

Summer 2018

## Enhancing Usability, Security, and Performance in Mobile Computing

Shanhe Yi

College of William and Mary - Arts & Sciences, yishanhe0203@gmail.com

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Yi, Shanhe, "Enhancing Usability, Security, and Performance in Mobile Computing" (2018). *Dissertations, Theses, and Masters Projects*. Paper 1530192793.

<http://dx.doi.org/10.21220/s2-pka0-2q70>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

Enhancing Usability, Security, and Performance in Mobile Computing

Shanhe Yi

Xiangyang, Hubei, China

Master of Science, Huazhong University of Science and Technology, 2013  
Bachelor of Engineering, Huazhong University of Science and Technology, 2010

A Dissertation presented to the Graduate Faculty  
of The College of William & Mary in Candidacy for the Degree of  
Doctor of Philosophy

Department of Computer Science

College of William & Mary  
May, 2018



## APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

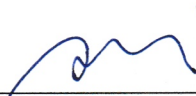
Doctor of Philosophy



---

Shanhe Yi

Approved by the Committee, May 2018



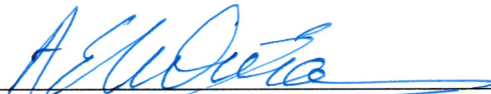
---

Committee Chair  
Professor Qun Li, Computer Science  
The College of William and Mary



---

Professor Weizhen Mao, Computer Science  
The College of William and Mary



---

Professor Andreas Stathopoulos, Computer Science  
The College of William and Mary



---

Assistant Professor Dmitry Evtushkin, Computer Science  
The College of William and Mary



---

Associate Professor Kai Zeng, Electrical and Computer Engineering  
George Mason University

## ABSTRACT

We have witnessed the prevalence of smart devices in every aspect of human life. However, the ever-growing smart devices present significant challenges in terms of usability, security, and performance. First, we need to design new interfaces to improve the device usability which has been neglected during the rapid shift from hand-held mobile devices to wearables. Second, we need to protect smart devices with abundant private data against unauthorized users. Last, new applications with compute-intensive tasks demand the integration of emerging mobile backend infrastructure. This dissertation focuses on addressing these challenges.

First, we present GlassGesture, a system that improves the usability of Google Glass through a head gesture user interface with gesture recognition and authentication. We accelerate the recognition by employing a novel similarity search scheme, and improve the authentication performance by applying new features of head movements in an ensemble learning method. As a result, GlassGesture achieves around 96% gesture recognition accuracy. Furthermore, GlassGesture accepts authorized users in nearly 92% of trials, and rejects attackers in nearly 99% of trials.

Next, we investigate the authentication between a smartphone and a paired smartwatch. We design and implement WearLock, a system that utilizes one's smartwatch to unlock one's smartphone via acoustic tones. We build an acoustic modem with sub-channel selection and adaptive modulation, which generates modulated acoustic signals to maximize the unlocking success rate against ambient noise. We leverage the motion similarities of the devices to eliminate unnecessary unlocking. We also offload heavy computation tasks from the smartwatch to the smartphone to shorten response time and save energy. The acoustic modem achieves a low bit error rate (BER) of 8%. Compared to traditional manual personal identification numbers (PINs) entry, WearLock not only automates the unlocking but also speeds it up by at least 18%.

Last, we consider low-latency video analytics on mobile devices, leveraging emerging mobile backend infrastructure. We design and implement LAVEA, a system which offloads computation from mobile clients to edge nodes, to accomplish tasks with intensive computation at places closer to users in a timely manner. We formulate an optimization problem for offloading task selection and prioritize offloading requests received at the edge node to minimize the response time. We design and compare various task placement schemes for inter-edge collaboration to further improve the overall response time. Our results show that the client-edge configuration has a speedup ranging from 1.3x to 4x against running solely by the client and 1.2x to 1.7x against the client-cloud configuration.

## TABLE OF CONTENTS

|   |      |
|---|------|
| Acknowledgments   | vii  |
| Dedication  | viii |
| List of Tables  | ix   |
| List of Figures   | x    |
| 1 Introduction  | 1    |
| 1.1 Challenges . . . . .  | 2    |
| 1.2 Head Gesture Interface of Smart Glasses . . . . .             | 4    |
| 1.2.1 Problem Statements . . . . .                                | 4    |
| 1.2.2 Contributions . . . . .                                     | 5    |
| 1.3 Smartwatch-assisted Smartphone Authentication . . . . .       | 5    |
| 1.3.1 Problem Statements . . . . .                                | 6    |
| 1.3.2 Contributions . . . . .                                     | 7    |
| 1.4 Edge Computing based Mobile Backend . . . . .                 | 7    |
| 1.4.1 Problem Statements . . . . .                                | 8    |
| 1.4.2 Contributions . . . . .                                     | 9    |
| 1.5 Dissertation Organization . . . . .                           | 9    |
| 2 GlassGesture: Exploring Head Gesture Interface of Smart Glasses | 10   |
| 2.1 Introduction . . . . .  | 10   |
| 2.2 Related Work . . . . .  | 14   |

|         |   |    |
|---------|---|----|
| 2.3     | GlassGesture System Design . . . . .              | 15 |
| 2.3.1   | System Overview . . . . .                         | 15 |
| 2.3.2   | Head Gesture Recognition . . . . .                | 16 |
| 2.3.2.1 | Head Gesture Library . . . . .                    | 17 |
| 2.3.2.2 | Activity Detector . . . . .                       | 19 |
| 2.3.2.3 | Gesture Detector . . . . .                        | 19 |
| 2.3.2.4 | Gesture Recognizer . . . . .                      | 20 |
| 2.3.2.5 | Efficient Similarity Search . . . . .             | 22 |
| 2.3.3   | Head-Gesture-based Authentication . . . . .       | 24 |
| 2.3.3.1 | Two-factor Authentication using Head Gesture . .  | 24 |
| 2.3.3.2 | Threat Model . . . . .                            | 25 |
| 2.3.3.3 | Authentication Setup . . . . .                    | 26 |
| 2.3.3.4 | Feature Set and Data Collection . . . . .         | 26 |
| 2.3.3.5 | Training and Classification . . . . .             | 27 |
| 2.3.3.6 | Feature Selection . . . . .                       | 28 |
| 2.4     | Evaluation . . . . .                              | 29 |
| 2.4.1   | Gesture Recognition . . . . .                     | 29 |
| 2.4.1.1 | Gesture Recognition with command gestures . .     | 29 |
| 2.4.1.2 | Number and Alphabet Input . . . . .               | 30 |
| 2.4.1.3 | Gesture Recognition Performance . . . . .         | 31 |
| 2.4.2   | Authentication Evaluation . . . . .               | 32 |
| 2.4.2.1 | Impact of Number of Training Samples . . . . .    | 33 |
| 2.4.2.2 | Authentication against Type-II attacker . . . . . | 34 |
| 2.4.2.3 | Impact of Peak Features . . . . .                 | 35 |
| 2.4.2.4 | Impact of Feature Selection . . . . .             | 35 |
| 2.4.2.5 | Imitator Attack . . . . .                         | 36 |
| 2.4.3   | System Performance . . . . .                      | 37 |

|         |   |    |
|---------|---|----|
| 2.4.4   | Other considerations . . . . .                                | 37 |
| 2.5     | Chapter Summary . . . . .                                     | 38 |
| 3       | WearLock: Unlocking Your Phone via Acoustics using Smartwatch | 39 |
| 3.1     | Introduction . . . . .  | 39 |
| 3.2     | System Overview . . . . .                                     | 43 |
| 3.2.1   | System Architecture . . . . .                                 | 43 |
| 3.2.2   | Smartwatch-assisted Unlocking Protocol . . . . .              | 44 |
| 3.3     | Acoustic Modem Design . . . . .                               | 45 |
| 3.3.1   | The Acoustic Channel . . . . .                                | 46 |
| 3.3.1.1 | Ambient Noise . . . . .                                       | 46 |
| 3.3.1.2 | Sound propagation and attenuation . . . . .                   | 46 |
| 3.3.1.3 | Multipath Effect . . . . .                                    | 48 |
| 3.3.1.4 | Microphone and Speaker Characteristics . . . . .              | 49 |
| 3.3.2   | OFDM Design . . . . .   | 49 |
| 3.3.2.1 | Modulation and Demodulation . . . . .                         | 49 |
| 3.3.2.2 | Sub-carrier Frequency Range . . . . .                         | 50 |
| 3.3.2.3 | Preamble Design . . . . .                                     | 51 |
| 3.3.2.4 | Silence Detection and Signal Detection . . . . .              | 51 |
| 3.3.2.5 | Synchronization . . . . .                                     | 51 |
| 3.3.2.6 | Channel Estimation and Equalization . . . . .                 | 52 |
| 3.3.2.7 | Adaptive Modulation . . . . .                                 | 53 |
| 3.4     | Secure Unlocking . . . . .                                    | 56 |
| 3.4.1   | Threat Model . . . . .  | 56 |
| 3.4.2   | One Time Password . . . . .                                   | 57 |
| 3.4.3   | Security Discussion. . . . .                                  | 57 |
| 3.4.3.1 | Brutal Force Attack . . . . .                                 | 57 |



|         |   |    |
|---------|---|----|
| 3.4.3.2 | Co-located Attack . . . . .                                     | 58 |
| 3.4.3.3 | Record and Replay Attack . . . . .                              | 58 |
| 3.4.3.4 | Relay Attack . . . . .  | 59 |
| 3.5     | Performance Optimizations . . . . .                             | 59 |
| 3.5.1   | Computation Offloading . . . . .                                | 60 |
| 3.5.2   | Computation Reduction . . . . .                                 | 61 |
| 3.5.2.1 | Leveraging Motion Sensor-based Filtering . . . . .              | 61 |
| 3.6     | Evaluation . . . . .  | 62 |
| 3.6.1   | Implementation Details . . . . .                                | 62 |
| 3.6.2   | Communication Range . . . . .                                   | 63 |
| 3.6.3   | Adaptive Modulation . . . . .                                   | 64 |
| 3.6.4   | Sensor-based Filtering . . . . .                                | 65 |
| 3.6.5   | System Delay . . . . .  | 65 |
| 3.6.6   | Field Test . . . . .  | 67 |
| 3.7     | Discussion and Limitations . . . . .                            | 68 |
| 3.7.1   | Non-omnidirectional Microphone/Speaker . . . . .                | 68 |
| 3.7.2   | Acoustic Frequency Range . . . . .                              | 68 |
| 3.7.3   | Bluetooth Proximity . . . . .                                   | 69 |
| 3.8     | Related Work . . . . .  | 69 |
| 3.8.1   | Acoustic Communication on Mobile Devices . . . . .              | 69 |
| 3.8.2   | Reduced-Effort Authentication . . . . .                         | 70 |
| 3.9     | Chapter Summary . . . . .                                       | 71 |
| 4       | LAVEA: Latency-aware Video Analytics on Edge Computing Platform | 72 |
| 4.1     | Introduction . . . . .  | 72 |
| 4.2     | Background and Motivation . . . . .                             | 75 |
| 4.2.1   | Edge Computing Network . . . . .                                | 75 |

|         |  |     |
|---------|--|-----|
| 4.2.2   | Serverless Architecture . . . . .                              | 77  |
| 4.2.3   | Video Edge Analytics for Public Safety . . . . .               | 78  |
| 4.3     | LAVEA System Design . . . . .                                  | 80  |
| 4.3.1   | Design Goals . . . . .   | 80  |
| 4.3.2   | System Overview . . . . .                                      | 80  |
| 4.3.2.1 | Edge Computing Node . . . . .                                  | 80  |
| 4.3.2.2 | Edge Client . . . . .  | 81  |
| 4.3.3   | Edge Computing Services . . . . .                              | 81  |
| 4.3.3.1 | Profiler Service . . . . .                                     | 81  |
| 4.3.3.2 | Monitoring Service . . . . .                                   | 82  |
| 4.3.3.3 | Offloading Service . . . . .                                   | 83  |
| 4.4     | Edge-front Offloading . . . . .                                | 83  |
| 4.4.1   | Task Offloading System Model and Problem Formulation . . . . . | 84  |
| 4.4.2   | Prioritizing Edge Task Queue . . . . .                         | 88  |
| 4.5     | Inter-edge Collaboration . . . . .                             | 89  |
| 4.5.1   | Motivation and Challenges . . . . .                            | 89  |
| 4.5.2   | Inter-Edge Task Placement Schemes . . . . .                    | 90  |
| 4.6     | System Implementation and Performance Evaluation . . . . .     | 93  |
| 4.6.1   | Implementation Details . . . . .                               | 93  |
| 4.6.2   | Evaluation Setup . . . . .                                     | 93  |
| 4.6.2.1 | Testbed . . . . .  | 93  |
| 4.6.2.2 | Datasets . . . . .   | 94  |
| 4.6.3   | Task Profiler . . . . .  | 94  |
| 4.6.4   | Offloading Task Selection . . . . .                            | 95  |
| 4.6.5   | Edge-front Task Queue Prioritizing . . . . .                   | 98  |
| 4.6.6   | Inter-Edge Collaboration . . . . .                             | 99  |
| 4.7     | Related Work . . . . .   | 101 |

|       |                                       |     |
|-------|---------------------------------------|-----|
| 4.7.1 | Distributed Data Processing . . . . . | 102 |
| 4.7.2 | Computation Offloading . . . . .      | 103 |
| 4.8   | Discussions and Limitations . . . . . | 103 |
| 4.9   | Chapter Summary . . . . .             | 104 |
| 5     | Conclusion and Future Work            | 106 |
|       | Bibliography                          | 110 |

## ACKNOWLEDGMENTS

This dissertation would not have been possible without the guidance, encouragements, patience, and support from many kind and talented people.

I would like to express my deepest gratitude to my advisor, Professor Qun Li. In the past 6 years, he helped me grow my academic career, by patient guidance, hearty encouragements, constructive discussions, and unconditional support. His passion, critical thinking, rigor, energy, hardworking, and wisdom will always inspire me. It has been my greatest honor to be one of his students.

I would like to thank my dissertation committee members, Professor Weizhen Mao, Professor Andreas Stathopoulos, Professor Dmitry Evtushkin and Professor Kai Zeng, for their tremendous encouragements, valuable feedbacks, and insightful comments. I would like to give special thanks to Dr. Kai Zeng, who introduced me to wireless network research and opened the door for me to pursue the academic path before I decided to apply to a PhD program.

I am grateful to my colleagues and collaborators including Dr. Zhengrui Qin, Zijiang Hao, Yutao Tang, Lele Ma, Dr. Ed Novak, Dr. Yafeng Yin, Yunlong Mao, Dr. Fengyuan Xu, Dr. Hao Han, Qingyang Zhang, Cheng Li, Dr. Hui Zeng, and Dr. Weisong Shi. I appreciate all the collaborations and helpful discussions that I had with them over the years.

I would like to thank our Computer Science Department Chair, Professor Robert Michael Lewis, Graduate Director Professor Denys Poshyvanyk and former Graduate Directors Professor Gang Zhou and Professor Evgenia Smirni, and the fantastic Computer Science administration team, Vanessa Godwin, Jacquelyn Johnson, and Dale Hayes.

I would like to dedicate this dissertation to my beloved wife and my parents for their endless love, support, and sacrifice.

## LIST OF TABLES

|     |  |    |
|-----|--|----|
| 2.1 | Head gesture candidates. . . . .                           | 18 |
| 2.2 | FPR and TPR of authentication on two gestures. . . . .     | 34 |
| 2.3 | Average Time Cost . . . . .                                | 37 |
| 3.1 | Sensor-based Filtering . . . . .                           | 65 |
| 3.2 | Field Test Result. The average BER is around 0.08. . . . . | 68 |

## LIST OF FIGURES

|     |  |    |
|-----|--|----|
| 2.1 | Head Movements . . . . .   | 12 |
| 2.2 | System Architecture . . . . .  | 15 |
| 2.3 | Collected Sensor Trace: The user sits still for about 17s, then stands up and walks for about 10s, then runs for a few seconds and stops. In each activities (marked in accelerometer plot), she performs several head gestures such as nodding, shaking, looking up/down/left/right (sensor coordinate reference [28]). . . . . | 16 |
| 2.4 | Gesture frequency of a user seated, working at a desk for about 20 minutes. The number in the name is the repetition count. “cw” is short for clockwise, “ccw” is short for counter-clockwise. . . . .   | 19 |
| 2.5 | Thresholds under different activities. The threshold will be set small when the user is sitting or standing, to enable even tiny head gesture detection (0.15). It will be set much larger when user is walking or running (0.7 and 1.3 respectively). . . . .   | 20 |
| 2.6 | DTW workflow . . . . .   | 22 |
| 2.7 | K-S test results for gesture Nod and Shake . . . . .   | 27 |
| 2.8 | (a) Confusion matrix of command gestures (sitting, tiny). TPR: 92.87%, FPR: 5.7%. (b) Confusion matrix of command gestures (sitting, normal). TPR: 96.99%, FPR: 2.4%. (c) Confusion matrix of command gestures (walking, normal). TPR: 94.88%, FPR: 4.6% . . . . .   | 30 |
| 2.9 | Accuracy changes with sampling lengths ( $n_{ED}$ ) and numbers of nearest neighbours. . . . .   | 31 |

|   |    |
|---|----|
| 2.10 Accuracy and scaled running time using DTW change with sampling lengths ( $n_{DTW}$ ). . . . .   | 31 |
| 2.11 Running time comparison between our scheme and linear scanning. . . . .  | 32 |
| 2.12 The average TPR (a) and FPR (b) change with different ratios of training samples. . . . .  | 33 |
| 2.13 The F1-score of certain category of features is excluded. . . . .  | 35 |
| 2.14 F1-Scores of one-class SVM with or without feature selection for gesture nod (a) and shake (b). . . . .  | 36 |
| 3.1 The architecture of WearLock. . . . .   | 43 |
| 3.2 The Protocol of WearLock. . . . .   | 45 |
| 3.3 The OFDM modem of WearLock. . . . .   | 46 |
| 3.4 Receiver's SPL in distance of different volume settings. Measured in a quiet room with the SPL of ambient noise about 15-20 dB in a line-of-sight scenario. . . . . | 47 |
| 3.5 The received signal comparison of LOS direct path (BER=0.0) and Body-blocked NLOS (BER=0.54). . . . .   | 48 |
| 3.6 The BER of different modulations changes with $E_b/N_0$ . . . . .   | 55 |
| 3.7 Time Cost (a) and Power Consumption (b) Comparison on Offloading and Local Processing on Wearable. . . . .  | 60 |
| 3.8 The BER in distances and transmission modes (near-ultrasound). . . . .  | 63 |
| 3.9 The BER in adaptive modulation under different BER constrains(near-ultrasound). . . . .   | 64 |
| 3.10 The BER under jamming and subchannel selection. (QPSK, audible sound) . . . . .  | 65 |
| 3.11 The computation delay of each phase on different devices. . . . .  | 66 |



|   |     |
|---|-----|
| 3.12 The communication delay between smartphone and smartwatch. .   | 66  |
| 3.13 Compare the total delay in different configurations with manually<br>entering pin codes. . . . .                                 | 67  |
| 4.1 An overview of edge computing environment . . . . .   | 76  |
| 4.2 Round trip time between client and edge/cloud. . . . .  | 77  |
| 4.3 Bandwidth between client and edge/cloud. . . . .  | 77  |
| 4.4 The architecture of edge computing platform . . . . .   | 82  |
| 4.5 The task graph of OpenALPR. . . . .   | 85  |
| 4.6 OpenALPR profile result of client type 1 (RPi2 quad-core 0.9 GHz)   | 95  |
| 4.7 OpenALPR profile result of client type 2 (RPi3 quad-core 1.2 GHz)   | 95  |
| 4.8 OpenALPR profile result of a type of edge node (i7 quad-core 2.30<br>GHz) . . . . .   | 96  |
| 4.9 OpenALPR profile of a type of cloud node (AWS EC2 t2.large Xeon<br>dual-core 2.40 GHz) . . . . .                                  | 96  |
| 4.10 The comparison of task selection impacts on edge offloading and<br>cloud offloading for wired clients (RPi2). . . . .            | 97  |
| 4.11 The comparison of task selection impacts on edge offloading and<br>cloud offloading for 2.4 GHz wireless clients (RPi3). . . . . | 97  |
| 4.12 The comparison result of three task prioritizing schemes. . . . .  | 98  |
| 4.13 Performance with no task placement scheme. . . . .   | 99  |
| 4.14 Performance of STTF. . . . .   | 99  |
| 4.15 Performance of SQLF. . . . .   | 99  |
| 4.16 Performance of SSLF. . . . .   | 99  |
| 4.17 Numbers of tasks placed by the edge-front node. . . . .  | 101 |

# Chapter 1

## Introduction

The rapid adoption of smart devices, including smartphones, wearables, smart appliances, has made the Internet more ubiquitously and intelligently connected. As a result, people always find themselves interfacing with various smart devices to accomplish everyday tasks. For example, a user at work receives a suspicious event notification on her **smartwatch** sent by her indoor home **security camera**. She checks the corresponding video clip on her **smartphone**, and finds out that her pet at home spills its food all over the kitchen floor. As a remedy, she instructs the **cleaner robot** to wipe the kitchen area via a voice command on her smartphone immediately. This example showcases several latest trends of mobile computing in the era of wearables and Internet of Things (IoT):

- Mobile devices are becoming the very first stop of computation requests. They can either solely carry out simple tasks, or chain up any smart devices or computing nodes in the edge or cloud network to accomplish complex assignments.
- Mobile devices are demanding multi-channel interfaces for better service coverage and better user experience. Such interfaces can be *humane-to-machine* interfaces that enable user to see (*graphical user interface*), talk (*conversational user interface* and *voice user interface*), and control ( *touch user interface* and *gesture interface*) the mobile devices [93]. The interfaces also include *machine-to-machine* interfaces

that coordinate smart devices to get things done automatically without any user intervention.

- Mobile devices are consuming and generating all kinds of data in large volume and at high rate. Unlike traditional applications (e.g., music/video streaming, web browsing) that consume data from remote servers, new types of applications, which sense the ambient environments, collect data from human body, exchange information with nearby devices, are generating data rapidly as well. Analyzing these data and making corresponding decisions needs tremendous system support from both the mobile device and its backend.

These trends are the results of impacts on human living styles from 1) the advancement of techniques in relevant domains (e.g., wireless communication, computer vision, speech recognition/synthesis, sensors); 2) the invention and adoption of new type of mobile devices (e.g., smart speaker, smartwatch, smart eyeglass, smart lock); and 3) the groundbreaking innovations in mobile applications (e.g. virtual reality, augment reality, live video broadcasting, real-time language translation).

## 1.1 Challenges

The prevalence of smart devices in every aspect of human life presents significant challenges in terms of usability, security, and performance. In this dissertation, we mainly address the challenges in three directions:

- **New interface design.** Evolving of mobile devices, from smartphones to wearables (e.g., smartwatch, smart glasses) witnesses significant usage pattern shift. To reduce the hardware and software development cost of wearables and improve their compatibility, manufacturers tend to re-purpose legacy mobile operating system and hardware that were originally designed for smartphones and tablets, while giving less consideration to the adaption of user interfaces in the era of wearable

devices. As a result, user interfaces may be difficult to use, error-prone, and insecure. We need to design new user interface to improve the usability of new smart devices.

- **User authentication.** Smartphone is becoming the hub of wearables, networked devices, and smart things [103], and valuable personal data are either processed at or flow through these devices. Therefore, it is crucially important to enhance security and preserve privacy of smart devices, since one corrupted device may compromise the whole chain of super-connected devices. As the first step of user interfacing, user authentication poses rigorous challenges on designing authentication schemes with well-balanced trade-off between security and convenience.
- **Mobile backend infrastructure.** Mobile devices are heterogeneous and resource-limited. The hardware upgrades can hardly keep up the pace with the resource demands such as new types of applications and services (e.g., AR/VR, video analytics, cognitive tasks). To make such applications scalable among mobile devices, existing mobile backends are usually deployed on cloud infrastructure. However, the prevalent cloud infrastructure suffers from unexpected latency and low bandwidth. Therefore, it is challenging to design and leverage new type of mobile backend infrastructure that can provide low latency and high bandwidth to support new applications with performance guarantee on mobile devices.

To explore the opportunities inherent in these challenges, we have designed and implemented three mobile systems. Generally speaking, our research focuses on the advanced interfaces and system design between user and device, between device and device, and between device and backend, to enhance the usability, security, and performance.

## 1.2 Head Gesture Interface of Smart Glasses

Along the emerging trends towards wearables, one typical category of wearable devices is smart glasses (eyewear), which is a pair of glasses equipped with a heads-up, near-eye display, a rich set of sensors, and an embedded mobile operating system. In this project, we investigate the interface design of smart glasses, whose current interfaces (legacy smartphone touch user interface and voice user interface) are difficult to use, error-prone, and provide no or insecure user authentication. For example, Google Glass uses a legacy smartphone touch user interface via a mounted touchpad and a voice user interface that a user can instruct the device using voice commands. On one hand, during operating, the user cannot precisely tap without seeing the touchpad, impeding user authentication that leverages unlocking touch gestures on the touchpad. On the other hand, the voice user interface cannot work in every scenario, for instance, when the user is talking directly with someone, or in a conference/meeting.

### 1.2.1 Problem Statements

Based on our observations on the legacy interfaces of smart glasses, we aim to design a new interface to improve the usability. Our main research problem is *how to design user interface for smart glasses that is easy-to-use, hand-free, and secure*. Toward this, we design and build GlassGesture, which improves Google Glass through a head gesture user interface with gesture recognition and gesture-based authentication. For gesture recognition, GlassGesture enables simple head gestures as input, which can be accurately recognized regardless of the noise of body movements in different activities. We propose a novel similarity search scheme to accelerate computation-intensive template matching during recognition. For gesture-based authentication, GlassGesture can identify owner through features extracted from head movements. We employ an ensemble learning method working with new features based on peak analyses to improve the authentication performance.

### **1.2.2 Contributions**

- For gesture recognition, our system increases the input space of the Glass by enabling small, easy-to-perform head gestures. We propose a reference gesture library exclusively for head movements. We utilize activity context information to adaptively set thresholds to separate head movements from body movements for robust gesture detection. We use a weighted dynamic time warping (DTW) algorithm to match templates for better accuracy. We accelerate the gesture matching with a novel template searching scheme, which reduces the time cost by at least 55%.
- For authentication, we propose that “head gestures can be used as passwords”. We design a two-factor authentication scheme, in which we ask users to perform head gestures to answer questions that show up in the private near-eye display. To characterize head gestures, we identify a set of useful features and propose new features based on peak analyses. We also exploit several optimizations such as one-class ensemble classifier, and one-class feature selection, to improve the authentication performance.
- We prototype our system on Google Glass. It is efficient, accurate, and extensible to general smart eyewears. We design experiments to evaluate gesture recognition in different user activities. We collect a total of around 6000 gesture samples from 18 users to evaluate the authentication performance. Our evaluation shows that GlassGesture achieves accurate gesture recognition. It can reliably accept the authorized users and reject attackers.

## **1.3 Smartwatch-assisted Smartphone Authentication**

Since mobile devices are collecting and storing a wide variety of sensitive information of the owner, it is critical to provide effective protection for smartphone. While most mobile

operating systems have built-in screen lock applications, a significant portion of users never lock their mobile devices. We have found that the root cause is the difficult reconciling of security and convenience. Naively reducing unlock frequency of existing authentication methods is usually at the cost of information security or ends up with even worse user experience. In the second project, we try to solve this problem without security sacrifice by finding the most suitable authentication method for mobile devices. Most authenticators on smartphone can be categorized into passwords, biometrics, and tokens. Complex passwords are recommended for strong security but they are also very hard to memorize and input. Biometrics, which is uniquely tied to human body, is impossible to replace once being compromised. Token-based methods authenticate users via a small piece of trusted hardware. Compared with passwords or biometrics, token-based authentication has unique advantages such as easy-to-use, replaceable, coming for free in the wearable era when consumers own or intent to own at least one trusted wearable device.

