UNIVERSITY OF CALIFORNIA

Los Angeles

Software Mechanisms for Pervasive and Autonomous Computing

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Salma Hosni Emam M. Elmalaki

2018

ABSTRACT OF THE DISSERTATION

Software Mechanisms for
Pervasive and Autonomous Computing

by

Salma Hosni Emam M. Elmalaki
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2018
Professor Mani B. Srivastava, Chair

Ubiquitous computing—that interacts and adapts to humans—is inevitable. In these pervasive systems, human reactions and behavior are observed and coupled into the loop of computation. The new generation of these autonomous systems has enabled a multitude of applications in the context of smart cities, healthcare, and automotive systems. By enabling autonomy into the essence of pervasive systems, these evolving systems not only provide services that are adaptable to the human context but also intervene and take actions that are tailored to the human reaction and behavior. *The objective of this dissertation is to weave the personalization and context-awareness into the very fabric of autonomous pervasive systems.*

The contributions of this dissertation are multi-fold. The first part of the thesis addresses the system software design to build context-aware applications that can adapt to different human and environment state. We introduce a framework for Android OS that can facilitate the implementation of the context-aware application which we named CAreDroid. The newly developed OS support is designed to decouple the application logic from the complex adaptation decisions in Android context-aware apps. In particular, several case studies implemented using the designed OS are shown to facilitate the implementation of personalized mobile apps by having at least half lines of code fewer and at least 10 more efficient in execution time compared to equivalent context-aware apps that use standard Android.

The second part of the thesis looks into the privacy concerns that arise from the adapta-

tion of personalized systems where the human interactions and behavior can leak sensitive information. We show that context-aware systems open the door for side-channel to leak sensitive personal information. That is, while context-aware autonomous applications adapt their behavior based on the current context of the user, this very act of changing the behavior can be used by malicious software to reverse engineer the human context. In this part, we studied the extent to which a malicious app can monitor the adaptations triggered by authentic context-aware apps and extract user's information. In particular, we showed a concrete instantiation of a new category of spyware which we refer to as Context-Aware Adaptation Based Spyware (SpyCon). Afterwards, we proposed a novel OS software mechanism to detect and mitigate SpyCon apps called VindiCo. Being a new spyware, traditional spyware detection methods that are based on code signature or app behavior are not adequate to detect SpyCon. Therefore, VindiCo proposes a novel information-based detection technique and several mitigation strategies.

The third part focuses on designing machine-learning based systems to build adaptation and personalization services. In this perspective, we show end-to-end applications that interact with humans and adapt to their needs and preferences. We focus on the area of context-aware driver assistance systems (ADAS). We show that by using the monitored human state to design driver-in-the-loop systems, these systems can provide personalized driving experience. We purpose Sentio, a Reinforcement Learning solution to take the human reactions and behavior into the loop of computation. We then discuss an architecture for personalized and autonomous IoT (IoPAT) by showing an example of personalized smart home application.

The dissertation of Salma Hosni Emam M. Elmalaki is approved.

Todd D. Millstein

Puneet Gupta

William J. Kaiser

Mani B. Srivastava, Committee Chair

University of California, Los Angeles

2018

To my beloved husband,
my beautiful babies Yahia and Layla
who—among so many other things—
gave me joy in life

