int_0^{iE[\beta_{n,i}]-\sum_{j=2}^{i} x_j} f_{\beta_{n,i}}(x_1)dx_1$ is less than 1, because the probability density function is integrated over a finite range of variable x_1, and the last equality comes from a change of variables in the integrals. Equation 7.15 is equal to $Pr(Tu_{n,i-1} > 0)$ and thus $Pr(Tu_{n,i} > 0) < Pr(Tu_{n,i-1} > 0)$. Similarly, we can show that:

$$Pr(Tu_{n,j} > 0) < Pr(Tu_{n,j-1} > 0), \forall j = 2, 3, \ldots, i \tag{7.16}$$

From the above equation, we can conclude that as i gets larger, $Pr(Tu_{n,i} > 0)$ gets smaller, and $\lim_{i\to\infty} Pr(Tu_{n,i} > 0) \to 0$ is satisfied. This concludes the proof. □

We can conclude from Theorem 5 that the rate of the increase in the gap between C3P and the non-ergodic solution decreases with increasing the number of sub-tasks and eventually the rate becomes zero for $R \to \infty$. Therefore, the gap becomes finite even for $R \to \infty$.

7.4.2 Performance of C3P w.r.t the Static Solution

In this section, we analyze the performance of C3P as compared to the static solution characterized in Section 7.3.2. The next theorem characterizes the task completion delay of C3P as well as the optimal task offloading policy.

Theorem 7. *The task completion delay of C3P approaches to*

$$T^{C3P} \approx \frac{R+K}{\sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}, \tag{7.17}$$

with increasing R and the number of offloaded tasks to helper n is approximated as

$$r_n^{C3P} \approx \frac{R+K}{E[\beta_{n,i}] \sum_{n=1}^{N} \frac{1}{E[\beta_{n,i}]}}. \tag{7.18}$$

$\square$

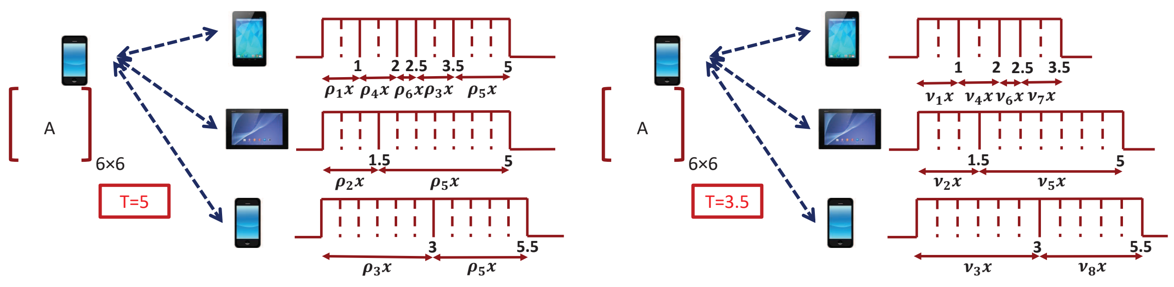
Theorem 7 shows that the task completion delay of C3P is getting close to the static solution T^{static} characterized in Section 7.3.2 with increasing R. The gap between T^{static} and $T^{\texttt{C3P}}$ is $\frac{K}{\sum_{n=1}^{N}\frac{1}{E[\beta_{n,i}]}}$ which is due to the coding overhead of Fountain codes, which becomes negligible for large R.

7.4.3 Performance of C3P w.r.t. Repetition Codes

In this section, we demonstrate the performance of C3P as compared to repetition coding with Round-robin (RR) scheduling through an illustrative example. Repetition codes with RR scheduling works as follows. Uncoded packets from the set $\Gamma = \{\rho_1, \rho_2, \dots, \rho_R\}$ is offloaded to helpers one by one (in round robin manner) depending on their sequence in Γ. For example, ρ_1 is offloaded to helper 1, ρ_2 is offloaded to helper 2, and so on. When all the packets are offloaded from Γ, we start again from the first packet in the set (so it is a repetition coding). Note that whenever a packet is computed and a corresponding ACK is received, the packet is removed from Γ. Thus, this RR scheduling continues until Γ becomes an empty set. We use $TTI_{n,i}$ in Equation 7.6 to determine the next scheduling time for helper n. The next example demonstrates the benefit of C3P as compared to this repetition coding mechanism with RR scheduling.

Example 4. *We consider the same setup in Example 3. We assume that per-packet runtimes are* $\beta_{1,1} = 1, \beta_{1,2} = 1, \beta_{1,3} = 0.5, \beta_{1,4} = 1, \beta_{1,5} = 1.5$*,* $\beta_{2,1} = 1.5, \beta_{2,2} = 3.5$*, and* $\beta_{3,1} = 3, \beta_{3,2} = 2.5$*, and the transmission times of packets are negligible.*

As seen in Figure 43(a), RR scheduler sends ρ_1*,* ρ_2*, and* ρ_3 *to helpers 1, 2, and 3, respectively at time* $t = 0$*. At time* $t = 1$*, the computed packet* $\rho_1\mathbf{x}$ *is received at the collector, and* ρ_4*, which is the next packet selected by RR scheduler, is transmitted to helper* 1*. Similarly, at time* $t = 1.5$*,* $\rho_2\mathbf{x}$ *is received*



(a) Repetition codes with RR scheduling

(b) `C3P`

Figure 43. Performance of `C3P` with respective to repetition codes with RR scheduling.

at the collector, and ρ_5 is transmitted to helper 2*. Similarly, the next packets are transmitted to helpers until the results for all packets are received at the collector, which is achieved at time $t = 5$. As seen, the resources of helper* 1 *is wasted while computing ρ_3, because those resources could have been used for computing a new packet.* `C3P` *addresses this problem thanks to employing Fountain codes.*

In particular, at time $t = 0$, three Fountain coded packets of ν_1, ν_2, ν_3 are created and transmitted to the three helpers, i.e., $p_{1,1} = \nu_1, p_{2,1} = \nu_2, p_{3,1} = \nu_3$*. At time $t = 1$, a new coded packet of ν_4 is created and transmitted as a second packet to helper* 1*,* i.e., $p_{1,2} = \nu_4$*. This continues until* 6 *computed coded packets (assuming that the overhead of Fountain codes,* i.e., *K is zero) are received at the collector, which is achieved at time $t = 3.5$.* □

Example 4 shows that the task completion delay is reduced from 5 to 3.5 when we use Fountain codes, which is significant. Section 7.5 shows extensive simulation results to support this illustrative example.

7.4.4 Efficiency of `C3P`

In this section, we characterize the efficiency of `C3P` in the worst case scenario when per task runtimes follow the shifted exponential distribution. We call it the worst case efficiency, because we take into account per packet under-utilization $Tu_{n,i}$ in efficiency calculation, but the fact that $Tu_{n,i}$ is monotonically decreasing, which is stated in Theorem 5, is not used.

Theorem 8. *Assume that the runtime of each packet,* i.e., $\beta_{n,i}$, *is a random variable according to an i.i.d shifted exponential distribution of*

$$F_{\beta_{n,i}}(t) = Pr(\beta_{n,i} < t) = 1 - e^{-\mu_n(t-a_n)}, \tag{7.19}$$

with mean $a_n + 1/\mu_n$ and shifted value of a_n. The expected value of the duration that helper n is underutilized per packet is characterized as:

$$E[Tu_{n,i}] = \begin{cases} \frac{1}{(e\mu_n)}\left(1 - e^{(\mu_n RTT_n^{\text{data}})}\right) + RTT_n^{\text{data}}, \text{if } RTT_n^{\text{data}} < \frac{1}{\mu_n} \\ \frac{1}{(e\mu_n)}, \quad \text{otherwise.} \end{cases} \tag{7.20}$$

□

We define the efficiency of helper n in the worst case as $\gamma_n = 1 - E[Tu_{n,i}]/E[\beta_{n,i}]$. Note that $E[Tu_{n,i}]$ is the expected time that helper n is underutilized per packet in the worst case, while

$E[\beta_{n,i}]$ is the expected runtime duration, *i.e.,* the expected time that helper n works per packet. Thus, $E[Tu_{n,i}]/E[\beta_{n,i}]$ becomes the under-utilization ratio of helper n in the worst case, so $\gamma_n = 1 - E[Tu_{n,i}]$ / $E[\beta_{n,i}]$ becomes the worst case efficiency. From Equation 7.20 and replacing $E[\beta_{n,i}]$ with $a_n + 1/\mu_n$, γ_n is expressed as the following:

$$\gamma_n = \begin{cases} \frac{1+a_n\mu_n-\mu_n RTT_n^{\text{data}}-1/e+\exp(\mu_n RTT_n^{\text{data}}-1)}{1+a_n\mu_n}, \text{if } RTT_n^{\text{data}} < 1/\mu_n \\ \frac{e(1+a_n\mu_n)-1}{e(1+a_n\mu_n)}, \quad \text{otherwise.} \end{cases} \tag{7.21}$$

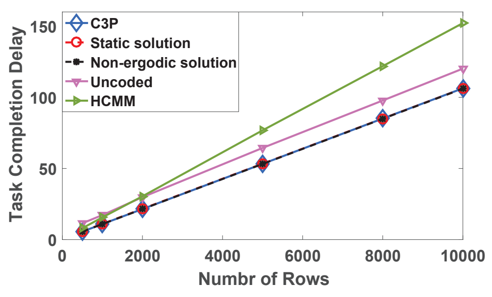
We show through simulations (in Section 7.5) that, (i) γ_n in Equation 7.21 is larger than 99%, which is significant as Equation 7.21 is the worst case efficiency, and (ii) C3P's efficiency is even larger than γ_n as γ_n in Equation 7.21 is the efficiency in the worst case, where the under-utilization time period has the maximum value.

7.5 Performance Evaluation of C3P

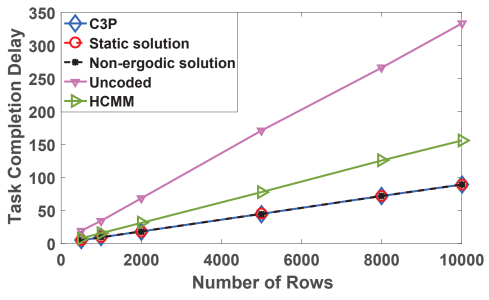
In this section, we evaluate the performance of our algorithm; Coded Cooperative Computation Protocol (C3P) via simulations and using real Android-based smartphones.

7.5.1 Simulation Results

We consider two scenarios: (i) Scenario 1, where the system resources for each helper vary over time. In this scenario, the runtime for computing each packet $p_{n,i}, \forall i$ at each helper n is an i.i.d. shifted exponential random variable with shifted value a_n and mean $a_n + 1/\mu_n$, and (ii) Scenario 2, where the runtime for computing packets in helper n does not change over time, *i.e.,* $\beta_{n,i} = \beta_n, \forall i$, and $\beta_n, \forall n \in \mathcal{N}$ is a shifted exponential random variable with shifted value a_n and mean $a_n + 1/\mu_n$.



(a) Scenario 1



(b) Scenario 2

Figure 44. Task completion delay vs. number of rows/packets for (i) Scenario 1, and (ii) Scenario 2, where the runtime for computing one row by helper n is selected from a shifted exponential distribution with $a_n = 0.5, \forall n \in \mathcal{N}$ and μ_n, which is selected uniformly from $\{1, 2, 4\}$.

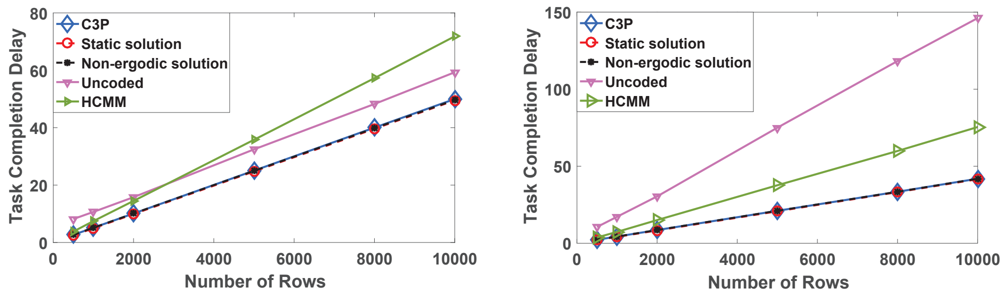
In our simulations, each simulated point is obtained by averaging over 200 iterations for $N = 100$ helpers. The transmission rate for sending each packet from the collector to each helper n and from helper n to the collector is a Poisson random variable with the average selected uniformly between 10 Mbps and 20 Mbps for each helper n. The size of a transmitted packet $p_{n,i}$ is set to $B_x = 8R$ bits, where R is the number of rows of matrix A, and it varies from 500 to $20,000$ in our simulations. The sizes of a computed packet $p_{n,i}\mathbf{x}$ and an acknowledgement packet are set to $B_r = 8$ bits and $B_{\text{ack}} = 1$ bit, respectively. These are the parameters that are used for creating all plots unless otherwise is stated.

Task Completion Delay vs. Number of Rows: We evaluate `C3P` for Scenarios 1 and 2 and compare its task completion delay with: (i) Static solution, which is the task completion delay characterized in Section 7.3.2 for both Scenarios 1 and 2. (ii) Non-ergodic solution, which is a realization of the non-ergodic problem characterized in Section 7.3.1 by knowing $\beta_{n,i}$ a priori at the collector and setting

$TTI_{n,i}$ as $\beta_{n,i}$. (iii) Uncoded: r_n packets without coding are assigned to each helper n, and the collector waits to receive computed values from all helpers. The number of assigned packets to each helper n is inversely proportional to the mean of $\beta_{n,i}$, *i.e.*, $r_n \propto \frac{1}{a_n+1/\mu_n}$. (iv) HCMM: Coded cooperative framework developed in [114] using block codes. We introduce 5% coding overhead for `C3P`, static, and non-ergodic solutions.

Figure 44(a) shows completion delay versus number of rows for Scenario 1, where the runtime for computing each packet by helper n, $\beta_{n,i}, \forall i$, is a shifted exponential random variable with shifted value of $a_n = 0.5$ and mean of $a_n + 1/\mu_n$, where μ_n is selected uniformly from $\{1, 2, 4\}$. As seen, `C3P` performs close to the static and non-ergodic solutions. This shows the effectiveness of our proposed algorithm. In addition, `C3P` performs better than the baselines. In particular, in average, 30% and 24% improvement is obtained by `C3P` over HCMM and no coding, respectively. Figure 44(b) considers the same setup but for Scenario 2, where the runtime for computing r_n packets by helper n is $r_n\beta_n$, where β_n is selected from a shifted exponential distribution with $a_n = 0.5, \forall n \in \mathcal{N}$ and μ_n, which is selected uniformly from $\{1, 2, 4\}$. As seen, for this scenario, `C3P` performs close to the static and non-ergodic solutions. `C3P` performs better than HCMM, and HCMM performs better than no coding. In particular, in average, 40% and 69% improvement is obtained by `C3P` over HCMM and no coding, respectively. Note that uncoded performs better than HCMM for Scenario 1, as HCMM is designed for Scenario 2, so it does not work well in Scenario 1. `C3P` performs well in both scenarios.

Figure 45 shows completion delay versus number of rows for both Scenarios 1 and 2, where the runtime for computing the rows by each helper n, is from a shifted exponential distribution with $\mu_n, n \in \mathcal{N}$ selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$ (different shifted values for different helpers). As



(a) Scenario 1 (b) Scenario 2

Figure 45. Task completion delay vs. number of rows/packets for (i) Scenario 1, and (ii) Scenario 2, where the runtime for computing one row by each helper n is selected from a shifted exponential distribution with μ_n, which is selected uniformly from $\{1, 3, 9\}$ for different helpers and $a_n = 1/\mu_n, \forall n \in \mathcal{N}$.

seen, `C3P` performs close to static and non-ergodic solutions and much better than the baselines. In particular, for Scenario 1, more than 30% and 15% improvement is obtained by `C3P` over HCMM and no coding, respectively. Also, for Scenario 2, in average, 42% and 73% improvement is obtained by `C3P` over HCMM and no coding, respectively.

Efficiency: We calculated the efficiency of helpers for different simulation setups and compared it with the theoretical efficiency obtained in Equation 7.21 for Scenario 1. For all simulation setups, the average efficiency over all helpers was around 99% and the theoretical efficiency was a little lower than the simulated efficiency. *E.g.*, for $R = 8000$ rows, where $\mu_n, n \in \mathcal{N}$ is selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$, the average of efficiency over all helpers is 99.7072% and the average of theoretical efficiency is 99.4115%. This is expected as the theoretical efficiency is calculated for the worst case scenario.

We also calculate the efficiency of helpers for Scenario 2. For all simulation setups, the average efficiency over all helpers was around 99%, *e.g.,* for $R = 8000$ rows, where $\mu_n, n \in \mathcal{N}$ is selected uniformly from $\{1, 3, 9\}$ and $a_n = 1/\mu_n$, the average of efficiency over all helpers was 99.9267%. Note that the theoretical efficiency for Scenario 1 is 100%. The simulated efficiency is lower than the theoretical one, because the simulation underutilizes the helpers when transmitting the very first packet to each helper, *i.e.,* before the collector estimates the resources of helpers.