### **1.3.1 Problem Statements**

As we are utilizing a wearable device as a secure token for authentication, the research problem becomes *how to securely and user-friendly unlock smartphone via a trusted companion wearable*. The difficulty is how to find the most suitable authentication channel and address technical challenges of the system. We present WearLock, a system that uses *acoustic tones* as the authentication channel to automate the unlocking securely. Unlike other authentication channels (e.g., NFC, WiFi, and Bluetooth), acoustics has the most desirable communication range for near-range authentication. We propose several optimizations to improve system performance. First, we build an acoustic modem with sub-channel selection and an adaptive modulation to maximize unlocking success rate against ambient noise. Second, we exploit the motion similarities via sensors when smartwatch and smartphone are on the same body to eliminate unnecessary unlockings. Last, we offload heavy signal processing tasks from the smartwatch to the smartphone to

speed up computation and save energy.

### **1.3.2 Contributions**

- We propose a novel automated and secure unlocking scheme for smartphone via a trusted wearable device. It requires minimal amount of effort from users. WearLock, the implemented system, secures the acoustic channel by adapting the transmission power and modulation configurations, and sends an one-time-password (OTP) tokens for validation via acoustics to unlock the smartphone.
- We are the first, to the best of our knowledge, to exploit the adaptive modulation of acoustics on common-of-the-shelf (COTS) mobile devices for robust data transmission. The acoustic modem can adapt to ambient noise levels and interfering signals.
- To optimize the system performance, we offload the heavy computation to the phone, and leverage multi-source information including wireless connectivities and motion similarities to reduce unnecessary audio transmissions.
- We build WearLock on unmodified COTS smartphone and smartwatch devices and evaluate the system extensively. WearLock's acoustic modem achieves an average bit error rate (BER) of 8% in our experiments. WearLock achieves at least 18% speedup of unlocking even on a low-end mobile device, compared to entering PINs.

## **1.4 Edge Computing based Mobile Backend**

Edge computing (a.k.a., fog computing [9], cloudlets [78], MEC [66], etc.) is proposed as a new computing paradigm to support latency-sensitive and bandwidth-hungry applications [36, 99]. Most mobile applications get help from backends deployed on remote servers or cloud nodes located in data centers. All the data generated on mobile devices



will be uploaded to the data center before processing. However, the role change of mobile device from data consumer to producer results in huge amount of data generated at the edge of the network. Transferring data at such scale to the distant cloud will add burdens to the network and lead to unacceptable response time, especially for data analytic applications.

#### 1.4.1 Problem Statements

There are many benefits to carry out video analytics at the edge of the network, in terms of, gathering more client side information, shortening the response time, saving network bandwidth, lowering the peak workload to the cloud, and so on. The ability to provide low latency video analytics is critical for applications in the fields of public safety, counter-terrorism, self-driving cars, VR/AR, etc [83]. In the last project, we consider the research problem *how to improve the performance of video analytic application on mobile device with edge computing based mobile backend*. The research problem is formulated as a response time minimization problem, and divided into three sub-problems. First, we select and offload client tasks for execution on the edge node to reduce time cost. We formulate this problem as a mathematical optimization problem to select offloading tasks and allocate bandwidth among clients. Unlike mobile cloud computing, we need to consider the resource contention and response time when more and more tasks are running on edge node whose resources are relatively limited compared with cloud node. Second, we prioritize the offloaded tasks at the edge node to minimize the makespan, because the offloaded tasks cannot be started when the corresponding inputs are not ready. In the ideal case, this problem can be formulated as a two-stage job shop model with an optimal solution. However, we need to address the problem with new constraints such as task priorities and dependencies. Last, we enable inter-edge collaboration to further improve the overall response time. We compare several task placement schemes and propose a prediction-based scheme that works efficiently in the edge computing network.

### 1.4.2 Contributions

- We have designed an edge computing platform based on a serverless architecture, which is able to provide flexible computation offloading to nearby clients to speed up computation-intensive and delay-sensitive applications. Our implementation is lightweight-virtualized, event-based, modular, and easy to deploy and manage on either edge or cloud nodes.
- We have formulated an optimization problem for offloading task selection and prioritized offloading requests to minimize the response time. The task selection problem decides the offloading decision and bandwidth allocation, under the latency constraint, which is tuned to adapt to the workload on the edge node as the offloading target. The task prioritizing is modeled as a two-stage job shop problem and a heuristic is proposed with the topological ordering constraint.
- We have evaluated several task placement schemes for inter-edge collaboration and proposed a predication-based method which efficiently estimates the response time.

## 1.5 Dissertation Organization

The dissertation is organized as follows. In Chapter 2, we introduce the work of designing head gesture interface for Google Glass. In Chapter 3, we present an novel user authentication system works on a smartphone-smartwatch pair within the acoustic channel. In Chapter 4, we present a system that supports low-latency video analytics on mobile device with edge computing backend. Finally, we conclude the dissertation and discuss the future work in Chapter 5.

## Chapter 2

# GlassGesture: Exploring Head Gesture Interface of Smart Glasses

### 2.1 Introduction

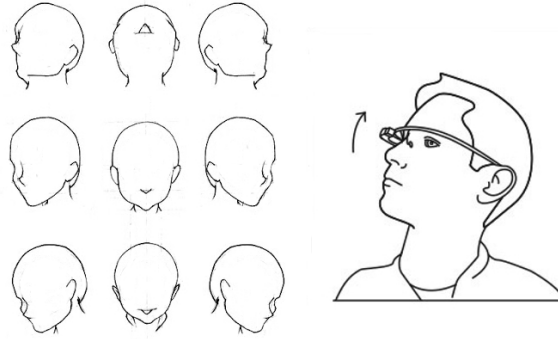
In recent years, we have seen an emerging trend towards wearables, which are designed to improve the usability of computers worn on the human body, while being more aesthetically pleasing and fashionable at the same time. One category of wearable devices is smart glasses (eyewear), which are usually equipped with a heads-up, near-eye display and various sensors, mounted on a pair of glasses. Among many kinds of smart eyewear, Google Glass (Glass for short) is the most iconic. However, since Glass is a new type of wearable device, *the user interface is less than ideal*.

On one hand, there is no virtual or physical keyboard attached to the Glass. Currently, there are two most prominent input methods for Glass. However, each of these input methods suffers in certain scenarios. First, there is a *touchpad* mounted on the right-hand side of the device. Tapping and swiping on the touchpad is error-prone: 1) The user needs to raise their hands and fingers to the side of their forehead to locate the touchpad and perform actions, which can be difficult or even dangerous when the user is walking or driving. 2) Since the touchpad is very narrow and slim, some gestures, such as slide

up/down, or tap can be easily confused. 3) When the user puts Glass on their head, or takes it off, it is very easy to accidentally touch the touchpad, causing erroneous input. Second, Glass supports *voice commands* and *speech recognition*. A significant drawback is that voice input cannot be applied in every scenario; for example, when the user is talking directly with someone, or in a conference or meeting. An even worse example is that other people can accidentally activate Glass using voice commands, as long as the command is spoken loudly enough to be picked by Glass. Additionally, disabled users are at a severe disadvantage using Glass if they cannot speak, or have lost control of their arms or fine motor skills.

On the other hand, *authentication* on Google Glass is very cumbersome and is based solely on the touchpad [30]. As a wearable device, Glass contains rich private information including point-of-view (POV) photo or video recording, deep integration of social and communication apps, and personal accounts of all kinds. There will be a severe information leak if Glass is accessed by malicious users. Thus, any user interface for Glass needs to provide schemes to reject unauthorized access. However, the current authentication on Glass is far from mature: a “password” is set by performing four consecutive swiping or tapping actions on the touchpad similar to a traditional four digit PIN code. This system has many problems. First, the entropy is low, as only five touchpad gestures (tap, swipe forward with one or two fingers, or swipe backward with one or two fingers) are available to form a limited set of permutations. Second, gestures are difficult to perform correctly on the narrow touchpad, especially when the user is in non-static activities. Third, unorthodox passwords are hard to remember. Finally, this system is susceptible to shoulder surfing attacks. Any attacker can easily observe the pattern from possibly several meters away, with no special equipment. If we want to reap the benefits of wearables, it must provide a user interface with high usability and security. We feel that the future of smart eyewears is very exciting, but is currently thwarted by poor user interfaces, which is one of the biggest problems.

To solve all of these problems, we propose the use of head gestures (gesture for short)



**Figure 2.1:** Head Movements

as an alternative user interface for smart eyewear devices like Google Glass. Because using head gestures is an intuitive option, we can leverage them as a hands-free and easy-to-use interface. A head gesture is a short burst of consecutive movements of the user’s head, as illustrated in Fig. 2.1. Motion sensors (i.e. the accelerometer and gyroscope) on Glass are able to measure and detect all kinds of head movements due to their high electromechanical sensitivity. However, smart eyewear presents new challenges for head gesture interface design. We need to answer questions such as “What are easy-to-perform head gestures?”, “How do we accurately recognize these gestures?”, “How do we make the system efficient on resource-limited hardware?”, and “How does the system reject unauthorized access?” and so on.

In this chapter, we present GlassGesture, a system aims at improving the usability of Glass by providing an advanced user interface built on various sensors (e.g., accelerometer and gyroscope) [102]. We are the first work, to the authors’ knowledge, to consider head-gesture-based recognition/authentication problems for smart glasses. First, GlassGesture leverages head gesture recognition for interactions. Because head gestures are easy-to-perform, intuitive, hands-free, user-defined, and accessible for the disabled. For example, it is usually considered inappropriate or even rude to operate Glass through the provided touchpad or voice commands. Head gestures, in comparison, can be tiny and not easily noticeable to mitigate the social awkwardness. Second, the head gesture user interface can authenticate users. In particular, head gestures have not been exploited in

authentication yet in the literature. We propose a novel head-gesture-based authentication scheme by using simple head gestures to answer security questions. For example, we ask user to answer a yes-or-no question, by shaking (no) or nodding (yes) her head. However, an attacker who knows the answer to the security questions can access the device. As a second factor, we further propose to leverage unique signatures extracted from these head gestures to identify the owner of the device. Compared to the original, touchpad-based authentication, our proposed head-gesture-based authentication is more resistant to shoulder surfing attacks , and requires less effort from the user.

In summary, we make the following contributions:

- For gesture recognition, our system increases the input space of the Glass by enabling small, easy-to-perform head gestures. We propose a reference gesture library exclusively for head movements. We utilize activity context to adaptively set thresholds for robust gesture detection. We use a weighted dynamic time warping (DTW) algorithm to match templates for better accuracy. We speed up the gesture matching with a novel scheme, which reduces the time cost by at least 55%.
- For authentication, we propose that “head gestures can be used as passwords”. We design a two-factor authentication scheme, in which we ask users to perform head gestures to answer questions shown in the near-eye display. To characterize head gestures, we identify a set of useful features and propose new features based on peak analyses. We also explore several optimizations such as one-class ensemble classifier, and one-class feature selection, to improve the authentication performance.
- We prototype our system on Google Glass. We design experiments to evaluate gesture recognition in different user activities. We collect a total of around 6000 gesture samples from 18 users to evaluate the authentication performance. Our evaluation shows that GlassGesture achieves accurate gesture recognition. It can reliably accept the authorized users and reject attackers.

## 2.2 Related Work

Our work is related to activity recognition, gesture recognition, and user authentication.

**Activity Recognition.** Activity recognition on smart mobile devices has been widely investigated. Researchers show that when the smart device is carried with the user, it can provide context information about the user such as if they are sitting or walking, or if they are using some transportation (e.g., cycling, driving) [57, 72, 74]. However, in this work, we are not aiming at improving upon the state-of-the-art activity recognition systems. We use a simply trained activity detector, to tune parameters for gesture detection.

**Gesture Recognition.** It has been shown that gestures as input can be precise, and fast. While there is a broad range of gesture recognition techniques based on vision, wireless signal, touch screen [17, 70, 94], we focus mainly on motion-sensor-based gesture recognition because it is low-cost, computationally feasible, and easy to deploy on mobile devices [55]. We differ from these works in that we propose a head gesture based interface for smart glasses. And we carefully design the system to work with head gestures which faces different challenges such as noise from user activities, performance on resource-constrained devices. For head gesture recognition, existing work mainly focuses on vision-based methods [58], while GlassGesture utilizes sensor mounted on user's head. For gesture recognition on Google Glass, Head Wake Up and Head Nudge [27] are two built-in gesture detectors as experimental features which monitor the angle of head. A similar open-sourced implementation can be found in [40]. In contrast, GlassGesture is more advanced which can recognize self-defined, free-form head gestures efficiently and accurately.

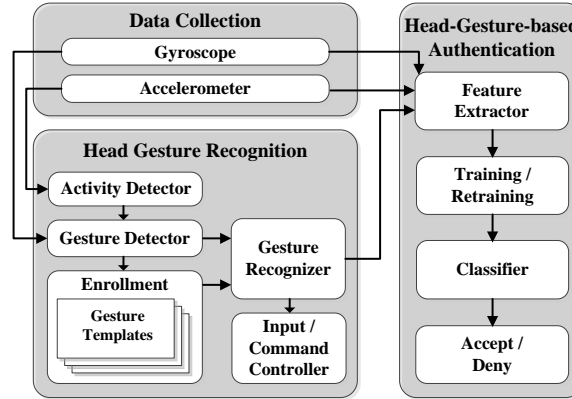
**User Authentication.** There has been research on authentication based on the unique patterns users exhibit while interacting with phone through touch screens and motion sensors [7, 16, 25, 52, 81, 87]. These systems show that such authentication schemes are less susceptible to shoulder surfing and don't require the user to memorize any password or pattern. For authentication on Google Glass, work [15] and [67] are touchpad-gesture-

based authentication, which needs continuous user effort to hold up fingers on the touchpad. Our work is orthogonal that tries to bring easy authentication to smart glasses using head gestures, which is simple, hands-free, and requires less effort.

## 2.3 GlassGesture System Design

In this section, we present the system design of GlassGesture. First, we give an overview of our system architecture. Then we introduce each module and elaborate its corresponding components.

### 2.3.1 System Overview



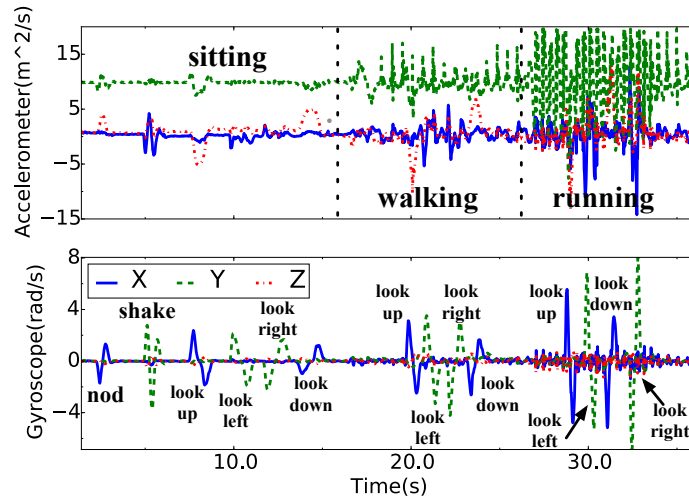
**Figure 2.2:** System Architecture

Our system consists of two modules, which together form our gesture-based interface. The first module allows users to input small gestures using their heads; the second module authenticates users based on their head gestures. The architecture of our system is illustrated in Fig. 2.2, which shows that the Gesture Recognition module is the cornerstone. We leverage an activity detector to tune the parameters for more accurate gesture detection, based on user activity context. An enrollment submodule is in charge of managing the gesture templates. The gesture recognizer runs a template matching algorithm to recognize potential gestures. The gesture-based authentication module is built on top



of the first module. It extracts features from the raw sensor data for training. With trained classifiers, we form a two-step authentication mechanism using simple head gestures. In the following sections, we present the design details of each module.

### 2.3.2 Head Gesture Recognition



**Figure 2.3:** Collected Sensor Trace: The user sits still for about 17s, then stands up and walks for about 10s, then runs for a few seconds and stops. In each activities (marked in accelerometer plot), she performs several head gestures such as nodding, shaking, looking up/down/left/right (sensor coordinate reference [28]).

We have made some preliminary observations from the collected trace in Fig. 2.3:

- Different activities add different amounts of noise. It is not easy to derive a general criterion for gesture detection in all of the many kinds of activities the user may be participating in at the time the gesture is made.
- Head gestures consist of mostly rotations rather than accelerations. We see obvious gyroscope readings while the user is performing head gestures in various activities, compared to relatively noisy accelerometer readings. Therefore it is possible to provide head gesture detection/recognition through the gyroscope data.







- Head gestures can be used rather frequently by the user. We need an efficient recognition scheme for performance considerations.

In summary, we face three challenges in designing this module.

1. **Head gesture library:** There is no library, which defines the most suitable head gestures for smart glasses.
2. **Noise:** Sensors on Glass are used to collect head movements, while at the same time may also collecting noise from other user activities, which will deteriorate the performance of the gesture recognition. It is challenging to derive a general criterion for head gesture detection in all kinds of activities.
3. **Computation:** In recognition tasks, computing-intensive algorithms may be invoked frequently, resulting in unsatisfactory performance. Therefore, it must be optimized to be extremely efficient, without sacrificing substantial recognition accuracy.

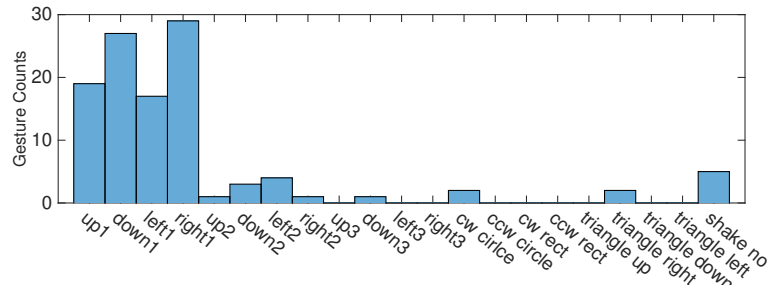
#### 2.3.2.1 Head Gesture Library

We need to provide a head gesture library as reference since head gestures are quite different from traditional hand gestures. For example, 1) head gestures mainly consist of rotational movement. 2) users moving their heads have limited freedom in 3D space. (e.g. usually humans can only look up and down in less than  $180^\circ$ . 3) In order to convey more information, we need a new set of head gestures beside the traditional ones that are already used (e.g., shaking for “no” and nodding for “yes”). In light of these constraints, we develop six basic candidate gesture categories adapted from work [55] and [2]: 1) nod, 2) look up/down/left/right, 3) shake, 4) circle, 5) triangle, and 6) rectangle. To clear up confusion when drawing (performing, acting out the gesture), we ask the user to move her head just like drawing something in the air in front of herself using the nose like a pen tip.

| Gesture   | Styles               | Number of strokes | Easy to perform | Frequency in Fig. 2.4 | Easy to repeat | Decision    |
|---|----------------------|-------------------|-----------------|-----------------------|----------------|-------------|
| 1  | up and down          | 3+                | 5.2             | low                   | no             | keep        |
| 2  | up/down/left/right   | 1                 | 4.9             | high                  | yes (81%)      | keep,repeat |
| 3  | left and right       | 3+                | 4.4             | low                   | no             | keep        |
| 4  | cw/ccw               | 1                 | 3.0             | very low              | neutral        | keep        |
| 5  | cw/ccw, directions   | 3                 | 2.2             | very low              | no             | drop        |
| 6  | cw/ccw, start points | 4                 | 1.4             | very low              | no             | drop        |

**Table 2.1:** Head gesture candidates.

With the purpose of trying to figure out what gestures are suitable, we performed a simple survey to rank them on how easy each category is to be performed for untrained users. It is important to note that the survey, and all data collections in the entire project, have gone through the IRB approval process. In total, we have received 22 effective responses. The study results are presented in Table 2.1. Our survey results indicate that nodding and shaking are popular and usually convey special social meanings (e.g. “yes” and “no”). Circles are easy to perform since they are single-stroke. The rectangle and triangle gestures are the least favored, due to the multiple strokes they entail. Simple “look up/down/left/right” gestures are easy and fast, but they appear frequently in daily head movement as shown in Fig. 2.4, another study we have done to understand the frequency of daily life head gestures. This leads us to believe there will be significant false positives if they are utilized naively. However, 81% of participants think these one-direction gestures are easy to be performed repeatedly. We decide to keep these gestures, as long as the user is willing to repeat them two or three times consecutively to reduce the false positive rate. It is important to note that this head gesture library is a reference. GlassGesture allows the user to define arbitrary head gestures. We also evaluate our gesture recognition system with “number” and “letter” input later in this work.



**Figure 2.4:** Gesture frequency of a user seated, working at a desk for about 20 minutes. The number in the name is the repetition count. “cw” is short for clockwise, “ccw” is short for counter-clockwise.

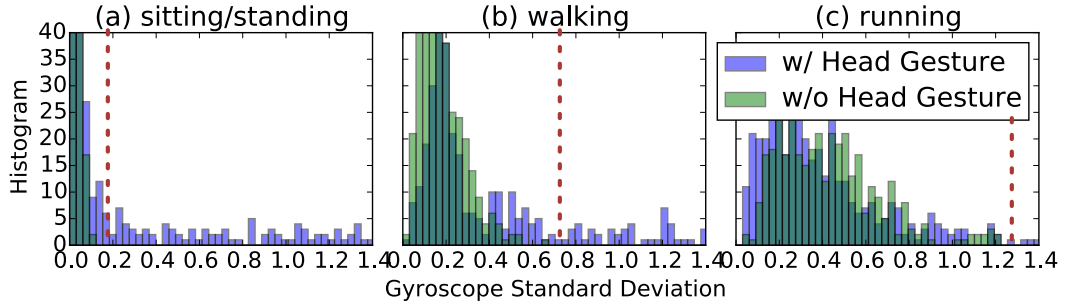
### 2.3.2.2 Activity Detector

The observations made in Fig. 2.3 motivate the need for a user activity context service, to help detect head gestures in different activities. Normally, Google Play Service provides activity APIs, which can be leveraged. Unfortunately, it is not supported on Glass at the time of writing. To fill this gap, we have implemented a simple activity detector using the accelerometer data. Samples from the accelerometer are chunked by an overlapping sliding window. We extract features, including *mean*, *standard deviation (std)*, *root mean square (rms)*, from each axis in every chunk. We collect traces of user wearing Glass in different activities and train a decision tree classifier due to its simplicity and efficiency. The classifier currently gives one of the four outputs: 1) sitting/standing, which indicates that the user’s head is fixed and the user’s body is not moving; 2) walking; 3) running; and 4) driving. By using a 50 Hz sensor sampling rate, and a 10-second window with a 5-second overlap, the classifier gives an average accuracy of **98%** in our preliminary experiments, which is adequate for use in our system.

### 2.3.2.3 Gesture Detector

The goal of the gesture detector is to capture every potential gesture from the sensor’s time series data. To find a potential gesture, we begin with windowing (30 samples) the gyroscope samples, and we calculate the rolling *standard deviation (std)*. A threshold

on the gyroscope rolling std for the gesture detector will be set according to the current activity context, the output of activity detector. To determine the thresholds, we collect user gyroscope data in different activities with and without the head gestures and apply a histogram-based method as shown in Fig. 2.5. In our current implementation, we disable the gesture recognition function when the user is running or driving for safety concerns. If the rolling std is below the current threshold, we know that there is no gesture, and the samples are discarded. Otherwise, we start to buffer both accelerometer and gyroscope readings. We keep these buffered samples until the rolling std drops below the threshold, indicating that user is no longer moving and the gesture has finished. We then check the sample length and drop all the buffered samples if the length is too short or too long (a head gesture usually ranges from 30 to 240 samples at 50 Hz sampling rate).



**Figure 2.5:** Thresholds under different activities. The threshold will be set small when the user is sitting or standing, to enable even tiny head gesture detection (0.15). It will be set much larger when user is walking or running (0.7 and 1.3 respectively).

#### 2.3.2.4 Gesture Recognizer

The gesture recognizer is the core of the gesture recognition module. A head gesture is defined as a time series of gyroscope samples about 0.5s to 2s long. The raw gyroscope sensor data,  $S$ , can be written as an infinite stream of four-tuples, i.e.  $S = \{(x, y, z, t)_1, (x, y, z, t)_2, \dots\}$ . Likewise, a gesture  $G$ , is defined as a subset of sequential elements in  $S$ , i.e.  $G = S_{t \in [t_1, t_2]} \subseteq S$ . We refer to gestures that the system has already learned as “gesture templates”, denoted as  $Gt$ . Because the system is passively listen-

ing, the user can perform any gesture at any time, so the problem becomes finding the gesture  $G$  in the infinite time series  $S$  and identifying which template  $Gt$  is the closest match.

**Gesture Template Enrollment.** GlassGesture selects templates, from the gestures recorded as the user does, in a gesture template enrollment procedure. This allows the system to be maximally accurate for its user. During enrollment, we require users to sit still when they are recording a new gesture. The recorded time series are normalized and error cases are filtered out. We will create templates from the recorded gestures using a clustering algorithm called affinity propagation, which has been proposed as an effective method [3]. The selected gesture, i.e. the affinity propagation cluster center, is stored as a gesture template in the system for recognition later.

**Weighted Dynamic Time Warping.** We use the weighted DTW algorithm to measure how well two gestures match, which has several advantages such as simplicity, working directly on raw data, and computational feasibility on wearables [55]. DTW calculates the distance between two gestures, by scaling one in the time domain until the distance is minimized. As the workflow shown in Figure 2.6, the algorithm takes two time series; a potential gesture  $G$  and a gesture template  $Gt$ . It calculates a distance measure between them. Assuming that  $G$  is of length  $l$  and  $Gt$  is of length  $l_t$ , where  $i \in [1, l], j \in [1, l_t]$ , given a 3-axis gyroscope time series, we have

$$\text{dtw}(G, Gt) = \sqrt{w_x D_{l,l_t}^2(x) + w_y D_{l,l_t}^2(y) + w_z D_{l,l_t}^2(z)} \quad (2.1)$$

The function  $D$  denotes the matching distance or cost, which is calculated as

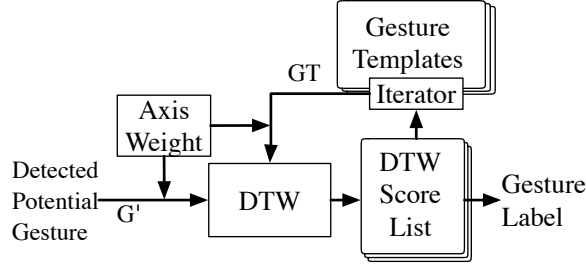
$$D_{i,j} = d(G(i), Gt(j)) + \min\{D_{i-1,j-1}, D_{i,j-1}, D_{i-1,j}\} \quad (2.2)$$

where  $d$  is a distance metric; we use *Euclidean distance (ED)*. We also add weights  $(w_x, w_y, w_z)$  to each axis to better capture the differences of gestures, since we have

found that head gestures have different movement distributions along each axis. For example, a nodding gesture has stronger component in the  $x$ -axis than the  $y$ -axis or  $z$ -axis. Weights are calculated by the *std* on each axis of the template as

$$w_x = \frac{\text{std}(Gt_x)}{\text{std}(Gt_x) + \text{std}(Gt_y) + \text{std}(Gt_z)} \quad (2.3)$$

The best match (minimal  $D_{l,l_t}$ ) is optimized in the sense of an optimal alignment of those samples. We can say that  $G$  matches  $Gt$ , if  $\text{dtw}(G, Gt)$  is below a certain threshold. To recognize which gesture is presented in a given window, we need to run DTW iterating all templates. Whichever template has the lowest DTW distance with the target, and is below a safety threshold, is selected as the recognition result.



**Figure 2.6:** DTW workflow

### 2.3.2.5 Efficient Similarity Search

DTW is a pair-wise template matching algorithm, which means that in order to detect a gesture naively, we need to traverse all gesture templates. It costs  $O(N^2)$  to compare two time series at length of  $N$  (we set  $l = l_t = N$  for simplicity), which is not efficient when there is a large number of gesture templates. We propose several schemes to optimize the performance.

Firstly, to reduce the search complexity, we want to build a  $k$ -dimensional ( $k$ -d) tree to do  $k$ -Nearest Neighbor ( $k$ NN) searches. However, tree branch pruning based on the triangle inequality will introduce errors if applied directly on DTW distances between gesture

templates, since DTW distance is a non-metric and does not satisfy the triangle inequality [96]. Therefore, we build the tree using *Euclidean distance (ED)* instead, which is a metric distance, and thus preserves the triangle inequality, allowing us to do pruning safely.