TABLE OF CONTENTS

1 Introduction **1**

1.1 What is Pervasive Autonomy? 2

1.2 Challenges 2

1.2.1 Adaptability 2

1.2.2 Privacy 3

1.2.3 Context Engines Support 4

1.3 Our Contribution 4

1.4 Organization Of This Dissertation 6

I Operating System Support for Pervasive Autonomous Systems **7**

2 CAreDroid: Adaptation framework for context-aware mobile application **8**

2.1 Introduction 8

2.1.1 Related Work 9

2.1.2 CAreDroid Contribution 12

2.2 System Architecture 13

2.3 Sensitivity Configuration File 16

2.3.1 Configuration File Structure 16

2.3.2 Configuration File Processing 18

2.3.3 Online Change of Context Ranges 21

2.4 CAreDroid context monitoring 23

2.4.1 Raw Context Monitoring 23

2.4.2 Inferred Context: Mobility State 24
2.5 CAreDroid Adaptation Engine 24
2.5.1 Dalvik Interpreter Extension 24
2.5.2 Which Polymorphic Implementation to Pick? 26
2.5.3 Conflict Resolution Cache 28
2.6 Evaluation 28
2.6.1 Case Study 1: A Simple Application 29
2.6.2 Case Study 2: A Context-Aware Phone Configuration 32
2.6.3 Case Study 3: Context-Aware Camera 35
2.6.4 Case study 4: A Context-Aware Image Processing Application 37
2.7 Discussion 39
2.7.1 Why is CAreDroid implemented inside the OS? 39
2.7.2 Privacy 39
2.7.3 Developer Matters 40
2.7.4 Limitations 40
2.7.5 Broader Uses of CAreDroid 41
2.8 Conclusion 41

3 CAMPS: Charging-aware Adaptation for Power Management in Mobile Operating System 43

3.1 Introduction 43
3.2 Smartphone Charging Profile 44
3.2.1 Effect of Charger Type 45
3.3 User Charging Behavior 47
3.3.1 SOC at Plug-In Event 47

3.3.2 Charging Duration 48
3.3.3 SOC at Un-plug Event 49
3.3.4 User Distribution 51
3.4 Opportunities for Task Deferral 51
3.4.1 Schedule Tasks After Unplugging 52
3.4.2 Schedule Tasks Within the Constant Current Phase 53
3.4.3 Schedule Tasks in the Power Headroom 53
3.5 Conclusion and Future Work 54

II Privacy Firewall for Personalized Autonomous Computing 56

4 SpyCon: Context-aware Adaptation Based Spyware 57
4.1 Introduction 57
4.1.1 Related Work 58
4.1.2 Chapter Contribution 58
4.2 Context-aware Adaptation based Spyware 59
4.2.1 Popular Phone Manager Apps 59
4.2.2 Spyware Description 59
4.2.3 SpyCon User Study 60
4.2.4 Experiment 1: Data Mining by Clustering 62
4.2.5 Experiment 2: Detection Using Current Antivirus Apps 64
4.2.6 Experiment 3: Beyond Location SpyCon 65
4.2.7 Experiment 4: How many SpyCons in the market? 66
4.3 Conclusion 66