C3P as Compared to Repetition Coding and Round Robin Scheduling: Figure 46 shows the percentage of improvement of C3P over repetition coding with RR scheduling in terms of task completion delay. The number of rows is selected as $R = 2000$ with 5% overhead for C3P and the number of helpers varies from $N = 100$ to $N = 600$. The transmission rate for sending each packet from the collector to each helper n and from helper n to the collector is a Poisson random variable with the average selected uniformly between 0.1 Mbps and 0.2 Mbps for each helper n. The other parameters are the same as the parameters used in Figure 44(a). As seen, by increasing the number of helpers, more improvement is gained by C3P compared to the repetition coding with RR scheduling.

7.5.2 Evaluation in a Testbed

We implemented a testbed of a collector and multiple helpers using real mobile devices, specifically Android 6.0.1 based Nexus 6P and Nexus 5 smartphones. All the helpers are connected to the collector device using Wi-Fi Direct connections. We conducted our experiments using our testbed in a lab

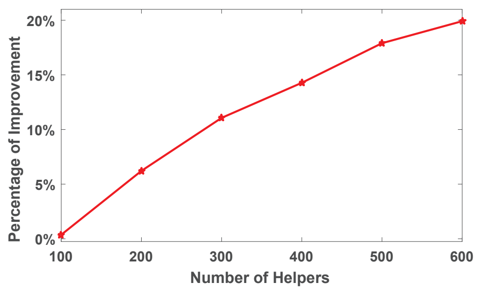


Figure 46. Percentage of improvement of `C3P` over repetition codes with RR scheduling in terms of the task completion delay.

environment where several other Wi-Fi networks were operating in the background. We located all the devices in close proximity of each other (within a few meters distance).

We implemented both `C3P` and repetition coding with RR scheduling in our testbed. The collector device would like to calculate matrix multiplication $\mathbf{y} = A\mathbf{x}$, where A is a 1K $\times$ 10K matrix and $\mathbf{x}$ is a 10K $\times$ 1 vector. Matrix A is divided into 20 sub-matrices, each of which is a 50 $\times$ 10K matrix. A sub-task to be processed by a helper is the multiplication of a sub-matrix with vector $\mathbf{x}$. There is one collector device (Nexus 5) and varying number of helpers (Nexus 6P).

Figure 47 shows task completion delay versus number of helpers for both `C3P` and repetition codes with RR scheduling. In this setup, each helper receives a sub-task, processes it, and waits for a random amount of time (exponential random variable with mean 10 seconds), which may arise due to other applications running at smartphones, and then sends the result back to the collector. As can be seen, the task completion delay reduces with increasing number of helpers in both algorithms. When there is one helper `C3P` performs worse, which is expected. In particular, `C3P` introduces coding overhead, and

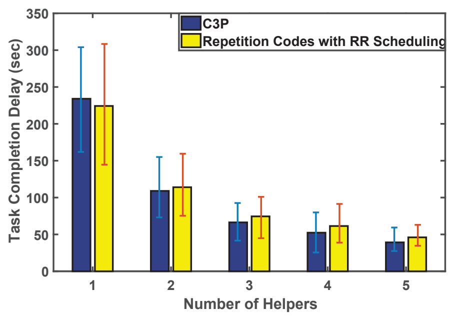


Figure 47. Task completion delay versus number of helpers.

the number of helpers is very small to see the benefit of coding. On the other hand, when the number of helpers increases, we start seeing the benefit of coding. For example, when the number of helpers is 5, C3P improves 14% over repetition codes with RR scheduling. This result confirms our simulation results in Figure 46 in a testbed with real Android-based smartphones.

Figure 48 shows the task completion delay versus per sub-task random delays at helpers. There are 5 helpers in this scenario. As can be seen, C3P improves more over repetition codes with RR scheduling when delay increases, as it increases heterogeneity, and C3P is designed to take into account heterogeneity.

7.6 Related Work

Mobile cloud computing is a rapidly growing field with the goal of providing extensive computational resources to mobile devices as well as higher quality of experience [120–122]. The initial approach to mobile cloud computing has been to offload resource intensive tasks to remote clouds by exploiting Internet connectivity of mobile devices. This approach has received a lot of attention which

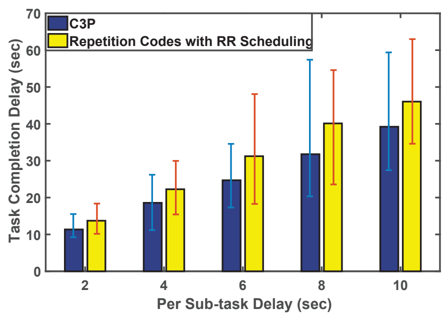


Figure 48. Task completion delay versus per sub-task delay.

led to extensive literature in the area [68, 123–126]. The feasibility of computation offloading to remote cloud by mobile devices [127] as well as energy efficient computation offloading [72, 73] has been considered in the previous work. As compared to this line of work, our focus is on edge computing rather than remote clouds.

There is an increasing interest in edge computing by exploiting connectivity among mobile devices [128]. This approach suggests that if devices in close proximity are capable of processing tasks cooperatively, then local area computation groups could be formed and exploited for computation. Indeed, cooperative computation mechanisms by exploiting device-to-device connections of mobile devices in close proximity are developed in [128] and [129]. A similar approach is considered in [130] with particular focus on load balancing across workers. As compared to this line of work, we consider coded cooperative computation.

Coded cooperative computation is shown to provide higher reliability, smaller delay, and reduced communication cost in MapReduce framework [131], where computationally intensive tasks are offloaded to distributed server clusters [132]. In [110] and [133], coded computation for matrix multiplication is considered, where matrix A is divided into sub-matrices and each sub-matrix is sent from the master node (called collector in our work) to one of the worker nodes (called helpers in our work) for matrix multiplication with the assumption that the helpers are homogeneous. In [110], workload of the worker nodes is optimized such that the overall runtime is minimized. Fountain codes are employed in [134] for coded computation, but for homogeneous resources. In [114], the same problem is considered, but with the assumption that workers are heterogeneous in terms of their resources. Compared to this line of work, we develop `C3P`, a practical algorithm that is (i) adaptive to the time-varying resources of helpers, and (ii) does not require any prior information about the computation capabilities of the helpers. As shown, our proposed method reduces the task completion delay significantly as compared to prior work.

CHAPTER 8

PRAC: PRIVATE AND RATELESS ADAPTIVE CODED COMPUTATION AT THE EDGE

The contents of this chapters are based on our work that is published in the proceedings of 2019 SPIE Disruptive Technologies in Information Sciences II [8] and a journal under submission. ©2019 SPIE. Reprinted, with permission, from [8].

Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables critical computations at the edge in applications such as Internet of Things (IoT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Coded computation is recently gaining interest, thanks to its higher reliability, smaller delay, and lower communication costs. In this chapter, we develop a private and rateless adaptive coded computation (PRAC) algorithm for distributed matrix-vector multiplication by taking into account (i) the privacy requirements of IoT applications and devices, and (ii) the heterogeneous and time-varying resources of edge devices. We show that PRAC outperforms known secure coded computing methods when resources are heterogeneous. We provide theoretical guarantees on the performance of PRAC and its comparison to baselines. Moreover, we confirm our theoretical results through simulations and implementations on Android-based smartphones.

8.1 Background

Edge computing is emerging as a new paradigm to allow processing data near the edge of the network, where the data is typically generated and collected. This enables computation at the edge in applications such as Internet of Things (IoT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

One of the promising solutions to handle computationally intensive tasks is computation offloading, which advocates offloading tasks to remote servers or cloud. Yet, offloading tasks to remote servers or cloud could be luxury that cannot be afforded by most of the edge applications, where connectivity to remote servers can be lost or compromised, which makes edge computing crucial.

Edge computing advocates that computationally intensive tasks in a device (master) could be offloaded to other edge or end devices (workers) in close proximity. However, offloading tasks to other devices leaves the IoT and the applications it is supporting at the complete mercy of an attacker. Furthermore, exploiting the potential of edge computing is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Thus, our goal is to develop a private, dynamic, adaptive, and heterogeneity-aware cooperative computation framework that provides both privacy and computation efficiency guarantees.

Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation [7, 110, 133, 135–144]. The following canonical example demonstrates the effectiveness of coded computation.

Example 5. *Consider the setup where a master device wishes to offload a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides the matrix A row-wise equally into two smaller matrices A_1 and A_2, which are then encoded using a* $(3, 2)$ *Maximum Distance Separable (MDS) code*[1] *to give $B_1 = A_1$, $B_2 = A_2$ and $B_3 = A_1 + A_2$, and sends each to a different worker. Also, the master device sends $\mathbf{x}$ to workers and ask them to compute $B_i\mathbf{x}$, $i \in \{1, 2, 3\}$. When the master receives the computed values (*i.e., $B_i\mathbf{x}$*) from at least two out of three workers, it can decode its desired task, which is the computation of $A\mathbf{x}$. The power of coded computations is that it makes $B_3 = A_1 + A_2$ acts as a "joker" redundant task that can replace any of the other two tasks if they end up straggling or failing.* □

The above example demonstrates the benefit of coding for edge computing. However, the very nature of task offloading from a master to worker devices makes the computation framework vulnerable to attacks. One of the attacks, which is also the focus of this work, is *eavesdropper adversary*, where one or more of workers can behave as an eavesdropper and can spy on the coded data sent to these devices for computations.[2] For example, $B_3 = A_1 + A_2$ in Example 5 can be processed and spied by

[1]*An (n, k) MDS code divides the master's data into k chunks and encodes it into n chunks ($n > k$) such that any k chunks out of n are sufficient to recover the original data.*

[2]Note that this work focuses specifically on *eavesdropper adversary* although there are other types of attacks; for example *Byzantine adversary*, which is out of scope of this work.

TABLE VI

EXAMPLE PRAC OPERATION IN HOMOGENEOUS AND STATIC SETUP.

Time	Worker 1	Worker 2	Worker 3
1	R_1	$\mathbf{A}_1 + A_3 + R_1$	$\mathbf{A}_3 + R_1$
2	▨	R_2	▨
3	$\mathbf{A}_2 + A_3 + R_2$	▨	▨
4	▨	▨	$\mathbf{A}_2 + R_2$

worker 3. Even though $A_1 + A_2$ is coded, the attacker can infer some information from this coded task. Thus, it is crucial to develop a private coded computation mechanism against eavesdropper adversary.

In this chapter, we develop a private and rateless adaptive coded computation (PRAC) mechanism. PRAC is (i) private as it is secure against eavesdropper adversary, (ii) rateless, because it uses Fountain codes [115–117] instead of Maximum Distance Separable (MDS) codes [145, 146], and (iii) adaptive as the master device offloads tasks to workers by taking into account their heterogeneous and time-varying resources. Next, we illustrate the main idea of PRAC through an illustrative example.

Example 6. *We consider the same setup in Example 5, where a master device offloads a task to 3 workers. The master has a large data matrix A and wants to compute matrix vector product $A\mathbf{x}$. The master device divides matrix A row-wise into 3 sub-matrices A_1, A_2, A_3; and encodes these matrices* *using a Fountain code*[1] *[115–117]. An example set of coded packets is* A_2*,* A_3*,* $A_1 + A_3$*, and* $A_2 + A_3$*. However, prior to sending a coded packet to a worker, the master generates a random key matrix* R *with the same dimensions as* A_i *and with entries drawn uniformly from the same field which contains the entries of* A*. The key matrix is added to the coded packets to provide privacy as shown in Table VI. In particular, a key matrix* R_1 *is created at the start of time slot* 1*, combined with* $A_1 + A_3$ *and* A_3*, and transmitted to workers* 2 *and* 3*, respectively.* R_1 *is also transmitted to worker* 1 *in order to obtain* $R_1\mathbf{x}$ *that will help the master in the decoding process.*

$\square$

This example shows that PRAC can take advantage of coding for computation, and provide privacy.

The use of Fountain codes in encoding the sub-tasks provides PRAC a flexibility in the number of stragglers and in the computing capacity of the workers, reflected by the number of sub-tasks assigned to each worker. In contrast, existing solutions for secure coded computing require the master to set a threshold on the number of stragglers that it can tolerate and pre-assign the sub-tasks to the workers based on this threshold.

Organization. The structure of the rest of this chapter is as follows. We start with presenting the system model in Section 8.2. Section 8.3 presents the design of private and rateless adaptive coded computation (PRAC). We characterize and analyze PRAC in Section 8.4. We present evaluation results in section 8.5. Section 8.6 presents related work.

[1]*Fountain codes are desirable here for two properties: (i) they provide a fluid abstraction of the coded packets so the master can always decode with high probability as long as it collects enough packets; (ii) They have low decoding complexity.*

8.2 System Model

Setup. We consider a master/workers setup at the edge of the network, where the master device M offloads its computationally intensive tasks to workers $w_i,\ i \in \mathcal{N}$, (where $|\mathcal{N}| = n$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. The master device divides a task into smaller sub-tasks, and offloads them to workers that process these sub-tasks in parallel.

Task Model. We focus on the computation of linear functions, *i.e.,* matrix-vector multiplication. We suppose the master wants to compute the matrix vector product $A\mathbf{x}$, where $A \in \mathbb{F}_q^{m\times\ell}$ can be thought of as the data matrix and $\mathbf{x} \in \mathbb{F}_q^{\ell}$ can be thought of as an attribute vector. We assume that the entries of A and $\mathbf{x}$ are drawn independently and uniformly at random[1] from $\mathbb{F}_q$. The motivation stems from machine learning applications where computing linear functions is a building block of several iterative algorithms [147, 148]. For instance, the main computation of a gradient descent algorithm with squared error loss function is

$$\mathbf{x}^{+} = \mathbf{x} - \alpha A^T(A\mathbf{x} - \mathbf{y}), \tag{8.1}$$

where $\mathbf{x}$ is the value of the attribute vector at a given iteration, $\mathbf{x}^{+}$ is the updated value of $\mathbf{x}$ at this iteration and the learning rate α is a parameter of the algorithm. Equation 8.1 consists of computing two linear functions $A\mathbf{x}$ and $A^T\mathbf{w} \triangleq A^T(A\mathbf{x} - \mathbf{y})$.

Worker and Attack Model. The workers incur random delays while executing the task assigned to them by the master device. The workers have different computation and communication specifications

[1]We abuse notation and denote both the random matrix representing the data and its realization by A. We do the same for $\mathbf{x}$.

resulting in a heterogeneous environment which includes workers that are significantly slower than others, known as stragglers. Moreover, the workers cannot be trusted with the master's data. We consider an *eavesdropper adversary* in this chapter, where one or more of workers can be eavesdroppers and can spy on the coded data sent to these devices for computations. We assume that up to z, $z < n$, workers can collude, *i.e.,* z workers can share the data they received from the master in order to obtain information about A. The parameter z can be chosen based on the desired privacy level; a larger z means a higher privacy level and vice versa. One would want to set z to the largest possible value for maximum, $z = n - 1$ security purposes. However, this has the drawback of increasing the complexity and the runtime of the algorithm. In our setup we assume that z is a fixed and given system parameter.

Coding & Secret Keys. The matrix A can be divided into b row blocks (we assume that b divides m, otherwise all-zero rows can be added to the matrix to satisfy this property) denoted by A_i, $i = 1\dots, m/b$. The master applies Fountain coding [115–117] across row blocks to create information packets $\nu_j \triangleq \sum_{i=1}^{m} c_{i,j} A_i$, $j = 1, 2, \dots,$ where the $c_{i,j} \in \{0, 1\}$. Note that an information packet is a matrix of dimension $m/b \times \ell$, i.e., $\nu_j \in \mathbb{F}_q^{m/b \times \ell}$. Such rateless coding is compatible with our goal to create adaptive coded cooperation computation framework. In order to maintain privacy of the data, the master device generates random matrices R_i of dimension $m/b \times \ell$ called *keys*. The entries of the R_i matrices are drawn uniformly at random from the field that contains the entries of A. Each information packet ν_j is *padded* with a linear combination of z keys $f_j(R_{i_1}, \dots, R_{i_z})$ to create a secure packet $s_j \in \mathbb{F}_q^{m/b \times \ell}$ defined as $s_j \triangleq \nu_j + f_j(R_{i_1}, \dots, R_{i_z})$.

The master device sends $\mathbf{x}$ to all workers, then it sends the keys and the s_j's to the workers according to our PRAC scheme described later. Each worker multiplies the received packet by $\mathbf{x}$ and sends the result back to the master. Since the encoding is rateless, the master keeps sending packets to the workers until it can decode $A\mathbf{x}$. The master then sends a stop message to all the workers.

Privacy Conditions. Our primary requirement is that any collection of z (or less) workers will not be able to obtain any information about A, in an information theoretic sense.