Secondly, to further reduce the computation, we down-sample the inputs before calculating the ED. Then we build the  $k$ -d tree. To recognize a target gesture, we first use the down-sampled target gesture to do the  $k$ NN search over the  $k$ -d tree. Then, we iterate over all  $k$  candidate templates to calculate the DTW distance with the target to find the best match with no down-sampling for the best accuracy.

The construction of a  $k$ -d tree is given in Alg. 1. And the  $k$ NN search is given in Alg. 2. Say we have  $m$  templates, which are all of length  $N$ . It costs  $O(m * N^2)$  when iterating over all the templates to match a target gesture, using DTW. The set of  $m$  gesture templates in  $N$ -space (each template is of length  $N$ ) can be firstly down-sampled to  $n_{ED}$ -space (each template is at  $n_{ED}$  length,  $n_{ED} \ll N$ ). We build a  $k$ -d tree of  $O(m)$  size in  $O(m \log m)$  time to process the down-sampled templates, of which the cost can be amortized. The  $k$ NN search query can be answered in  $O(m^{\frac{1}{n_{ED}}} + k)$ , where  $k$  is the number of query results. In total, the time cost is  $O(m^{\frac{1}{n_{ED}}} + k + k * N^2)$ .

---

**Algorithm 1** Build KD-Tree

---

```

1: procedure Build KDTrees(  $\mathbf{T}$ ,  $n_{ED}$ )
2:   for each template  $t$  in  $\mathbf{T}$  do
3:     downsampling to length- $n_{ED}$ 
4:     stored in  $\mathbf{T}_{down}$ .
5:   end for
6:   Build a KD Tree from  $\mathbf{T}_{down}$  using Euclidean distance, as  $\mathbf{Tr}$ 
7: end procedure

```

---

Lastly, we can also down-sample the gesture data before running DTW after the  $k$ NN search. The time cost will become  $O(m^{\frac{1}{n_{ED}}} + k + k * n_{DTW}^2)$  where  $n_{DTW} \ll N$  is the down-sampled length for DTW. However, it is non-trivial to choose proper  $n_{DTW}$ , since we don't want the down-sampling to remove important features of the time series. If this is the case, then the DTW algorithm may fail at differentiating two slightly different gestures.



---

**Algorithm 2** kNN search.

---

```
1: procedure kNN Search( $\mathbf{Tr}, t, k$ )  
2:   put  $k$  nearest neighbors of target  $t$  in tree  $\mathbf{Tr}$  into  $\mathbf{C}$ .  
3:   for each candidate in  $\mathbf{C}$  do  
4:     run DTW on target and candidate.  
5:   end for  
6:   return index of minimal DTW distances  
7: end procedure
```

---

We decide  $n_{DTW}$  through our experiments in the evaluation section.

### 2.3.3 Head-Gesture-based Authentication

Our system provides gesture-based authentication as an enhancement to secure the head gesture interface. One important question motivates us is that “can user head gestures be used as a password?”. In addition, we also need to answer challenging questions such as “what are the most suitable head gestures for authentication?”, “how do we select relevant features to distinguish different users?”, and “how difficult are those gestures to be forged?”.

#### 2.3.3.1 Two-factor Authentication using Head Gesture

As we mentioned previously, Glass does not have a secure and convenient authentication scheme. To secure the head gesture interface in GlassGesture, we propose the use of signatures extracted from simple head gestures. Specifically, we will ask the user to perform one or two simple gestures. In order to lead the user to perform a natural and instinctual gesture, when a user authenticates through GlassGesture, a “yes or no” security question, that can be answered using head gestures, is presented on the near-eye display. The user answers the question with head movements. In this way, the instinctual gestures (nodding and shaking) can be considered as containing signature head movements. After that, the answer (gestures) will be verified by the system. Features are extracted from motion sensors, then fed into a trained classifier. If the answer is correct and the classifier labels the gesture as belonging to the user, the user will be accepted.

Otherwise, it will reject the user. Thus, we form a two-factor authentication scheme. The user must know the answer to the security question (give the answer through a head gesture) and perform the gesture in the correct way. While we mainly test the feasibility of the “nod” and “shake” gestures, since they convey social meanings in answering questions, we do not rule out the possibility of other head gestures. This scheme has several advantages over the existing authentication done on the touchpad. First, the user does not have to remember anything, as the signatures we extract are inherent in their movement/gesture. Second, nod and shake are simple gestures taking almost no effort from user. Finally, an attacker cannot brute-force this system even with significant effort, because 1) the near-eye display is a private display, which can prevent shoulder surfing on the secure questions; 2) the signature of the head gestures are hard to observe by the human eye, unaided by any special equipment. Furthermore they are difficult to forge even with explicit knowledge of the features. Even with a recording of the user performing the gesture correctly, the attacker will have to wear the Glass and perform the gesture themselves in order to authenticate. This makes enumerating all possible inputs nearly impossible.

#### **2.3.3.2 Threat Model**

We have identified three types of possible attackers. The *Type-I attacker* has no prior information whatsoever. This attacker simply has physical access to the user’s Glass and attempts to authenticate as the user. Type-I attack is very likely to fail and ultimately amounts to a brute force attack, which can be mitigated by locking the device after a few consecutive authentication failures. The *Type-II attacker* may know the answer to the user specific security questions, but will try to authenticate with head gestures in their own natural styles (not trying to imitate the correct user’s motions or features). The *Type-III attacker*, the most powerful attacker, not only knows the answers to the security questions, but also is able to observe authentication instances (e.g., through a video clip). The

attacker can try to perform the gesture in a similar manner as the owner, in an attempt to fool the system. Note that, there is no security mechanism which can guarantee that the attacker will not be able to obtain the data on the device once the attacker has the physical access. The proposed authentication method can slow the attacker down, foil naive or inexperienced attackers, and make the task of extracting data from the device more difficult.

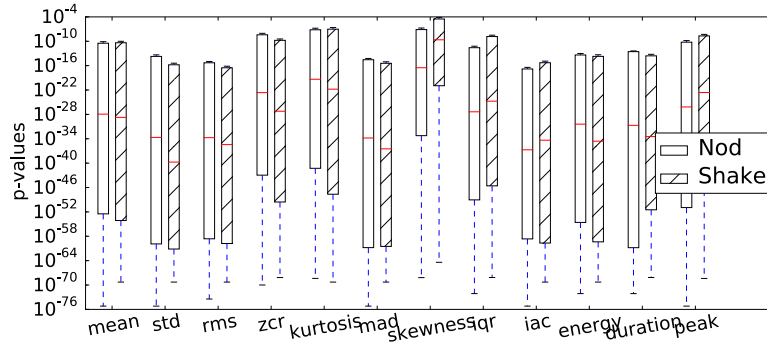
#### **2.3.3.3 Authentication Setup**

In the offline setup phase, the user first needs to establish a large set of security questions with answers. These questions could be something like “Is Ford the maker of your first car?”, “Is red your favorite color?” etc. Next, the user is asked to contribute an initial training set, from which a classifier model can be built. Because the classifier requires some training samples before sufficient accuracy is achieved (>30 training samples in our evaluation), the system can leverage the gesture recognition module to opportunistically collect instances of the “nod” and “shake” gestures. Whenever GlassGesture recognizes these gestures, we store these instances for classifier training in the authentication module.

#### **2.3.3.4 Feature Set and Data Collection**

We select statistical features such as, *mean*, *standard deviation (std)*, *root mean square (rms)*, *kurtosis*, *skewness*, *median absolute deviation (mad)*, *zero-crossing rate (zcr)* and *inter-quartile range (iqr)*. We also add features like *energy*, *duration* and *inter-axis correlation (iac)*. Additionally, we add a new category of features called *peak features* (including *average peak-to-peak duration*, *average peak-to-peak amplitude*, and *peak number*) by analysing peaks in the motion sensor data, which we have found effective at characterizing movements like head gestures. We collect motion sensor data of gesture samples from 18 users (gender: m/f: 14/4; age: 20-30: 11, 30-40: 5, 40+: 2.) while they are

answering yes or no questions using head gestures. We extract features from the raw accelerometer and gyroscope data on each axis, in total 84 unique features, for each sample. To test the effectiveness of the selected features, we run a two-sample Kolmogorov-Smirnov test (K-S test) to see whether the features of different users are from differentiable distributions. From the results in Fig. 2.7, we can find that all the  $p$ -values, returned by K-S test, are smaller than the significant level (0.05), which indicates the effectiveness of selected features.



**Figure 2.7:** K-S test results for gesture Nod and Shake

### 2.3.3.5 Training and Classification

SVM classifiers have been widely used in biometric-based authentication systems and radial basis function (RBF) kernels have been shown to have good performance [7, 81]. For the authentication problem, a one-class classifier is the most practical model since, at the training phase, the system can only gather training samples from the authorized user. However, ideally, if the system is able to gather enough negative instances, the one-class classifier might be outperformed by a two-class classifier, eventually. Therefore, for practicality concern, we report the one-class classifier results to assess our gesture-based authentication system. The training phase happens offline. We use grid search to get the optimal parameters for the one-class SVM classifier (OCSVM) with RBF kernel in 10-fold cross validation. To further improve the classification performance, we employ a one-class ensemble SVM method (OCESVM) [68] to combine multiple classifiers. The

basic idea is that we collect and rank multiple sets of parameters by the true positive rate (TPR) with a constraint on the false positive rate ( $FPR < 1\%$ ) during the grid search. Then the top- $r$  (we set  $r = 5$  empirically) classifiers are chosen to form an ensemble classifier using majority voting on the classification decisions. We use the trained model to classify the test samples. The test samples can be labeled in one of two ways: 1) samples from the authorized user; 2) samples from unauthorized users. We will present the evaluation of our training and classification in the next section.

#### **2.3.3.6 Feature Selection**

The rationale of our feature selection method is that different gestures have different weights in a 3-D space. While our features are extracted from three axes, it is possible that a gesture in 3D space may be well characterized by features extracted from data of only one (1D) or two axes (2D). Therefore, we apply recursive feature elimination (RFE) [33] to eliminate redundant or useless features, which will increase accuracy and reduce delay. In RFE, the training process will recursively select a subset of the features. However, RFE usually works with multi-class classifiers, not one-class classifiers. Therefore, we propose a new way of applying RFE in one-class classification. The training set in one-class classification are all positive instances (same class labels). The basic idea is to divide the training set into several groups evenly and manually assign each group a different virtual class label, to turn the one-class training set into a multi-class one. Then we apply RFE on the “fake” multi-class training set, during which we use a 10-fold cross validation and vote on the features in each run. Since features which top the voting result contribute most in differentiating those virtual groups, we eliminate features with more than  $e$  votes and finally train a one-class SVM classifier with the rest of features. The problem here is how to determine the value of  $e$ . Through our experiments we empirically set  $e = 3$ , which gives the best performance in most of our trials. We will evaluate this feature selection scheme in experiments.

## 2.4 Evaluation

Currently, we have implemented GlassGesture as a Google Glass application using Glass SDK. We adopt FastDTW implementation [76] to build the gesture recognition module. For gesture authentication module, we compile libSVM [14] as native code to implement the classifier via Android NDK. The model is trained offline on a remote machine (MacBook Air, i5-1.3GHz and 4GB-RAM). In this section, we will evaluate our system in two modules individually. We will report performance metrics in terms of TPR ( $\frac{tp}{tp+fn}$ ), FPR ( $\frac{fp}{fp+tn}$ ), accuracy and F1-Score ( $\frac{2tp}{2tp+fp+fn}$ ) for gesture recognition and authentication.

### 2.4.1 Gesture Recognition

We prime our system with eight command gesture templates: *nod* and *shake*, *left* and *right* 3 times, *triangle* and *rectangle*, and *cw/ccw circle*. We also prepare our system for *number* and *alphabet* input. We choose those head gestures in order to evaluate the capability of gesture recognition on various gestures.

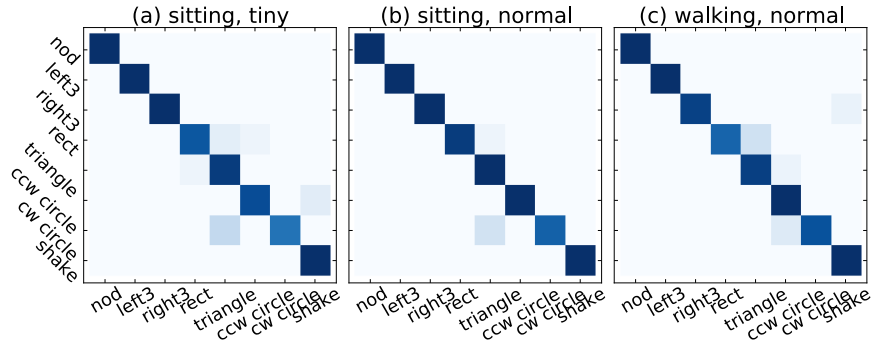
#### 2.4.1.1 Gesture Recognition with command gestures

For each of these command gestures we conduct gesture data collection three times, once with as little head movement as possible (tiny) in sitting, and a second time with a normal/natural amount of head movement (normal) in sitting, and a third time in a normal amount of head movement in walking. This experiment is repeated for multiple rounds with each round collecting about 10 gestures. The results of accuracy in Figure 2.8 is in the form of confusion matrices.

**Gesture in sitting.** From results in Figure 2.8 (a, b), we can see that for several gestures, such as *nod*, *left3*, *right3*, *shake*, the accuracy is perfect, even in the tiny gesture case. The reason behind is that the gesture has a repeating pattern in itself, which distinguishes it from other miscellaneous movements. The most easily confused gestures are *clockwise circle* and *triangle*, because of similar shapes in a clockwise direction. When

the user tries to make her gesture very tiny, the head movement space is suppressed greatly, which will make these two gestures indistinguishable. Since our system allows users to define their own gestures, we can notify them in case new gestures are too similar to any pre-existing templates to ensure the best user experience.

**Gesture in walking.** When a user is walking, it is rather natural that the user will perform gestures in an obvious, unconstrained way. Otherwise, this gesture will just be buried in the noise of her walking. From the confusion matrix in Figure 2.8 (c), we find minor deterioration of accuracy in recognizing gestures such as *right3* and *rect*, which we believe is caused by noise of walking movement. However, the *triangle* and *clockwise circle* are more distinguishable, which as we find is easier for the user to perform while walking rather than sitting.

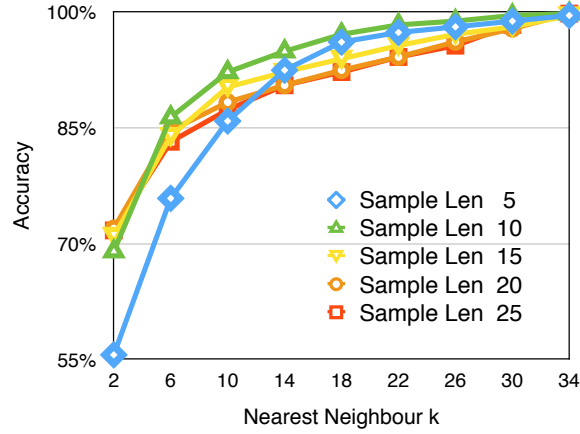


**Figure 2.8:** (a) Confusion matrix of command gestures (sitting, tiny). TPR: 92.87%, FPR: 5.7%. (b) Confusion matrix of command gestures (sitting, normal). TPR: 96.99%, FPR: 2.4%. (c) Confusion matrix of command gestures (walking, normal). TPR: 94.88%, FPR: 4.6%

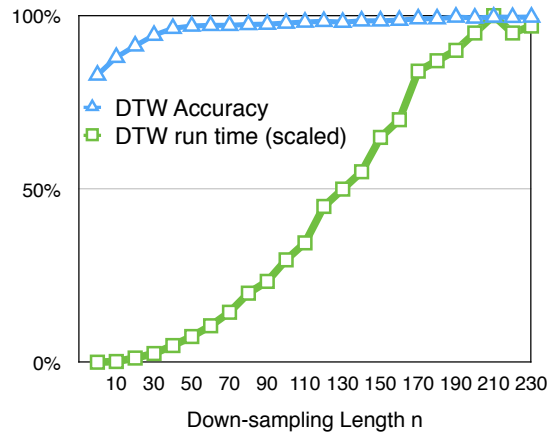
#### 2.4.1.2 Number and Alphabet Input

Next, we evaluate gesture recognition accuracy when we use head gesture as *number* and *alphabet* input method. Users are asked to draw 0-9 and a-z for at least 10 times to evaluate the accuracy. While 35 of total 36 gestures are 100% identified, the only error is one instance of number 9 is mis-recognized as number 7. The failures are due to the limitation of template matching, i.e. writing 7 and 9 are just too similar if user doesn't

write them carefully via head movement. One way to improve the result in this case is to explore different styles of writing, which is out of the scope of this dissertation.



**Figure 2.9:** Accuracy changes with sampling lengths ( $n_{ED}$ ) and numbers of nearest neighbours.

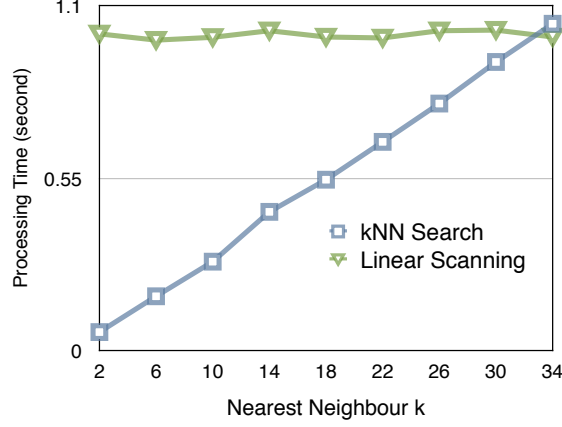


**Figure 2.10:** Accuracy and scaled running time using DTW change with sampling lengths ( $n_{DTW}$ ).

### 2.4.1.3 Gesture Recognition Performance

To demonstrate the performance of gesture recognition, we evaluate it with the processing of 36 gestures of *number* (0-9) and *alphabet* (a-z). Firstly, we want to determine the proper down-sampling length  $n_{ED}$  for calculating *Euclidean distant* used in  $k$ NN search and  $n_{DTW}$  for calculating DTW used in template matching. In Figure 2.9, we evaluate the



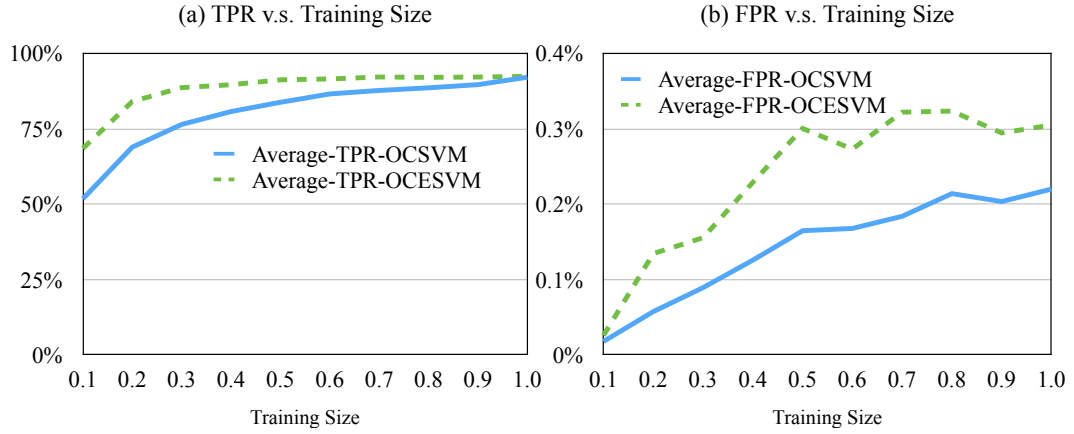


**Figure 2.11:** Running time comparison between our scheme and linear scanning.

gesture recognition accuracy at different down-sampling lengths ( $n_{ED}$ ) and numbers of nearest neighbours ( $k$ ). We found that when the  $n_{ED}$  is set as 10, it gives best accuracy. In Figure 2.10, we change the  $n_{DTW}$  in the linear scanning using DTW distance metric. The time cost grows exponentially with the input length, while the accuracy can reach a satisfactory level when down-sampling length is as small as 40 or 50. Next, we show the processing speedup of our scheme against the linear scanning baseline. The results are shown in Figure 2.11. We set  $n_{ED} = 10$ ,  $n_{DTW} = 50$ . The number of nearest neighbours to be searched can be set to 10-14, which is a reasonable trade-off between processing speed and accuracy based on Figure 2.11 and Figure 2.9. The running time will be reduced by 70% when  $k = 10$  and 55% when  $k = 14$ . We use  $k = 14$  in our system.

#### 2.4.2 Authentication Evaluation

We have collected motion sensor data from 18 users while they are answering yes-and-no questions using head gestures. We have gathered around 100-150 trials for each gesture of each user. Those data are pre-processed and used for feature extraction, model training and evaluation.



**Figure 2.12:** The average TPR (a) and FPR (b) change with different ratios of training samples.

#### 2.4.2.1 Impact of Number of Training Samples

Before training the model, we want to decide an appropriate size of training samples since it will be a trade-off between authentication accuracy and user convenience. We run the one-class SVM (OCSVM) training process with 10-fold cross validation. Based on trained models, we also build a one-class ensemble SVM classifier (OCESVM). As plotted in Figure 2.12, we increase the percentage of training samples from 0.1 to 1.0, use the rest as test samples, and report average TPR and FPR of all users. We find that 30 samples (0.2 ratio) is sufficient to achieve an average TPR higher than 70% and keep an average FPR lower than 0.3%. OCESVM shows great gain of TPR and slight deterioration of FPR when the sizes of training samples are small. Therefore, in our system, we build the training set passively and continuously in the background every time the user performs those gestures. We employ OCESVM when the size of training samples is insufficient, and fall back to OCSVM for system overhead concern when the gathered training samples are adequate. This scheme eases the burden of training on users significantly while maintaining high TPR and low FPR at the same time.

| Single                             | TPR              | FPR             |
|------------------------------------|------------------|-----------------|
| GlassGesture Nod                   | 92.43% (+/-3.09) | 0.09% (+/-0.22) |
| GlassGesture Shake                 | 92.33% (+/-3.32) | 0.17% (+/-0.33) |
| GlassGesture Left3                 | 89.08% (+/-6.36) | 0.48% (+/-0.79) |
| GlassGesture Right3                | 89.61% (+/-5.99) | 0.52% (+/-0.87) |
| Multiple and Comparison            | TPR              | FPR             |
| GlassGesture (2 gestures)          | 99.16%           | 0.61%           |
| Touchpad+ Voice (5 events) [67]    | 97.14%           | 1.27%           |
| Touchscreen GEAT (3 gestures) [81] | 98.2%            | 1.1%            |

**Table 2.2:** FPR and TPR of authentication on two gestures.

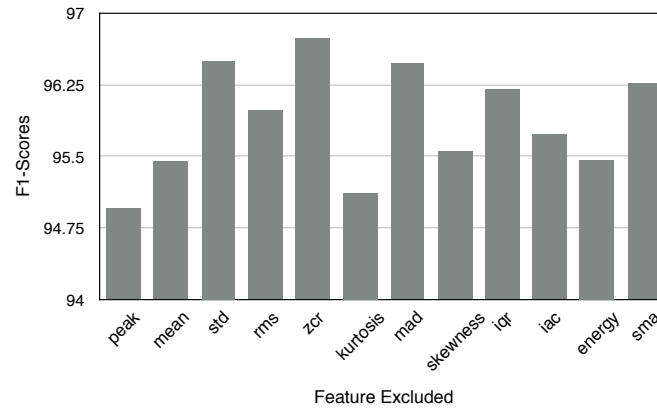
#### 2.4.2.2 Authentication against Type-II attacker

In order to understand the authentication performance, we evaluate the authentication against Type-II attackers, which are more powerful than Type-I attackers. We utilize the whole data set to train the model with 10-fold cross-validation for each user. While training model for a certain user, we use data samples from all other users as Type-II attacking trials. The result is shown in Table 2.2 in metrics of TPR and FPR and compared with several existing works. From the result, we can tell that our authentication system can identify authorized users with a high TPR as average 92.38% and defend against Type-II attackers with a low FPR as average 0.13% if using Nod and Shake gestures. We are careful to bother no authorized user with occasional false positives. However, since gestures are very short, cost nothing, and are easy to perform, we assume that the user is willing to go through authentication multiple times which can basically eliminate the false positives. We compare authentication performance using two consecutive gestures with work [67] and [81], where both one class classifiers are used. Work [67] combines touchpad and voice commands to authenticate user in Google Glass. Our scheme requires fewer gestures, less effort, and produces better result. Work [81] is about touchscreen-based authentication on smartphone, while we show that we can achieve competitive performance using head gestures on Google Glass. Another work [15] uses a two-class

SVM classifier, which only reports the average error rate (AER, defined as  $\frac{1}{2}(1 - tpr + fpr)$ ) as 0.04 while using 5 touchpad gestures on Glass. Our scheme requires fewer gestures, and better result when multiple gestures are combined.

### 2.4.2.3 Impact of Peak Features

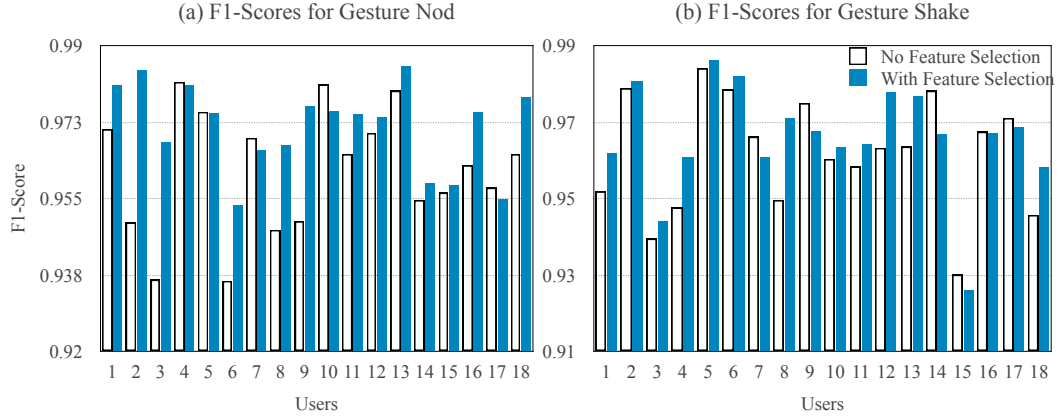
With the intention to investigate the impact of peak features, we use an feature-excluding method to verify the effectiveness of peak features. We collect number of true positive, false negative and false positive to calculate the F1-score as a metric to show the overall performance of the classification. In Figure 2.13, the F1-score is lowered the most when peak features are excluded. Some other important features are *mean*, *energy*, *kurtosis* and *skewness*.



**Figure 2.13:** The F1-score of certain category of features is excluded.

### 2.4.2.4 Impact of Feature Selection

To show how our feature selection method helps in our case, we compare the F1-scores of classification with and without feature selection. From Figure 2.14, we can find that for majority of users (13/18 and 12/18 respectively), feature selection improves the classification.



**Figure 2.14:** F1-Scores of one-class SVM with or without feature selection for gesture nod (a) and shake (b).

#### 2.4.2.5 Imitator Attack

In this evaluation, we want to know whether an Type-III attacker (imitator), can fool the authentication system. We start by taking a short video of a victim while she is performing gesture-based authentication, and then present this video to attackers. We give attackers enough time to learn, practice, and mimic the victim. And we only start the authentication process whenever each attacker feels she is ready. We give 5 attackers 10 chances for each gesture and unlimited access to the reference video. In all of our tests (100 trials), attackers are never able to fool the system and (falsely) identified as authorized users. From the experiment, we find that an imitator fails in mimicking the head gesture because 1) it is not easy to recover every details of head gestures recorded by sensitive motion sensors through vision observation; 2) it is not easy to control the head movement precisely and make it like a natural movement during mimicking. We suspect that the different muscle distributions of head and neck in human individuals will add different features to the sensor recordings.

| DTW<br>per instance | Training<br>per user | Classification<br>per instance |
|---------------------|----------------------|--------------------------------|
| 30.2 ms             | 42.8 s               | 28.6 ms                        |

**Table 2.3:** Average Time Cost

### 2.4.3 System Performance

We report the running time of several important functions in Table 2.3: DTW time cost in gesture recognition, training time cost (offline on a remote machine), and classification time cost in authentication. The time cost of gesture recognition grows linearly with the number of templates, while the time of running one instance of DTW is rather small as 30.2 ms. The training is offloaded to a remote machine and cost average 42.8 seconds per user, which is affordable since the request of training and retraining is relatively rare after the initial setup. Classification runs on the Glass, of which the cost (28.6 ms) of single instance is almost unnoticeable by users.