5 VindiCo: Privacy Safeguard Against Context-aware Adaptation Based Spyware . . . 68

5.1 Introduction . . . 68

5.1.1 Related Work . . . 68

5.1.2 Contribution . . . 69

5.2 VindiCo System Architecture . . . 69

5.2.1 Threat Model and Design Objectives . . . 70

5.2.2 Context-adaptation Registration . . . 71

5.2.3 Information-Based Detection Engine . . . 72

5.2.4 Mitigation Engine . . . 73

5.3 Implementation . . . 77

5.3.1 VindiCo Context-adaptation Registration . . . 77

5.3.2 VindiCo Detection Engine . . . 78

5.3.3 VindiCo Mitigation Engine . . . 81

5.4 Evaluation . . . 81

5.4.1 Experiment 5: Performance of Information-Based Detection . . . 82

5.4.2 Experiment 6: Performance of Mitigation Algorithms . . . 84

5.4.3 Experiment 7: Timing Analysis of VindiCo . . . 85

5.4.4 Experiment 8: Effect on Benign Applications . . . 85

5.5 Conclusion . . . 87

III Personalization of Pervasive Autonomy 89

6 Sentio: Driver-in-the-Loop Forward Collision Warning Using Multisample Reinforcement Learning . . . 90

6.1 Introduction . . . 90

6.1.1 Related Work . . . 92
6.1.2 Contribution . . . 94
6.2 Sentio System Architecture . . . 94
6.2.1 Human Context-Inference . . . 94
6.2.2 Vehicle and Environment Context-Inference . . . 95
6.2.3 Context-Aware Adaptation Engine . . . 95
6.3 Human Driver as a Markov Decision Process . . . 98
6.4 Dynamic & Time-Varying Rewards . . . 100
6.4.1 Reward Function Definition . . . 101
6.4.2 Random Human Actions and Erroneous Rewards . . . 102
6.5 Multisample Q-Learning . . . 102
6.5.1 Standard Q-Learning . . . 102
6.5.2 Multisample Q-Learning Algorithm . . . 103
6.6 Experimental Results . . . 106
6.6.1 Parameter Tuning . . . 109
6.6.2 Human Driving Experience . . . 117
6.6.3 Execution Time Analysis . . . 120
6.7 Discussions . . . 120
6.8 Conclusion . . . 121

7 IoPAT: Internet of Personalized and Autonomous Things . . . 122

7.1 IoPAT Systems . . . 123
7.2 Architecture for IoPAT Edge Devices . . . 124
7.2.1 Resilient Context Fusion . . . 124
7.2.2 Reinforcement Learning Controller . . . 125

7.2.3 Information-Based Firewall . . . 126
7.3 Case Study . . . 126
7.3.1 Thermal Model of a House . . . 127
7.3.2 Human Thermal Comfort . . . 128
7.3.3 RL-based Controller for IoPAT . . . 129
7.3.4 Numerical Results . . . 129
7.4 Conclusion . . . 130

8 Conclusion and Future Research . . . 131
8.1 Conclusion . . . 131
8.1.1 CAreDroid . . . 131
8.1.2 CAMPS . . . 132
8.1.3 SpyCon . . . 132
8.1.4 VindiCo . . . 133
8.1.5 Sentio . . . 133
8.1.6 IoPAT . . . 134
8.2 Future Research . . . 134
8.2.1 Mobile-Assisted, Context-aware, and Personalized Automotive Systems 134
8.2.2 Context-Aware Internet-of-Things for Personalized Healthcare . . . 136
8.2.3 Context-Aware Personalized Differentiated Learning . . . 136

References . . . 138

LIST OF FIGURES

2.1 CAreDroid System Architecture. The developer provides a set of polymorphic methods and provides a configuration file describing how these methods shall be called. At runtime, CAreDroid monitors the phone context and adapts the application behavior accordingly. 13

2.2 Snippet of a CAreDroid configuration file. 17

2.3 CAreDroid's extended installation flow. CAreDroid intercepts the installation process of the app on the device in order to parse the sensitivity configuration file. The final outcome of this process is two data structures named "Replacement Map" and "Table of Range Identifiers." 18

2.4 Flow of CAreDroid Adaptation Engine at runtime. The CAreDroid extended interpreter intercepts the execution of the Dalvik opcode *invoke-virtual-quick* to check whether the method invoked is sensitive or not. If the method is sensitive, then the CAreDroid adaptation engine checks the current context and picks the correct polymorphic method. This process is done through leveraging the information in the TRI and RM data structures along with the conflict resolution mechanism implemented using the decision graph. Finally, to speed up the process, CAreDroid uses a resolution cache, which exploits the temporal locality of the context. 25

2.5 An example of a Replacement Map(RM) (right) and its associated decision graph (left). The nodes at each level correspond to the ORIs in the same level of the associated RM. The edges in the decision graph correspond to the five methods shown in the RM. The shaded nodes correspond to the ORIs that match the current context. The solid arrows correspond to the active paths that match both the RM and the current context. Finally the path marked in green corresponds to the method that satisfies all the ORIs, and therefore this method is the one picked by CAreDroid for execution. 27