In particular, let P_i, $i = 1\dots,n$, denote the collection of packets sent to worker w_i. For any set $\mathcal{B} \subseteq \{1,\dots,n\}$, let $P_{\mathcal{B}} \triangleq \{P_i, i \in \mathcal{B}\}$ denote the collection of packets given to worker w_i for all $i \in \mathcal{B}$. The privacy requirement[1] can be expressed as

$$H(A|P_{\mathcal{Z}}) = H(A), \quad \forall \mathcal{Z} \subseteq \{1,\dots,n\} \text{ s.t. } |\mathcal{Z}| \leq z. \tag{8.2}$$

$H(A)$ denotes the entropy, or uncertainty, about A and $H(A|P_{\mathcal{Z}})$ denotes the uncertainty about A after observing $P_{\mathcal{Z}}$.

Delay Model. Each packet transmitted from the master to a worker $w_i,\ i = 1,2,...,n$, experiences the following delays: (i) transmission delay for sending the packet from the master to the worker, (ii) computation delay for computing the multiplication of the packet by the vector $\mathbf{x}$, and (iii) transmission delay for sending the computed packet from the worker w_i back to the master. We denote by $\beta_{t,i}$ the computation time of the t^{th} packet at worker i and RTT_i denotes the round-trip time spent to send and receive a packet from worker i. The time spent by the master is equal to the time taken by the $(z+1)^{\text{st}}$ fastest worker to finish its assigned tasks.

[1]In some cases the vector $\mathbf{x}$ may contain information about A and therefore must not be revealed to the workers. We explain in Appendix B how to generalize our scheme to account for such cases.

TABLE VII

SUMMARY OF NOTATIONS.

Symbol	Meaning
M	master
w_i	worker i
n	number of workers
A	$m \times \ell$ data matrix
$\mathbf{x}$	$1 \times \ell$ attribute vector
z	number of colluding workers
m	number of rows in A
ε	overhead of Fountain codes
A_i	i^{th} row block of data matrix A
R	random matrix
RTT_i	round trip time to send and receive packet i
$\beta_{t,i}$	computation time of the t^{th} packet at w_i
ν	Fountain coded packet of A_i's
s	secure Fountain coded packet
T_i	time to compute a packet at w_i
$T_{(d)}$	d^{th} order statistic of T_i's
T	time spent by M to decode $A\mathbf{x}$

8.3 Design of PRAC

8.3.1 Overview

We present the detailed explanation of PRAC. Let $p_{t,i} \in \mathbb{F}_q^{m/b \times \ell}$ be the t^{th} packet sent to worker w_i. This packet can be either a key or a secure packet. For each value of t, the master sends z keys denoted by $R_{t,1}, \ldots, R_{t,z}$ to z different workers and up to $n - z$ secure packets $s_{t,1}, \ldots, s_{t,n-z}$ to the remaining workers. The master needs the results of $m + \epsilon$ information packets, *i.e.,* $\nu_{t,i}\mathbf{x}$, to decode the final result $A\mathbf{x}$, where ϵ is the overhead required by Fountain coding[1]. To obtain the results of $m + \epsilon$ information packets, the master needs the results of $m + \epsilon$ secure packets, $s_{t,i}\mathbf{x} = (\nu_{i,j} + f_j(R_{t,i}, \ldots, R_{t,z}))\mathbf{x}$, together with all the corresponding[2] $R_{t,i}\mathbf{x}, i = 1, \ldots, z$. Therefore, only the results of the $s_{t,i}\mathbf{x}$ for which all the computed keys $R_{t,i}\mathbf{x}, i = 1, ..., z$, are received by the master can account for the total of $m + \epsilon$ information packets.

8.3.2 Dynamic Rate Adaptation

The dynamic rate adaptation part of PRAC is based on [7]. In particular, the master offloads coded packets gradually to workers and receives two ACKs for each transmitted packet; one confirming the receipt of the packet by the worker, and the second one (piggybacked to the computed packet) showing that the packet is computed by the worker. Then, based on the frequency of the received ACKs, the master decides to transmit more/less coded packets to that worker. In particular, each packet $p_{t,i}$

[1]The overhead required by Fountain coding is typically as low as 5% [117], *i.e.,* $\epsilon = 0.05m$.

[2]Recall that $f_j(R_{t,i}, \ldots, R_{t,z})$ is a linear function, thus it is easy to extract $(f_j(R_{t,i}, \ldots, R_{t,z}))\mathbf{x}$ from $(R_{t,i})\mathbf{x}, i = 1, ..., z$.

is transmitted to each worker w_i before or right after the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master. For this purpose, the average per packet computing time $\mathbb{E}[\beta_{t,i}]$ is calculated for each worker w_i dynamically based on the previously received ACKs. Each packet $p_{t,i}$ is transmitted after waiting $\mathbb{E}[\beta_{t,i}]$ from the time $p_{t-1,i}$ is sent or right after packet $p_{t-1,i}\mathbf{x}$ is received at the master, thus reducing the idle time at the workers. This policy is shown to approach the optimal task completion delay and maximizes the workers' efficiency and is shown to improve task completion time significantly compared with the literature [7].

8.3.3 Coding

We explain the coding scheme used in PRAC. We start with an example to build an intuition and illustrate the scheme before going into details.

Example 7. *Assume there are $n = 4$ workers out of which any $z = 2$ can collude. Let A and $\mathbf{x}$ be the data owned by the master and the vector to be multiplied by A, respectively. The master sends $\mathbf{x}$ to all the workers. For the sake of simplicity, assume A can be divided into $b = 6$ row blocks,* i.e., $A = \begin{bmatrix} A_1 & A_2 & \dots & A_6 \end{bmatrix}^T$*. The master encodes the A_i's using Fountain code. We denote by* round *the event when the master sends a new packet to a worker. For example, we say that worker 1 is at round 3 if it has received 3 packets so far. For every round t, the master generates $z = 2$ random matrices*

$R_{t,1}$, $R_{t,2}$ and encodes them using an $(n, z) = (4, 2)$ systematic maximum distance separable (MDS) code by multiplying $R_{t,1}$, $R_{t,2}$ by a generator matrix G as follows

$$G \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix} \triangleq \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} R_{t,1} \\ R_{t,2} \end{bmatrix}. \tag{8.3}$$

The following ($R_{t,1}$, $R_{t,2}$, $R_{t,1} + R_{t,2}$, $R_{t,1} + 2R_{t,2}$) is the resulting encoding of the $R_{t,i}$'s. Now let us assume that workers can be stragglers. At the beginning the master initializes all the workers at round 1. Afterwards, when a worker w_i finishes its task, the master checks how many packets this worker has received so far and how many other workers are at this round. If this worker w_i is the first or second to be at round t, the master generates $R_{t,1}$ or $R_{t,2}$, respectively, and sends it to w_i. Otherwise, if w_i is the jth worker ($j > 2$) to be at round t, the master multiplies $\begin{bmatrix} R_{t,1} & R_{t,2} \end{bmatrix}^T$ by the jth row of G, adds it to a generated coded packet, and sends it to w_i. The master keeps sending packets to the workers until it can decode $A\mathbf{x}$. We illustrate the idea in Table VIII.

We now explain the details of PRAC in the presence of z colluding workers.

1. *Initialization:* The master divides A into b row blocks $A_1, \dots, A_b$ and sends the vector $\mathbf{x}$ to the workers. Let $G \in \mathbb{F}_q^{n \times z}$, $q > n$, be the generator matrix of an (n, z) systematic MDS code. For example one may use systematic Reed-Solomon codes that use Vandermonde matrix as generator matrix, see for example [149]. The master generates z random matrices $R_{1,1}, \dots, R_{1,z}$ and encodes them using G. Each coded key can be denoted by $\mathbf{g}_i\mathcal{R}$ where $\mathbf{g}_i$ is the i^{th} row of G

and $\mathcal{R} \triangleq \begin{bmatrix} R_{1,1} & \dots & R_{1,z} \end{bmatrix}^T$. The master sends the z keys $R_{1,1}, \dots, R_{1,z}$ to the first z workers, generates $n-z$ Fountain coded packets of the A_i's, adds to each packet an encoded random key $\mathbf{g}_i\mathcal{R}$, $i = z+1, \dots n$, and sends them to the remaining $n-z$ workers.

2. *Encoding and adaptivity:* When the master wants to send a new packet to a worker (noting that a packet $p_{t,i}$ is transmitted to worker w_i before, or right after, the computed packet $p_{t-1,i}\mathbf{x}$ is received at the master according to the strategy described in Section 8.3.2), it checks at which round this worker is, *i.e.,* how many packets this worker has received so far, and checks how many other workers are at this round. Assume the worker is at round t and $j-1$ other workers are at this round. If $j \leq z$, the master generates and sends $R_{t,j}$ to the worker. However, if $j > z$ the master generates a Fountain coded packet of the A_i's (*e.g.,* $A_1 + A_2$), adds to it $\mathbf{g}_j\mathcal{R}$ and sends the packet $(A_1 + A_2 + \mathbf{g}_j\mathcal{R})$ to the worker. Each worker computes the multiplication of the received packet by the vector $\mathbf{x}$ and sends the result to the master.

3. *Decoding and speed:* Let τ_i denote the number of packets received by worker i. At the end of the process, the master has $R_{t,i}\mathbf{x}$ for all $t = 1, \dots, \tau_{max}$ and all $i = 1, \dots, z$, where $\tau_{max} \triangleq \max_i \tau_i$. The master can subtract the $R_{t,i}$'s from all received secure information packets, thus can decode the A_i's using the Fountain code decoding process. The number of secure packets that can be used to decode is dictated by the $(z+1)^{\text{st}}$ fastest worker, *i.e.,* the master can only use the results of secure information packets computed at a given round if at least $z+1$ workers have completed that round. If for example the z fastest workers have completed round 100 and the $(z+1)^{\text{st}}$ fastest worker has completed round 20, the master can only use the packets belonging to the first 20 rounds. The reason is that the master needs all the keys corresponding to a given round in order

to use the secure information packet for decoding. In Lemma 10 we prove that this scheme is optimal, *i.e.,* in private coded computing the master cannot use the packets computed at rounds finished by less than $z + 1$ workers irrespective of the coding scheme.

8.4 Performance Analysis of PRAC

8.4.1 Privacy

In this section, we provide theoretical analysis of PRAC by particularly focusing on its privacy properties.

Theorem 9. *PRAC is a rateless real-time adaptive coded computing scheme that allows a master device to run distributed linear computation on private data A via n workers while satisfying the privacy constraint given in Equation 8.2 for a given $z < n$.*

Proof. Since the random keys are generated independently at each round, it is sufficient to study the privacy of the data on one round and the privacy generalizes to the whole algorithm. We show that for any subset $Z \subset \{1, \dots, n\}, |Z| = z$, the collection of packets $p_Z \triangleq \{p_{t,i}, i \in Z\}$ sent at round t reveals no information about the data A as given in (Equation 8.2), *i.e.,* $H(A) = H(A|p_Z)$. Let K denote the random variable representing all the keys generated at round t, then it is enough to show that $H(K|A, p_Z) = 0$ as detailed in Appendix C. Therefore, we need to show that given A as side information, any z workers can decode the random keys $R_{t,1}, \dots, R_{t,z}$. Without loss of generality assume the workers are ordered from fastest to slowest, *i.e.,* worker w_1 is the fastest at the considered round t. At each round the master sends z packets as the random keys to the fastest z workers, *i.e.,* $p_{i,t} = R_{t,i}, i = 1, \dots, z$. The remaining $n - z$ packets are secure information packets sent to the remaining $n - z$ workers, *i.e.,* $p_{t,i} = s_{t,i} = \nu_{t,i} + f(R_{t,1}, \ldots, R_{t,z})$, where $\nu_{t,i}$ is a linear combination of row blocks of A and $f(R_{t,1}, \ldots, R_{t,z})$ is a linear combination of the random keys generated at round t. Given the data A as side information, any collection of z packets can be expressed as z codewords of the (n, z) MDS code encoding the random keys. Thus, given A any collection of z packets is enough to decode all the keys and $H(K|S, p_Z) = 0$ which concludes the proof. □

Remark 1. *PRAC requires the master to wait for the* $(z+1)^{st}$ *fastest worker in order to be able to decode* $A\mathbf{x}$*. We show in Lemma 10 that this limitation is a byproduct of all private coded computing schemes.*

Remark 2. *PRAC uses the minimum number of keys required to guarantee the privacy constraints. At each round PRAC uses exactly z random keys which is the minimum amount of required keys.(c.f. Equation* (Equation C.5) *in Appendix C).*

Lemma 10. *Any private coded computing scheme for distributed linear computation limits the master to the speed of the* $(z+1)^{st}$ *fastest worker.*

Proof. The proof of Lemma 10 is provided in Appendix D. □

8.4.2 Task Completion Delay

In this section, we characterize the task completion delay of PRAC and compare it with Staircase codes [135], which are secure against eavesdropping attacks in a coded computation setup with homogeneous resources. First, we start with task completion delay characterization of PRAC.

Theorem 11. *Let b be the number of row blocks in A, let $\beta_{t,i}$ denote the computation time of the t^{th} packet at worker i and let RTT_i denote the round-trip time spent to send and receive a packet from worker i. The task completion time of PRAC is approximated as*

$$\begin{aligned} T_{PRAC} &\approx \max_{i\in\{1,\dots,n\}}\{RTT_i\} + \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,w_i}]}, && (8.4)\\ &\approx \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,w_i}]}, && (8.5)\end{aligned}$$

where w_i are ordered indices of the workers from fastest to slowest, i.e., $w_1 = \arg\min_i \mathbb{E}[\beta_{t,w_i}]$.

Proof. The proof of Theorem 11 is provided in Appendix E. □

Now that we characterized the task completion delay of PRAC, we can compare it with the state-of-the-art. Secure coded computing schemes that exist in the literature usually use static task allocation, where tasks are assigned to workers a priori. The most recent work in the area is Staircase codes, which is shown to outperform all existing schemes that use threshold secret sharing [135]. Therefore, we restrict our focus on comparing PRAC to Staircase codes.

Staircase codes assigns a task of size $b/(k-z)$ row blocks to each worker.[1] Let T_i be the time spent at worker i to compute the whole assigned task. Denote by $T_{(i)}$ the i^{th} order statistic of the T_i's and by $T_{\text{SC}}(n,k,z)$ the task completion time, *i.e.,* time the master waits until it can decode $A\mathbf{x}$, when using Staircase codes. In order to decode $A\mathbf{x}$ the master needs to receive a fraction equal to $(k-z)/(d-z)$

[1]Note that in addition to n and z, all threshold secret sharing based schemes require a parameter $k,\ z<k<n$, which is the minimum number of non stragglers that the master has to wait for before decoding $A\mathbf{x}$.

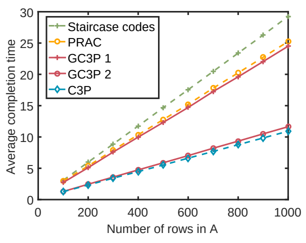


Figure 49. Scenario 1 with the fastest 13 workers as eavesdropper for GC3P 1 and the slowest workers as eavesdropper for GC3P 2.

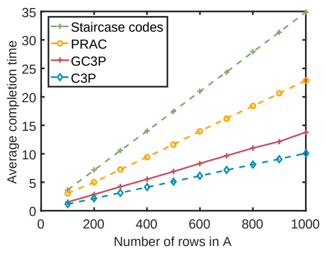


Figure 50. Scenario 2 with 13 workers picked at random to be eavesdroppers.

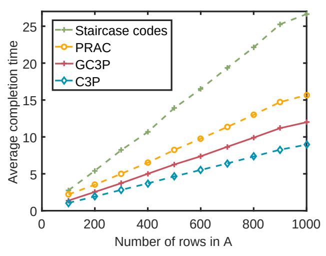


Figure 51. Scenario 3 with 13 workers picked at random to be eavesdroppers.

Figure 52. Comparison between PRAC and GC3P in different scenarios with $n = 50$ workers and $z = 13$ colluding eavesdroppers for different values of the number of rows m. For each value of m we run 100 experiments and average the results. When the eavesdropper are chosen to be the fastest workers, PRAC has very similar performance to GC3P. When the eavesdroppers are picked randomly, the performance of PRAC becomes closer to this of GC3P when the non adversarial workers are more heterogeneous.

of the task assigned to each worker from any d workers where $k \leq d \leq n$. The task completion time of the master can then be expressed as

$$T_{\text{SC}}(n,k,z) = \min_{d \in \{k,\ldots,n\}} \left\{ \frac{k-z}{d-z} T_{(d)} \right\}. \tag{8.6}$$

Theorem 12. *The gap between the completion time of PRAC and coded computation using staircase codes is lower bounded by:*

$$\mathbb{E}[T_{SC}] - \mathbb{E}\left[T_{PRAC}\right] \geq \frac{bx - \epsilon y}{y(x+y)}, \tag{8.7}$$

where $x = \frac{n-d^*}{E[\beta_{t,w_n}]}$, $y = \frac{d^*-z}{E[\beta_{t,d^*}]}$ *and* d^* *is the value of* d *that minimizes Equation 8.6.*

Proof. The proof of Theorem 12 is provided in Appendix F. □

Theorem 12 shows that the lower bound on the gap between secure coded computation using staircase codes and PRAC is in the order of number of row block of A. Hence, the gap between secure coded computation using staircase codes and PRAC is linearly increasing with the number of row blocks of A. Note that, ϵ, the required overhead by fountain coding used in PRAC, becomes negligible by increasing b.