### 2.4.4 Other considerations

Due to space limit, we briefly discuss other practical considerations. 1) Authentication Frequency: The frequency is depend on the usage pattern of user. The default setting is to authenticate user after booting or being taken-off, which is infrequent. 2) Biometric Invariance: We have been keeping collecting gesture samples from several users during a week. We have not noticed much difference in recognition/authentication accuracy. However, we do add template adaptation [55] and classifier retraining to our system in case of any performance deterioration. And we have fail-safe authentication after consecutive failures. We are still lack of enough data to claim that human head gesture is invariant in a long term. We plan to investigate this problem in the future. 3) Power Consumption: Based on the energy consumption reported in [72] and [54], the battery life of constantly sampling sensors is 265 mins (300 mins daily in normal usage). We are expecting a much longer lifetime since our implementation is not always-on. The device will enter a

low-power mode after a short period of inactive. It responds to wake-up events [27] and then the gesture interface will be enabled accordingly.

## **2.5 Chapter Summary**

In this chapter, we propose GlassGesture to improve the usability of Google Glass. Currently, Glass relies on touch input and voice command and suffers from several drawbacks which limits its usability. GlassGesture provides a new gesture-based user interface with gesture recognition and authentication, which enables users to use head gestures as input and protects Glass from unauthorized attackers. We utilize activity context information to adaptively set threshold for gesture detection. We optimize the performance of GlassGesture using several novel schemes including efficient similarity search, weighted dynamic time warping (DTW) and an ensemble scheme of one class SVM classifiers. We propose a new category of features which are effective at characterizing head gestures. GlassGesture achieves high gesture recognition accuracy. For authentication, GlassGesture can accept authorized users and reject attackers with high confidence.

## Chapter 3

# WearLock: Unlocking Your Phone via Acoustics using Smartwatch

### 3.1 Introduction

Smartphone stores a wide variety of sensitive information of the owner, such as identities, locations, banking accounts, photos and videos, addresses, contacts, emails, text messages, and social account profiles, etc. Effective protection of smartphone data is critical against the compromise of personal information. When not in use, unattended smartphones should be locked with a personalized credential to prevent access from unauthorized persons. Every smartphone operating system now has a built-in screen lock application, which enables users to unlock their smartphones via PINs, passwords, patterns, etc. However, the reality is that a significant portion of users never lock their smartphones. A recent study [89] indicated that 53 out of 150 (35%) of participants have never enabled any sort of screen locks and the primary reason was due to the inconvenient input methods of screen locks. In another study [37], a large portion of participants (57.1%) indicated that they use none or naive screen lock (e.g., slide-to-unlock) while lots of participants (46.8%) agreed that unlocking their phones can be annoying and many of them (25.5%) admitted that they want a way to unlock their phone much easier. There-



fore, the problem of user authentication on mobile devices is how to balance the security and the user experience [1].

To address this problem, one direction is to reduce the number of unlocks upon existing authentication mechanisms. There are two common approaches. One is to provide partial functionality on lock screens, such that the user can interact with the smartphone before unlocking. This technique potentially reduces the number of unlocks, thus easing the unlock burden on users but at the cost of information security. For example, this approach may display several lines of an incoming email on a locked screen for user. However, these few lines may contain sensitive data. Further judgments from users are needed to determine what functionality or information is safe on the locked screen. The other approach is to choose the right moment to surface the authentication instead of enforcing it at every user session [75], which eventually relies on some sort of implicit authentication methods. This scheme is not suitable for screen lock due to noticeable delays [37].

Another more promising direction to solve this problem without security tradeoff is to find the most suitable authentication method for mobile devices. The commonly seen authenticators on smartphone can be categorized into *passwords* (“what you know”), *biometrics* (“who you are”), and *tokens* (“what you have”) [63]. The term *password* here includes words, phrases, patterns, PINs, or their combinations, which are used as secrets for authentication. However, this approach is problematic on mobile devices for several reasons. First, simple passwords are easy to guess while strong passwords are hard to remember. Second, the input environments on mobile devices introduce difficulties for users to enter passwords consisting of characters, digits, and symbols. Third, even though pattern or graphical passwords are much easier to input but all those passwords including previously mentioned are susceptible to shoulder surfing attacks or smudge attacks.

Alternatively, *biometric*-based authentication uses unique features (e.g. fingerprints, eye iris, faces, voices, etc.) extracted from the human body and is considered convenience and secure. Recent work has also considered various gestures and inputting

habits [16, 35, 87, 102] as biometrics. However, one big disadvantage of biometric-based authentication is that those biometrics are uniquely tied to human body and are not as replaceable as passwords or tokens when being compromised or disclosed [10, 48, 110]. The *token*-based authentication usually includes contact-less proximity card, smart card with static or dynamic tokens. The advantages of token are easy-to-use, no need to memorize passwords, while the disadvantage is the cost of additional hardware.

In this work, we seek a smartphone authentication solution in line with *token*-based method. Ideally, we want a secure screen lock that 1) authenticates user on every user session; 2) is resistance to malicious observers; 3) requires minimal effort from user. Originally, the token-based solution is less favored due to the cost of additional hardware. However, due to the increasing popularity of smart things and wearables, this solution has re-gained attentions [8, 49, 90]. Based on a market research of Kantar Wearable Technology [46] and Morgan Stanley [59], 12% of US consumers own at least one wearable device while 55% of consumers have intentions to buy at least one wearable device. Hence, we envision that many smartphone users will possess at least one peripheral wearable device, such as a smartwatch or smartband, in the near future. Therefore, we decide to leverage pervasively co-located trusted devices for token-based authentication to create an automated and secure screen lock approach. Nevertheless, it is not easy to find a proper channel to conveniently establish a secure range to associate smartphone with co-located trusted devices (e.g. a smartwatch in our system). Solutions utilizing Near Field Communications (NFC) tags as trusted devices require users to manually attach a tag close to the phone's NFC antenna to achieve proximity of 10 cm or less. Solutions based on Bluetooth-enabled wearables, speakers and cars, can constantly connect to smartphones, but the connection range of Bluetooth cannot be guaranteed. Variants such as device model, paired device, and local environment may sustain a Bluetooth connection up to 100 meters in distance [31]. In our preliminary experiment, we have confirmed that "android trusted device" based on Bluetooth does not lock one's phone until the trusted device is 10-15 meters away in line-of-sight case or 2-3 rooms away in none-line-of-sight

case. If someone takes your smartphone and stays not too far away from your trusted device, he may access your unlocked phone since your trusted device is still connected via Bluetooth.

To address these concerns, in this project, we propose to exploit the acoustic channel to build the trusted relationship between a smartphone and its associated smartwatch and automatically unlock the phone when the smartwatch is nearby. To this end, we build WearLock, a system to automatically unlock smartphones via an acoustic channel between a smartphone and its associated wearable (a smartwatch in this work) [100]. To be noted that our system is not meant to replace current smartphone authentication schemes (password or biometric based authentications), but to provide a secure and efficient alternative which can significantly reduce authentication effort of users. The assumption is that with a given noise level, we can maintain a secure and ranged acoustic channel within roughly 1m distance between two devices using speaker and microphone. Microphones and speakers are commonly available on these devices, eliminating the need for extra hardware additions. The communication range of acoustic channel is shorter than the Bluetooth and longer than NFC or magnetic-based channel [43], which is more desirable for the purpose of unlocking smartphones. One challenge is to build a robust and reliable acoustic modem scheme to secure the acoustic channel when devices are nearby. The other is to carefully design a system to accommodate the limited battery capacity and computation power of wearable hardware.

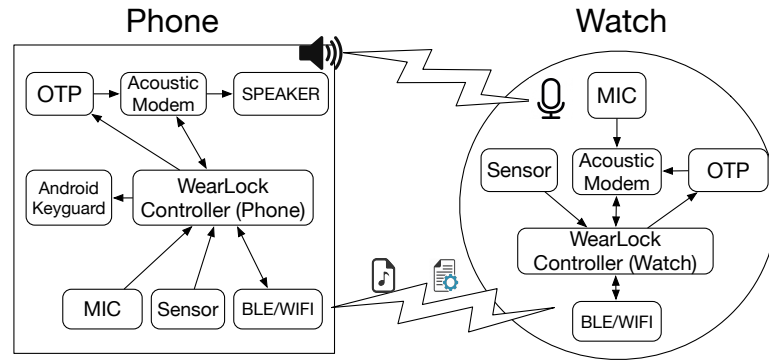
In summary, we make the following contributions:

- We proposed a novel automated and secure unlocking scheme for smartphone via a trusted wearable device. It requires minimal amount of effort from user.
- We are the first, to the best of our knowledge, to exploit the adaptive modulation of acoustics on common off-the-shelf (COTS) mobile devices for robust data transmission. The acoustic modem can adapt to ambient noise levels and interfering signals.

- We built WearLock on unmodified COTS smartphone and smartwatch devices and evaluated the system extensively.

## 3.2 System Overview

In this section, we describe the system architecture of WearLock and the smartwatch-assisted unlocking protocol.



**Figure 3.1:** The architecture of WearLock.

### 3.2.1 System Architecture

Figure 3.1 illustrates the architecture of WearLock, which consists of a smartphone and a smartwatch. The smartphone usually has a speaker and microphone, a wireless interfaces (Bluetooth or WiFi), and optionally motion sensors. The smartwatch usually has a microphone, a wireless interface and optionally motion sensors. Each device runs an instance of WearLock Controller, as the agent executing our proposed unlocking protocol, which takes input from underlying hardware and controls the the output channels such as speaker for emitting acoustics, wireless radio for sending configurations, and Android Keyguard for enabling or disabling lock screen. The one time password (OTP) module is responsible for the one time password generation and verification. The acoustic modem

is an OFDM modem which enables data such as OTP to be transmitted over the acoustic channel using proper modulation schemes.

The smartphone and the smartwatch communicate with each other through both the wireless and the acoustic channels. The wireless channel serves as the secure control channel, transmitting acoustic channel configuration information, including the pilot sub-channel, the null sub-channel, and the data sub-channel. The acoustic channel conveys data payload in data sub-channels along with pilot sub-channels. The motion sensor will be used to construct a pre-filter to skip unnecessary unlocking requests by matching the motion pattern. In the following sections, we will provide further details on the acoustic OFDM modem design, the secure unlocking scheme, and several system optimizations.

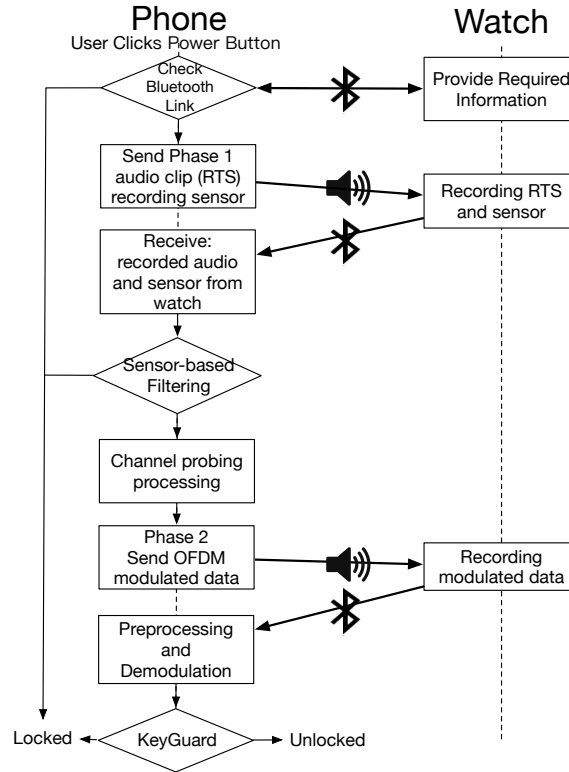
### **3.2.2 Smartwatch-assisted Unlocking Protocol**

Figure 3.2 illustrates the overall protocol of WearLock between the smartphone and the smartwatch. The protocol has two phases: 1) Phase 1 is Request-to-Send/Clear-to-Send (RTS/CTS) phase for channel probing; and 2) Phase 2 is the data transmission phase for OFDM modulated OTP token.

*Smartphone's view:* To avoid continuous probing and monitoring, we design to start our protocol when the user clicks the power button. The smartphone detects the presence or absence of the wireless link with the smartwatch. When the wireless link presents, the smartphone continues to evaluate the motion patterns of the smartphone and the smartwatch, respectively. If the motion patterns match, it is assumed that both devices are co-located and the smartphone continues to operate by verifying recorded audio token from the smartwatch. If the token is valid, then the Android Keyguard service will set the smartphone in screen unlocked state. During this process, if the wireless link, or the motion pattern, or the token validation fails, subsequent computations will be skipped and the Android Keyguard will remain the smartphone locked.

*Smartwatch's view:* The smartwatch runs a thin client, which cooperates with the

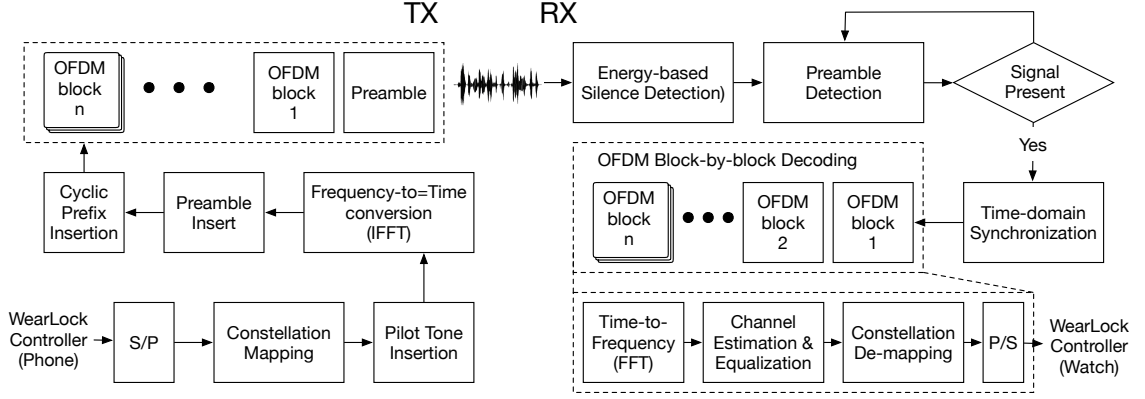
smartphone controller. The smartwatch transmits information such as Bluetooth/WiFi status, sensor data, and recorded acoustics over the wireless channel to the smartphone.



**Figure 3.2:** The Protocol of WearLock.

### 3.3 Acoustic Modem Design

We designed and implemented a software modem for reliable data transmission over the acoustic channel. The goal is to meet the challenge of achieving robust communication under different ambient noise environments. We first discuss important characteristics of the acoustic channel. Then, we will describe our modem design, which includes signal detection using preamble identification, time synchronization using preamble and cyclic prefix, channel estimation and equalization with pilot tone, and signal modulation/demodulation. Figure 3.3 shows the block diagram of the OFDM modem design.



**Figure 3.3:** The OFDM modem of WearLock.

### 3.3.1 The Acoustic Channel

Before diving into the design of acoustic modem, it is necessary to understand the important aspects of the acoustic channel, which significantly shape our design decisions. Next, we will discuss details of our OFDM modem design followed by practical considerations of our implementation.

#### 3.3.1.1 Ambient Noise

Ambient noise directly affects the Signal-Noise-Ratio (SNR) at the receiver side. While ambient noise introduces challenges, it also provides opportunities for co-location detection [47]. To measure the sound or noise power, we use the sound pressure level (SPL), which is defined as

$$\text{SPL} = 20 \log_{10} \frac{p}{p_{\text{ref}}} \quad (3.1)$$

where  $p$  is the root mean square (RMS) power and  $p_{\text{ref}}$  is a reference value.

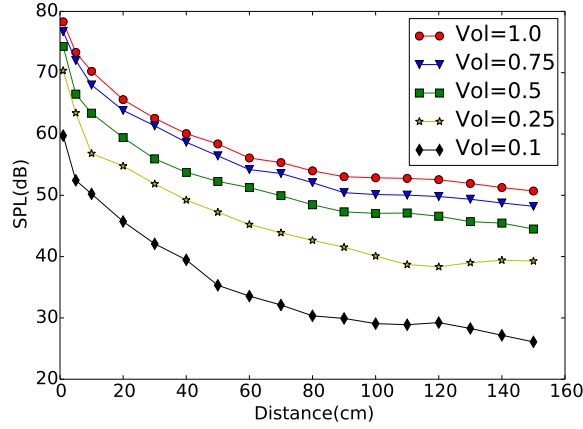
#### 3.3.1.2 Sound propagation and attenuation

In the open air, the sound attenuation is mainly caused by spreading loss. Assume that  $\text{SPL}_{tx}$  and  $\text{SPL}_{rx}$  are the sound pressure levels at the transmitter and the receiver, respectively, and the distance between the transmitter and the receiver is  $d$ , then the sound

attenuation in the open air is defined as:

$$\text{SPL}_{\text{tx}} - \text{SPL}_{\text{rx}} = 20g \log_{10}\left(\frac{d}{d_0}\right) \quad (3.2)$$

where  $g$  is a geometric constant, with  $g = 1$  for spherical propagation from a point source, and  $d_0$  is a reference distance, i.e., the distance between transmitter's microphone and speaker [19].



**Figure 3.4:** Receiver's SPL in distance of different volume settings. Measured in a quiet room with the SPL of ambient noise about 15-20 dB in a line-of-sight scenario.

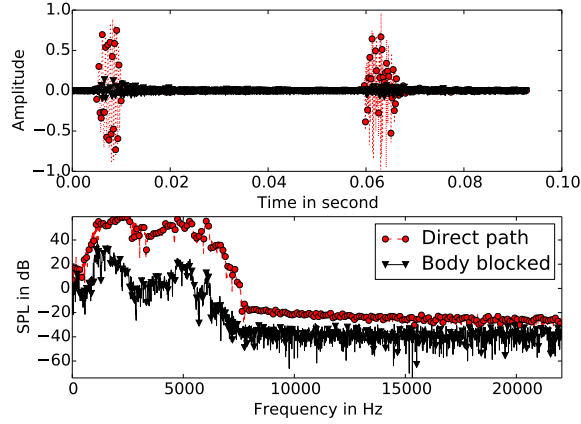
In WearLock, we control the propagation range of acoustic signal by adjusting the speaker volume. We have measured the SPL at the receiver under line-of-sight (LOS) scenarios with different distances and volume settings, and the results are shown in Figure 3.4. From the figure, we can see that SPL attenuation matches well with the theoretical value in spherical propagation, decreasing by about 6 dB when the distance is doubled. Therefore, the Signal-to-Noise (SNR) at the receiver side can be estimated by

$$\text{SNR}_{\text{rx}} = \text{SPL}_{\text{rx}} - \text{SPL}_{\text{noise}} \quad (3.3)$$

where  $\text{SPL}_{\text{noise}}$  is the SPL of ambient noise.

While in most cases of LOS propagation, the signal experiences only spreading loss,





**Figure 3.5:** The received signal comparison of LOS direct path (BER=0.0) and Body-blocked NLOS (BER=0.54).

it is possible that occasional signal transmission occurs through human body when the LOS path is partially blocked by the user's body. Human body signal occlusion between the transmitter and receiver results in heavy absorption loss and delay spreading. We have measured the same signal transmitted under both signal paths. From the results plotted in Figure 3.5, we can see a drop as large as 20dB in the received signal strength, with increased bit error rate (BER) of the acoustic transmission. Careful attention to both signal path configurations is needed in order to maintain a robust acoustic communication system.

### 3.3.1.3 Multipath Effect

Signals that reflect off a local structural features form additional signal components that contain the same information as the original signal but may be delayed in time or weaker in signal strength. An experiment using COTS smart devices located randomly within 5 meter rang has shown that in a typical indoor environment severe multi-path propagation exists [50], which means it is possible that one symbol will interfere with other symbols from other multipath components. In our preliminary evaluation, our acoustic signal is very short (about 1-2 OFDM symbols) and its power is controlled for transmission in a

1-2 meter short range, we have not observed significant multipath fading. We always received the strongest direct path component, followed by very weak multipath components sometimes. However, we do observe a delay spreading of multipath effect in NLOS cases.

#### 3.3.1.4 Microphone and Speaker Characteristics

*Ringling effect* and *rise effect* adversely affect speaker and microphone performance [61]. Ringing is the effect that the speaker generates a longer output than the real length of input with a reverberation tail slowly reduced to zero. Similarly, rise effect is due to the fact that the speaker unit cannot reach to its highest power instantly. To overcome these effects, we define a guard interval  $T_g$  larger than the largest reverberation length to reduce the inter-symbol interference (ISI), and we also apply fading at the beginning of the signal.

### 3.3.2 OFDM Design

WearLock leverages orthogonal frequency division multiplexing (OFDM) modulation to encode the acoustic token. OFDM efficiently utilizes the spectrum by allowing overlap in the frequency domain. It is also more resistant to frequency selective fading by enabling sub-channel selection and equalization techniques.

#### 3.3.2.1 Modulation and Demodulation

The OFDM modulation and demodulation are implemented through Fast Fourier's Transformation (FFT) algorithms. Considering a data sequence input to the inverse FFT (IFFT),

$$X = [X_0, X_1, \dots, X_k, \dots, X_{N-2}, X_{N-1}] \quad (3.4)$$

where  $X_k = X_I(k) + jX_Q(k)$ , which is in the form of quadrature amplitude modulation (QAM). The real and imaginary axes are often called the in phase, or I-axis, and the

quadrature, or Q-axis. Usually, the conversion back and forth between a binary data and the QAM-represented data input is done through a constellation mapping/de-mapping. To get the baseband modulated time-domain signal, we apply the IFFT:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_A(k) e^{j(\frac{2\pi}{N} f_k t_n + X_P(k))} \quad (3.5)$$

where  $f_k = k/(N\Delta t)$ ,  $F_s = 1/\Delta t$  is the sampling rate, and  $t_n = n\Delta t$ , and

$$X_A(k) = \sqrt{X_I(k)^2 + X_Q(k)^2} \quad (3.6)$$

$$X_P(k) = \arctan(X_Q(k)/X_I(k)) \quad (3.7)$$

Then the final representation of the signal is its real part  $s_n = \text{Re}(x_n)$ . We directly use this base-band signal as our output acoustic signal and send it through the speaker. To demodulate a received time-domain signal, we apply the FFT and then look at the complex representation of  $X_k$  in the result, and de-map it according to the constellation diagram. However, due to the characteristics of the acoustic channels, which present delay, attenuation and phase distortion issues, we need to implement synchronization, sub-carrier selection, channel estimation and channel equalization.

### 3.3.2.2 Sub-carrier Frequency Range

Originally, we want to work on the near-ultrasound frequency ranged from 15kHz to 20kHz for the following reasons: 1) the frequency range of most ambient noise in our scenarios is below 15kHz; 2) humans are most sensitive to frequencies between 2,000 and 5,000 Hz; and 3) many new smart devices support native 44.1kHz or even higher sampling rate which indicates that the frequency response is acceptable below 20kHz. However, in real device experiment (A Moto 360 Android watch), we have found that there is a mandatory built-in low-pass filter, which limits the frequency range no higher than 7kHz, where

the signal fades significantly from 5kHz to 7kHz<sup>1</sup>. Therefore, our final design supports a smartphone-smartwatch pair utilizing audible acoustic signals (1kHz-6kHz) and an emulated smartphone-smartphone pair utilizing inaudible near-ultrasound acoustic signals (15kHz-20kHz).

### 3.3.2.3 Preamble Design

Existing preambles used in OFDM modems are usually based on PN-sequence or linearly frequency modulated (LFM) signals. The PN-sequence signal is a sequence of signal that has very strong auto-correlation output and weak cross-correlation output. The LFM signal is also known as Chirp signal or Sweep signal, which has nice Doppler-shift insensitivity and can be accurately detected in matched filtering. In our modem, we adopted a chirp signal for signal detection and coarse synchronization. The chirp signal increases from  $f_{\min}$  to  $f_{\max}$  in a time frame  $T_p$ .

### 3.3.2.4 Silence Detection and Signal Detection

The purpose of signal detection is to find the target signal in the recorded acoustic stream. First, we use an energy-based detector to filter out the section of silence. When there is a strong signal with SPL that surpasses our predefined noise level, we will perform the signal detection, relying on the detection of a known preamble. A cross-correlator calculates a normalized score and compares against a threshold value. Once we have detected a target signal, we will send this audio buffer to the next processing block.

### 3.3.2.5 Synchronization

Finding the start of a frame is critical to all the following operations and thus the system performance. Our synchronization has two steps: a coarse time-domain synchronization and a fine time-domain synchronization. The coarse synchronization in time-domain is

---

<sup>1</sup>We deem the reason of filtering as the main microphone usage in Android wear is speech recognition. We are planning to test on more android wear models.

done during the preamble detection through cross-correlation of the received signal and the known preamble. The preamble is a chirp signal, which correlates well with the original chirp even if there is a frequency shift. This property ensures that we can always find a coarse start of the frame. During the processing of the OFDM symbol, we perform the fine time-domain synchronization by leveraging the cyclic prefix. The cyclic prefix is a technique prefixing a symbol with a repetition of its end, which usually serves as a guard interval to eliminate ISI and is a technique to improve the robustness of multi-path propagation. For the purpose of fine time-domain synchronization, we use a window-based method, to iteratively find the best match of the head and tail of the signal after various delay adjustments. Assume the time domain signal is  $x(t)$ , and the length of cyclic prefix is  $T_g$ , we have

$$\underset{t_f}{\operatorname{argmin}} \sum_{t=t_c+t_f}^{t_c+t_f+T_g} x(t)x(t+T_s), \quad \forall t_f \in [-\tau, \tau] \quad (3.8)$$

where  $T_s$  is length of symbol excluding the guard interval,  $t_c$  is the coarse delay, and  $\tau$  is the searching range for  $t_f$  of a finer synchronization.

### 3.3.2.6 Channel Estimation and Equalization

Acoustic channel requires channel estimation and equalization techniques to overcome the distortions caused by fast fading, delay spreading, and multipath propagation. We insert equal-spaced unit-powered pilot tones for the purpose of equalization. To get the channel estimation, we extract pilot tones in frequency domain after proper synchronization as  $z(k)$  where  $k \in \mathbb{P}$ , the pilot sub-channel set. Since it is equal-spaced in the frequency domain, we then apply a FFT-based interpolation with a proper interpolation length to expand it to estimate the data channel frequency response  $H(k)$ ,  $k \in \mathbb{P} \cup \mathbb{D}$ , where  $\mathbb{D}$  is the data sub-channel set. And  $H(k) = z(k)$  when  $k \in \mathbb{P}$ . Then, the equalization on the

pilot and data channel is calculated as follows:

$$\hat{s}(k) = \frac{z(k)}{H(k)}, \quad k \in \mathbb{P} \cup \mathbb{D} \quad (3.9)$$

By equalizing the known *a-priori* pilot sub-channels to unit-power, we equalize the data channels at the same time.