Thus, PRAC outperforms secure coded computation using Staircase codes in heterogeneous systems. The more heterogeneous the workers are, the more improvement is obtained by using PRAC. However, Staircase codes can slightly outperform PRAC in the case where the slowest $n - z$ workers are homogeneous, *i.e.,* have compute service times T_i. Staircase codes outperform PRAC in homogeneous case, because both algorithms are restricted to the slowest $n - z$ workers (see Lemma 10), but PRAC incurs an ϵ overhead of tasks (due to using Fountain codes) which is not needed for Staircase codes. In particular, from Equation 8.5 and Equation 8.6, when the $n - z$ slowest workers are homogeneous, the task completion time of PRAC and Staircase codes are equal to $\frac{b+\epsilon}{n-z}\mathbb{E}[\beta_{t,w_n}]$ and $\frac{b}{n-z}\mathbb{E}[\beta_{t,w_n}]$, respectively.

8.5 Performance Evaluation

8.5.1 Simulations

In this section, we present simulations run on MATLAB, and compare PRAC with the following baselines: (i) Staircase codes [135], (ii) C3P [7] (which is not secure as it is not designed to be secure), and (iii) Genie C3P (GC3P) that extends C3P by assuming a knowledge of the identity of the eavesdroppers and ignoring them. We note that GC3P serves as a lower bound on private coded computing schemes for heterogeneous systems[1] the following reason: for a given number of z colluding workers the ideal coded computing scheme knows which workers are eavesdroppers and ignores them to use the remaining workers without need of randomness. If the identity of the corrupted workers is unknown, coded computing schemes require randomness and become limited to the $(z + 1)^{\text{st}}$ slowest worker (Lemma 10). GC3P and other coded computing schemes have similar performance if the z colluding workers are the fastest workers. If the z colluding workers are the slowest, then GC3P outperforms any coded computing scheme. In terms of comparing PRAC to secret sharing, we restrict our attention to Staircase codes which are a class of secret sharing schemes that enjoys a flexibility in the number of

[1]If the system is homogeneous Staircase codes outperform GC3P, because pre-allocating tasks to the workers avoids the overhead needed by Fountain codes.

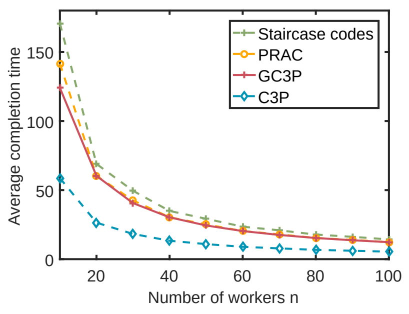


Figure 53. Task completion time as a function of the number of workers with $z = n/4$.

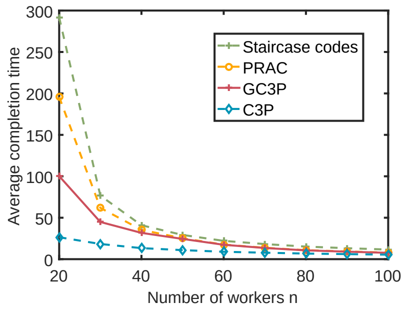


Figure 54. Task completion time as a function of the number of workers with $z = 13$.

Figure 55. Comparison between PRAC, Staircase codes and GC3P in scenario 1 for different values of the number workers and number of colluding workers. We fix the number of rows to $m = 1000$. For each value of the x-axis we run 100 experiments and average the results. We observe that the difference between the completion time of PRAC and this of GC3P is small for small number of colluding workers and increases with the increase of z.

workers needed to decode the matrix-vector multiplication. Staircase codes are shown to outperform any coded computing scheme that requires a threshold on the number of stragglers [135].

In our simulations, we model the computation time of each worker w_i by an independent shifted exponential random variable with rate λ_i and shift c_i, *i.e.,* $F(T_i = t) = 1 - \exp(-\lambda_i(t - c_i))$. We take $c_i = 1/\lambda_i$ and consider three different scenarios for choosing the values of λ_i's for the workers as follows:

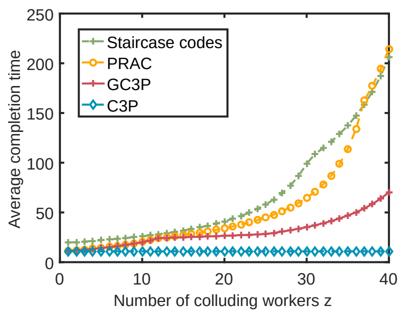


Figure 56. Task completion time as a function of the number of colluding workers for $n = 50$. Computing time of the workers are chosen according to scenario 1.

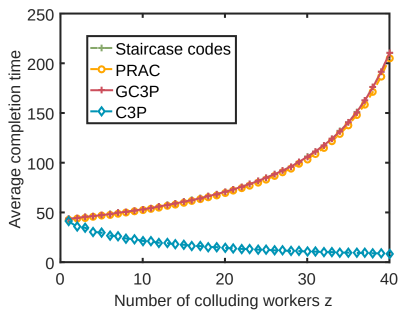


Figure 57. Task completion time for $n = 50$ workers and variable z. Computing times of the workers are chosen such that the $n - z$ slowest workers are homogeneous.

Figure 58. Comparison between PRAC and Staircase codes average completion time as a function of number of colluding workers z. We fix the number of rows to $m = 1000$. Both codes are affected by the increase of number of colluding helpers because their runtime is restricted to the slowest $n - z$ workers. We observe that PRAC outperforms Staircase codes except when the $n - z$ slowest workers are homogeneous.

- *Scenario 1*: we assign $\lambda_i = 3$ for half of the workers, then we assign $\lambda_i = 1$ for one quarter of the workers and assign $\lambda_i = 9$ for the remaining workers.
- *Scenario 2*: we assign $\lambda_i = 1$ for one third of the workers, the second third have $\lambda_i = 3$ and the remaining workers have $\lambda_i = 9$.

- *Scenario 3*: we draw the λ_i's independently and uniformly at random from the interval $[0.5, 9]$.

When running Staircase codes, we choose the parameter k that minimizes the task completion time for the desired n and z. We do so by simulating Staircase codes for all possible values of $z \leq k \leq n$ and choose the one with the minimum completion time.

We take $b = m$, *i.e.,* each row block is simply a row of A. The size of each element of A and vector $\mathbf{x}$ are assumed to be 1 Byte (or 8 bits). Therefore, the size of each transmitted packet $p_{t,i}$ is $8 * \ell$ bits. For the simulation results, we assume that matrix A is a square matrix, *i.e.,* $l = m$. We take $m = 1000$, unless explicitly stated otherwise. C_i denotes the average channel capacity of each worker w_i and is selected uniformly from the interval $[10, 20]$ Mbps. The rate of sending a packet to worker w_i is sampled from a Poisson distribution with mean C_i.

In Figure 52 we show the effect of the number of rows m on the completion time at the master. We fix the number of workers to 50 and the number of colluding workers to 13 and plot the completion time for PRAC, C3P, GC3P and Staircase codes. Notice that PRAC and Staircase codes have close completion time in scenario 1 (Figure Figure 49) and this completion time is far from that of C3P. The reason is that in this scenario we pick exactly 13 workers to be fast ($\lambda_i = 9$) and the others to be significantly slower. Since PRAC assigns keys to the fastest z workers, the completion time is dictated by the slow workers. To compare PRAC to Staircase codes notice that the majority of the remaining workers have $\lambda_i = 3$ therefore pre-allocating equal tasks to the workers is close to adaptively allocating the tasks.

In terms of lower bound on PRAC, observe that when the fastest workers are assumed to be adversarial, GC3P and PRAC have very similar task completion time. However, when the slowest workers are assumed to be adversarial the completion of GC3P is very close to C3P and far from PRAC. This observation is in accordance of Lemma 2. In scenarios 2 and 3 we pick the adversarial workers uniformly at random and observe that the completion time of PRAC becomes closer to GC3P when the workers are more heterogeneous. For instance, in scenario 3, GC3P and PRAC have closer performance when the workers' computing times are chosen uniformly at random from the interval $[0.5, 9]$.

In Figure 55, we plot the task completion time as a function of the number of workers n for a fixed number of rows $m = 1000$ and λ_i's assigned according to scenario 1. In Figure 55(a), we change the number of workers from 10 to 100 and keep the ratio $z/n = 1/4$ fixed. We notice that with the increase of n the completion time of PRAC becomes closer to GC3P. In Figure 55(b), we change the number of workers from 20 to 100 and keep $z = 13$ fixed. We notice that with the increase of n, the effect of the eavesdropper is amortized and the completion time of PRAC becomes closer to C3P. In this setting, PRAC always outperforms Staircase codes.

In Figure 58, we plot the task completion time as a function of the number of colluding workers. In Figure 58(a), we choose the computing time at the workers according to scenario 1. We change z from 1 to 40 and observe that the completion time of PRAC deviates from that of GC3P with the increase of z. More importantly, we observe two inflection points of the average completion time of PRAC at $z = 13$ and $z = 37$. Those inflection points are due to the fact that we have 12 fast workers ($\lambda = 9$) and 25 workers with medium speed ($\lambda = 3$) in the system. For $z > 36$, the completion time of Staircase codes becomes less than that of PRAC because the 14 slowest workers are homogeneous. Therefore, pre-allocating the tasks is better than using Fountain codes and paying for the overhead of computations. To confirm that Staircase codes always outperforms PRAC when the slowest $n - z$

workers are homogeneous, we run a simulation in which we divide the workers into three clusters. The first cluster consists of $\lfloor z/2 \rfloor$ fast workers ($\lambda = 9$), the second consists of $\lfloor z/2 \rfloor + 1$ workers that are regular ($\lambda = 3$) and the remaining $n - z$ workers are slow ($\lambda = 1$). In Figure 58(b) we fix n to 50 and change z from 1 to 40. We observe that Staircase codes always outperform PRAC in this setting. In contrast to non secure C3P, Staircase codes and PRAC are always restricted to the slowest $n - z$ workers and cannot leverage the increase of the number of fast workers. For GC3P, we assume that the fastest workers are eavesdroppers. We note that as expected from Lemma 10, when the fastest workers are assumed to be eavesdroppers the performance of GC3P and PRAC becomes very close.

8.5.2 Experiments

Setup. The master device is a Nexus 5 Android-based smartphone running 6.0.1. The worker devices are Nexus 6Ps running Android 8.1.0. The master device connects to worker devices via Wi-Fi Direct links and the master is the group owner of Wi-Fi Direct group. The master device is required to complete one matrix multiplication ($y = Ax$) where A is of dimensions 60×10000 and x is a 10000×1 vector. A is further divided by each row. The matrix multiplication is completed by offloading to the workers. There is also an introduced delay at the workers following an exponential distribution. The introduced delays serve to emulate applications running in the background of the devices. When each worker device is done calculating and the introduced delay is passed, it returns the result to the master. Furthermore, we assume that there is one unknown worker that is adversarial among all the workers and we want to protect our data by adding random keys. The experiments are conducted in a lab environment where there are other Wi-Fi networks operating in the background.

Baselines. Our PRAC algorithm is compared to three baseline algorithms: (i) Staircase codes that preallocate the tasks based on the number of workers n, the minimum number of workers required to reconstruct the information k, and the number of colluding workers z; (ii) GC3P in which we assume the adversarial worker is known and excluded during the task allocation. In this setup we run C3P on $n-z$ workers; (iii) Non secure C3P in which the security problem is ignored and the master device will utilize every resource without randomness.

Results. Figure 59 presents the task completion time with increasing number of workers for the homogeneous setup, *i.e.,* when all the workers have similar computing times. Computing delay for each packet follows an exponential distribution with mean $\mu = 1/\lambda = 3$ seconds in all workers. C3P performs the best in terms of completion time, but C3P do not provide any privacy guarantees. PRAC outperforms Staircase codes when the number of workers is 5. The reason is that PRAC performs better than staircase codes in heterogeneous setup, and when the number of workers increases, the system becomes a bit more heterogeneous. GC3P significantly outperforms PRAC in terms of completion time. Yet, it requires prior knowledge of which worker is adversarial, which is often not available in real world scenarios.

Now, we focus on heterogeneous setup. We group the workers into two groups; fast workers (per task delay follows exponential delay with mean 2 seconds) and slow workers (per task delay follows exponential distribution with mean 5 seconds). Figure 60 presents the completion time as a function of number of workers. In this setup, for the n-worker scenario, there are $\lceil \frac{n}{2} \rceil$ fast and $\lfloor \frac{n}{2} \rfloor$ slow workers. The difference between the setups of Figure 60(a) and Figure 60(b) is that we remove a fast worker (as adversarial) for GC3P in the former, whereas in the latter, we assume that the eavesdropper is a

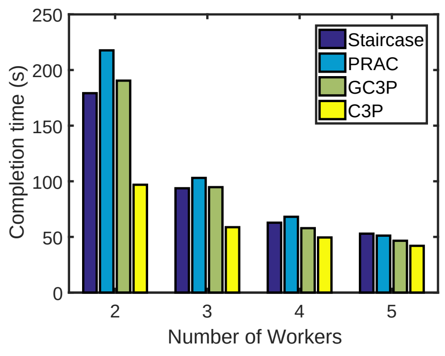
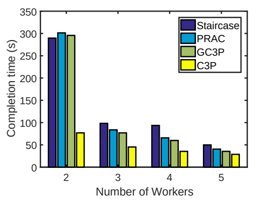


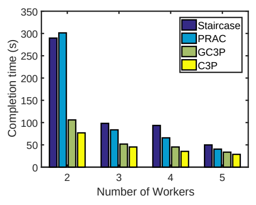
Figure 59. Completion time as function of the number of workers in homogeneous setup.

slow worker. As illustrated in Figure 60, for the 2-worker case, due to the 5% overhead introduced by Fountain codes, PRAC performs worse than Staircase code. However, PRAC outperforms staircase code in terms of completion time for 3, 4, and 5 worker cases. This is due to the fact that PRAC can utilize results calculated by slow workers more effectively when the number of workers is large. On the other hand, the results computed by slow workers are often discarded in Staircase codes, which is a waste of computation resources. If a fast worker is removed as adversarial for GC3P, the difference between the performance of GC3P and PRAC becomes smaller. This result is intuitive as, in PRAC, the master has to wait for the $(z+1)^{\text{st}}$ fastest worker to decode $A\mathbf{x}$, which is also the case for GC3P in this setting.

In Figure 61, we consider the same setup with the exception that for the n-worker scenario, there are $\lceil \frac{n}{2} \rceil$ slow and $\lfloor \frac{n}{2} \rfloor$ fast workers.



(a) We assume a fast worker is adversarial for GC3P.



(b) We assume a slow worker is adversarial for GC3P.

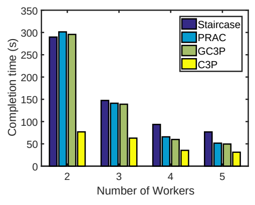
Figure 60. Completion time as function of the number of workers in heterogeneous setup.

Staircase codes performs more closely to PRAC in the 3-worker case as compared to Figure 60 since Staircase codes and PRAC have similar number of computations. Yet, for 5-worker case, PRAC outperforms Staircase codes when comparing to Figure 60 since PRAC is adaptive to time-varying resources while Staircase codes assigns tasks a priori in a static manner.

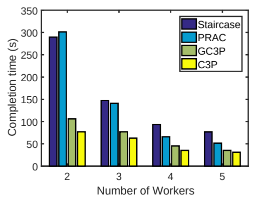
Note that in all experiments when $n - z$ slowest workers are homogeneous Staircase codes outperform GC3P and PRAC. This happens because pre-allocating the tasks to the workers avoids the overhead of sub-tasks required by Fountain codes and utilizes all the workers to their fullest capacity.

8.6 Related work

Mobile cloud computing is a rapidly growing field with the aim of providing better experience of quality and extensive computing resources to mobile devices [61, 62]. The main solution to mobile computing is to offload tasks to the cloud or to neighboring devices by exploiting connectivity of the devices. With task offloading comes several challenges such as heterogeneity of the devices, time vary-



(a) We assume a fast worker is adversarial for GC3P.



(b) We assume a slow worker is adversarial for GC3P.

Figure 61. Completion time as function of the number of workers in heterogeneous setup.

ing communication channels and energy efficiency, see *e.g.,* [50, 54, 63, 72]. We refer interested reader to [7] and references within for a detailed literature on edge computing and mobile cloud computing.

The problem of stragglers in distributed systems is initially studied by the distributed computing community, see *e.g.,* [131, 150–166]. Research interest in using coding theoretical techniques for straggler mitigation in distributed content download and distributed computing is rapidly growing. The early body of work focused on content download, see *e.g.,* [167–171]. Using codes for straggler mitigation in distributed computing started in [110] where the authors proposed the use of MDS codes for distributed linear machine learning algorithms in homogeneous workers setting.

Following the work of [110], coding schemes for straggler mitigation in distributed matrix-matrix multiplication, coded computing and machine learning algorithms are introduced and the fundamental limits between the computation load and the communication cost are studied, see *e.g.,* [140, 172] and references within for matrix-matrix multiplication, see [110, 136, 139, 141–143, 173–180] for machine learning algorithms and [133, 137, 138, 181] for other topics.

Codes for privacy and straggler mitigation in distributed computing are first introduced in [135] where the authors consider a homogeneous setting and focus on matrix-vector multiplication. Beyond matrix-vector multiplication, the problem of private distributed matrix-matrix multiplication and private polynomial computation with straggler tolerance is studied [182–187]. The main difference between those works and PRAC is that the former works are deigned for the homogeneous setting in which the master pre-assigns the sub-tasks equally to the workers and sets a threshold on the number of stragglers that it can tolerate. Works on privacy-preserving machine learning algorithms are also related to our work. However, the privacy constraint in this line of work is computational privacy and the proposed solutions do not take stragglers into account, see *e.g.,* [188–190].

TABLE VIII

DEPICTION OF PRAC IN THE PRESENCE OF STRAGGLERS. THE MASTER KEEPS GENERATING PACKETS USING FOUNTAIN CODES UNTIL IT CAN DECODE Ax. THE MASTER ESTIMATES THE AVERAGE TASK COMPLETION TIME OF EACH WORKER AND SENDS A NEW PACKET TO AVOID IDLE TIME. EACH NEW PACKET SENT TO A WORKER MUST BE SECURED WITH A NEW RANDOM VECTOR. THE MASTER CAN DECODE $A_1\mathbf{X}, \ldots, A_6\mathbf{X}$ AFTER RECEIVING ALL THE PACKETS NOT HAVING $R_{4,1}$ OR $R_{4,2}$ IN THEM.

Time	Worker 1	Worker 2	Worker 3	Worker 4
1	$R_{1,1}$	$R_{1,2}$	$\mathbf{A}_4 + R_{1,1} + R_{1,2}$	$\mathbf{A}_3 + A_4 + A_6 + R_{1,1} + 2R_{1,2}$
2				$R_{2,1}$
3	$R_{2,2}$			
4		$\mathbf{A}_3 + R_{2,1} + R_{2,2}$	$\mathbf{A}_4 + A_5 + R_{2,1} + 2R_{2,2}$	
5		$R_{3,1}$		
6	$\mathbf{A}_2 + R_{3,1} + R_{3,2}$			$R_{3,2}$
7		$R_{4,1}$	$\mathbf{A}_1 + R_{3,1} + 2R_{3,2}$	
8	$R_{4,2}$			$\mathbf{A}_2 + A_3 + R_{4,1} + R_{4,2}$

CHAPTER 9

CONCLUSION

In this thesis, we focus on design, optimization, and implementation of communication and computation algorithms by particularly focusing on Internet of Things and Edge Computing. In particular,

1. In our work *Device-Centric Cooperation in Mobile Networks*, we considered a cooperation scenario among mobile devices for video streaming. We developed a device-centric cooperation scheme; DcC. We showed that DcC reduces; (i) overhead; *i.e.,* the number of control packets that should be transmitted over cellular links, and (ii) the amount of delay that each packet experiences. Simulations demonstrate significant improvement in terms of overhead and delay.
2. In our work *Energy-Aware Cooperative Computation in Mobile Devices*, we considered that a group of cooperative mobile devices, within proximity of each other, (i) use their cellular or Wi-Fi (802.11) links as their primary networking interfaces, and (ii) exploit their D2D connections (Wi-Fi Direct) for cooperative computation. We showed that if mobile devices cooperate to utilize their aggregate processing power, it significantly improves transmission rates. Thus, for this scenario, we developed an *energy-aware cooperative computation* framework to effectively utilize processing power and energy. This framework provides a set of algorithms including flow, computation and energy controls as well as cooperation and scheduling. We implemented these algorithms in a testbed, which consists of real mobile devices. The experiments in the testbed

show that our *energy-aware cooperative computation* framework brings significant performance benefits.

3. In the work *Device-Aware Routing and Scheduling in Multi-Hop Device-to-Device Networks*, we developed a *device-aware routing and scheduling algorithm* (DARS) over D2D networks by taking into account device capabilities such as computing power, energy, and incentives. Our approach is grounded on a network utility maximization formulation of the problem and its solution. We developed a multi-hop D2D testbed using real mobile devices. We implemented DARS over this testbed. The experimental results demonstrate the benefits of our algorithm.

4. In our work *Predictive Edge Computing with Hard Deadlines*, we developed a predictive edge computing algorithms `PrComp` with hard deadline constraints for serial and parallel tasks. Our algorithms (i) predict the uncertain dynamics of resources of edge devices, and (ii) make task offloading decisions by taking into account the predicted available resources, as well as the hard deadline constraints of tasks. We evaluate `PrComp` on a testbed consisting of real Android-based smartphones. The experiments show that `PrComp` algorithms significantly improve energy consumption of edge devices as well as task completion delay as compared to baselines.

5. In our work *the Evolving Nature of Disaster Management in the Internet and Social Media Era*, social media usage appears to mimic the usage of everyday communication (*e.g.,* telephony) during disasters, and could therefore effectively complement other communication channels in disaster situations. Having an intelligent engine processing social media (*e.g.,* tweets) in real-time can help coordinate efficient, fine-grained dissemination of requests/offers of assistance to all the intended/relevant recipients, whether it is authorities or ordinary people. Geo-tagged tweets can

be of great help where Automatic Location Information (ALI) of 911 service is not functioning. Despite the benefits of social media, privacy (sending personal information online like location, *etc.*) and false information (starting rumors, *etc.*) are some of the important issues that social media-based crisis response methods face [81]. It is important to further perfect their use through the design of efficient, secure and reliable dissemination architectures.

6. In our work *Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge*, we designed a Computation Control Protocol (`C3P`), where heterogeneous edge devices with computation capabilities and energy resources are connected to each other. In `C3P`, a collector device divides tasks into sub-tasks, offloads them to helpers by taking into account heterogeneous resources. `C3P` is (i) a dynamic algorithm that efficiently utilizes the potential of each helper, and (ii) adaptive to the time-varying resources at helpers. We analyzed the performance of `C3P` in terms of task completion delay and efficiency. Simulation and experiment results in an Android testbed confirm that `C3P` is efficient and reduces the completion delay significantly as compared to baselines.

7. In our work *PRAC: Private and Rateless Adaptive Coded Computation at the Edge*, we develop a secure edge computing mechanism to mitigate the computational bottleneck of IoT devices by allowing these devices to help each other in their computations, with possible help from the cloud if available. Our key tool is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation. Focusing on eavesdropping attacks, we designed a private and rateless adaptive coded computation (PRAC) mechanism considering (i) the privacy requirements

of IoT applications and devices, and (ii) the heterogeneous and time-varying resources of edge devices. Our proposed PRAC model can provide adequate security and latency guarantees to support real-time computation at the edge. We showed through analysis, MATLAB simulations, and experiments on Android-based smartphones that PRAC outperforms known secure coded computing methods when resources are heterogeneous.

APPENDICES

Appendix A

PROOF OF THEOREM 3

Stability: Let $\boldsymbol{H}(t) = \{\boldsymbol{S(t)}, \boldsymbol{U(t)}, \boldsymbol{Q(t)}, \boldsymbol{Z(t)}\}$, where $\boldsymbol{S(t)} = \{S_n(t)\}_{\forall n \in \mathcal{N}}$, $\boldsymbol{U(t)} = \{U_{n,k}(t)\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}}$, $\boldsymbol{Q(t)} = \{Q_{n,k}(t)\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}}$, and $\boldsymbol{Z(t)} = \{Z_{n,k}(t)\}_{\forall n \in \mathcal{N}, k \in \mathcal{N}}$. Let the Lyapunov function be;

$$L(\boldsymbol{H}(t)) = \sum_{n \in \mathcal{N}} S_n(t)^2 + \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} U_{n,k}(t)^2 + \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Q_{n,k}(t)^2 + \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Z_{n,k}(t)^2 \tag{A.1}$$

The Lyapunov drift is;

$$\Delta(\boldsymbol{H}(t)) = E[L(\boldsymbol{H}(t+1)) - L(\boldsymbol{H}(t)) | \boldsymbol{H}(t)] \tag{A.2}$$

which is expressed as;

$$\Delta(\boldsymbol{H}(t)) = E[\sum_{n \in \mathcal{N}} S_n(t+1)^2 - \sum_{n \in \mathcal{N}} S_n(t)^2 + \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} U_{n,k}(t+1)^2 - \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} U_{n,k}(t)^2 + \\ \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Q_{n,k}(t+1)^2 - \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Q_{n,k}(t)^2 + \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Z_{n,k}(t+1)^2 - \sum_{n \in \mathcal{N}} \sum_{k \in \mathcal{N}} Z_{n,k}(t)^2 | \boldsymbol{H}(t)] \tag{A.3}$$

Appendix A (Continued)

Considering the fact that $(\max[Q-b,0]+A)^2 \leq Q^2+A^2+b^2+2Q(A-b)$, (Equation A.3) is expressed as;

$$\Delta(\boldsymbol{H}(t)) \leq E\Big[\sum_{n\in\mathcal{N}}\Big(S_n(t)^2+\Big(\sum_{k\in\mathcal{N}}x_{k,n}(t)\Big)^2+x_n(t)^2+2S_n(t)\Big(x_n(t)-\sum_{k\in\mathcal{N}}x_{k,n}(t)\Big)\Big)- \sum_{n\in\mathcal{N}}S_n(t)^2+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)^2+x_{n,k}(t)^2+d_{n,k}(t)^2+2U_{n,k}(t)\Big(x_{n,k}(t)-d_{n,k}(t)\Big)\Big)- \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}U_{n,k}(t)^2+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(Q_{n,k}(t)^2+\Big(d_{n,k}(t)\alpha_{n,k}(t)\Big)^2+e_{n,k}(t)^2+2Q_{n,k}(t)\Big(d_{n,k}(t) \alpha_{n,k}(t)-e_{n,k}(t)\Big)\Big)-\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}Q_{n,k}(t)^2+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(Z_{n,k}(t)^2+h_{n,k}(t)^2+e_{n,k}(t)^2+2Z_{n,k}(t) \Big(e_{n,k}(t)-h_{n,k}(t)\Big)\Big)-\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}Z_{n,k}(t)^2|\boldsymbol{H}(t)\Big] \tag{A.4}$$

which is expressed as

$$\Delta(\boldsymbol{H}(t)) \leq E\Big[\sum_{n\in\mathcal{N}}\Big(\Big(\sum_{k\in\mathcal{N}}x_{k,n}(t)\Big)^2+x_n(t)^2+2S_n(t)\Big(x_n(t)-\sum_{k\in\mathcal{N}}x_{k,n}(t)\Big)\Big)+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} \Big(x_{n,k}(t)^2+d_{n,k}(t)^2+2U_{n,k}(t)\Big(x_{n,k}(t)-d_{n,k}(t)\Big)\Big)+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(\Big(d_{n,k}(t)\alpha_{n,k}(t)\Big)^2+e_{n,k}(t)^2 +2Q_{n,k}(t)\Big(d_{n,k}(t)\alpha_{n,k}(t)-e_{n,k}(t)\Big)\Big)+\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(h_{n,k}(t)^2+e_{n,k}(t)^2+2Z_{n,k}(t)\Big(e_{n,k}(t)- h_{n,k}(t)\Big)\Big)|\boldsymbol{H}(t)\Big] \tag{A.5}$$

There always exists a finite positive constant B satisfying

$$B \geq E\Big[\sum_{n\in\mathcal{N}}\Big(\Big(\sum_{k\in\mathcal{N}} x_{k,n}(t)\Big)^2 + x_n(t)^2\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(x_{n,k}(t)^2 + d_{n,k}(t)^2\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} \Big(\Big(d_{n,k}(t)\alpha_{n,k}(t)\Big)^2 + e_{n,k}(t)^2\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(h_{n,k}(t)^2 + e_{n,k}(t)^2\Big)|\boldsymbol{H}(t)\Big] \tag{A.6}$$

because the maximum values of $x_n(t)$, $x_{n,k}(t)$, $d_{n,k}(t)$, $h_{n,k}(t)$, $e_{n,k}(t)$, and $\alpha_{n,k}(t)$ terms are bounded by finite positive constants by our EaCC algorithm.

By taking into account (Equation A.6), (Equation A.5) is expressed as

$$\Delta(\boldsymbol{H}(t)) \leq B + E\Big[\sum_{n\in\mathcal{N}}\Big(2S_n(t)\Big(x_n(t) - \sum_{k\in\mathcal{N}} x_{k,n}(t)\Big)\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(2U_{n,k}(t)\Big(x_{n,k}(t) - d_{n,k}(t)\Big)\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(2Q_{n,k}(t)\Big(d_{n,k}(t)\alpha_{n,k}(t) - e_{n,k}(t)\Big)\Big) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(2Z_{n,k}(t)\Big(e_{n,k}(t) - h_{n,k}(t)\Big)\Big)|\boldsymbol{H}(t)\Big] \tag{A.7}$$

The minimization of the right hand side of the drift inequality in (Equation A.7) corresponds to the decoder control in (Equation 3.8) , energy control in (Equation 5.1), and scheduling & cooperation in (Equation 3.10).

Appendix A (Continued)

If the arrival rates satisfy $E[x_n(t)] = A_n$ and (A_n) is inside the stability region Λ, then there exists a randomized policy with solution; $\overset{*}{x}_n(t)$, $\overset{*}{x}_{n,k}(t)$, $\overset{*}{d}_{n,k}(t)$, $\overset{*}{h}_{n,k}(t)$, and $\overset{*}{e}_{n,k}(t)$, satisfying

$$\begin{aligned}
&-E[\sum_{k\in\mathcal{N}} \overset{*}{x}_{k,n}(t) - \overset{*}{x}_n(t)] \leq -\delta_1, \forall n \in \mathcal{N} \\
&-E[\overset{*}{d}_{n,k}(t) - \overset{*}{x}_{n,k}(t)] \leq -\delta_2, \forall n \in \mathcal{N}, k \in \mathcal{N} \\
&-E[\overset{*}{e}_{n,k}(t) - \overset{*}{d}_{n,k}(t)\alpha_{n,k}(t)] \leq -\delta_3, \forall n \in \mathcal{N}, k \in \mathcal{N} \\
&-E[\overset{*}{h}_{n,k}(t) - \overset{*}{e}_{n,k}(t)] \leq -\delta_4, \forall n \in \mathcal{N}, k \in \mathcal{N}
\end{aligned} \tag{A.8}$$

where $\delta_1, \delta_2, \delta_3, \delta_4$ are positive small constants.

Since our EaCC algorithm minimizes the right hand side of (Equation A.7), the following inequalities satisfy: (i) $-E[\sum_{k\in\mathcal{N}} x_{k,n}(t) - x_n(t)] \leq -E[\sum_{k\in\mathcal{N}} \overset{*}{x}_{k,n}(t) - \overset{*}{x}_n(t)] \leq -\delta_1$, (ii) $-E[d_{n,k}(t) - x_{n,k}(t)] \leq -E[\overset{*}{d}_{n,k}(t) - \overset{*}{x}_{n,k}(t)] \leq -\delta_2$, (iii) $-E[e_{n,k}(t) - d_{n,k}(t)\alpha_{n,k}(t)] \leq -E[\overset{*}{e}_{n,k}(t) - \overset{*}{d}_{n,k}(t)$ $\alpha_{n,k}(t)] \leq -\delta_3$, and (iv) $-E[h_{n,k}(t) - e_{n,k}(t)] \leq -E[\overset{*}{h}_{n,k}(t) - \overset{*}{e}_{n,k}(t)] \leq -\delta_4$. Thus, the following inequality satisfy

$$\begin{aligned}
&\Delta(\boldsymbol{H}(t)) \leq B - 2\sum_{n\in\mathcal{N}} S_n(t)\delta_1 - 2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} U_{n,k}(t)\delta_2 - 2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} Q_{n,k}(t)\delta_3 - \\
&2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} Z_{n,k}(t)\delta_4
\end{aligned} \tag{A.9}$$

Since there exists $\delta > 0$ satisfying $\delta \leq \min[\delta_1, \delta_2, \delta_3, \delta_4]$, the time average of the Lyapunov drift in (Equation A.9) is expressed as

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \frac{\Delta(\boldsymbol{H}(t))}{2} \leq \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \Big(\frac{B}{2} - \sum_{n\in\mathcal{N}} S_n(t)\delta - \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} U_{n,k}(t)\delta - \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}}$$
$$Q_{n,k}(t)\delta - \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} Z_{n,k}(t)\delta \Big) \tag{A.10}$$

which leads to

$$\limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \Big(\sum_{n\in\mathcal{N}} S_n(t) + \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} \Big(U_{n,k}(t) + Q_{n,k}(t) + Z_{n,k}(t) \Big) \Big) \leq \frac{B}{2\delta} \tag{A.11}$$

concluding that the time average of the sum of the queues are bounded. This concludes the stability analysis part of the proof.