### 3.3.2.7 Adaptive Modulation

WearLock supports modulations such as BASK/QASK, BPSK/QPSK, 8PSK and 16QAM. We adopt an adaptive modulation scheme, which has a Request-to-Send/Clear-to-Send (RTS/CTS) phase before the data transmission phase. The motivation of adaptive modulation is that in every round, we want to make sure that the acoustic signal can be delivered reliably from smartphone to the nearby smartwatch in spite of the ambient noise and interfering signals. As is well known, the higher the order of modulation, the higher the data rate  $R$ .  $R$  can be calculated by

$$R = \frac{|D|r_c \log_2 M}{T_g + T_s} \quad (3.10)$$

where  $M$  is the modulation order,  $|D|$  is size of data sub-channel set,  $r_c$  is the coding rate for channel coding, and  $r_c = 1$  if no channel coding is used. Higher order modulations are more vulnerable to ambient noise and interference. This usually requires a higher SNR to maintain the same error rate as a lower order modulation. Therefore, dynamically adaptive modulation are adopted by many communication systems, in which they sense the channel quality and adapt the modulation under certain constraints. Unlike traditional adaptive modulation for communication systems which seeks to maximize the system data rate, our design goal is to utilize the propagation loss in transmission to select a modulation mode to maintain a BER under a target BER. In the RTS/CTS phase, WearLock sends out a preamble with a block-based pilot symbol as a channel probing packet, which will serve the purpose of sub-channel selection and modulation selection.

**Channel probing and sub-channel selection:** It is important for WearLock to find the long-term or short-term noise which lasts for at least the time of transmission, like periodically-restarting air conditioner, which overlays certain frequencies for undefined duration. By sending a channel probing packet, WearLock can get an estimate of the channel state information and rank all the candidate sub-channels by the noise power. WearLock also chooses sub-channels in a priority order from low frequency to high frequency, and from low noise power to high noise power. We will assess the performance of sub-channel selection in our evaluation.

**Pilot-based SNR indicator:** From the channel probing result, we can also estimate the pilot signal SNR as an indicator for adaptive modulation. In order to measure and compare the performance of different modulation schemes, we use a normalized signal-to-noise ratio (SNR) as metric:  $E_b/N_0$ , which is the ratio of the energy per bit to noise power spectral density. It can be calculated as

$$\frac{E_b}{N_0} = \frac{C}{N} \cdot \frac{B}{R} \propto \text{PSNR} \cdot \frac{B}{R} \quad (3.11)$$

where  $B$  is the bandwidth, and  $R$  is the data rate, as we have discussed previously.

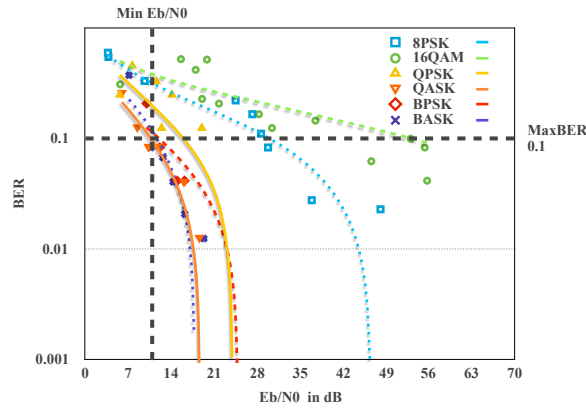
The  $\frac{C}{N}$  is the carrier to noise power ratio, which will be estimated using a pilot-based SNR [95], that can be calculated from the spectrum result:

$$\text{PSNR} = \frac{\mathbb{E}_{k \in \mathbb{P}} [X(k) \cdot X^*(k)] - \mathbb{E}_{k \in \mathbb{N}} [X(k) \cdot X^*(k)]}{\mathbb{E}_{k \in \mathbb{N}} [X(k) \cdot X^*(k)]} \quad (3.12)$$

where  $\mathbb{N}$  is the null sub-channel set.

**Deciding transmission mode:** We have measured how BERs of different modulations change in terms of different  $\frac{E_b}{N_0}$  in a quiet room (15-20db SPL) and LOS. We control the ambient noise by an external speaker playing white noise audio. The result is shown in Figure 3.6, in which the scatter plots are fitted by logarithmic trend-lines. The ranking order of our measures closely matches the theoretic result [85]. Due to hardware limita-

tions, 16QAM is not usable in real experiments or at least needs heavy error correction techniques. Also due to the uneven responses of amplitude modulation and phase modulation of the audio hardware, amplitude-shift keying needs less SNR per bit than phase-shift keying. Therefore, we setup three transmission modes in total: QASK, QPSK, and 8PSK.



**Figure 3.6:** The BER of different modulations changes with  $E_b/N_0$

**Ambient noise measurement:** The ambient noise is measured in the first processing phase at both sides. The smartphone also conducts a self-recording while the smartwatch is actively recording the incoming signals. By detecting the preamble existing in those recordings, we can coarsely align the two time series. The time series before the preamble are used to calculate the ambient noise. The ambient noise similarity is used to filter the cases that those devices are apparently not co-located. The noise level is also used to set proper speaker volume to control the transmission range.

**How adaptive modulation works:** According to our preliminary measurements in Fig. 3.4, in the first phase, a probing packet is sent out using a SPL(volume) that surpasses the SPL of noise at least a minimal SNR around 1 meters:

$$\text{SPL}_{\text{tx}} - 20 \log_{10}\left(\frac{1.0}{d_0}\right) - \text{SPL}_{\text{noise}} > \text{SNR}_{\min} \quad (3.13)$$

where  $\text{SNR}_{\min}$  can be decided from a minimal  $E_b/N_0$ , such as marked in Fig. 3.6. This



ensures that the receiver in the range receives this probing packet. WearLock has no explicit ranging and we use this as the bound on the transmission range, if a receiver falls within this range, it will be able to receive the signal which is beyond the minimal SNR. The actual received SNR is estimated by the pilot-based SNR and will be reported in the CTS signal. After the transmitter gets the  $\text{SNR}_{\text{rx}}$ , this one is used to select the modulation scheme that can reach a BER at least smaller than a decided bound, the MaxBER as we have also marked in Fig. 3.6. For example, if the rx's SNR converts to  $E_b/N_0 = 35\text{dB}$  and  $\text{MaxBER} = 0.1$ , we can send the signal using 8PSK, since we can get a guaranteed BER. If  $\text{MaxBER} = 0.01$ , then we can choose the modulation scheme as either QPSK or QASK.

### 3.4 Secure Unlocking

Existing work uses SIC to secure information transmitted in the acoustic channel. However, in our scenario, it is not feasible since most android wearable devices are not shipped with speakers. Therefore, we employed one time password (OTP) scheme to make use the acoustic channel with no secret disclosed.

#### 3.4.1 Threat Model

We assume that the wireless link is securely established, and used as a secure control channel for the OFDM communication. The acoustic channel is assumed to be insecure and an attacker can eavesdrop. We also assume that the attacker cannot take possession of the smart watch since it is hard to steal the watch from user's wrist without being caught. An attacker may take control of the phone and try to peak into it for the purpose of online payment, private photos and emails, etc. In order to fool the WearLock system, we assume that an attacker may try to perform various attacks. One is the co-located attack, in which the attacker holds the user's phone to get as close to the target as possible without being discovered. Another one is a record-and-replay attack, in which the

attacker makes use of recording and replay devices to capture the acoustic signal and replay it to the smartphone. Jamming or Denial-of-Service attacks are not considered, since we can simply turn back to traditional locking scheme on smartphones. Currently, our design cannot protect acoustic channel against sophisticated relay attack which relies on some sort of relay to extend the range of between those two devices. However, we will argue the difficulty of launching this attack in acoustic channel, then discuss potential counter-measures.

### **3.4.2 One Time Password**

To defend against replay attacks, we employ a counter based one time password scheme (i.e., IETF RFC 4226 [60]). Assume that the phone and watch have negotiated a secret key  $k$  and a counter  $c$  through the wireless control channel (e.g., Bluetooth), which can also be updated at anytime. The one time password is generate by keyed hash message authentication code (HMAC) using  $\text{HMAC-SHA-1}$ , as  $\text{HMAC}(k, c)$ . Then a dynamic truncation (DT) technique is used to extract a 32 bit binary from the 160-bit result, which ensures that the outputs on different counter inputs are uniformly distributed. The final digits are generated by the DT result taking modulo  $10^{\text{Digit}}$ , where Digit is the number of digits.

### **3.4.3 Security Discussion.**

As we have mentioned, an attacker possessing the victim's phone, will try various attacks. We have identified the following attacks and explained why our system can defend against or at least mitigate these attacks.

#### **3.4.3.1 Brutal Force Attack**

An attacker who takes possession of the victim's phone, will try to mount brutal force attack when the victim wearing the smartwatch is in another room or quite far away while the Bluetooth connection is still live. The attacker needs to properly guess the acoustic

modem parameters and guess the OTP. A 32 bits OTP has a large keyspace as  $2^{32}$  and we can easily increase the keyspace by adding more data channels or using higher order modulations. The smartphone will be locked up after three consecutive failures, which makes the brutal force attack unrealistic.

#### **3.4.3.2 Co-located Attack**

Being similar to brutal force attack, the attack just tries to get close enough to the victim to perform a successful unlock. The defense against this attack lies in the design of the modem that there is high bit error rate when the transmission distance is beyond around 1 meter. Getting closer to the user and covering the smartphone stealthily may not work, since it will obstruct the direct path and result in significant loss when acoustic channel becomes NLOS.

#### **3.4.3.3 Record and Replay Attack**

Since an attacker can monitor the acoustic channels, disclosing the OTP token may suffer from a replay attack, in which an attacker can record the token signal and replay it to the watch like in the man-in-the-middle (MITM) attack. This attack is defeated by examining the timing window, since in the protocol, we can measure the software stack delay and wireless round-trip-time. A MITM attacker with recorder and player in the loop definitely adds delay in the acoustic path. Every time the power button is pressed, a Bluetooth message is sent to the watch indicating the start of the protocol, and the watch replies a Bluetooth message and starts recording. Then the smartphone starts to send acoustic token, after which smartphone also sends a Bluetooth message of stopping recording. And the watch will stop recording as well. This procedure has two phases, and it is interactive, which means we can examine the result of the first phase, and abort the second phase if there is anything suspicious. Because the OTP token is sent in the second phase, we avoid the disclosure of OTP token in such attack.

#### **3.4.3.4 Relay Attack**

Sophisticated relay attack will try to use record and replay in a live manner, to circumvent the time window based protection. If this attack can be performed in ideal case, our current design cannot protect acoustic channel against this attack. However, this attack is very hard to mount since it needs very flat frequency and phase response speaker and microphone to avoid acoustic distortions in the ADC and DAC. Otherwise, we can use fingerprinting method to unique identify those acoustic hardwares to check if there are relays. Additionally, high quality speaker/microphone usually cannot be made in small sizes, which enlarges the chance being spotted by victim. Another potential counter-measure is to employ distance bounding protocol [11].

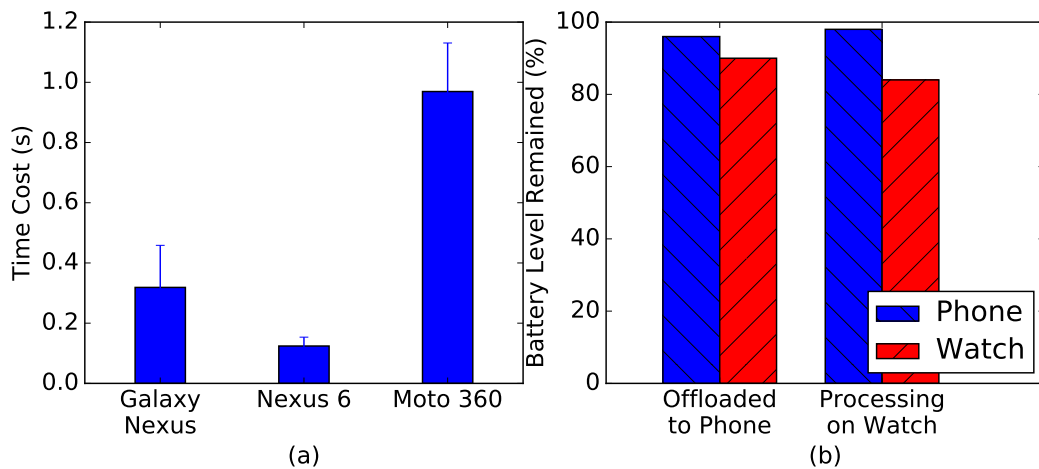
### **3.5 Performance Optimizations**

WearLock Controllers are the running instances of our system on the smartphone and smartwatch. One task of WearLock Controller is to gather information from various sources and make the final decisions on questions such as where to run the computation and when to abort a transmission, which gives us plenty of opportunities for performance optimizations. The rationale is that the change of the way of unlocking smartphone using a paired smartwatch does not actually reduce the frequency of unlocking. Every audio transmission is followed by a series of computations, which would be heavy burdens on wearable devices. Even though the microphone and speaker power consumptions are relatively low, digit signal processing computations such as cross correlation, FFT based Modulation and Demodulation, FFT based interpolation are all relatively computationally intensive, consuming more power. We believe that by well addressing those questions, we can not only save energy for wearable devices but also reduce the delay of processing. We conduct computation load balance and computation reduction as two main solutions.

### 3.5.1 Computation Offloading

To mitigate the power drain on wearables, we leverage the natural computation pattern of the smartphone and its paired wearable, offloading heavy computation tasks from the smartwatch to the smartphone. Since all the acoustic modem and digital signal processing libraries are implemented as a common module shared by both phone-side and watch-side apps, we can easily partition the computations among these two devices.

In order to understand the trade-off here, we have measured the time cost of processing after the recording and the corresponding rough power consumption, in Figure 3.7. The processing mainly consists of a sliding window based cross correlator and an OFDM demodulator. Since it is not possible to tear apart the Android smartwatch and connect it to a power meter, we run our system for 50 rounds of acoustic unlocking and rely on the Android OS battery status to roughly measure the power consumption by the API provided by Android Framework. To be noted that, this energy consumption measure is pretty rough, as the measurement procedure keeps the device awake, violating the life-cycle design pattern of an Android wear app. We anticipate more energy saving in daily usage. From the result, we can see that by offloading to the smartphone, it not only saves energy but also reduces the computation time.



**Figure 3.7:** Time Cost (a) and Power Consumption (b) Comparison on Offloading and Local Processing on Wearable.

### 3.5.2 Computation Reduction

The basic idea of the computation reduction is to leverage a series of filters using information such as wireless network, ambient noise and motion sensors, to avoid unnecessary follow-up heavy computation. For example, we can set a rule that the WearLock only works when the Bluetooth link exists. Therefore, if there is no Bluetooth link, all the protocols and algorithms will not run. Alternatively, the technique used in Sound-Proof [47] is complementary to WearLock by leveraging the similarity of ambient noise, to eliminate unnecessary acoustic transmission, which can be implemented in the RTS/CTS phase of adaptive modulation. If the ambient noise similarity is below a threshold, we believe those two devices are not co-located with a high confidence and then the transmission is aborted. Additionally, we can also leverage the activity context information or hand movement derived from sensor units to reduce the number of acoustic transmissions.

#### 3.5.2.1 Leveraging Motion Sensor-based Filtering

When the user is engaged in activities, or the smartphone is held by the same hand that wears the watch, we can use the raw inertial sensor data to detect the device movement similarity. This will serve as a filter that can eliminate unnecessary acoustic transmission if the similarity distance is lower or higher than predefined thresholds. In order to use sensor traces, we need to convert the 3-axis sensors to its magnitude representation by  $s \leftarrow \sqrt{s_x^2 + s_y^2 + s_z^2}$ , since it is a challenge to obtain accurate relative orientation between those two devices. The alignment of the sensor time series is not necessary since we use Dynamic Time Warping (DTW) to find the best alignment in the time domain [55]. The procedure is presented in Alg. 3.

Even though the time complexity of DTW is  $O(n^2)$  assuming two inputs are both in length of  $n$ , it is very cheap since  $n$  is usually small ranging from 50 to 150 samples. We will verify the feasibility and measure the time cost in the evaluation.

---

**Algorithm 3** Sensor-based Filter

---

```
1: procedure Sensor-based Filtering
2:   for each first phase do
3:     while recording do
4:        $sp_{x,y,z} \leftarrow$  phone accelerometer
5:        $sw_{x,y,z} \leftarrow$  watch accelerometer
6:     end while
7:      $sp \leftarrow \text{Normalized}(\text{Magnitude}(sp_{x,y,z}))$ 
8:      $sw \leftarrow \text{Normalized}(\text{Magnitude}(sw_{x,y,z}))$ 
9:     if  $\text{DTW}(sp, sw) > d_h$  then
10:      abort protocol
11:    else if  $\text{DTW}(sp, sw) < d_l$  then
12:      skip second phase
13:    else
14:      continue to the second phase
15:    end if
16:  end for
17: end procedure
```

---

▷ save the computation

▷ save the computation

## 3.6 Evaluation

In this section, we will first briefly discuss the implementation details. Then, we will evaluate our system in terms of communication range, adaptive modulation, sensor-based filtering, system delay, a field test and a case study. The metrics we are reporting are mostly bit error rate (BER) and time cost.

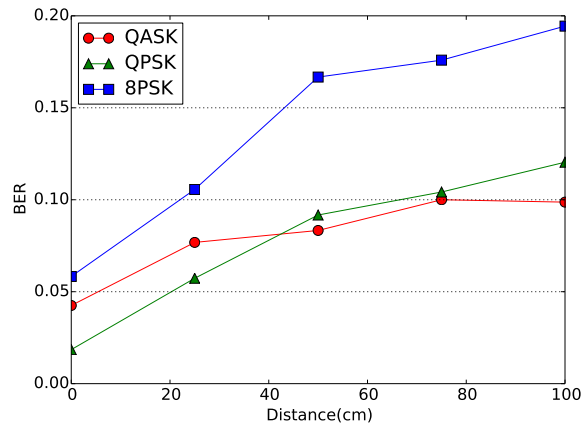
### 3.6.1 Implementation Details

We have implemented our system on Android OS, consisting of an Android phone app and an Android wear app. We have wrapped the MessageAPI and ChannelAPI of Android Wear SDK for implicit message/file transferring so that we do not need to handle the underlying networking using either Bluetooth or WiFi. We have also ported the wear app to a smartphone in order to test near-ultrasound frequency in WearLock. The OFDM modem is written in pure JAVA libraries, which can be running on both sides. The digital signal processing library is also written in JAVA and we plan to move on native DSP library in the future. The default FFT size is 256 and the sampling rate is 44.1 kHz, which gives about 172Hz sub-channel bandwidth. We index our channels from 1-256. and in

default we pick channel  $\{16, 17, 18, 20, 21, 22, 24, 25, 26, 28, 29, 30\}$  as data channels, and  $\{7, 11, 15, 19, 23, 27, 31, 35\}$  as pilot channels for working at 1-6kHz frequency band. The rests are null channels. We shift this channel assignment with higher index when we want 15-20kHz frequency band. This channel assignments will be adjusted during sub-channel selection. The preamble size is 256 samples, the post-preamble guard size is 1024 samples and the CP duration is 128 samples. All those parameters can be easily tuned in the setting activity of our app.

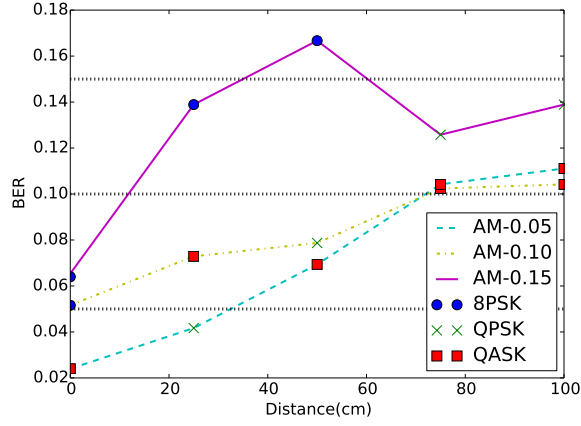
### 3.6.2 Communication Range

The communication range is a very important performance metric. Ideally, we want to the communication range to be strictly constrained within one meter. However, the performance varies due to different modulations and ambient noise. In Figure 3.8, we show the communication range of the acoustic modem in terms of BER in three different transmission modes. They are measured at an office room with a line-of-sight setup. We can see that by constraining the max BER we can adaptively change the transmission mode to guarantee that the signal fades significantly when the current communication range is increased.



**Figure 3.8:** The BER in distances and transmission modes (near-ultrasound).

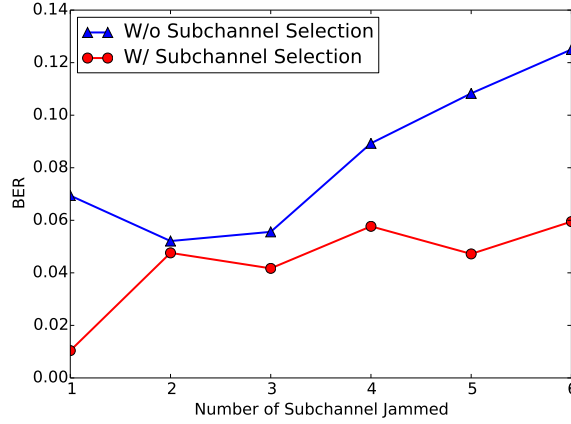




**Figure 3.9:** The BER in adaptive modulation under different BER constraints(near-ultrasound).

### 3.6.3 Adaptive Modulation

To understand the performance of adaptive modulation, we have conducted two experiments. First, we enable adaptive modulation selection in the previous measurements to show the effectiveness of adaptive modulation. In Figure 3.9, by constraining the BER, we can adaptively change the modulation schemes, which can allow us to have shorter packets or more redundant bits. It also guarantees that an eavesdropper located nearby will have a larger BER since a higher order modulation is more vulnerable to noise and interference. Next, we demonstrate WearLock adaptation to ambient noise in sub-channel selections. We use audible frequency range for this experiment and employ an external tone generator as an acoustic jammer, the Audacity, which supports at most 6 mono-tracks simultaneously. We use QPSK modulation with the smartwatch and smartphone placed at a fixed distance about 15cm. The jammed sub-channel indexes are randomly selected every time. The result, depicted in Figure 3.10, shows that when the sub-channel selection is enabled, the modem is able to avoid the noisy or interfered sub-channels and maintain a stable BER. Since we have a wide bandwidth, in which there are 24 data sub-channel candidates and 32 sub-channels in total. For a 6-digit pin code (24 bit in binary) using QPSK, it needs 12 sub-channels. A jammer or noise source needs to block at least more than half of those sub-channels.



**Figure 3.10:** The BER under jamming and subchannel selection. (QPSK, audible sound)

### 3.6.4 Sensor-based Filtering

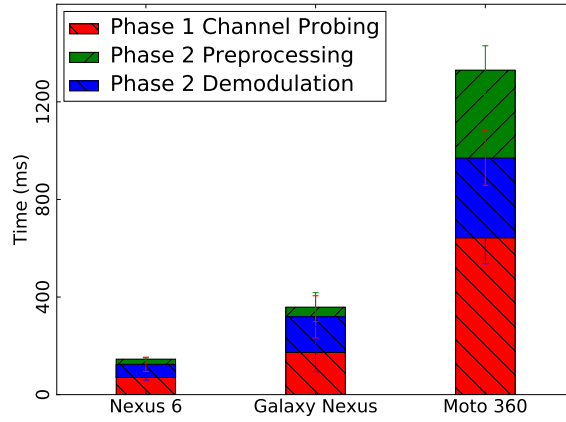
We have also evaluated the sensor-based filtering to see how much similarity in sensor data we can leverage to reduce the number of acoustic transmissions. We tested the smartphone and the smartwatch in same activities such as sitting, walking and jogging, and also in different activity combinations. The normalized DTW scores and the running time are reported in Table 3.1. The activity context can be queried through Google Play Service APIs. By setting a threshold on the DTW scores (0.1 in our case), we can reduce the Max BER or skip the second phase when the DTW score is under the threshold.

| Activities | sitting | walking | running | different | cost(ms) |
|------------|---------|---------|---------|-----------|----------|
| DTW Scores | 0.05    | 0.02    | 0.06    | 0.20      | 45.9     |

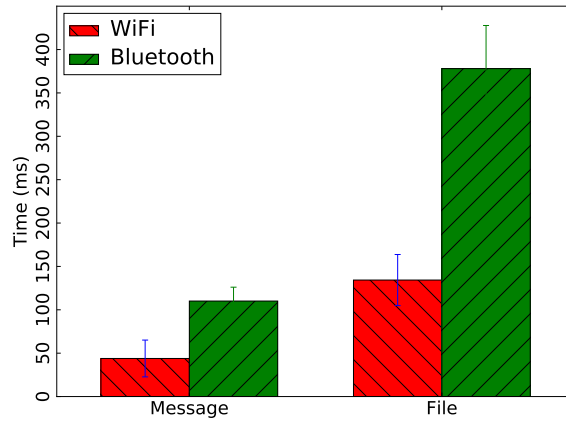
**Table 3.1:** Sensor-based Filtering

### 3.6.5 System Delay

The system delay is important since users will lose their patience with the WearLock technique if it is much slower than entering a password. There are two types of delay: computation delay and communication delay. We have broken down the computation delay into *phase 1 channel probing processing*, *phase 2 pre-processing* and *phase 2 demodulation* in Figure 3.11 when the computation is carried out on different devices. We



**Figure 3.11:** The computation delay of each phase on different devices.



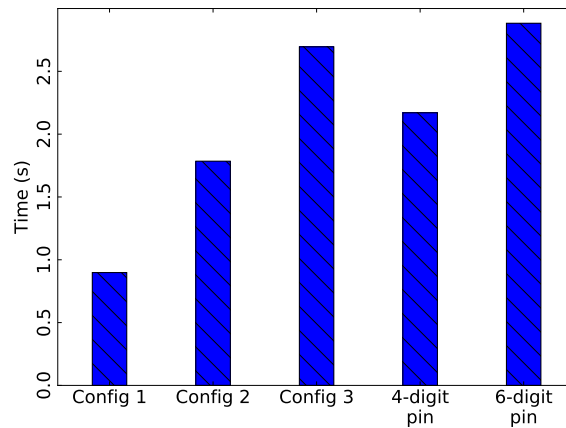
**Figure 3.12:** The communication delay between smartphone and smartwatch.

have also measured the communication delay in WiFi/Bluetooth message and file transfer in Figure 3.12. Every experiment is repeated at least 20 times. We did not measure the modulation since the generation is very fast. Part of them can be generated ahead-of-time and therefore the cost can be amortized. For purpose of comparison, we also measured the time cost for a user entering 4/6-digit PIN codes on an Android device using similar method as [37]. The results are also aligned to the medians of measurements in [37]. We compare the results with three different configurations as shown in Figure 3.13:

- *Config1*: the fastest case where the smartwatch offloads computation via WiFi to a high end smartphone (Nexus 6)

- *Config2*: the slowest case where the smartwatch offloads computation via Bluetooth to a low end smartphone (Galaxy Nexus)
- *Config3*: local processing case where the processing is on the smartwatch (Moto 360)

The results indicate that WearLock has a delay advantage over manually unlocking even on a low end device and slow Bluetooth link with a speedup of at least 17.7%. For the fastest case, the WearLock speedup is at least 58.6%. Notably, WearLock experiences less delay and only needs the user to click the power button.



**Figure 3.13:** Compare the total delay in different configurations with manually entering pin codes.

### 3.6.6 Field Test

We tested WearLock with the smartphone and smartwatch hold or worn in different configurations: same hand and different hands. We also tested them in different locations as offices, classrooms, cafes and grocery stores where the typical sounds in those scenarios are human voice and noises from sources such as keyboard typing, cafe machines, air conditioners, etc. We report the BER results in Table 3.2. From the results, we find that near-ultrasound may have less interference but significant signal fade due to direct path blocking in the same hand case. The audible sound is less convenient but more usable in

most noise cases. It would be better to use inaudible sound in quiet spaces and audible sound in noisy spaces as long as the volume is controlled. We can easily integrate this choice to current mobile OS since it is in line with how smartphone users set their the sound notification preferences.

| BER v.s. Locations           | Office       | Class Room   | Cafe         | Grocery Store |
|------------------------------|--------------|--------------|--------------|---------------|
| Diff. Hand (Audible)         | 0.0486(8PSK) | 0.0333(8PSK) | 0.0263(QPSK) | 0.0119(QPSK)  |
| Same Hand (Audible)          | 0.0889(8PSK) | 0.0512(8PSK) | 0.0655(QPSK) | 0.0648(QPSK)  |
| Diff. Hand (Near-ultrasound) | 0.0556(8PSK) | 0.0417(QPSK) | 0.0233(QPSK) | 0.0139(QPSK)  |
| Same Hand (Near-ultrasound)  | 0.1054(QPSK) | 0.1875(QPSK) | 0.1971(QPSK) | 0.2060(QPSK)  |

**Table 3.2:** Field Test Result. The average BER is around 0.08.

## 3.7 Discussion and Limitations

### 3.7.1 Non-omnidirectional Microphone/Speaker

Even though the microphone/speaker units are omni-directional, they are not exposed fully to the outside of smart device cases, which all results in certain direction-limited effect, just like uni-directional units. During our experiments, the real performance of the acoustic modem is heavily affected by whether the speaker-mic pair are directed to each other (i.e., line of sight). This effect increases the BER and shorts the working range. However, the imposed limitation of working range will not hinder the security and can be mitigated by relaxing the acceptable BER threshold. We can also leverage the diverse specifications of smartphone and smartwatch in which multiple microphones and speakers are built-in at various positions for the purpose of stereo play/recording and noise cancellation.