Optimality: Let us define a drift-plus-penalty function as $\Delta(\boldsymbol{H}(t)) - \sum_{k\in\mathcal{N}} ME[g_n(x_n(t))|\boldsymbol{H}(t)]$ which is, considering the bound in (Equation A.7), expressed as

$$\Delta(\boldsymbol{H}(t)) - \sum_{n\in\mathcal{N}} ME\Big[g_n(x_n(t))|\boldsymbol{H}(t)\Big] \leq B - 2E\Big[\sum_{n\in\mathcal{N}} \Big(S_n(t) \Big(\sum_{k\in\mathcal{N}} x_{k,n}(t) - x_n(t) \Big) \Big) +$$
$$\sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} \Big(U_{n,k}(t) \Big(d_{n,k}(t) - x_{n,k}(t) \Big) \Big) + \sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} \Big(Q_{n,k}(t) \Big(e_{n,k}(t) - d_{n,k}(t)\alpha_{n,k}(t) \Big) \Big) +$$
$$\sum_{n\in\mathcal{N}} \sum_{k\in\mathcal{N}} \Big(Z_{n,k}(t) \Big(h_{n,k}(t) - e_{n,k}(t) \Big) \Big) |\boldsymbol{H}(t) \Big] - \sum_{n\in\mathcal{N}} ME\Big[g_n(x_n(t))|\boldsymbol{H}(t)\Big] \tag{A.12}$$

Note that the minimization of the right hand side of the drift inequality in (Equation A.12) corresponds the flow control part of EaCC in Section 3.5 as well as the decoder control in (Equation 3.8), energy control in (Equation 3.9), and scheduling & cooperation in (Equation 3.10). Since there exists a randomized policy as discussed in the stability part above, the right hand side of (Equation A.12) is bounded as

$$\Delta(\boldsymbol{H}(t)) - \sum_{n\in\mathcal{N}} ME\Big[g_n(x_n(t))|\boldsymbol{H}(t)\Big] \leq B - 2\sum_{n\in\mathcal{N}} S_n(t)\delta - 2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)+ Q_{n,k}(t) + Z_{n,k}(t)\Big)\delta - \sum_{n\in\mathcal{N}} ME[g_n(A_n+\delta)] \tag{A.13}$$

where $\sum_{n\in\mathcal{N}} g_n(A_n)$ is the maximum time average of the sum utility function that can be achieved by any control policy that stabilizes the system. We can rewrite (Equation A.12) as

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\Delta(\boldsymbol{H}(\tau)) - \sum_{n\in\mathcal{N}} ME[g_n(x_n(\tau))]\Big] \leq \limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[B - 2\sum_{n\in\mathcal{N}} S_n(t)\delta - 2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)+Q_{n,k}(t)+Z_{n,k}(t)\Big)\delta - \sum_{n\in\mathcal{N}} MU_n(A_n+\delta)\Big] \tag{A.14}$$

Let us first consider the stability of the queues. If both sides of (Equation A.14) are divided by δ and the terms are arranged, we have

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big(\sum_{n\in\mathcal{N}} S_n(t) + \sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)+Q_{n,k}(t)+Z_{n,k}(t)\Big)\Big) \leq \frac{B}{2\delta} + \limsup_{t\to\infty} \frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{n\in\mathcal{N}}\frac{M}{\delta}E[g_n(x_n(\tau))]\Big] - \sum_{n\in\mathcal{N}}\frac{Mg_n(A_n+\delta)}{\delta}. \tag{A.15}$$

This concludes that EaCC stabilizes the queues in the system when the flow control algorithm in Section 3.5 is employed. Next, we consider the optimality of EaCC. If both sides of (Equation A.14) are divided by M, we have

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[-\sum_{n\in\mathcal{N}}E[g_n(x_n(\tau))]\Big] \leq \limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\frac{B}{M}-2\sum_{n\in\mathcal{N}}S_n(t)\frac{\delta}{M}-2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}} \Big(U_{n,k}(t)+Q_{n,k}(t)+Z_{n,k}(t)\Big)\frac{\delta}{M}-\sum_{n\in\mathcal{N}}g_n(A_n+\delta)\Big] \quad (A.16)$$

which is expressed as

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{n\in\mathcal{N}}E[g_n(x_n(\tau))]\Big] \geq \limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{n\in\mathcal{N}}g_n(A_n+\delta)-\frac{B}{M}+2\sum_{n\in\mathcal{N}}S_n(t)\frac{\delta}{M}+ 2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)+Q_{n,k}(t)+Z_{n,k}(t)\Big)\frac{\delta}{M}\Big] \quad (A.17)$$

Since $\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{n\in\mathcal{N}}2\sum_{n\in\mathcal{N}}S_n(t)\frac{\delta}{M}+2\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{N}}\Big(U_{n,k}(t)+Q_{n,k}(t)+Z_{n,k}(t)\Big)\frac{\delta}{M}\Big] \geq 0$, the following inequality holds

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\Big[\sum_{n\in\mathcal{N}}E[g_n(x_n(\tau))]\Big] \geq \sum_{n\in\mathcal{N}}g_n(A_n+\delta)-\frac{B}{M}. \quad (A.18)$$

This proves that the flow rates achieved by EaCC converge to the utility optimal operating point with increasing M. This concludes the optimality part of the proof.

Appendix B

HIDING THE VECTOR

In machine learning applications, the master runs iterative algorithms in which the vector $\mathbf{x}$ contains information about A and needs to be hidden from the workers. We describe how PRAC can be generalized to achieve privacy for both A and $\mathbf{x}$. The idea is to divide the n workers into two disjoint groups and ask each of them to privately multiply A by a vector that is statistically independent of $\mathbf{x}$. In addition, the master should be able to decode $A\mathbf{x}$ from the results of both multiplications. The scheme works as follows. The master divides the workers into two groups of cardinality n_1 and n_2 such that $n_1 + n_2 = n$ and chooses the security parameters $z_1 < n_1$ and $z_2 < n_2$. To hide $\mathbf{x}$, the master generates a random vector $\mathbf{u}$ of same size as $\mathbf{x}$ and sends $\mathbf{x} + \mathbf{u}$ to the first group and $\mathbf{u}$ to the second group. Afterwards, the master applies PRAC on both groups. According to our scheme, the master decodes $A(\mathbf{x} + \mathbf{u})$ and $A\mathbf{u}$ after receiving enough responses from the workers of each group. Hence, the master can decode $A\mathbf{x}$. Note that no information about $\mathbf{x}$ is revealed because it is one-time padded by $\mathbf{u}$. Note that here we assume workers from group 1 do not collude with workers from group 2. The same idea can be generalized to the case where workers from different groups can collude by creating more groups and encoding $\mathbf{x}$ using an appropriate secret sharing scheme. For instance, if the master divides the workers into 3 groups and workers from any 2 different groups can collude, the master encodes $\mathbf{x}$ into $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_1 + \mathbf{u}_2 + \mathbf{x}$ and sends each vector to a different group.

Appendix C

EXTENSION OF PROOF OF PRIVACY (*i.e.,* THEOREM 9)

Since at each round we generate new random matrices, it is enough to study the privacy condition at one round. Consider a given round t of PRAC. Let P_i denote the random variable representing packet p_i sent to worker w_i. For any subset $Z \subset \{1, \dots, n\}, |Z| = z$, denote by P_Z the collection of packets indexed by Z, *i.e.,* $P_Z = \{p_i; i \in Z\}$. We prove that the perfect secrecy constraint $H(A \mid P_Z) = H(A)$, given in (Equation 8.2), is equivalent to $H(K \mid P_Z, A) = 0$. The proof is standard [191–193] but we reproduce it here for completeness. In what follows, the logarithms in the entropy function are taken base q, where q is a power of prime for which all matrices can be defined in a finite field $\mathbb{F}_q$. We can write,

$$H(A \mid P_Z) = H(A) - H(P_Z) + H(P_Z \mid A) \tag{C.1}$$

$$= H(A) - H(P_Z) + H(P_Z \mid A) - H(P_Z \mid A, K) \tag{C.2}$$

$$= H(A) - H(P_Z) + I(P_Z; K \mid A) \tag{C.3}$$

$$= H(A) - H(P_Z) + H(K \mid A) - H(K \mid P_Z, A) \tag{C.4}$$

$$= H(A) - H(P_Z) + H(K) - H(K \mid P_Z, A) \tag{C.5}$$

$$= H(A) - z + z - H(K \mid P_Z, A) \tag{C.6}$$

$$= H(A) - H(K \mid P_Z, A). \tag{C.7}$$

Equation (Equation C.2) follows from the fact that given the data A and the keys $R_1, \dots, R_z$ all packets generated by the master can be decoded, in particular the packets P_Z received by any z workers can be decoded, *i.e.,* $H(P_Z \mid A, K) = 0$. Equation (Equation C.5) follows because the random matrices are chosen independently from the data matrix A and equation (Equation C.6) follows because PRAC uses z independent random matrices that are chosen uniformly at random from the field $\mathbb{F}_q$. Therefore, proving that $H(A|P_Z) = H(A)$ is equivalent to proving that $H(K \mid P_Z, A) = 0$. In other words, we need to prove that the random matrices can be decoded given the collection of packets sent to any z workers and the data matrix A. This is the main reason behind encoding the random matrices using an (n, z) MDS code. We formally prove that $H(K \mid P_Z, A) = 0$ in the proof of Theorem 9. Note from equation (Equation C.5) that for any code to be information theoretically private, $H(K)$ cannot be less then $H(P_Z) = z$. This means that a secure code must use at least z independent random matrices.

Appendix D

PROOF OF LEMMA 10

We prove the lemma by contradiction. Assume that there exists a private coded computing scheme for distributed linear computation that is secure against z colluding workers and allows the master to decode $A\mathbf{x}$ using the help of the fastest z workers. Without loss of generality, assume that the workers are ordered from the fastest to the slowest, *i.e.,* worker w_1 is the fastest and worker w_n is the slowest. The previous assumption implies that the results sent from the first z workers contain information about $A\mathbf{x}$, otherwise the master would have to wait at least for the $(z+1)^{\text{st}}$ fastest worker to decode $A\mathbf{x}$. By linearity of the multiplication $A\mathbf{x}$, decoding information about $A\mathbf{x}$ from the results of z workers implies decoding information about A from the packets sent to those z workers. Hence, there exists a set of z workers for which $H(S|P_Z) \neq 0$, where P_Z denotes the tasks allocated to a subset $Z \subset \{1, \dots, n\}$ of z workers, hence violating the privacy constraint. Therefore, any private coded computing scheme for linear computation limits the master to the speed of the $(z+1)^{\text{st}}$ fastest worker in order to decode the wanted result.

Appendix E

PROOF OF THEOREM 11

The total delay for receiving τ_i computed packets from worker w_i is equal to

$$T_i \approx RTT_i + \tau_i \mathbb{E}[\beta_{t,i}] \approx \tau_i \mathbb{E}[\beta_{t,i}]$$

where RTT_i is the average transmission delay for sending one packet to worker w_i and receiving one computed packet from the worker, $\beta_{t,i}$ is the computation time spent on multiplying packet $p_{t,i}$ by $\mathbf{x}$ at worker w_i, and the average $\mathbb{E}[\beta_{t,i}]$ is taken over all τ_i packets. The reason is that PRAC is a dynamic algorithm that sends packets to each worker w_i with the interval of $\mathbb{E}[\beta_{t,i}]$ between each two consecutive packets and it utilizes the resources of workers fully [194]. The reason behind counting only one round-trip time (RTT) in T_i is that in PRAC, the packets are being transmitted to the workers while the previously transmitted packets are being computed at the worker. Therefore, in the overall delay only one RTT_i is required for sending the first packet $p_{1,i}$ to worker w_i and receiving the last computed packet $p_{\tau_i,i}\mathbf{x}$ at the master. To approximate the total delay, we assume that the transmission delay of one packet is negligible compared to the computing delay of all τ_i packets, which is a valid assumption in practice for IoT-devices at the edge.

On the other hand, in PRAC, the master stops sending packets to workers as soon as it collectively receives $b + \epsilon$ computed packets from the $n - z$ slowest workers (note that $b + \epsilon$ is the number of computed packets required for successful decoding, where ϵ is the overhead due to Fountain Coding), *i.e.,* $\sum_{i=z+1}^{n} \tau_i = b+\epsilon$. Note that the z fastest workers are assigned for computing the keys as described in the previous sections. Due to efficiently using the resources of workers by PRAC, all $n-z$ workers will finish computing τ_i packets approximately at the same time, *i.e.,* $T_{PRAC} \approx T_i \approx \tau_i \mathbb{E}[\beta_{t,i}], i = z+1, ..., n$. By replacing τ_i with $\frac{T_{PRAC}}{\mathbb{E}[\beta_{t,i}]}$ in $\sum_{i=z+1}^{n} \tau_i = b+\epsilon$, we can show that $T_{PRAC} \approx \frac{b+\epsilon}{\sum_{i=z+1}^{n} 1/\mathbb{E}[\beta_{t,i}]}$. Note that the approximated value approaches the exact value by increasing b. The reason is that the workers' efficiency increases with increasing b.

Appendix F

PROOF OF THEOREM 12

We express $\mathbb{E}[T_{\text{SC}}]$ as a function of the computing time $\beta_{t,i}$ of worker w_i, $i = 1, \ldots, n$, as

$$\mathbb{E}[T_{\text{SC}}] = \min_{d \in \{k,\ldots,n\}} \left\{ \frac{k-z}{d-z} \mathbb{E}[T_{(d)}] \right\} \tag{F.1}$$

$$= \min_{d \in \{k,\ldots,n\}} \left\{ \frac{b}{d-z} \mathbb{E}[\beta_{t,d}] \right\}, \tag{F.2}$$

where w_d is the d^{th} fastest worker. Next, we find a lower bound on $\mathbb{E}[T_{\text{SC}}] - \mathbb{E}\left[T_{PRAC}\right]$ as follows

$$\mathbb{E}[T_{\text{SC}}] - \mathbb{E}\left[T_{PRAC}\right] = \frac{b}{\frac{d-z}{\mathbb{E}[\beta_{t,d}]}} - \frac{b+\epsilon}{\sum_{i=z+1}^{n} \frac{1}{\mathbb{E}[\beta_{t,i}]}} \tag{F.3}$$

$$= \frac{b}{\frac{d-z}{\mathbb{E}[\beta_{t,d}]}} - \frac{b+\epsilon}{\sum_{i=z+1}^{d} \frac{1}{\mathbb{E}[\beta_{t,i}]} + \sum_{i=d+1}^{n} \frac{1}{\mathbb{E}[\beta_{t,i}]}} \tag{F.4}$$

$$\geq \frac{b}{\frac{d-z}{\mathbb{E}[\beta_{t,d}]}} - \frac{b+\epsilon}{(d-z)\frac{1}{\mathbb{E}[\beta_{t,d}]} + (n-d)\frac{1}{\mathbb{E}[\beta_{t,n}]}} \tag{F.5}$$

$$= \frac{\frac{b(n-d)}{\mathbb{E}[\beta_{t,n}]} - \frac{\epsilon(d-z)}{\mathbb{E}[\beta_{t,d}]}}{\frac{d-z}{\mathbb{E}[\beta_{t,d}]}\left(\frac{d-z}{\mathbb{E}[\beta_{t,d}]} + \frac{n-d}{\mathbb{E}[\beta_{t,n}]}\right)} \tag{F.6}$$

$$= \frac{bx - \epsilon y}{y(x+y)}, \tag{F.7}$$

where $x = \frac{n-d}{\mathbb{E}[\beta_{t,n}]}$ and $y = \frac{d-z}{\mathbb{E}[\beta_{t,d}]}$ and the inequality (Equation F.5) comes from the fact that $z \leq k \leq d \leq n$ and the workers are ordered from the fastest to the slowest.

Appendix G

COPYRIGHT PERMISSIONS

This Appendix includes the copyright permissions granted from the IEEE and SPIE to use published work in thesis.

Copyright Permission from IEEE: The following statement has been copied from the CopyRight Clearance Center (Rightslink).

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you must follow the requirements listed below:

<u>Textual Material</u>

Using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line ©2011 IEEE.

In the case of illustrations or tabular material, we require that the copyright line ©[Year of original publication] IEEE appear prominently with each reprinted figure and/or table.

If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

<u>Full-Text Article</u>

If you are using the entire IEEE copyright owned article, the following IEEE copyright/ credit notice should be placed prominently in the references: ©[year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]

Appendix G (Continued)

Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.

In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Copyright Permission from SPIE: The following statement has been copied from SPIE Article-Sharing Policies (`https://www.spiedigitallibrary.org/article-sharing-policies`).

Authors may reproduce their original figures and text in new publications. The SPIE source publication should be referenced.

CITED LITERATURE

1. Usense dataset. https://crawdad.org/copelabs/usense/20170127/”.

2. Seferoglu, H. and Xing, Y.: Device-centric cooperation in mobile networks. In 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pages 217–222, Oct 2014.

3. Singh, A., Xing, Y., and Seferoglu, H.: Energy-aware cooperative computation in mobile devices. In 2016 IFIP Networking Conference (IFIP Networking) and Workshops, pages 368–376, May 2016.

4. Xing, Y. and Seferoglu, H.: Device-aware routing and scheduling in multi-hop device-to-device networks. In 2017 Information Theory and Applications Workshop (ITA), pages 1–7, Feb 2017.

5. Xing, Y. and Seferoglu, H.: Predictive edge computing with hard deadlines. In 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 13–18, June 2018.