### 3.7.2 Acoustic Frequency Range

Due to the frequency range limitation of the mobile acoustic hardware, the implemented system can work on audio range (1-6Khz) on a phone-watch pair, and near-ultra sound range (15-20Khz) in a phone-phone pair. This brings the limitation that the acoustic is

either audible or can be possibly heard by babies or animals. This is one limitation of our work and we leave this to the smartphone manufacture when devices with higher sampling rate will be shipped. For example, several latest models of Samsung Galaxy Note supports 96kHz and higher audio recording/playback. Device with higher sampling rate can utilize higher and more frequency bands with less noise and more bandwidth.

### **3.7.3 Bluetooth Proximity**

According to the document of Bluetooth proximity profile that even the link between devices has been securely enabled, the device can be spoofed into assuming that the other device is close due to the internal design of Bluetooth protocol, which means that naively using Bluetooth proximity profile for secure distance measurement is not encouraged [92]. Currently secure distance measurement using Bluetooth requires additional development upon existing stacks. Comparatively, our system on mobile and wearable devices can be easily implemented in the application level and ported to other devices. However, we do admit that a solution via Bluetooth is promising, and we leave this in our future work to explore secure ranged and easy-to-implement token-based authentication in wireless channel.

## **3.8 Related Work**

There are two main areas related to our work. First, we will briefly outline the acoustic communication on mobile device and justify the difference of our work. Then, we will discuss the existing work about reduced-effort authentication.

### **3.8.1 Acoustic Communication on Mobile Devices**

WearLock extends acoustic communications to more smart devices. Dhvani [61] aims to replace NFC with an acoustic orthogonal frequency division multiplexing (OFDM) modem

secured by a self-interference cancellation (SIC) technique. Dolphin [51] and PriWhisper [105] also leverage similar idea for secure acoustic channel. However, their schemes are not suitable for practical and efficient implementation on phone-watch pairs, since most smartwatches have no speakers and generating a cancellation signal imposes both energy and processing burdens on wearable devices. We use a different secure scheme tailored for smartwatch which acts as a listener in acoustic channel and conduct offloading to shift computation and energy burdens on smartwatch to more capable smartphone. Work [50] used On-off keying on chirp signals to overcome one of the main limitations of acoustic communication on mobile devices: the short communication range. However, our work make a good use of the relatively short communication range, and we use OFDM which yields much higher data rate. Google NearBy [29] is an Android Framework API to provide near field communication and interaction using Bluetooth, WiFi and acoustics. The acoustic signal is modulated in Dual-tone multi-frequency signaling (DTMF), which is slower and less spectrum-efficient compared to the OFDM. However, the NearBy API requires the devices to support near-ultrasound in 18.5kHz-20kHz and therefore is not supported on Android Wear devices yet. Other work requires the provision of special acoustic communication hardware [56, 77, 90]. WearLock requires no additional special hardware.

### **3.8.2 Reduced-Effort Authentication**

The reduced-effort authentication is about techniques that seek to reduce or eliminate the human effort involved in the process of authentication. The simplest schemes utilize short-range radio communication using Bluetooth or NFC. ZIA [18] is one of the earliest work with zero-interaction authentication, leveraging an authentication token. WearLock can be taken as a natural extension from PC and electronic tokens to the nowadays common smartphone-smartwatch pairs. Work [75] has proposed the combination of multiple signals to define a security confidence level and subsequent the authentication only at certain

levels. Their scheme can reduce the authentication frequency but requires large effort in data collection and training. Similarly, work [53] has proposed a method to lock the device when the user's physical separation is detected. Their method is complementary to ours and can be combined. Another way of reducing effort in authentication is to leverage device co-location or localization [34, 47, 88, 107]. Sound-Proof [47] has proposed to leverage similarities in ambient noise signals for user authentication. Sound-of-silent [88] has proposed to utilize the silence patterns in recordings to provide co-location context. However, these techniques cannot defend against co-located attackers due to the co-location granularity issues. WearLock relies on the presence of a validated acoustic signal that is designed not to be detectable more than one meter away from the generating device.

### **3.9 Chapter Summary**

In this chapter, we show that a convenient and secure smartphone unlocking can be achieved by leveraging a paired smartwatch. We argue that the smartwatch is an ideal wearable token device that is theft-proof and has constant connections to the phone. Smartphone users can save much effort from unlocking. WearLock, the implemented system, secures the acoustic channel by adapting the transmission power and modulation configurations, and sends an OTP tokens for validation via acoustics to unlock the smartphone. To optimize the system performance, we offload the heavy computation to the phone, and leverage multi-source information including sensor data to reduce unnecessary audio transmissions. WearLock can achieve an average bit error rate of 8% in our experiments. WearLock achieves at least 18% speedup even on a low-end device, compared to entering PINs.



## Chapter 4

# LAVEA: Latency-aware Video Analytics on Edge Computing Platform

### 4.1 Introduction

Edge computing (also termed fog computing [9], cloudlets [78], MEC [66], etc.) has brought us better opportunities to achieve the ultimate goal of a world with pervasive computation [78]. This new computing paradigm is proposed to overcome the inherent problems of cloud computing and provide supports to the emerging Internet of Things (IoT) [36, 84, 99]. When using the cloud, all the data generated shall be uploaded to the cloud data center before processing. However, considering nowadays a huge amount of data is being intensively generated at the edge of the network, transferring the data at such scale to the distant cloud for processing will add burdens to the network and lead to unacceptable response time, especially for latency-sensitive applications. More specifically, as for edge computing, we aim to provide *edge analytics*, which focuses on data analytics at or near the places (the network edge) where data is generated [80]. Data analytics done at the edge of the network has many benefits such as gathering more client

side information, cutting short the response time, saving network bandwidth, lowering the peak workload to the cloud, and so on.

Among many edge analytic applications, in this project, we focus on delivering video analytics at the edge. The ability to provide low latency video analytics is critical for applications in the fields of public safety, counter-terrorism, self-driving cars, VR/AR, etc [83]. In video edge analytic applications, we consider typical client devices such as mobile phones, body-worn cameras or dash cameras mounted on vehicles, web cameras at toll stations or highway checkpoints, security cameras in public places, or even video captured by UAVs [91]. For example, in “Amber Alert”, our system can automate and speedup the searching of objects of interest by vehicle recognition, vehicle license plate recognition and face recognition utilizing various web cameras deployed at highway entrances, or dash cameras or cameras of smartphones mounted on cars.

Simply uploading all the captured video or redirecting video streams to the cloud cannot meet the requirement of latency-sensitive applications, because the computer vision algorithms involved in object tracking, object detection, object recognition, face and optical character recognition (OCR) are either computation intensive or bandwidth hungry. In addressing these problems, mobile cloud computing (MCC) is proposed to run heavy tasks on resource rich cloud node to improve the response time or energy cost. This technique utilizes both the mobile and cloud for computation. An appropriate partition of tasks that makes trade-off between local and remote execution can speed up the computation and preserve mobile energy at the same time [20, 32, 38, 62, 82]. However, there are still concerns of cloud about the limited bandwidth, the unpredictable latency, and the abrupt service outage. Existing work has explored adding intermediate servers (cloudlets) between mobile client and the cloud. Cloudlet is an early implementation of the cloud-like edge computing platform with virtual machine (VM) techniques. The edge computing platform in our work has a different design on top of lightweight OS-level virtualization which is modular – easy to deploy, manage, and scale. Compared to VM, the OS-level virtualization provides resource isolation in a much lower cost. The adoption of container

technique leads to a server-less platform where the end user can deploy and enable edge computing platform on heterogeneous devices with minimal efforts. The user programs (scripts or executable binaries) will be encapsulated in containers, which provide resource isolation, self-contained packaging, anywhere deploy, and easy-to-configure clustering. The end user only needs to register events of interest and provide corresponding handler functions to our system, which automatically handles the events behind the scene.

In this chapter, we present the **Latency-Aware Video Edge Analytics (LAVEA)** system [98]. We are considering a 3-tier mobile-edge-cloud deployment and we put most of our efforts into the mobile-edge side and inter-edge side design. We divide the response time minimization problem into three sub-problems. First, we select client tasks that benefit from being offloaded to edge node in term of time cost. We formulate this problem as a mathematical optimization problem to choose tasks for offloading and allocate bandwidth among clients. Unlike existing work in mobile cloud computing, we cannot make the assumption that edge node is as powerful as cloud node with unlimited resources. Therefore, we consider the increasing resource contention and response time when more and more tasks are running on the edge node by adding a latency constraint to the optimization problem. Second, upon receiving offloading task requests at each epoch, the edge node runs these tasks in an order to minimize the makespan. However, the offloaded tasks cannot start when the corresponding inputs are not ready. To address this problem, we employ a classic two-stage job shop model and adapt the Johnson's rule [44] with topological ordering constraint in a heuristic to prioritize the tasks. Last, we enable inter-edge collaboration which leverages nearby edge nodes to reduce the overall task completion time. We have investigated the performance of several task placement schemes for inter-edge collaboration. The findings provide us insights that lead to an efficient prediction-based task placement scheme.

In summary, we make the following contributions:

- We have designed an edge computing platform with serverless architecture, which

is able to provide flexible computation offloading to nearby clients to speed up computation-intensive and delay-sensitive applications. Our implementation is lightweight-virtualized, event-based, modular, and easy to deploy and manage on either edge or cloud nodes.

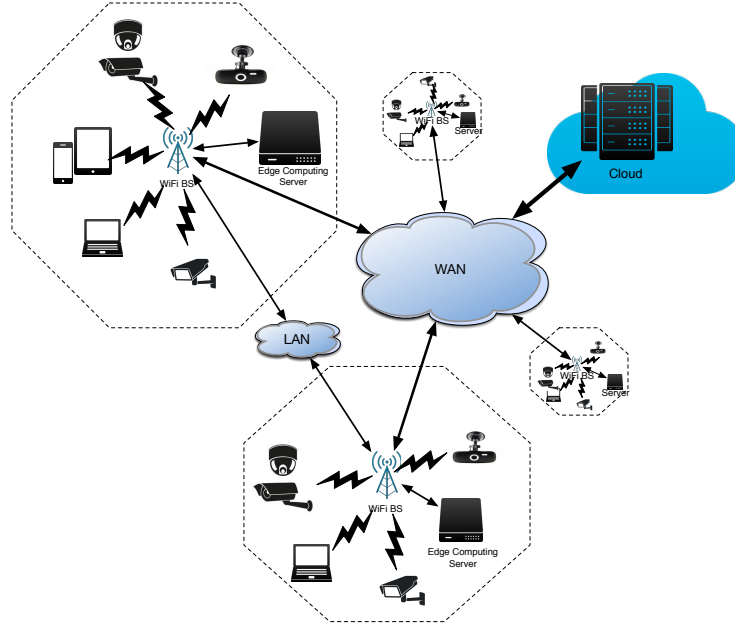
- We have formulated an optimization problem for offloading task selection and prioritized offloading requests to minimize the response time. The task selection problem co-optimizes the offloading decision and bandwidth allocation , and is constrained by the latency requirement, which can be tuned to adapt to the workload on edge node for offloading. The task prioritizing is modeled as a two-stage job shop problem and a heuristic is proposed with the topological ordering constraint.
- We have evaluated several task placement schemes for inter-edge collaboration and proposed a predication-based method which efficiently estimates the response time.

## **4.2 Background and Motivation**

In this section, we briefly introduce the background of edge computing and relevant techniques, present our observations from preliminary measurements, and discuss the scenarios that motivate us.

### **4.2.1 Edge Computing Network**

In this work, we consider an edge computing network as shown in Figure 4.1, in which we focus on two types of nodes, the client node (we call it client for short) and the edge server node (we call it edge, edge node, or edge server for short). We assume that clients are one-hop away from edge server via wired or wireless links. When a client connects to the edge node, it implicitly indicates that the client will first connect to the correspond access points (APs) using cable or wireless link and then utilize the services provided

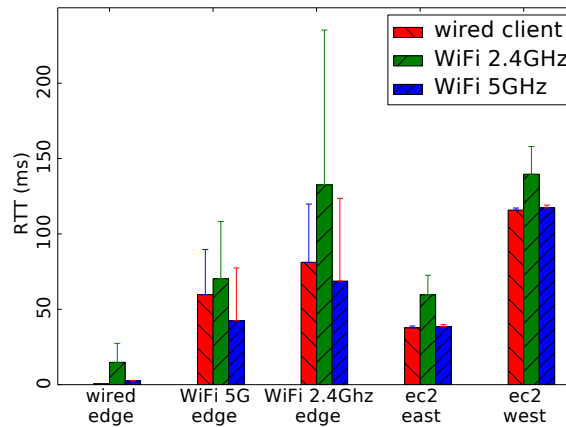


**Figure 4.1:** An overview of edge computing environment

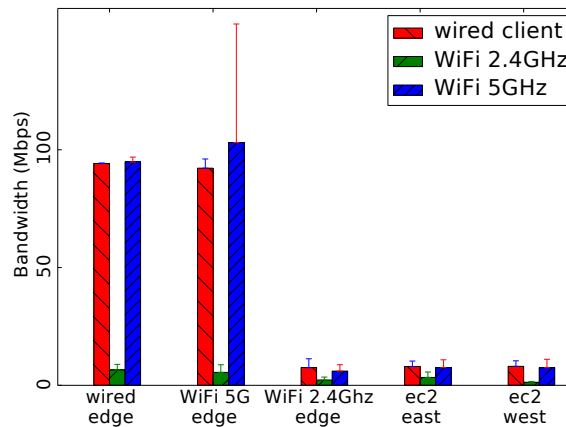
by connecting to the co-located edge node. In a sparse edge node deployment, a client will only connect to one of the available edge nodes nearby at certain location. While in a dense deployment, a client may have multiple choices of selecting the multiple edge servers for service access. Implicitly, we assume that there is a remote cloud node which can be reached via the wide area network (WAN).

To understand the factors that impact the feasibility of realizing practical edge computing systems, we have performed several preliminary measurements in different network setups and shown the results in Figure 4.2 and Figure 4.3. In these experiments, we measure the latency and bandwidth of combinations between clients nodes with different network interfaces connecting to edge (or cloud) nodes. According to the measurements of bandwidth, all clients have benefits in utilizing a wire-connected or advanced-wireless (802.11ac 5 GHz) edge computing node. In terms of latency, wire-connected edge nodes is the best while the 5 GHz wireless edge computing nodes have larger means and variances in latency compared to the cloud node in the closest region due to the intrinsic nature of wireless channels. Based on the observations, in this work, we pragmatically

assume that edge nodes are wired to APs via cables to deliver services with better latency and bandwidth against the cloud. Therefore, in such a setup, the cloud node can be considered as a backup computing node, which will be utilized only when the edge node is saturated and experiences very long response time.



**Figure 4.2:** Round trip time between client and edge/cloud.



**Figure 4.3:** Bandwidth between client and edge/cloud.

#### 4.2.2 Serverless Architecture

Serverless architecture or Function-as-a-Service (FaaS), such as AWS Lambda, Google Cloud Functions, Azure Functions, is an agile solution for developer to build cloud computing services without the heavy lifting of managing cloud instances. To use AWS Lambda

as an example, AWS Lambda is an event-based, micro-service framework, in which a user-supplied Lambda function as the application logic will be executed in response to the event of interest. The AWS cloud will take care of the provisioning and resource management for running Lambda functions. At the first time a Lambda function is created, a container will be built and launched based on the configurations provided. Each container will also be provided a small disk space as transient cache during multiple invocations. AWS has its own way to run Lambda functions with either reusing an existing container or creating a new one. Recently, there is AWS Lambda@Edge [4], that allows using serverless functions at the AWS edge location to apply moderate computations in response to content distribution network (CDN) events. We strongly advocate the adoption of serverless architecture at the edge computing layers, as serverless architecture naturally solves two very important problems for edge computing: 1) serverless programming model greatly reduces the burden on users or developers in developing, deploying and managing edge applications, as there is no need to understand the complex underlying procedures or distributed system management to run the applications; 2) the function is a very good abstract that is flexible to run on either edge or cloud, which lowers the barrier of edge-cloud inter-operability and federation. Recent works have shown the potentials of such architecture in low latency video processing tasks [24] and distributed computing tasks [45], and there have been research efforts of incorporating serverless architecture in edge computing [21].

#### **4.2.3 Video Edge Analytics for Public Safety**

Video surveillance is of great importance for public safety. Besides the “Amber Alert” example, there are many other applications in this field. For example, secure cameras deployed at public places (e.g. the airport) can quickly spot unattended bags [109], police with body-worn cameras can identify suspects and suspicious vehicles during approaching, and so on. Because those scenarios are urgent and critical, the applications need to

provide the quickest responses with best efforts. However, most tasks in video analytics are undoubtedly computationally intensive [71]. If such application is running solely on resource constrained mobile clients or IoT devices directly, the latency in computation, battery drain (if battery-powered), or even heat dissipation will eventually ruin the user experience, failing to achieve the application performance goals. If deployed on cloud nodes, transferring large volume of multimedia data will incur unacceptable transmission latency and additional bandwidth cost. Being proposed as a dedicated solution, the edge computing platform enables the quickest responses to these video analytics tasks which require both low latency and high bandwidth.

In this work, we mainly focus on building the video edge analytics platform and we demonstrate our platform with the application of Automated License Plate Recognition (ALPR). Even though we integrate specific application, our edge platform is a general design and can be extended for other applications with minor modifications. An ALPR system usually has four stages: 1) image acquisition, 2) license plate extraction, 3) license plate analysis, and 4) character recognition [5, 23]. Each stage involves various computer vision, pattern recognition, and machine learning algorithms. Migrating the execution of some algorithms to powerful edge/cloud node can significantly reduce the response time [86]. However, offloaded tasks require intermediate data, application state variables, and corresponding configurations to be uploaded. For example, some computational tasks that produce a large amount of intermediate data will add delay to the whole processing time if offloaded to the remote cloud. In general, we believe that an edge computing platform that is carefully designed to handle the computation offloading will assist ALPR system to expand on more resource-constrained devices at more locations and provide better response time at the same time.



## 4.3 LAVEA System Design

In this section, we present our system design. First, we will discuss our design goals. Then, we will overview our system design and introduce several important edge computing services.

### 4.3.1 Design Goals

- **Latency.** The ability to provide low latency services is recognized as one of the essential requirements of edge computing system design.
- **Flexibility.** Edge computing system should be able to flexibly utilize the hierarchical resources from client nodes, nearby edge nodes and remote cloud nodes.
- **Edge-first.** By edge-first, we mean that the edge computing platform is the first choice of our computation offloading target.

### 4.3.2 System Overview

LAVEA is intrinsically an edge computing platform, which supports low-latency video processing. The main components are edge computing node and edge client. Whenever a client is running tasks and the nearby edge computing node is available, a task can be decided to run either locally on the client or remotely on the edge server. We present the architecture of our edge computing platform in Figure 4.4.

#### 4.3.2.1 Edge Computing Node

In LAVEA, the edge computing node provides edge computing services to nearby mobile devices. The edge computing node attached to the same access point or base station as clients is called the *edge-front*. By deploying edge computing node with access point or base station, we ensure that edge computing service can be as ubiquitous as Internet access. Multiple edge computing nodes can collaborate and the edge-front will always

serve as the master and be in charge of the coordination with other edge nodes and cloud nodes. As shown in Figure 4.4, we use the light-weight virtualization technique to provide resource allocation and isolation for different clients. Any client can submit tasks to the platform via client APIs. The platform will be responsible for shaping workload, managing queue priorities, and scheduling tasks. Those functions are implemented via internal APIs provided by multiple micro-services such as queueing service, scheduling service, data store service, etc. We will introduce several important services later in this section.

#### **4.3.2.2 Edge Client**

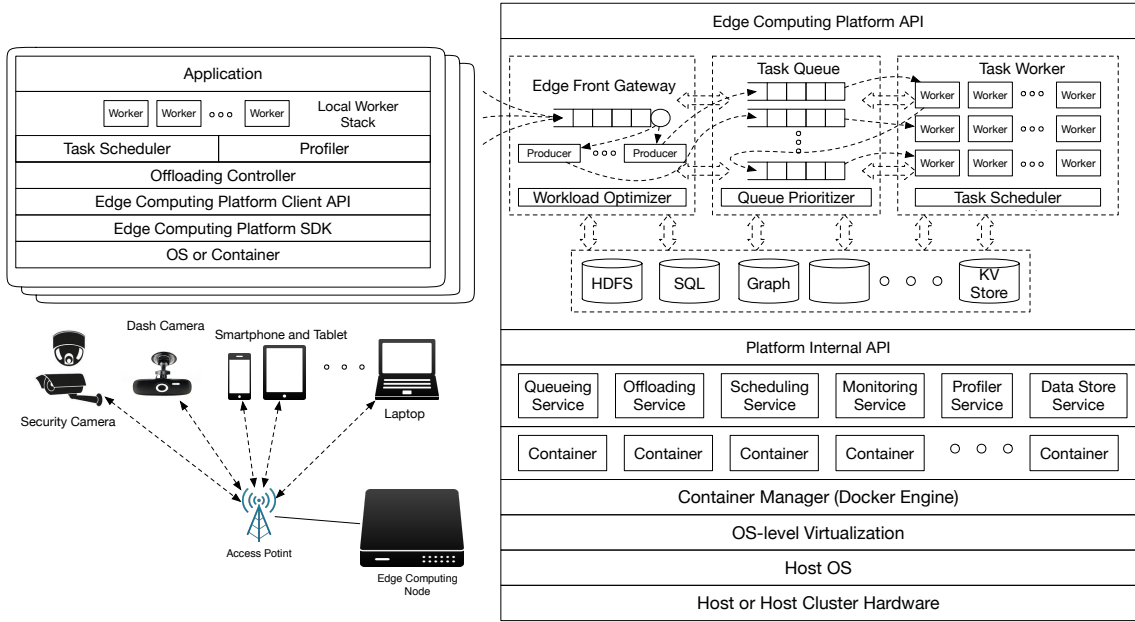
As a resource constrained device, an edge client prefers to run lightweight data processing tasks locally and offload heavy tasks to the edge computing node nearby. In LAVEA, the edge client has a thin client design, to make sure all the clients can run it without introducing too much overhead. According to the available computation resource, the device spawns at least one worker to make progress on the assigned job. The most important components of client node design are the profiler and the offloading controller, acting as participants in the corresponding profiler service and offloading service. With profiler and offloading controller, a client can provide offloading information to the edge-front node and fulfill offloading decisions received.

### **4.3.3 Edge Computing Services**

Here, we will briefly introduce some important edge computing services.

#### **4.3.3.1 Profiler Service**

Similar to work [20, 62, 82], our system uses a profiler to collect metrics of task performance on various devices, since it is difficult to derive an analytic model to accurately capture the behavior of the whole system. However, we have found that the execution of video process tasks on certain device is relatively stable, when the input and algorithmic



**Figure 4.4:** The architecture of edge computing platform

configurations are fixed. Therefore, we add a profiling phase during the deployment on every new type of client devices and edge devices. The profiler will execute instrumented tasks multiple times with different inputs and configurations on the device and measure metrics including but not limited to the execution time, input/output data size, etc. The time-stamped logs will be gathered to build the task execution graph for specific tasks, inputs, configurations, and devices. The profiler service will collect those information, on which LAVEA relies for offloading decisions.

#### 4.3.3.2 Monitoring Service

Unlike profiler service which gathers pre-run-time execution information with pre-defined inputs and configurations, the monitoring service is used to continuously monitor and collect run-time information such as the network, system load, etc., from not only the clients but also nearby edge nodes. Monitoring the network between client and edge-front is necessary since most edge clients connect to edge-front server via wireless links. The condition of wireless link is changing from time to time. Therefore, we need to constantly

monitor the wireless link, to estimate the bandwidth and the latency. Monitoring the system load on the edge clients and edge servers provides flexible workload shaping and task offloading from the client to the edge. This information is also broadcasted among nearby edge nodes. When an edge-front node is saturated or unstable, some tasks will be assigned to nearby edge nodes, according to the system load, the network bandwidth, and network delay between edge nodes, as long as there is still benefit compared to assigning tasks to cloud node.

#### **4.3.3.3 Offloading Service**

The offloading controller tracks tasks running locally at the client, and exchanges information with the offloading service running on the edge-front server. The data gathered in profiler and monitoring services will be used as inputs to the offloading decision problem which is formulated as an optimization problem to minimize the response time. Every time when a new client registers itself to the offloading services, after the edge-front node collects enough prerequisite information and statistics, the optimization problem is solved again and the updated offloading decisions will be sent to all the clients. Periodically, the offloading service also solves the optimization problem, and updates offloading decisions with its clients.

### **4.4 Edge-front Offloading**

In this section, we consider selecting tasks to run on the edge as a computation offloading problem. Traditional offloading problems are about offloading schemes between clients and remote powerful cloud servers. In literature [20, 62, 82], these system models usually assume the task will be instantly finished remotely once the task is offloaded to the server. However, we argue that this assumption will not hold in edge computing environment as we need to consider the various delays at the server side especially when lots of clients

are sending offloading requests. We call it *edge-front computation offloading* from the perspective of client:

- Tasks will be only offloaded from client to the nearest edge node, which we call the edge front.
- The underlying scheduling and processing is agnostic to clients.
- When a mobile node is disconnected from any edge node or even cloud node, it will resort to local execution of all the tasks.

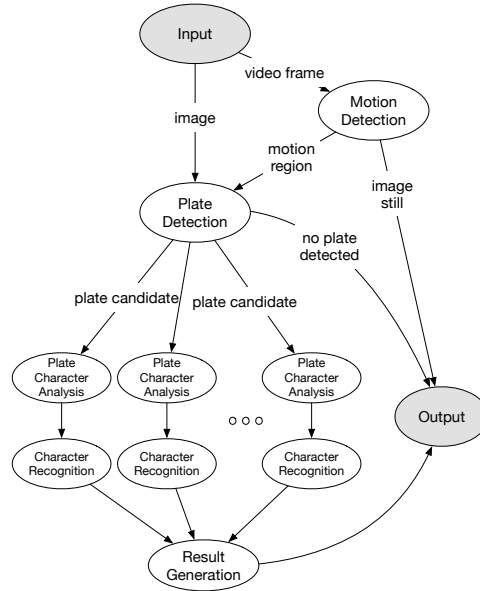
We assume that edge node is wire-connected to the access point, which indicates that the out-going traffic can go through edge node with no additional communication cost. The only difference between offloading task to edge node and cloud node, is that the task running on edge node may experience resource contention and scheduling delay while we assume task offloaded to cloud node will get enough resource and be scheduled to run immediately. In light work load case, if there is any response time reduction when this task is offloaded to cloud, then we know that there is definitely benefit when this task is offloaded to the edge. The reasons are 1) an edge server is as responsive as the server in the cloud data center, 2) running a task on edge server experiences shorter data transmission delay as client-edge link has much larger bandwidth than edge-cloud link which is usually limited and imbalanced by the Internet service providers (ISPs). Therefore, in this section, we focus on the task offloading only between client and edge server, and we will discuss integrating nearby edge nodes for the heavy work load scenario in the next section.

#### **4.4.1 Task Offloading System Model and Problem Formulation**

Throughout the chapter, we call a running instance of the application a job, which consists a set of tasks. The job is the unit of work that user submits to our system while the task is the unit of work for our system to make scheduling and optimization decisions. These

tasks from each application will be queued and processed either locally or remotely. By remotely, we mean run the task on an edge node. In our edge application scenario, all clients are running instances of applications processing same kind of jobs. However, our system can be easily extended to support heterogeneous applications.