6. Jahanian, M., Xing, Y., Chen, J., Ramakrishnan, K. K., Seferoglu, H., and Yuksel, M.: The evolving nature of disaster management in the internet and social media era. In 2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), pages 79–84, June 2018.

7. Keshtkarjahromi, Y., Xing, Y., and Seferoglu, H.: Dynamic heterogeneity-aware coded cooperative computation at the edge. In 2018 IEEE 26th International Conference on Network Protocols (ICNP), pages 23–33, Sep. 2018.

8. Bitar, R., Xing, Y., Keshtkarjahromi, Y., Dasari, V., Rouayheb, S. E., and Seferoglu, H.: Prac: private and rateless adaptive coded computation at the edge. In Proc. SPIE 11013, Disruptive Technologies in Information Sciences II, volume 11013, May. 2019.

9. Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021.

10. Cisco visual networking index: Global mobile data traffic forecast update, 2014-2019.

11. Ericsson mobility report”, Nov. 2013.

12. Asadi, A., Wang, Q., and Mancuso, V.: A survey on device-to-device communication in cellular networks. IEEE Communications Surveys Tutorials, 16(4):1801–1819, Fourthquarter 2014.

13. A multilayer application for multi-hop messaging on android devices. http://anrg.usc.edu/ee579_2012/Group02/index.html.

14. Casetti, C., Chiasserini, C.-F., Pelle, L. C., Valle, C. D., Duan, Y., and Giaccone, P.: Content-centric routing in wi-fi direct multi-group networks. http://arxiv.org/abs/1412.0880.

15. Neely, M. J.: Stochastic network optimization with application to communication and queueing systems. Morgan & Claypool, 2010.

16. Cisco visual networking index: Global mobile data traffic forecast update, 2010-2015.

17. Keller, L., Le, A., Cici, B., Seferoglu, H., Fragouli, C., and Markopoulou, A.: Microcast: Cooperative video streaming on smartphones. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, pages 57–70, New York, NY, USA, 2012. ACM.

18. Seferoglu, H., Keller, L., Cici, B., Le, A., and Markopoulou, A.: Cooperative video streaming on smartphones. In 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 220–227, Sep. 2011.

19. Tassiulas, L. and Ephremides, A.: Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. IEEE Transactions on Automatic Control, 37(12):1936–1948, Dec 1992.

20. Tassiulas, L. and Ephremides, A.: Dynamic server allocation to parallel queues with randomly varying connectivity. IEEE Transactions on Information Theory, 39(2):466–478, March 1993.

21. Neely, M. J., Modiano, E., and Li, C.: Fairness and optimal stochastic control for heterogeneous networks. IEEE/ACM Transactions on Networking, 16(2):396–409, April 2008.

22. Chiang, M., Low, S. H., Calderbank, A. R., and Doyle, J. C.: Layering as optimization decomposition: A mathematical theory of network architectures. Proceedings of the IEEE, 95(1):255–312, Jan 2007.

23. Gupta, P. and Kumar, P. R.: The capacity of wireless networks. IEEE Transactions on Information Theory, 46(2):388–404, March 2000.

24. Seferoglu, H. and Xing, Y.: Device-centric cooperation in mobile networks, tech. report. http://www.linuxjournal.com/content/queueing-linux-network-stack?page=0,2.

25. JTC1/SC29/WG11, I.: Information technology - dynamic adaptive streaming over http (dash) – part 1: Media presentation description and segment formats, 2012.

26. http://techblog.netflix.com/2010/12/html5-and-video-streaming.html.

27. Ioannidis, S., Chaintreau, A., and Massoulie, L.: Optimal and scalable distribution of content updates over a mobile social network. In IEEE INFOCOM 2009, pages 1422–1430, April 2009.

28. Han, B., Hui, P., Kumar, V. A., Marathe, M. V., Pei, G., and Srinivasan, A.: Cellular traffic offloading through opportunistic communications: A case study. In Proceedings of the 5th ACM Workshop on Challenged Networks, CHANTS '10, pages 31–38, New York, NY, USA, 2010. ACM.

29. Whitbeck, J., Amorim, M., Lopez, Y., Leguay, J., and Conan, V.: Relieving the wireless infrastructure: When opportunistic networks meet guaranteed delays. In 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, pages 1–10. IEEE, 2011.

30. Hui, P., Crowcroft, J., and Yoneki, E.: Bubble rap: Social-based forwarding in delay-tolerant networks. IEEE Transactions on Mobile Computing, 10(11):1576–1589, 2010.

31. Boldrini, C., Conti, M., and Passarella, A.: Exploiting users' social relations to forward data in opportunistic networks: The hibop solution. Pervasive and Mobile Computing, 4(5):633–657, 2008.

32. Ramadan, M., El Zein, L., and Dawy, Z.: Implementation and evaluation of cooperative video streaming for mobile devices. In 2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, pages 1–5. IEEE, 2008.

33. Li, S. and Chan, S.-H. G.: Bopper: Wireless video broadcasting with peer-to-peer error recovery. In 2007 IEEE International Conference on Multimedia and Expo, pages 392–395. IEEE, 2007.

34. Xiaojun Lin, Shroff, N. B., and Srikant, R.: A tutorial on cross-layer optimization in wireless networks. IEEE Journal on Selected Areas in Communications, 24(8):1452–1463, Aug 2006.

35. Li, C. and Modiano, E.: Receiver-based flow control for networks in overload. IEEE/ACM Transactions on Networking, 23(2):616–630, April 2015.

36. Wi-fi direct. http://www.wi-fi.org/discover-and-learn/wi-fi-direct.

37. Android. http://developer.android.com/develop/index.html.

38. Nexus tech specs. https://support.google.com/nexus/answer/610 2470?hl=en.

39. Rappaport, T., Sun, S., Mayzus, R., Zhao, H., Azar, Y., Wang, K., Wong, G., Schulz, J., Samimi, M., and Gutierrez, F.: Millimeter wave mobile communications for 5g cellular: It will work! Access, IEEE, 1:335–349, 2013.

40. Ostermann, J., Bormans, J., List, P., Marpe, D., Narroschke, M., Pereira, F., Stockhammer, T., and Wedi, T.: Video coding with h.264/avc: tools, performance, and complexity. Circuits and Systems Magazine, IEEE, 4(1):7–28, First 2004.

41. Horowitz, M., Joch, A., Kossentini, F., and Hallapuro, A.: H.264/avc baseline profile decoder complexity analysis. Circuits and Systems for Video Technology, IEEE Transactions on, 13(7):704–716, July 2003.

42. Sundararajan, J. K., Shah, D., Jakubczak, M., Mitzenmacher, M., and Barros, J.: Network coding meets tcp: Theory and implementation. Proceedings of the IEEE, pages 490–512, March 2011.

43. Vingelmann, P., Zanaty, P., Fitzek, F., and Charaf, H.: Implementation of random linear network coding on opengl-enabled graphics cards. In Wireless Conference, 2009. EW 2009. European, pages 118–123, May 2009.

44. Shojania, H., Li, B., and Wang, X.: Nuclei: Gpu-accelerated many-core network coding. In INFOCOM 2009, IEEE, pages 459–467, April 2009.

45. Arora, S. and Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

46. Syrivelis, D., Iosifidis, G., Delimpasis, D., Chounos, K., Korakis, T., and Tassiulas, L.: Bits and coins: Supporting collaborative consumption of mobile internet. In Proc. IEEE Infocom, Hong Kong, April 2015.

47. Asadi, A., Wang, Q., and Mancuso, V.: A survey on device-to-device communication in cellular networks. Technical report - arxiv:1310.0720v6[cs.GT], April 2014.

48. Chesterfield, J., Chakravorty, R., Pratt, I., Banerjee, S., and Rodriguez, P.: Exploiting diversity to enhance multimedia streaming over cellular links. In Proc. of IEEE INFOCOM, March 2005.

49. Chesterfield, J., Chakravorty, R., Pratt, I., Banerjee, S., and Rodriguez, P.: A system for peer-to-peer video streaming in resource constrained mobile environments. In Proc. of ACM U-NET, December 2009.

50. Lomotey, R. K. and Deters, R.: Architectural designs from mobile cloud computing to ubiquitous cloud computing - survey. In Proc. IEEE Services, Anchorage, Alaska, June 2014.

51. Bonomi, F., Milito, R., Zhu, J., and Addepalli, S.: Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

52. Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V.: Mobile edge computing?a key technology towards 5g. ETSI white paper, 11(11):1–16, 2015.

53. Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L.: Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5):637–646, Oct 2016.

54. Penner, T., Johnson, A., Slyke, B. V., Guirguis, M., and Gu, Q.: Transient clouds: Assignment and collaborative execution of tasks on mobile devices. In Proc. IEEE GLOBECOM, Austin, TX, December 2014.

55. Satyanarayanan, M., Smaldone, S., Gilbert, B., Harkes, J., and Iftode, L.: Bringing the cloud down to earth: Transient pcs everywhere. In MobiCASE'10, pages 315–322, 2010.

56. Miluzzo, E., Caceres, R., and Chen, Y.: Vision: mclouds - computing on clouds of mobile devices. In ACM workshop on Mobile cloud computing and services, Low Wodd Bay, Lake District, UK, June 2012.

57. Android transcoder. https://github.com/ypresto/android-transcoder.

58. Boilsoft video splitter, January 2017. http://www.boilsoft.com/videosplitter/.

59. Cisco visual networking index: Global mobile data traffic forecast update, 2016-2021.

60. Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B.: A survey of computation offloading for mobile systems. Mob. Netw. Appl., 18(1):129–140, February 2013.

61. Dinh, H. T., Lee, C., Niyato, D., and Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wireless Communications and Mobile Computing, 13(8), October 2011.

62. Fernando, N., Loke, S. W., and Rahayu, W.: Mobile cloud computing: A survey. Future Generation Computer Systems, 29(1):84 – 106, 2013.

63. Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R.: Heterogeneity in mobile cloud computing: Taxonomy and open challenges. Communications Surveys Tutorials, IEEE, 16(1):369–392, First 2014.

64. Khan, A., Othman, M., Madani, S., and Khan, S.: A survey of mobile cloud computing application models. Communications Surveys Tutorials, IEEE, 16(1):393–413, First 2014.

65. Lei, L., Zhong, Z., Zheng, K., Chen, J., and Meng, H.: Challenges on wireless heterogeneous networks for mobile cloud computing. Wireless Communications, IEEE, 20(3):34–44, June 2013.

66. Gordon, M., Jamshidi, D., Mahlke, S., Mao, Z., , and Chen, X.: Comet: Code offload by migrating execution transparently. In Proc. OSDI, Hollywood, CA, October 2012.

67. Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P.: Maui: Making smartphones last longer with code offload. In Proc. ACM MobiSys, San Francisco, CA, June 2010.

68. Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S.: Refactoring android java code for on-demand computation offloading. In OOPSLA, Tuscon, AZ, October 2012.

69. Kemp, R., Palmer, N., Kielmann, T., and Bal, H.: Cuckoo: A computation offloading framework for smartphones. Mobile Computing, Applications, and Services, 76:59–79, 2012.

70. Hoang, D. T., Niyato, D., and Wang, P.: Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In Wireless Communications and Networking Conference (WCNC), 2012 IEEE, pages 3145–3149, April 2012.

71. Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X.: Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In INFOCOM, 2012 Proceedings IEEE, pages 945–953, March 2012.

72. Geng, Y., Hu, W., Yang, Y., Gao, W., and Cao, G.: Energy-efficient computation offloading in cellular networks. In 2015 IEEE 23rd International Conference on Network Protocols (ICNP), pages 145–155, Nov 2015.

73. Zhang, W., Wen, Y., and Wu, D. O.: Energy-efficient scheduling policy for collaborative execution in mobile cloud computing. In INFOCOM, 2013 Proceedings IEEE, pages 190–194, April 2013.

74. Li, Y. and Wang, W.: Can mobile cloudlets support mobile applications? In IEEE INFOCOM, pages 1060–1068, April 2014.

75. Chen, M., Hao, Y., Li, Y., Lai, C. F., and Wu, D.: On the computation offloading at ad hoc cloudlet: architecture and service modes. IEEE Communications Magazine, 53(6):18–24, June 2015.

76. Ferrari, A., Giordano, S., and Puccinelli, D.: Reducing your local footprint with anyrun computing. Computer Communications, 81:1 – 11, 2016.

77. Meng, X., Wang, W., and Zhang, Z.: Delay-constrained hybrid computation offloading with cloud and fog computing. IEEE Access, 5:21355–21367, 2017.

78. Fan, J., Wei, X., Wang, T., Lan, T., and Subramaniam, S.: Deadline-aware task scheduling in a tiered iot infrastructure. In GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pages 1–7, Dec 2017.

79. Zhou, C. and Tham, C.: Deadline-aware peer-to-peer task offloading in stochastic mobile cloud computing systems. In 2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), pages 1–9, June 2018.

80. Wikipedia: Hurricane Harvey. `https://en.wikipedia.org/wiki/Hurricane_Harvey`.

81. Stelter, B.: How social media is helping Houston deal with Harvey floods. https://tinyurl.com/y9v3zyfa.

82. Yoshitsugu, Y.: Roles of social media at the time of major disasters observed in the great east Japan earthquake:. https://www.nhk.or.jp/bunken/english/reports/summary/201107/02.html, July 2011.

83. Wikipedia: Hurricane Irma. https://en.wikipedia.org/wiki/Hurricane_Irma.

84. Myers, A. L.: Tropical Storm Harvey victims use social media when 911 fails . https://tinyurl.com/y955ryeq.

85. Meadows-Fernandez, R.: What Harvey and Irma Taught Us About Using Social Media in Emergency Response. https://tinyurl.com/yba29o5z.

86. MacMillan, D.: In Irma, Emergency Responders' New Tools: Twitter and Facebook . https://tinyurl.com/y8znmk93.

87. Palen, L.: Online social media in crisis events. Educause Quarterly, 31(3):76–78, 2008.

88. Utz, S., Schultz, F., and Glocka, S.: Crisis communication online: How medium, crisis type and emotions affected public reactions in the fukushima daiichi nuclear disaster. Public Relations Review, 39(1):40–46, 2013.

89. Gao, H., Barbier, G., and Goolsby, R.: Harnessing the crowdsourcing power of social media for disaster relief. IEEE Intelligent Systems, 26(3):10–14, 2011.

90. Matias, Y.: Helping people in a crisis. https://blog.google/products/search/helping-people-crisis/.

91. Google: Person finder help. https://support.google.com/personfinder.

92. Google: Crisis map help. https://support.google.com/crisismaps.

93. Google: Public alerts help. https://support.google.com/publicalerts/.

94. Facebook: Disaster response faq. https://www.facebook.com/help/141874516227713.

95. Maas, P.: Facebook disaster maps: Methodology. https://research.fb.com/facebook-disaster-maps-methodology/.

96. Chen, J., Arumaithurai, M., Fu, X., and Ramakrishnan, K.: CNS: content-oriented notification service for managing disasters. In ICN, 2016.

97. Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L.: Networking Named Content. In CoNEXT, 2009.

98. Wikipedia: Timeline of relief efforts after the 2010 Haiti earthquake. https://tinyurl.com/yczzrx93.

99. datahub: Haiti Crisis Map. datahub.io/dataset/ushahidi/resource/81d058a8-173a-49d9-8ce9-4edf5e7cafc9.

100. FCC: Hurricane Harvey. https://www.fcc.gov/harvey.

101. FCC: Hurricane Irma. https://www.fcc.gov/irma.

102. Apache Lucene. https://lucene.apache.org/.

103. Amadeo, K.: Hurricane Irma: Facts, Damage, and Costs . https://www.thebalance.com/hurricane-irma-facts-timeline-damage-costs-4150395.

104. World Vision: Hurricane Irma: Facts, FAQs, and how to help. https://www.worldvision.org/disaster-relief-news-stories/hurricane-irma-facts.

105. Energy Information Administration: Hurricane Irma cut power to nearly two-thirds of Florida's electricity customers. https://www.eia.gov/todayinenergy/detail.php?id=32992.

106. World Population Review: Florida Population 2018. http://worldpopulationreview.com/states/florida-population/.

107. Florida Department of Management Services: Florida E911 Plan. https://www.dms.myflorida.com/business_operations/telecommunications/enhanced_911/florida_e911_plan.

108. Liu, J., Kato, N., Ma, J., and Kadowaki, N.: Device-to-device communication in lte-advanced networks: A survey. IEEE Communications Surveys Tutorials, 17(4):1923–1940, Fourthquarter 2015.

109. Khabbaz, M. J., Assi, C. M., and Fawaz, W. F.: Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges. IEEE Comm. Sur. & Tut., 14(2):607–640, 2012.

110. Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K.: Speeding up distributed machine learning using codes. IEEE Transactions on Information Theory, 64(3):1514–1529, March 2018.

111. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. N.: Learning to rank using gradient descent. In Proceedings of the 22nd International Conference on Machine learning (ICML-05), pages 89–96, 2005.

112. Zhang, T.: Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning, page 116. ACM, 2004.

113. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010, pages 177–186. Springer, 2010.