In our ALPR application, each task is usually a computer vision algorithm. For example, We have analyzed an open source ALPR project called OpenALPR [64] and illustrate its task graph in Figure 4.5. We choose to work on the granularity of task since these tasks are modularized and can be flexibly pipelined with tuned parameters to make trade-off between quickness and accuracy.



**Figure 4.5:** The task graph of OpenALPR.

In the model, we consider there are  $N$  clients and only one edge server connected as shown in Figure 4.1. The edge server could be a single server or a cluster of servers. Each client  $i, i \in [1, N]$ , will process the upcoming job upon request (e.g., recognizing the license plates in video streams). We expect that such job consists of heavy computation tasks could benefit from offloading some tasks to the edge server. Without loss of generality, we use a graph of task to represent the complex task dependencies inside a job,

which is essentially similar to the method call graph in [20], but in a more coarse granularity. For a certain kind of job, we start with its directed acyclic graph (DAG),  $G = (V, E)$ , which gives the task execution sequence. Each vertex  $v \in V$  weight is the computation or memory cost of a task ( $c_v$ ), while each edge  $e = (u, v), u, v \in V, e \in E$  weight represents the data size of intermediate results ( $d_{uv}$ ). Thus, our offloading problem can be taken as a graph partition problem, in which we need to assign a directed graph of tasks to different computing nodes (local, edge, or cloud), with the purpose to minimize certain cost, in our problem, which is the job finish time.

The remote response time includes the communication delay, the network transmission delay of sending data to the edge server, and the task execution time on that server. We use an indicator  $I_{v,i} \in \{0, 1\}$  for all  $v$  in  $V$  and for all  $i \in [1, N]$ . If  $I_{v,i} = 1$ , the task  $v$  at client  $i$  will run locally, otherwise, it will run on the remote edge server. For those tasks running locally, the total execution time for client  $i$  is a summation:

$$T_i^{local} = \sum_{v \in V} I_{v,i} c_v / p_i \quad (4.1)$$

where  $p_i$  is the processor speed of client  $i$ .

Similarly, we use

$$\bar{T}_i^{local} = \sum_{v \in V} (1 - I_{v,i}) c_v / p_i \quad (4.2)$$

to represent the execution time of the path not taken which is running the offloaded tasks locally instead. In the network, when there is an offloading decision, the client need to send the intermediate data (outputs of previous task, application status, configurations, etc) to the edge server in order to continue the computing. The network delay is modeled as

$$T_i^{net} = \sum_{(u,v) \in E} |I_{u,i} - I_{v,i}| d_{uv} / r_i + \beta_i \quad (4.3)$$

where  $r_i$  is the connection rate assigned for this client connecting to the edge server and  $\beta_i$  is the communication latency which can be estimated using round trip time between

the client  $i$  and the edge server.

For each client, the remote execution time is

$$T_i^{remote} = \sum_{v \in V} (1 - I_{v,i})(c_v/p_0) \quad (4.4)$$

where  $p_0$  is the processor speed of the edge server.

Then our offloading task selection problem can be formulated as

$$\min_{\mathbf{I}, r_i} \sum_{i=1}^N (T_i^{local} + T_i^{net} + T_i^{remote}) \quad (4.5)$$

The offloading task selection is represented by the indicator matrix  $\mathbf{I}$ . This optimization problem is subject to the following constraints:

- The total bandwidth

$$\text{s.t.} \quad \sum_{i=1}^N r_i \leq R \quad (4.6)$$

- Like existing work, we restrict the data flow to avoid ping-pong effect in which intermediate data is transmitted back and forth between client and edge server.

$$\text{s.t.} \quad I_{v,i} \leq I_{u,i}, \quad \forall e(u, v) \in E, \forall i \in [1, N] \quad (4.7)$$

- Unlike existing offloading frameworks for mobile cloud computing, we take the resource contention or scheduling delay at the edge side into consideration by adding an end-to-end delay constraint.

$$\text{s.t.} \quad \bar{T}_i^{local} - (T_i^{net} + T_i^{remote}) > \tau, \quad \forall i \in [1, N] \quad (4.8)$$

where  $\tau$  can be tuned to avoid selecting borderline tasks that if offloaded will get no gain due to the resource contention or scheduling delay at the edge.



**Optimization Solver.** The proposed optimization is a mixed integer non-linear programming problem (MINLP), where the integer variable stands for the offloading decision and the continuous variable stands for the connection rate. To solve this optimization problem, we start from relaxing the integer constraints and solve the non-linear programming version of the problem using Sequential Quadratic Programming method, a constrained nonlinear optimization method. This solution is optimal without considering the integer constraints. Starting from this optimal solution, we optionally employ branch and bound (B&B) method to search for the optimal integer solution or simply do an exhaustive search when the number of clients and the number of tasks of each job are small.

#### 4.4.2 Prioritizing Edge Task Queue

The offloading strategy produced by the task selection optimizes the “flow” time of each type of job. At each time epoch of running, the edge-front node receives a large number of offloaded tasks from the clients. Originally, we follow the first come first serve rule to accommodate all the client requests. For each request at the head of the task queue, the edge-front server first checks whether the input or intermediate data (e.g. images or videos) is ready; otherwise the server waits. This scheme is easy to implement but substantial computation is wasted if the network IO is busy with a large size file and there is no task that is ready for processing. Therefore, we improve the task scheduling with a task queue prioritizer to maintain a task sequence which minimizes the makespan for the task scheduling of all offloading task requests received at a certain time epoch. Since the edge node can execute the task only when the input data has been fully received or the depended tasks have finished execution, we consider that an offloaded task has to go through two stages: the first stage is the retrieval of input or intermediate data and state variables; the second stage is the execution of the task.

We study our scheduling problem using the flow job shop model and apply the Johnson's rule [44]. This scheme is optimal and the makespan is minimized, when the number

of stages is two. Nevertheless, this model only fits in the case that all submitted job requests are independent and have no priorities. When considering task dependencies, a successor can only start after its predecessor finishes. By enforcing the topological ordering constraints, the problem can be solved optimally using the B&B method [12]. However, this solution hardly scales against the number of tasks. In this case, we adapt the method in [6], i.e., grouping tasks with dependencies and executing all tasks in a group sequentially. The basic idea is applying Johnson's rule in two levels. The first level is to decide the sequence of tasks within each group. The difference in our problem is that we need to decide the best sequence among all valid topological orderings. The bottom level is a job shop scheduling problem in terms of grouped jobs (i.e., a group of tasks with dependencies in topological ordering), in which we can utilize Johnson's rule directly.

## **4.5 Inter-edge Collaboration**

In this section, we improve our edge-first design in the case when the incoming workload saturates our edge-front node. We will first discuss our motivation and list the corresponding challenges. Then we will introduce several collaboration schemes we have proposed and investigated.

### **4.5.1 Motivation and Challenges**

The Edge computing node possesses more resources than client nodes but less than the cloud node. While serving an increasing number of client nodes nearby, the edge-front node will be eventually overloaded and become non-responsive to new requests. As a baseline, we can optionally choose to offload further requests to the remote cloud. We assume that the remote cloud has unlimited resources and is capable of handling all the requests. However, running tasks remotely in the cloud, the application need to bear with unpredictable latency and limited bandwidth, which is not the best choice especially when there are other nearby edge nodes that can accommodate those tasks. We as-

sume that under the condition when all available edge nodes nearby are exhausted, the mobile-edge-cloud computing paradigm will simply fall back to the mobile cloud computing paradigm. The fallback design is a future work. In this chapter, we mainly investigate the inter-edge collaboration with the primary purpose to alleviate the burden on edge-front node.

When the edge-front node is saturated with requests, it can collaborate with nearby edge nodes by placing some tasks to these not-so-busy edge nodes, such that all the tasks can get scheduled in a reasonable time. This is slightly different from balancing the workload among the edge nodes and the edge-front node, in that the goal of inter-edge collaboration is to better serve the client nodes with submitted requests, rather than simply making the workload balanced. For example, an edge-front node that is not overloaded does not need to place any tasks to the nearby edge nodes, even when they are idle.

The challenges of inter-edge collaboration are two-fold: 1) we need to design a proper inter-edge task placement scheme that fulfills our goal of reducing the workload on the edge-front node while offloading a proper amount of workload to the qualified edge nodes; 2) the task placement scheme should be lightweight, scalable, and easy-to-implement.

#### **4.5.2 Inter-Edge Task Placement Schemes**

We have investigated three task placement schemes for inter-edge collaboration.

- Shortest Transmission Time First (STTF)
- Shortest Queue Length First (SQLF)
- Shortest Scheduling Latency First (SSLF)

The STTF task placement scheme tends to place tasks on the edge node that has the shortest estimated latency for the edge-front node to transfer the tasks. The edge-front node maintains a table to record the latency of transmitting data to each available edge

node. The periodical re-calibration is necessary because the network condition between the edge-front node and other edge nodes may vary from time to time.

The SQLF task placement scheme, on the other hand, tends to transfer tasks from the edge-front node to the edge node which has the least number of tasks queued upon the time of query. When the edge-front node is saturated with requests, it will first query all the volunteer edge nodes about their current task queue length, and then transfer tasks to the edge node that has the shortest queue reported.

The SSLF task placement scheme tends to transmit tasks from the edge-front node to the edge node that is predicted to have the shortest response time. The response time is the time interval between the time when the edge-front node submits a task to an available edge node and the time when it receives the result of the task from that edge node. Unlike the SQLF task placement scheme, in which the edge-front node keeps querying all the edge nodes and may have a performance issue when the number of nodes is very large, we have designed a novel method for the edge-front node to measure the scheduling latency efficiently. During the measurement phase before the edge-front node chooses the task placement target, edge-front node sends a request message to each available edge node, which appends a special task to the tail of the task queue. When the special task is executed, the edge node simply sends a response message to the edge-front node. The edge-front node receives the response message and records the response time. Periodically, the edge-front node maintains a series of response times for each available edge node. When the edge-front node is saturated, it will start to reassign tasks to the edge node having the shortest response time. Unlike the STTF and SQLF task assignment schemes, which choose the target edge node based on the current or most recent measurements, the SSLF scheme predicts the current response time for each edge node by applying regression analysis to the response time series recorded so far. The reason is that the edge nodes are also receiving task requests from client nodes, and their local workload may vary from time to time, so the most recent response time cannot serve as a good predictor of the current response time for the edge nodes. As the local

workload in the real world on each edge node usually follows certain pattern or trend, applying regression analysis to the recorded response times is a good way to estimate the current response time. To this end, we record the measurements of response times from each edge node, and offload tasks to the edge node that is predicted to have the least current response time. Once the edge-front node starts to place tasks to a certain edge node, the estimation will be updated using piggybacking of the redirected tasks, which amortizes the overhead of measurement.

Each of the task placement schemes described above has some advantages and disadvantages. For instance, the STTF scheme can quickly reduce the workload on the edge-front node. But there is a chance that tasks may be placed to an edge node which already has intensive workload, as STTF scheme gathers no information of the workload on the target. The SQLF scheme works well when the network latency and bandwidth are stable among all the available edge nodes. When the network overheads are highly variant, this scheme fails to factor the network condition and always chooses edge node with the lowest workload. When an intensive workload is placed under a high network overhead, this scheme potentially deteriorates the performance as it needs to measure the workload frequently. The SSLF task placement scheme estimates the response time of each edge node by following the task-offloading process, and the response time is a good indicator of which edge node should be chosen as the target of task placement in terms of the workload and network overhead. The SSLF scheme is a well trade-off between previous two schemes. However, the regression analysis may introduce a large error to the predicted response time if inappropriate models are selected. We believe that the decision of which task placement scheme should be employed for achieving good system performance should always give proper considerations on the workload and network conditions. We evaluated those three schemes through a case study in the next section.

## **4.6 System Implementation and Performance Evaluation**

In this section, we first brief the implementation details of our system. Next, we introduce our evaluation setup and present evaluation results.

### **4.6.1 Implementation Details**

Our implementation aims at a serverless edge computing architecture. As shown in the system architecture of Figure 4.4, our implementation is based on docker container for the benefits of quick deployment and easy management. Every component has been docker-ized and its deployment is greatly simplified via distributing pre-built images. The creation and destruction of docker instances is much faster than that of VM instances. Inspired by the IBM OpenWhisk [42], each worker container contains an action proxy, which uses Python to run any scripts or compile and execute any binary executable. The worker container communicates with others using a message queue, as all the inputs/outputs will be jsonified. However, we don't jsonified image/video and use its path reference in a shared storage. The task queue is implemented using Redis as it is in memory with good performance. The end user only needs to 1) deploy our edge computing platform on heterogeneous devices with just a click, 2) define the event of interests using provided APIs, and 3) provide a function (scripts or binary executable) to process such event. The function we have implemented utilizes the open source project OpenALPR [64] as the task payload for workers.

### **4.6.2 Evaluation Setup**

#### **4.6.2.1 Testbed**

We have built a testbed consisting of four edge computing nodes. One of the edge nodes is the edge-front node, which is directly connected to a wireless router using a cable. Other three nodes are set as nearby edge computing nodes for the evaluation of inter-

edge collaboration. These four machines have the same hardware specifications. They all have a quad-core CPU and 4 GB main memory. The three nearby edge nodes are directly connected to the edge-front node through a network cable. We make use of two types of Raspberry Pi (RPi) nodes as clients: one type is RPi 2 which is wired to the router while the other type is RPi 3 which is connected to router using built-in 2.4 GHz WiFi.

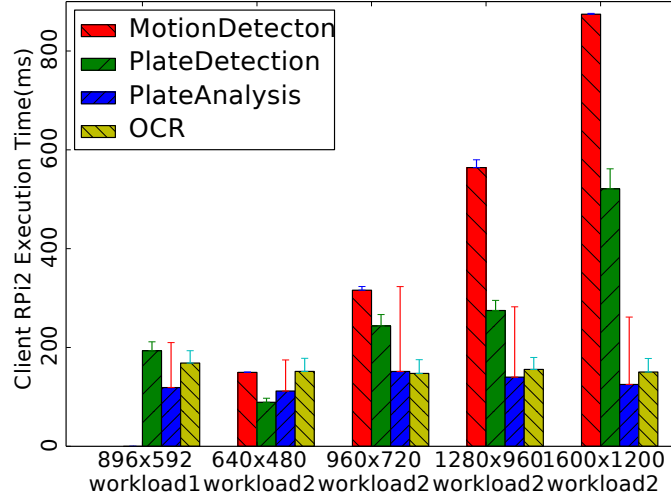
#### **4.6.2.2 Datasets**

We have employed three datasets for evaluation. One dataset is the Caltech Vision Group 2001 testing database, in which the car rear image resolution (126 images with resolution 896x592) is adequate for license plate recognition [69]. Another dataset is a self-collected 4K video containing rear license plates taken on an Android smartphone and is converted into videos of different resolutions (640x480, 960x720, 1280x960, and 1600x1200). The other dataset used in inter-edge collaboration evaluation contains 22 car images, with the various resolution ranging from 405x540 pixels to 2514x1210 pixels (file size 316 KB to 2.85 MB). The task requests use the car images as input in a round-robin way, one car image for each task request.

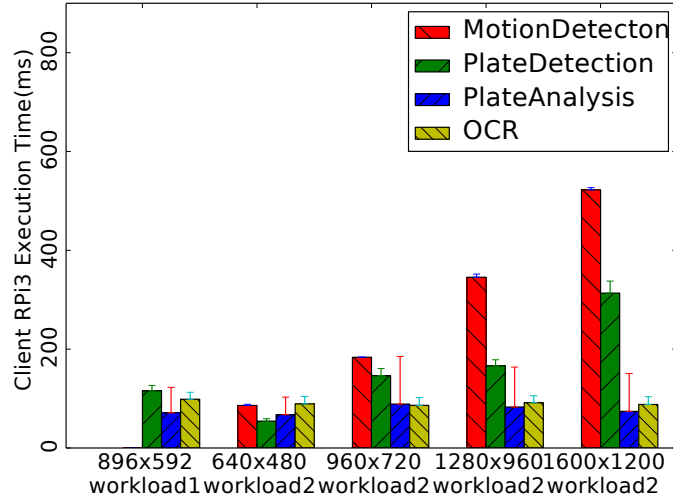
#### **4.6.3 Task Profiler**

Beside the round trip time and bandwidth benchmark we have presented in Figure 4.2 and Figure 4.3 to characterize the edge computing network, we have done the profiling of the OpenALPR application on various client, edge and cloud nodes.

In this experiment, we use both dataset 1 (workload 1) and dataset 2 (workload 2) at various resolutions. The execution time of each task is shown in Figure 4.6, Figure 4.7, Figure 4.8, and Figure 4.9. The results indicate that by utilizing an edge node, we can get a comparable amount of computation power close to clients for computation-intensive tasks. Another observations is that, due to the uneven optimization on heterogeneous CPU architectures, some tasks are better to keep local while some others should be



**Figure 4.6:** OpenALPR profile result of client type 1 (RPi2 quad-core 0.9 GHz)



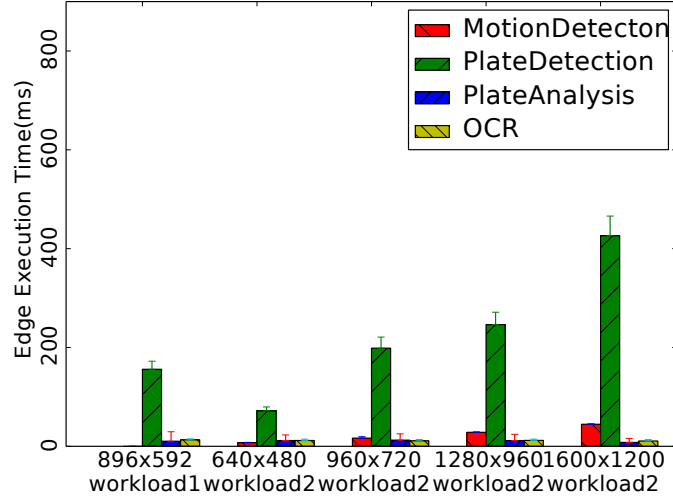
**Figure 4.7:** OpenALPR profile result of client type 2 (RPi3 quad-core 1.2 GHz)

offloaded to edge computing nodes. This observation justifies the need of computation offloading between clients and edge nodes.

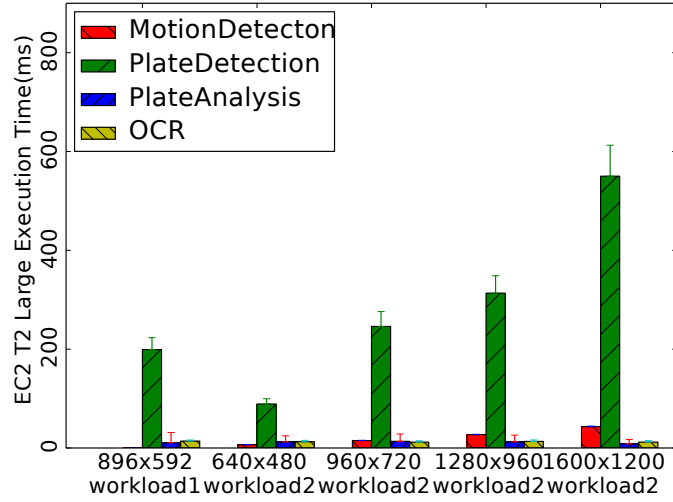
#### 4.6.4 Offloading Task Selection

To understand how much execution time can be reduced by splitting tasks between the client and the edge, or between the client and the cloud, we design an experiment with



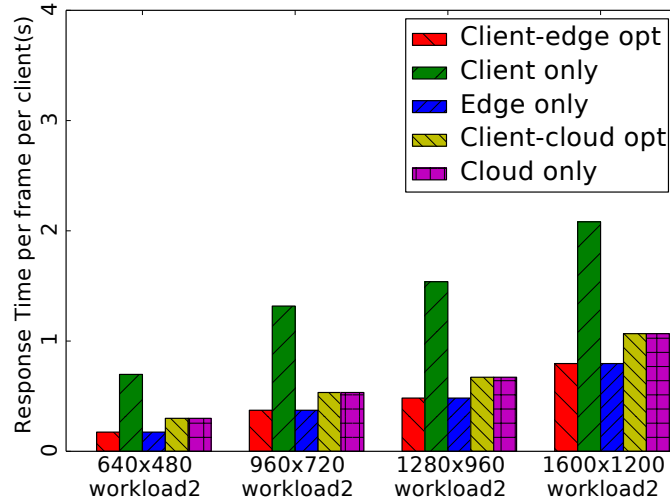


**Figure 4.8:** OpenALPR profile result of a type of edge node (i7 quad-core 2.30 GHz)

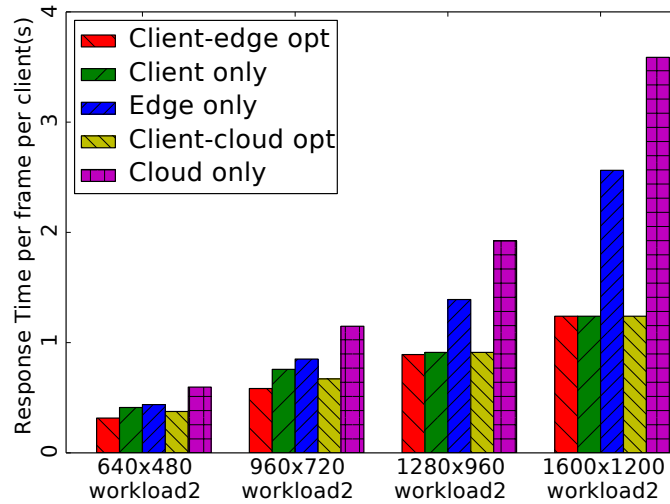


**Figure 4.9:** OpenALPR profile of a type of cloud node (AWS EC2 t2.large Xeon dual-core 2.40 GHz)

workloads generated from dataset 2 on two setups: 1) one edge node provides service to three wired client nodes that have the best network latency and bandwidth; 2) one edge node provides service to three wireless 2.4 GHz client nodes that have latency with high variance and relatively low bandwidth. The result of the first case is very straightforward: the clients simply upload all the input data and run all the tasks on the edge node in edge offloading or cloud node in cloud offloading, as shown in Figure 4.10. This is mainly



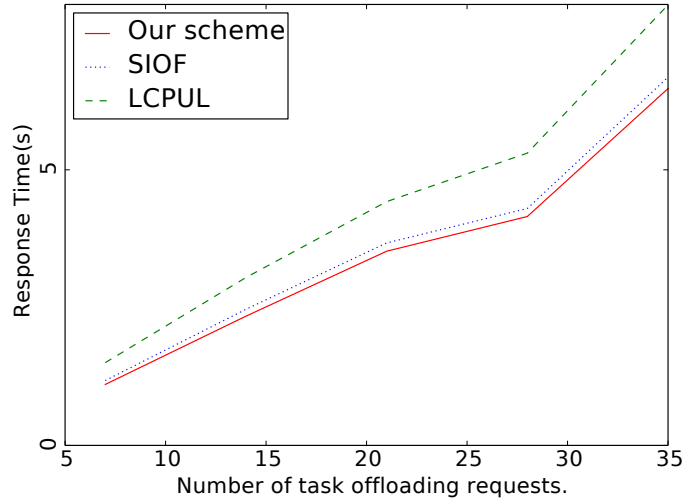
**Figure 4.10:** The comparison of task selection impacts on edge offloading and cloud offloading for wired clients (RPI2).



**Figure 4.11:** The comparison of task selection impacts on edge offloading and cloud offloading for 2.4 GHz wireless clients (RPI3).

because using Ethernet cable can stably provide lowest latency and highest bandwidth, which makes offloading to edge very rewarding. We didn't evaluate 5 GHz wireless client since this interface is not supported on our client hardware while we anticipate similar results as the wired case. We plot the result of a 2.4 GHz wireless client node with offloading to an edge node or a remote cloud node in the second case in Figure 4.11. Overall, the

results showed that by offloading tasks to an edge computing platform, the application we had chosen experienced a speedup up to 4.0x on wired client-edge configuration compared to local execution, and up to 1.7x compared to a similar client-cloud configuration. For clients with 2.4 GHz wireless interface, the speedup is up to 1.3x on client-edge configuration compared to local execution, and is up to 1.2x compared to similar client-cloud configuration.



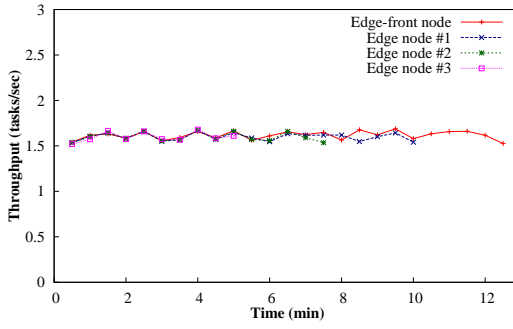
**Figure 4.12:** The comparison result of three task prioritizing schemes.

#### 4.6.5 Edge-front Task Queue Prioritizing

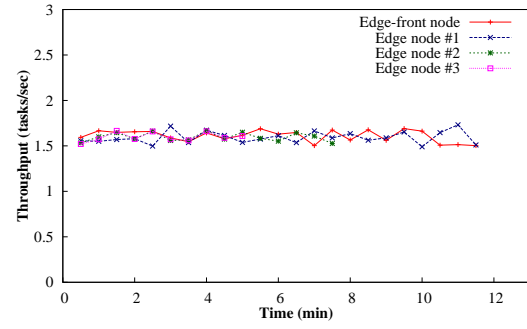
To evaluate the performance of the task queue prioritizing, we collect the statistical results from our profiler service and monitoring service on various workloads for simulation. We choose the simulation method because we can set the numbers and types of client and edge nodes to overcome the limitation of our current testbed to evaluate more complex deployments. We add two simple schemes as baselines: 1) shortest IO first (SIOF): sorting all the tasks against the time cost of the network transmission; 2) longest CPU last (LCPUL): sorting all the tasks against the time cost of the processing on the edge node. In the simulation, based on the combination of client device types, workloads and offloading decisions, we have in total seven types of jobs to run on the edge node. We

increase the total number of jobs and evenly distributed them among the seven types and report the makespan time in Figure 4.12. The result shows that LCPUL is the worst among those three schemes and our scheme outperforms the shortest job first scheme.

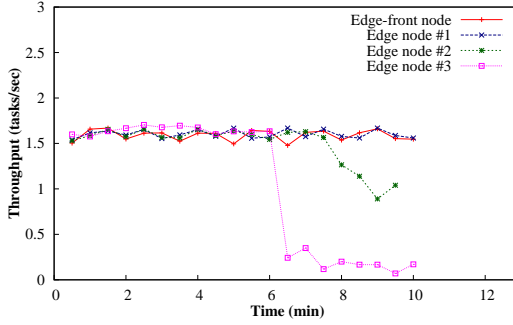
#### 4.6.6 Inter-Edge Collaboration



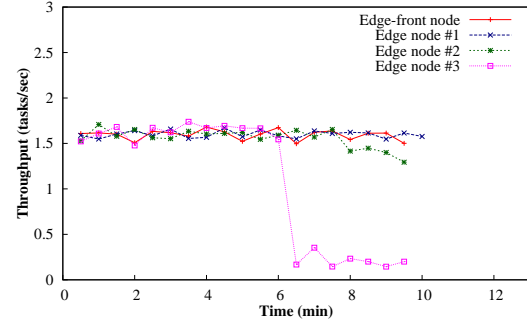
**Figure 4.13:** Performance with no task placement scheme.



**Figure 4.14:** Performance of STTF.



**Figure 4.15:** Performance of SQLF.



**Figure 4.16:** Performance of SSLF.

We also evaluate the three task placement schemes (i.e., STTF, SQLF and SSLF) discussed in Section 4.5, through a controlled experiment on our testbed. For evaluation purpose, we configure the network in the edge computing system as follows. The first edge node, denoted as “edge node #1”, has 10 ms RTT and 40 Mbps bandwidth to the edge-front node. The second edge node, “edge node #2”, has 20 ms RTT and 20 Mbps bandwidth to the edge-front node. The third edge node, “edge node #3”, has 100 ms RTT and 2 Mbps bandwidth to the edge-front node. Thus, we emulate the situation where

three edge nodes are in different distances to the edge-front node, from near to far.