114. Reisizadeh, A., Prakash, S., Pedarsani, R., and Avestimehr, S.: Coded computation over heterogeneous clusters. In 2017 IEEE International Symposium on Information Theory (ISIT), pages 2408–2412. IEEE, 2017.

115. Luby, M.: Lt codes. In The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., pages 271–280. IEEE, 2002.

116. Shokrollahi, A.: Raptor codes. IEEE TRANSACTIONS ON INFORMATION THEORY, 52(6):2551, 2006.

117. MacKay, D. J.: Fountain codes. IEE Proceedings-Communications, 152(6):1062–1068, 2005.

118. Ho, T., Koetter, R., Medard, M., Karger, D., and Effros, M.: The benefits of coding over routing in a randomized setting. In IEEE International Symposium on Information Theory, 2003. Proceedings., page 442. IEEE, 2003.

119. Lin, S., Costello, D. J., and Miller, M. J.: Automatic-repeat-request error-control schemes. IEEE Communications magazine, 22(12):5–17, 1984.

120. Dinh, H. T., Lee, C., Niyato, D., and Wang, P.: A survey of mobile cloud computing: architecture, applications, and approaches. Wireless communications and mobile computing, 13(18):1587–1611, 2013.

121. Fernando, N., Loke, S. W., and Rahayu, W.: Mobile cloud computing: A survey. Future generation computer systems, 29(1):84–106, 2013.

122. Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R.: Heterogeneity in mobile cloud computing: Taxonomy and open challenges. IEEE COMMUNICATIONS SURVEYS & TUTORIALS, 16(1), 2014.

123. Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X.: {COMET}: Code offload by migrating execution transparently. In Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pages 93–106, 2012.

124. Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P.: Maui: making smartphones last longer with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services, pages 49–62. ACM, 2010.

125. Kemp, R., Palmer, N., Kielmann, T., and Bal, H.: Cuckoo: a computation offloading framework for smartphones. In International Conference on Mobile Computing, Applications, and Services, pages 59–79. Springer, 2010.

126. Hoang, D. T., Niyato, D., and Wang, P.: Optimal admission control policy for mobile cloud computing hotspot with cloudlet. In 2012 IEEE Wireless Communications and Networking Conference (WCNC), pages 3145–3149, April 2012.

127. Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X.: Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In 2012 Proceedings IEEE Infocom, pages 945–953. IEEE, 2012.

128. Lomotey, R. K. and Deters, R.: Architectural designs from mobile cloud computing to ubiquitous cloud computing-survey. In 2014 IEEE World Congress on Services, pages 418–425. IEEE, 2014.

129. Miluzzo, E., Cáceres, R., and Chen, Y.-F.: Vision: mclouds-computing on clouds of mobile devices. In Proceedings of the third ACM workshop on Mobile cloud computing and services, pages 9–14. ACM, 2012.

130. Penner, T., Johnson, A., Van Slyke, B., Guirguis, M., and Gu, Q.: Transient clouds: Assignment and collaborative execution of tasks on mobile devices. In 2014 IEEE Global Communications Conference, pages 2801–2806. IEEE, 2014.

131. Dean, J. and Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

132. Li, S., Maddah-Ali, M. A., and Avestimehr, A. S.: Coded mapreduce. In 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), pages 964–971. IEEE, 2015.

133. Dutta, S., Cadambe, V., and Grover, P.: Short-dot: Computing large linear transforms distributedly using coded short dot products. In Advances In Neural Information Processing Systems, pages 2100–2108, 2016.

134. Mallick, A., Chaudhari, M., and Joshi, G.: Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication. arXiv preprint arXiv:1804.10331, 2018.

135. Bitar, R., Parag, P., and El Rouayheb, S.: Minimizing latency for secure distributed computing. In Information Theory (ISIT), 2017 IEEE International Symposium on, pages 2900–2904. IEEE, 2017.

136. Li, S., Maddah-Ali, M. A., and Avestimehr, A. S.: A unified coding framework for distributed computing with straggling servers. In Globecom Workshops (GC Wkshps), 2016 IEEE, pages 1–6. IEEE, 2016.

137. Dutta, S., Cadambe, V., and Grover, P.: Coded convolution for parallel and distributed computing within a deadline. arXiv preprint arXiv:1705.03875, 2017.

138. Yang, Y., Grover, P., and Kar, S.: Computing linear transformations with unreliable components. IEEE Transactions on Information Theory, 2017.

139. Halbawi, W., Azizan-Ruhi, N., Salehi, F., and Hassibi, B.: Improving distributed gradient descent using reed-solomon codes. arXiv preprint arXiv:1706.05436, 2017.

140. Yu, Q., Maddah-Ali, M., and Avestimehr, S.: Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In Advances in Neural Information Processing Systems, pages 4403–4413, 2017.

141. Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N.: Gradient coding: Avoiding stragglers in distributed learning. In International Conference on Machine Learning, pages 3368–3376, 2017.

142. Li, S., Maddah-Ali, M. A., and Avestimehr, A. S.: Fundamental tradeoff between computation and communication in distributed computing. In IEEE International Symposium on Information Theory (ISIT), 2016.

143. Karakus, C., Sun, Y., Diggavi, S., and Yin, W.: Straggler mitigation in distributed optimization through data encoding. In Advances in Neural Information Processing Systems, pages 5434–5442, 2017.

144. Aktas, M. F., Peng, P., and Soljanin, E.: Effective straggler mitigation: Which clones should attack and when? ACM SIGMETRICS Performance Evaluation Review, 45(2):12–14, 2017.

145. Lin, S. and Costello, D.: Error-Correcting Codes. Prentice-Hall, Inc, 1983.

146. MacWilliams, F. J. and Sloane, N. J. A.: The theory of error-correcting codes. Elsevier, 1977.

147. Seber, G. A. and Lee, A. J.: Linear regression analysis, volume 329. John Wiley & Sons, 2012.

148. Suykens, J. A. and Vandewalle, J.: Least squares support vector machine classifiers. Neural processing letters, 9(3):293–300, 1999.

149. Lacan, J., Roca, V., Peltotalo, J., and Peltotalo, S.: Reed-solomon forward error correction (fec) schemes. Technical report, 2009.

150. Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., and Stoica, I.: Improving mapreduce performance in heterogeneous environments. In Osdi, volume 8, page 7, 2008.

151. Dean, J. and Barroso, L. A.: The tail at scale. Communications of the ACM, 56(2):74–80, 2013.

152. Chen, J., Monga, R., Bengio, S., and Jozefowicz, R.: Revisiting distributed synchronous sgd. arXiv preprint arXiv:1604.00981, 2016.

153. Ananthanarayanan, G., Kandula, S., Greenberg, A. G., Stoica, I., Lu, Y., Saha, B., and Harris, E.: Reining in the outliers in map-reduce clusters using mantri. In OSDI, volume 10, page 24, 2010.

154. Narayanamurthy, S., Weimer, M., Mahajan, D., Condie, T., Sellamanickam, S., and Keerthi, S. S.: Towards resource-elastic machine learning. In NIPS 2013 BigLearn Workshop, 2013.

155. Recht, B., Re, C., Wright, S., and Niu, F.: Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pages 693–701, 2011.

156. Mitliagkas, I., Zhang, C., Hadjis, S., and Ré, C.: Asynchrony begets momentum, with an application to deep learning. In Communication, Control, and Computing (Allerton), 2016 54th Annual Allerton Conference on, pages 997–1004. IEEE, 2016.

157. Recht, B., Re, C., Tropp, J., and Bittorf, V.: Factoring nonnegative matrices with linear programs. In Advances in Neural Information Processing Systems, pages 1214–1222, 2012.

158. Zhuang, Y., Chin, W.-S., Juan, Y.-C., and Lin, C.-J.: A fast parallel sgd for matrix factorization in shared memory systems. In Proceedings of the 7th ACM conference on Recommender systems, pages 249–256. ACM, 2013.

159. Yun, H., Yu, H.-F., Hsieh, C.-J., Vishwanathan, S., and Dhillon, I.: Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. Proceedings of the VLDB Endowment, 7(11):975–986, 2014.

160. Liu, J., Wright, S. J., Ré, C., Bittorf, V., and Sridhar, S.: An asynchronous parallel stochastic coordinate descent algorithm. Journal of Machine Learning Research, 16(285-322):1–5, 2015.

161. Duchi, J., Jordan, M. I., and McMahan, B.: Estimation, optimization, and parallelism when data is sparse. In Advances in Neural Information Processing Systems, pages 2832–2840, 2013.

162. Wang, Y.-x., Sadhanala, V., Dai, W., Neiswanger, W., Sra, S., and Xing, E. P.: Asynchronous parallel block-coordinate frank-wolfe. stat, 1050:22, 2014.

163. Hsieh, C.-J., Yu, H.-F., and Dhillon, I. S.: Passcode: Parallel asynchronous stochastic dual coordinate descent. In ICML, volume 15, pages 2370–2379, 2015.

164. Mania, H., Pan, X., Papailiopoulos, D., Recht, B., Ramchandran, K., and Jordan, M. I.: Perturbed iterate analysis for asynchronous stochastic optimization. arXiv preprint arXiv:1507.06970, 2015.

165. Chilimbi, T. M., Suzue, Y., Apacible, J., and Kalyanaraman, K.: Project adam: Building an efficient and scalable deep learning training system. In OSDI, volume 14, pages 571–582, 2014.

166. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al.: Large scale distributed deep networks. In Advances in neural information processing systems, pages 1223–1231, 2012.

167. Huang, L., Pawar, S., Zhang, H., and Ramchandran, K.: Codes can reduce queueing delay in data centers. In IEEE International Symposium on Information Theory (ISIT), 2012.

168. Joshi, G., Liu, Y., and Soljanin, E.: Coding for fast content download. In 50th Annual Allerton Conference on Communication, Control, and Computing, 2012.

169. Liang, G. and Kozat, U. C.: TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes. In IEEE International Conference on Computer Communications, 2014.

170. Kadhe, S., Soljanin, E., and Sprintson, A.: Analyzing the download time of availability codes. In IEEE International Symposium on Information Theory (ISIT), 2015.

171. Peng, P. and Soljanin, E.: On distributed storage allocations of large files for maximum service rate. arXiv preprint, arXiv:1808.07545, 2018.

172. Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S.: Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. arXiv preprint, arXiv:1801.07487, 2018.

173. Maity, R. K., Rawat, A. S., and Mazumdar, A.: Robust gradient descent via moment encoding with ldpc codes. arXiv preprint arXiv:1805.08327, 2018.

174. Chen, L., Charles, Z., Papailiopoulos, D., et al.: Draco: Robust distributed training via redundant gradients. arXiv preprint arXiv:1803.09877, 2018.

175. Kiani, S., Ferdinand, N., and Draper, S. C.: Exploitation of stragglers in coded computation. In 2018 IEEE International Symposium on Information Theory (ISIT), pages 1988–1992. IEEE, 2018.

176. Ozfaturay, E., Gunduz, D., and Ulukus, S.: Speeding up distributed gradient descent by utilizing non-persistent stragglers. arXiv preprint arXiv:1808.02240, 2018.

177. Ye, M. and Abbe, E.: Communication-computation efficient gradient coding. arXiv preprint arXiv:1802.03475, 2018.

178. Ferdinand, N. and Draper, S.: Anytime stochastic gradient descent: A time to hear from all the workers. arXiv preprint arXiv:1810.02976, 2018.

179. Li, S., Kalan, S. M. M., Avestimehr, A. S., and Soltanolkotabi, M.: Near-optimal straggler mitigation for distributed gradient methods. In 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 857–866. IEEE, 2018.

180. Raviv, N., Tamo, I., Tandon, R., and Dimakis, A. G.: Gradient coding from cyclic mds codes and expander graphs. arXiv preprint arXiv:1707.03858, 2017.

181. Sheth, U., Dutta, S., Chaudhari, M., Jeong, H., Yang, Y., Kohonen, J., Roos, T., and Grover, P.: An application of storage-optimal matdot codes for coded matrix multiplication: Fast k-nearest neighbors estimation. arXiv preprint, arXiv:1811.11811, 2018.

182. D'Oliveira, R. G., Rouayheb, S. E., and Karpuk, D.: Gasp codes for secure distributed matrix multiplication. arXiv preprint arXiv:1812.09962, 2018.

183. Yu, Q., Li, S., Raviv, N., Kalan, S. M. M., Soltanolkotabi, M., and Avestimehr, S.: Lagrange coded computing: Optimal design for resiliency, security and privacy. arXiv preprint arXiv:1806.00939v4, 2019.

184. Chang, W.-T. and Tandon, R.: On the capacity of secure distributed matrix multiplication. In 2018 IEEE Global Communications Conference (GLOBECOM), pages 1–6. IEEE, 2018.

185. Kakar, J., Ebadifar, S., and Sezgin, A.: Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation. arXiv preprint arXiv:1810.13006, 2018.

186. Yang, H. and Lee, J.: Secure distributed computing with straggling servers using polynomial codes. IEEE Transactions on Information Forensics and Security, 14(1):141–150, 2018.

187. Yu, Q., Raviv, N., So, J., and Avestimehr, A. S.: Lagrange coded computing: Optimal design for resiliency, security and privacy. arXiv preprint, arXiv:1806.00939, 2018.

188. Takabi, H., Hesamifard, E., and Ghasemi, M.: Privacy preserving multi-party machine learning with homomorphic encryption. In 29th Annual Conference on Neural Information Processing Systems (NIPS), 2016.

189. Hall, R., Fienberg, S. E., and Nardi, Y.: Secure multiple linear regression based on homomorphic encryption. Journal of Official Statistics, 27(4):669, 2011.

190. Gade, S. and Vaidya, N. H.: Private learning on networks: Part ii. arXiv preprint arXiv:1703.09185, 2017.

191. Shah, N. B., Rashmi, K. V., and Kumar, P. V.: Information-theoretically secure regenerating codes for distributed storage. In Proc. IEEE Global Communications Conference, 2011.

192. Rouayheb, S. E., Soljanin, E., and Sprintson, A.: Secure network coding for wiretap networks of type II. IEEE Transactions on Information Theory, 58(3):1361–1371, March 2012.

193. Bitar, R. and El Rouayheb, S.: Staircase codes for secret sharing with optimal communication and read overheads. IEEE Transactions on Information Theory, PP(99):1–1, 2017.

194. Keshtkarjahromi, Y., Xing, Y., and Seferoglu, H.: Dynamic heterogeneity-aware coded cooperative computation at the edge. arXiv preprint, rXiv:1801.04357v3, 2018.

VITA

YUXUAN XING

Education	Ph.D. Electrical and Computer Engineering University of Illinois at Chicago	2012 – 2019
	B.Eng. Communication Engineering North China Electric Power University	2008 – 2012

Publications

Rawad Bitar, Yuxuan Xing, Yasaman Keshtkarjahromi, Venkat Dasari, Salim El Rouayheb, Hulya Seferoglu, “PRAC: Private and Rateless Adaptive Coded Computation at the Edge,” *in Proc. of SPIE*, 2019.

Yasaman Keshtkarjahromi, Yuxuan Xing, Hulya Seferoglu, “Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge,” *in Proc. of IEEE ICNP*, 2018.

Mohammad Jahanian, Yuxuan Xing, Jiachen Chen, K. K. Ramakrishnan, Hulya Seferoglu, Murat Yuksel, “The Evolving Nature of Disaster Management in the Internet and Social Media Era,” *in Proc. of IEEE LANMAN*, 2018.

Yuxuan Xing, Hulya Seferoglu, “Predictive Edge Computing with Hard Deadlines,” *in Proc. of IEEE LANMAN*, 2018.

Yuxuan Xing, Hulya Seferoglu, “Device-aware routing and scheduling in multi-hop Device-to-Device networks,” *in Proc. of IEEE ITA Workshop*, 2017.

Ajita Singh, Yuxuan Xing, Hulya Seferoglu, “Energy-aware cooperative computation in mobile devices,” *in Proc. of IEEE IFIP Networking Conference*, 2016.

Hulya Seferoglu, Yuxuan Xing, “Device-Centric Cooperation in Mobile Networks,” *in Proc. of IEEE CloudNet*, 2014.

Yuxuan Xing, Ajita Singh and Hulya Seferoglu, “Energy-aware cooperative computation in mobile devices,” *IEEE Transactions on Mobile Computing*, under submission.

	Yuxuan Xing, Shanyu Zhou and Hulya Seferoglu, “Predictive Edge Computing with Hard Deadlines,” *IEEE/ACM Transactions on Networking*, under submission.
	Rawad Bitar, Yuxuan Xing, Yasaman Keshtkarjahromi, Venkat Dasari, Salim El Rouayheb, and Hulya Seferoglu, “Device-Centric Cooperation in Mobile Networks,” *IEEE Transactions on Information Forensics and Security*, under submission.
Presentations	“Predictive Edge Computing with Hard Deadlines” IEEE LANMAN, Washington DC, Jun. 2018.
	“Modeling and Development of Resilient Communication Demo ”, PSCR, San Diego, CA, Jun. 2018.
	“ReDiCom: Resilient Communications for Dynamic First Responder Teams in Disaster Management Demo”, PSCR, Chicago, IL, JuL. 2019.
Experience	Research Assistant at University of Illinois at Chicago (Jan. 2014 - Oct. 2019)