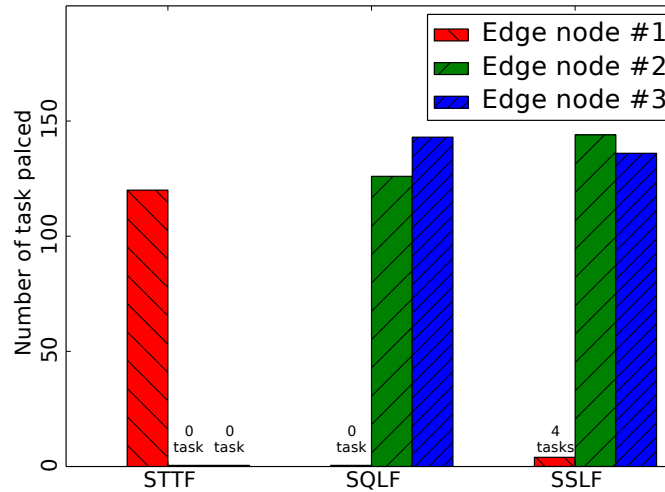
We use the third dataset to synthesize a workload as follows. In the first 4 minutes, the edge-front node receives 5 task requests per second, edge node #1 receives 4 task requests per second, edge node #2 receives 3 task requests per second, and edge node #3 receives 2 task requests per second, respectively. No task comes to any of the edge nodes after the first 4 minutes. For the SSLF task placement scheme, we implement a simple linear regression to predict the scheduling latency of the task being transmitted, since the workload we have injected is uniform distributed.

Figure 4.13 illustrates the throughput on each edge node, when no task placement scheme is enabled on the edge-front node. The edge-front node has the heaviest workload and it takes about 12.36 minutes to finish all the tasks. We consider this result as our baseline.

Figure 4.14 is the throughput result of STTF scheme. In this case, the edge-front node only transmits tasks to edge node #1, because edge node #1 has the highest bandwidth and the shortest RTT to the edge-front node. Figure 4.17 reveals that the edge-front node transmits 120 tasks to edge node #1 and no task to other edge nodes. As edge node #1 has heavier workload than edge node #2 and edge node #3, the STTF scheme has limited improvement on the system performance: the edge-front node takes about 11.29 minutes to finish all the tasks. Figure 4.15 illustrates the throughput result of SQLF scheme. This scheme works better than the STTF scheme, because the edge-front node transmits more tasks to less-saturated edge nodes, efficiently reducing the workload on the edge-front node. However, the edge-front node intends to transmit many tasks to edge node #3 at the beginning, which has the lowest bandwidth and the longest RTT to the edge-front node. As such, the task placement may incur more delay than expected. From Figure 4.17, the edge-front node transmits 0 task to edge node #1, 132 tasks to edge node #2, and 152 tasks to edge node #3. The edge-front node takes about 9.6 minutes to finish all the tasks.

Figure 4.16 demonstrates the throughput result of SSLF scheme. This scheme consid-

ers both the transmission time of the task being placed and the waiting time in the queue on the target edge node, and therefore achieves the best performance of the three. As mentioned, edge node #1 has the lowest transmission overhead but the heaviest workload among the three edge nodes, while edge node #3 has the lightest workload but the highest transmission overhead. In contrast, edge node #2 has modest transmission overhead and modest workload. The SSLF scheme takes all these situations into consideration, and places the most number of tasks on edge node #2. As shown in Figure 4.17, the edge-front node transmits 4 tasks to edge node #1, 152 tasks to edge node #2, and 148 tasks to edge node #3 when working with the SSLF scheme. The edge-front node takes about 9.36 minutes to finish all the tasks, which is the best result among the three schemes. We infer that the third scheme will further improve the task completion time if more tough network conditions and workloads are considered.



**Figure 4.17:** Numbers of tasks placed by the edge-front node.

## 4.7 Related Work

The emergence of edge computing has drawn attentions due to its capabilities to reshape the land surface of IoTs, mobile computing, and cloud computing [13, 36, 83, 84, 97, 99,

101]. Satyanarayanan [79] has briefed the origin of edge computing, also known as fog computing [9], cloudlet [78], mobile edge computing [66] and so on. Here we will review several relevant research fields towards video edge analytics, including distributed data processing and computation offloading in various computing paradigms.

#### **4.7.1 Distributed Data Processing**

Distributed data processing has close relationship to the edge analytics in the sense that those data processing platforms [22, 104] and underlying techniques [39, 65, 73] can be easily deployed on a cluster of edge nodes. In this chapter, we pay specially attentions to distributed image/video data processing systems. VideoStorm [106] made insightful observation on vision-related algorithms and proposed resource-quality trade-off with multi-dimensional configurations (e.g. video resolution, frame rate, sampling rate, sliding window size, etc.). The resource-quality profiles are generated offline and a online scheduler is built to allocate resources to queries to optimize the utility of quality and latency. Their work is complementary to ours, in that we do not consider the trade-off between quality and latency goals via adaptive configurations. Vigil [109] is a wireless video surveillance system that leveraged edge computing nodes with emphasis on the content-aware frame selections in a scenario where multiple web cameras are at the same location to optimize the bandwidth utilization, which is orthogonal to the problems we have addressed here. Firework [108] is a computing paradigm for big data processing in collaborative edge environment, which is complementary to our work in terms of shared data view and programming interface.

While there should be more on-going efforts for investigating the adaptation, improvement, and optimization of existing distributed data processing techniques on edge computing platform, we focus more on the task/application-level queue management and scheduling, and leave all the underlying resource negotiating, process scheduling to the container cluster engine.

### 4.7.2 Computation Offloading

Computation offloading (a.k.a. Cyber foraging [78]) has been proposed to improve resource utilization, response time, and energy consumption in various computing environments [20, 32, 62, 82]. Work [41] has quantified the impact of edge computing on mobile applications and found that edge computing can improve response time and energy consumption significantly for mobile devices through offloading via both WiFi and LTE networks. Mocha [86] has investigated how a two-stage face recognition task from mobile device can be accelerated by cloudlet and cloud. In their design, clients simply capture image and sends to cloudlet. The optimal task partition can be easily achieved as it has only two stages. In LAVEA, our application is more complicated in multiple stages and we leverage client-edge offloading and other techniques to improve the resource utilization and optimize the response time.

## 4.8 Discussions and Limitations

In this section, we will discuss alternative design options, point out current limitations, and identify future work that can improve the system.

**Measurement-based Offloading.** In this work, we utilize a measurement-based offloading (static offloading), i.e, the offloading decisions are based on the outcome of periodic measurements. We consider this as one of the limitations of our implementations, as stated in [38] and there are several dynamic computation offloading schemes have been proposed [26]. We are planning to improve the measurement-based offloading in the future work.

**Video Streaming.** Our current data processing is image-based, which is one of the limitations of our implementation. The input is either in the format of image or in video stream which is read into frames and sent out. We believe that utilizing existing video streaming techniques in between our system components for data sharing will further im-



proves the system performance and opens more potential opportunities for optimization.

**Discovering Edge Nodes.** There are different ways for the edge-front node to discover the available edge nodes nearby. For example, every edge node intending to serve as a collaborator may open a designated port, so that the edge-front node can periodically scan the network and discover the available edge nodes. This is called the “pull-based” method. In contrast, there is also a “push-based” method, in which the edge-front node opens a designated port, and every edge node intending to serve as a collaborator will register to the edge-front node. When the network is in a large scale, the pull-based method usually performs poorly because the edge-front node may not be able to discover an available edge node in a short time. For this reason, the edge node discovery should be implemented in a push-based method, which guarantees good performance regardless of the network scale.

## 4.9 Chapter Summary

In this chapter, we have investigated how to provide video analytic services to latency-sensitive applications in edge computing environment. As a result, we have built LAVEA, a low-latency video edge analytic system, which collaborates nearby client, edge and remote cloud nodes, and transfers video feeds into semantic information at places closer to the users in early stages. We have utilized an edge-front design and formulated an optimization problem for offloading task selection and prioritized task queue to minimize the response time. Our result indicates that by offloading tasks to the closest edge node, the client-edge configuration has a 1.3x to 4x (1.2x to 1.7x) speedup against running locally (client-cloud) under various network conditions and workloads. In case of a saturating workload on the front edge node, we have proposed and compared various task placement schemes that are tailed for inter-edge collaboration. The proposed prediction-based shortest scheduling latency first task placement scheme considers both the transmission time of the tasks and the waiting time in the queue, and outputs better overall performance

than the other schemes.

## Chapter 5

# Conclusion and Future Work

In this dissertation, we outline the latest trends of mobile computing and highlight three challenges. We address these challenges in three mobile systems.

The first one is GlassGesture, an head-gesture user interface for smart glasses. We examine the interface of smart glasses and identify flawed designs. Then we design and implement a head gesture user interface which not only recognizes the gestures for control and input, but also authenticates the user by biometric signatures extracted from user head movements. We design efficient similarity search for weighted dynamic time warping to accelerate the gesture recognition. We propose peak features which are effective at characterizing head gestures and apply ensemble learning scheme to improve the one-class SVM classifiers. Evaluation results show that the gesture recognition is accurate and efficient, and the gesture-based authentication rejects attackers in most trials.

The second system is WearLock, which assists smartphone unlocking via token-based authentication through a paired smartwatch. We secure the acoustic channel by adapting the transmission power and modulation configurations, and sends OTP tokens for validation via acoustics to unlock the smartphone. We offload the heavy computation from the smartwatch to the smartphone, and leverage multi-source information including motion similarities to reduce unnecessary audio transmissions. Compared to entering

PINs, WearLock not only automatically unlocks the smartphone but also accelerates the process.

The last system is LAVEA, a low-latency video edge analytic system, which collaborates nearby mobile client, edge and remote cloud nodes, and transfers video feeds into semantic information at places closer to the users in early stages. We consider the response time minimization problem in three sub-problems. We formulate an optimization problem for offloading task selection and task queue prioritizing problem to minimize the response time. We also propose and compare various task placement schemes to allow one edge node to collaborate with nearby edge nodes. We show that a client application leverages LAVEA achieves up to 4x speedup against running locally.

In summary, our work explores new opportunities in latest mobile computing trends and addresses the associated challenges through novel system designs and technical contributions. It is challenging to thoroughly address all exposed problems of usability, security, and performance of mobile computing systems, which are changing rapidly from time to time. We believe that the projects presented in this dissertation can serve as early explorers along the line of enhancing usability, security, and performance of modern mobile computing systems in the era of wearables and IoTs. We envision that the following research directions can be pursued:

- **Advanced User Interface for Wearables**

The future of wearable user interface will be multi-modal. Graphical User Interface (GUI) will co-exist with other types of user interfaces. The gesture-based user interface in GlassGesture can be easily extended to general sensor-based user interface in which the interactions are done via various sensor data collection and processing. We are integrating the Google Glass with electroencephalogram (EEG) sensor that to allow the wearer to trigger gesture recognition using attention detection based on the EEG signals.

- **Privacy-preserving Sensor Data Evaluations for Mobile Devices**

Smart devices are constantly adding new types of sensors. The data collected by sensors (time-series) can be used to infer various sensitive information of the user. The data will be implicitly or explicitly uploaded to remote servers, for in-depth analysis or advanced functionality. Currently, users are at the risks of privacy leakage while using these sensors. Secure computation is able to support function evaluation and preserves the data privacy. Existing solutions based solely on garbled circuits or homomorphic encryption are not computationally efficient. We are planning to combine homomorphic encryption and oblivious transfer to form a more secure and efficient scheme that optimized for mobile devices.

- **Long-term Direction: Hyper-converging at Mobile, Edge, and Cloud**

In this dissertation, we started to converge mobile computing and edge computing through low-latency video analytics. The edge computing platform we have designed and implemented can support offloading of computation tasks from clients to edge or cloud nodes. One direction is to extend the LAVEA offloading framework to manage not only computation resource but also storage and network resources. How to offload tasks according to different resource requests with the consideration of maintaining quality of experience (QoE) is a big challenge. Another direction is to expand the offloading framework from client-edge two-layer design to client-edge-cloud three-layer design. Thus the serverless architecture will support flexible task offloading among client, edge, and cloud nodes. Furthermore, as a long-term direction, it would be beneficial if we can build a realistic worldwide edge computing testbed, which can be used by us or other researchers to deploy and evaluate new edge computing services and applications. We are planning to build incentive mechanism via blockchain and cryptocurrency techniques to recruit edge computing nodes in a “bring-your-own-edge-node” style. The challenges include how to build trusted edge computing services via blockchain and smart-contract technologies, how to manage large-scale geographically distributed edge nodes, how to incen-

tivize the edge nodes to join the network, and how to incentivize the edge clients and edge nodes to follow the rules. We believe the future of hyper-converging of mobile, edge, and cloud is very promising while significant challenges ahead are waiting for us to address.

# Bibliography

- [1] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42(12):40–46, 1999.
- [2] Ahmad Akl, Chen Feng, and Shahrokh Valaee. A novel accelerometer-based gesture recognition system. *IEEE Transactions on Signal Processing*, 2011.
- [3] Ahmad Akl and Shahrokh Valaee. Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing. In *ICASSP '10*, 2010.
- [4] Amazon Web Service. Aws lambda@edge. <http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html>, 2017.
- [5] Christos-Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Ioannis D Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on intelligent transportation systems*, 9(3):377–391, 2008.
- [6] KR Baker. Scheduling groups of jobs in the two-machine flow shop. *Mathematical and Computer Modelling*, 13(3):29–36, 1990.
- [7] Cheng Bo, Lan Zhang, Xiang-Yang Li, et al. Silentsense: silent user identification via touch and movement behavioral biometrics. In *MobiCom '13*, 2013.
- [8] Hristo Bojinov and Dan Boneh. Mobile token-based authentication on a budget. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 14–19. ACM, 2011.
- [9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.
- [10] Russell Brandom. The Verge: Your phone’s biggest vulnerability is your fingerprint - can we still use fingerprint logins in the age of mass biometric databases? <https://goo.gl/VmNKsP>, May 2016.
- [11] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 344–359. Springer, 1993.

- [12] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, 49(1):107–127, 1994.
- [13] Yu Cao, Songqing Chen, Peng Hou, and Donald Brown. Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation. In *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pages 2–11. IEEE, 2015.
- [14] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *TIST '11*, 2011.
- [15] Jagmohan Chauhan et al. Gesture-based continuous authentication for wearable devices: the google glass case. *arXiv preprint*, 2014.
- [16] Yimin Chen, Jingchao Sun, Rui Zhang, and Yanchao Zhang. Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices. In *INFO-COM'15*, pages 2686–2694, Hong Kong, China, Apr. 2015.
- [17] Andrea Colaço et al. Mime: Compact, low power 3d gesture sensing for interaction with head mounted displays. In *UIST '13*, 2013.
- [18] Mark D Corner and Brian D Noble. Zero-interaction authentication. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 1–11. ACM, 2002.
- [19] Malcolm J. Crocker. *Handbook of acoustics*. John Wiley & Sons, 1998.
- [20] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.
- [21] Eyal de Lara, Carolina S Gomes, Steve Langridge, S Hossein Mortazavi, and Meysam Roodi. Hierarchical serverless computing for the mobile edge. In *Edge Computing (SEC), IEEE/ACM Symposium on*, pages 109–110. IEEE, 2016.
- [22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [23] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. Automatic license plate recognition (alpr): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology*, 23(2):311–325, 2013.
- [24] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winsten. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In *14th USENIX Symposium on Networked Systems Design*



*and Implementation (NSDI 17)*, pages 363–376, Boston, MA, 2017. USENIX Association.

- [25] Michael Frank, Ralf Biedert, En-Di Ma, Ivan Martinovic, and Dong Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *TIFS '13*, 8, 2013.
- [26] Wei Gao, Yong Li, Haoyang Lu, Ting Wang, and Cong Liu. On exploiting dynamic execution patterns for workload offloading in mobile cloud applications. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 1–12. IEEE, 2014.
- [27] Google. Head wake up/nudge. <https://goo.gl/6lfFWg>.
- [28] Google. Locations and sensors. <https://goo.gl/0j6Mqg>.
- [29] Google. Nearby API. <https://developers.google.com/nearby/>.
- [30] Google. Screen lock. <https://goo.gl/Knf7p1>.
- [31] Google. Trusted bluetooth devices. <https://goo.gl/xEZSCw>.
- [32] Mark S Gordon, Davoud Anoushe Jamshidi, Scott A Mahlke, Zhuoqing Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *OSDI*, volume 12, pages 93–106, 2012.
- [33] Isabelle Guyon et al. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.
- [34] Hao Han, Shanhe Yi, Qun Li, Shen Guobin, Yunxin Liu, and Edmund Novak. AMIL: localizing neighboring mobile devices through a simple gesture. In *The 35th Annual IEEE International Conference on Computer Communications (INFOCOM 2016)*. IEEE, 2016.
- [35] Zijiang Hao and Qun Li. Towards user re-authentication on mobile devices via on-screen keyboard. In *Hot Topics in Web Systems and Technologies (HotWeb), 2016 Fourth IEEE Workshop on*, pages 78–83. IEEE, 2016.
- [36] Zijiang Hao, Ed Novak, Shanhe Yi, and Qun Li. Challenges and software architecture for fog computing. *Internet Computing*, 2017.
- [37] Marian Harbach, Emanuel von Zezschwitz, Andreas Fichtner, Alexander De Luca, and Matthew Smith. It’s a hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 213–230, 2014.
- [38] Mohammed A Hassan, Kshitiz Bhattarai, Qi Wei, and Songqing Chen. Pomac: Properly offloading mobile applications to clouds. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

- [39] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.
- [40] Takahiro Horikawa. Head gesture detector. <https://goo.gl/uRgVnu>.
- [41] Wenlu Hu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai, and Mahadev Satyanarayanan. Quantifying the impact of edge computing on mobile applications. In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, page 5. ACM, 2016.
- [42] IBM. Apache openwhisk. <http://openwhisk.org/>, April 2017.
- [43] Rong Jin, Liu Shi, Kai Zeng, Amit Pande, and Prasant Mohapatra. Magpairing: Pairing smartphones in close proximity using magnetometers. *IEEE Transactions on Information Forensics and Security*, 11(6):1306–1320, 2016.
- [44] Selmer Martin Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.
- [45] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99%. *arXiv preprint arXiv:1702.04024*, 2017.
- [46] Kantar. Wearable technology report. <https://goo.gl/N6DEV4>, 2016.
- [47] Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. Sound-proof: usable two-factor authentication based on ambient sound. In *USENIX Security 15*, pages 483–498, 2015.
- [48] Zoe Kleinman. BBC News: Politician’s fingerprint ‘cloned from photos’ by hacker. <http://www.bbc.com/news/technology-30623611>, 2014.
- [49] Manuel Koschuch, Matthias Hudler, Hubert Eigner, and Zsolt Saffer. Token-based authentication for smartphones. In *Data Communication Networking (DCNET), 2013 International Conference on*, pages 1–6. IEEE, 2013.
- [50] Hyewon Lee, Tae Hyun Kim, Jun Won Choi, and Sunghyun Choi. Chirp signal-based aerial acoustic communication for smart devices. In *Proc. of IEEE Conf. on Computer Communications (INFOCOM), Hong Kong SAR, PRC*, 2015.
- [51] Lingjun Li, Guoliang Xue, and Xinxin Zhao. The power of whispering: Near field assertions via acoustic communications. In *ASIA CCS’15*, pages 627–632. ACM, 2015.
- [52] Lingjun Li, Xinxin Zhao, and Guoliang Xue. Unobservable re-authentication for smartphones. In *NDSS ’13*, 2013.
- [53] Tao Li, Yimin Chen, Jingchao Sun, Xiaocong Jin, and Yanchao Zhang. ilock: Immediate and automatic locking of mobile devices against data theft. In *CCS’16*, pages 933–944, Vienna □ Austria, Oct. 2016.

- [54] Robert LiKamWa, Zhen Wang, Aaron Carroll, et al. Draining our glass: An energy and heat characterization of google glass. In *APSys '14*, 2015.
- [55] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5, 2009.
- [56] Kaikai Liu, Xinxin Liu, and Xiaolin Li. Guoguo: Enabling fine-grained indoor localization via smartphone. In *Mobisys*, pages 235–248. ACM, 2013.
- [57] Hong Lu, Jun Yang, et al. The jigsaw continuous sensing engine for mobile phone applications. In *Sensys '10*, 2010.
- [58] Louis-Philippe Morency and Trevor Darrell. Head gesture recognition in intelligent interfaces: the role of context in improving recognition. In *IUI '06*, 2006.
- [59] Morgan Stanley. Wearable devices the ‘internet of things’ becomes personal. <https://goo.gl/6MC02M>, 2014.
- [60] D M’Raihi, M Bellare, F Hoornaert, D Naccache, and O Ranen. Hotp: An HMAC-based one-time password algorithm. *The Internet Society, Network Working Group. RFC4226*, 2005.
- [61] Rajalakshmi Nandakumar, Krishna Kant Chintalapudi, Venkat Padmanabhan, and Ramarathnam Venkatesan. Dhvani: Secure peer-to-peer acoustic nfc. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 63–74, New York, NY, USA, 2013. ACM.
- [62] Ryan Newton, Sivan Toledo, Lewis Girod, Hari Balakrishnan, and Samuel Madden. Wishbone: Profile-based partitioning for sensornet applications. In *NSDI*, volume 9, pages 395–408, 2009.
- [63] Lawrence O’Gorman. Comparing passwords, tokens, and biometrics for user authentication. *Proceedings of the IEEE*, 91(12):2021–2040, 2003.
- [64] OpenALPR. OpenALPR – Automatic License Plate Recognition. <http://www.openalpr.com/>, April 2017.
- [65] Kay Ousterhout, Patrick Wendell, Matei Zaharia, and Ion Stoica. Sparrow: distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 69–84. ACM, 2013.
- [66] M Patel, B Naughton, C Chan, N Sprecher, S Abeta, A Neal, et al. Mobile-edge computing introductory technical white paper. *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.
- [67] Ge Peng, Gang Zhou, David T Nguyen, Xin Qi, Qing Yang, and Shuangquan Wang. Continuous authentication with touch behavioral biometrics and voice on wearable glasses. *IEEE Transactions on Human-Machine Systems*, 47(3):404–416, 2017.

- [68] Roberto Perdisci, Guofei Gu, et al. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *ICDM '06*, pages 488–498. IEEE, 2006.
- [69] Brad Philip and Paul Updike. Caltech vision group 2001 testing database. <http://www.vision.caltech.edu/html-files/archive.html>, 2001.
- [70] Qifan Pu, Sidhant Gupta, Shyamnath Gollakota, and Shwetak Patel. Whole-home gesture recognition using wireless signals. In *MobiCom '13*, 2013.
- [71] Kari Pulli, Anatoly Baksheev, Kirill Korniyakov, and Victor Eruhimov. Real-time computer vision with opencv. *Communications of the ACM*, 55(6):61–69, 2012.
- [72] Swati Rallapalli, Aishwarya Ganesan, Krishna Chintalapudi, et al. Enabling physical analytics in retail stores using smart glasses. In *MobiCom '14*, 2014.
- [73] Jeff Rasley, Konstantinos Karanasos, Srikanth Kandula, Rodrigo Fonseca, Milan Vojnovic, and Sriram Rao. Efficient queue management for cluster scheduling. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 36. ACM, 2016.
- [74] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *AAAI '05*, volume 5, 2005.
- [75] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. Progressive authentication: deciding when to authenticate on mobile phones. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 301–316, 2012.
- [76] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *KDD '04*. Citeseer, 2004.
- [77] G Enrico Santagati and Tommaso Melodia. U-wear: Software-defined ultrasonic networking for wearable devices. In *Mobisys*, pages 241–256. ACM, 2015.
- [78] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal communications*, 8(4):10–17, 2001.
- [79] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.
- [80] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.
- [81] Muhammad Shahzad, Alex X Liu, and Arjmand Samuel. Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it. In *MobiCom '13*, 2013.

- [82] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. Cosmos: computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM international symposium on Mobile ad hoc networking and computing*, pages 287–296. ACM, 2014.
- [83] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.
- [84] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.
- [85] Bernard Sklar. *Digital communications*, volume 2. Prentice Hall NJ, 2001.
- [86] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000059–000066. IEEE, 2012.
- [87] Jingchao Sun, Rui Zhang, Jinxue Zhang, and Yanchao Zhang. Touchin: Sightless two-factor authentication on multi-touch mobile devices. In *CNS’16*, pages 436–444, San Francisco, CA, Oct. 2014.
- [88] Wai-Tian Tan, Mary Baker, Bowon Lee, and Ramin Samadani. The sound of silence. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, page 19. ACM, 2013.
- [89] Dirk Van Bruggen, Shu Liu, Mitch Kajzer, Aaron Striegel, Charles R Crowell, and John D’Arcy. Modifying smartphone user locking behavior. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, page 10. ACM, 2013.
- [90] Wei Wang, Lin Yang, and Qian Zhang. Touch-and-guard: secure pairing through hand resonance. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 670–681. ACM, 2016.
- [91] Xiaoli Wang, Aakanksha Chowdhery, and Mung Chiang. Skyeyes: adaptive video streaming from uavs. In *Proceedings of the 3rd Workshop on Hot Topics in Wireless*, pages 2–6. ACM, 2016.
- [92] PUID WG. Bluetooth proximity profile spec doc v1.0.1, July 2015.
- [93] Wikipedia. User interface. [https://en.wikipedia.org/wiki/User\\_interface](https://en.wikipedia.org/wiki/User_interface).
- [94] Jacob O Wobbrock et al. Gestures without libraries, toolkits or training: a \$1 recognizer for user interface prototypes. In *UIST ’07*, 2007.
- [95] Hai Yan, Lei Wan, Shengli Zhou, Zhijie Shi, Jun-Hong Cui, Jie Huang, and Hao Zhou. Dsp based receiver implementation for ofdm acoustic modems. *Physical Communication*, 5(1):22–32, 2012.

- [96] Byoung-Kee Yi, HV Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE '98*, pages 201–208. IEEE, 1998.
- [97] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *Hot Topics in Web Systems and Technologies (HotWeb), 2015 Third IEEE Workshop on*, pages 73–78. IEEE, 2015.
- [98] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing, SEC '17*, pages 15:1–15:13, New York, NY, USA, 2017. ACM.
- [99] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: Concepts, applications and issues. In *Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata '15*, pages 37–42. ACM, 2015.
- [100] Shanhe Yi, Zhengrui Qin, Nancy Carter, and Qun Li. Wearlock: Unlocking your phone via acoustics using smartwatch. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 469–479, June 2017.
- [101] Shanhe Yi, Zhengrui Qin, and Qun Li. Security and privacy issues of fog computing: A survey. In *Wireless Algorithms, Systems, and Applications*, pages 685–695. Springer, 2015.
- [102] Shanhe Yi, Zhengrui Qin, Ed Novak, Yafeng Yin, and Qun Li. Glassgesture: Exploring head gesture interface of smart glasses. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.
- [103] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 27–32. ACM, 2015.
- [104] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.
- [105] Bingsheng Zhang, Qin Zhan, Si Chen, Muyuan Li, Kui Ren, Cong Wang, and Di Ma. PriWhisper : Enabling keyless secure acoustic communication for smartphones. *IEEE Internet of Things Journal*, 1(1):33–45, Feb 2014.
- [106] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, 2017. USENIX Association.

- [107] Huanle Zhang, Wan Du, Pengfei Zhou, Mo Li, and Prasant Mohapatra. Dopenc: acoustic-based encounter profiling using smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 294–307. ACM, 2016.
- [108] Quan Zhang, Qingyang Zhang, Weisong Shi, and Hong Zhong. Firework: Data processing and sharing for hybrid cloud-edge analytics. *IEEE Transactions on Parallel and Distributed Systems*, 2018.
- [109] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 426–438. ACM, 2015.
- [110] Yulong Zhang, Zhaofeng Chen, Hui Xue, and Tao Wei. Fingerprints on mobile devices: Abusing and leaking. In *Black Hat Conference*, 2015.