2.6 Energy consumption results for case study 1 showing (a) energy used when context monitoring is running alone (b) energy consumed by the context switching subsystem and (c) the total energy consumed. On each case, we plot the energy consumed by the pure Java implementation as well as the must fit and best fit implementations of CAreDroid. The results in (a) show that bypassing the HAL layer and implementing the context monitoring inside the OS allowed CAreDroid to use 36% less energy within the 2.5 hours lapse of the experiment. The results in (b) show that both Must fit and Best fit adaptation significant outperform the pure Java implementation in terms of energy consumed (and hence battery lifetime). The overall results (c) show that CAreDroid consumes only 6.73% energy compared with the non-context aware implementation and provides 69.33% energy saving compared to the pure Java implementation. 33

2.7 Photos taken by the Smart Camera application developed for case study 3: (a) the photo taken while the phone holder is standing still, (b) the photo taken while the phone holder is walking and no context-awareness is taking place, and (c) photo taken while the phone holder is walking and using the Smart Camera application built on top of CAreDroid. 36

2.8 Accuracy results of the different polymorphic methods used for the context-aware image processing application used in case study 4. 37

2.9 Results of different edge detection algorithms used in case study 4. This figure shows the percentage of false positives and false negatives versus CPU execution time for different algorithms. 38

3.1 Illustration of typical battery charging current and voltage characteristics. 44

3.2 Nexus 4 charging characteristics from 0% to 100% SOC. The smartphone was powered on and idle for each test. 45

3.3 Mean SOC at plug-in events for each user. 48

3.4 Charging behavior for three distinct users. Each row represents an exemplar user whose behavior follows different charging behavior trends from Classes 1, 2, and 3, which are colored accordingly as blue, green, and red. 49

3.5 Histogram of charging duration for all charging events across all users. 50

3.6 Charging duration vs. SOC when plugged in for all charging events across all users. Blue $*$ represents users of Class 1. Green $\bullet$ represents users of Class 2. Red $+$ represents users of Class 3. 50

3.7 Effect of running an app during vs. after the charging period. 52

3.8 Effect of running an app early vs. late in the CC phase. 53

3.9 Effect of running an app in the CC vs. CV phases. 54

4.1 One example of profile timeline from user #2. (a) The golden output. (b) Clustering result based on full observation. (c) Clustering result based on dominant features derived by feature selection algorithm. 62

4.2 Accuracy of leaking information about user from data collected by 45 of the most downloaded free apps; (blue) accuracy of identifying the semantics of the user location when a location-based context-aware app (Tasker/Locale) is used, and (red) accuracy of identifying user calendar profile when a calendar-based context-aware app (Silence 2.0) is used. 65

5.1 Snippet of a VindiCo registry file. 72

5.2 VindiCo architecture. The context-aware application is registered in VindiCo by context-adaptation registration module. The behavior of the context-aware app is monitored, and a possible SpyCon is detected by the Detection Engine. Adequate mitigation technique is then taken by the Mitigation Engine. 75

5.3 Context-adaptation registration. At installation time, VindiCo checks the existence of a registry file and starts processing it accordingly. Registry file processing constructs all necessary data structures that are needed by the detection and mitigation engines. 76

5.4 VindiCo Detection Engine. When a context-aware app calls a *setter* API to adapt to user context, VindiCo Service intercepts the call and checks if this API call is in the *Adaptation Record*. Next, the mutual information algorithm updates the corresponding Mutual Information Table based on the new data recorded in the Adaptation Record. Whenever an app calls a *getter* API that matches one of the API in the *Protection Lists*, the mutual information corresponding to the *getter* API is retrieved and assigned to this app as a suspicion score. 79

5.5 Profile timeline of user #2 after VindiCo applies the mitigation techniques. (a)Suppression mitigation *(3 rows)* (b)Row-masking mitigation *(p=0.4)* (c)Feature-masking mitigation *(p=0.4)* 82

5.6 Mutual information (MI) and clustering accuracy difference (Acc diff)—with respect to the baseline accuracy—after applying different mitigation methods. When mitigation magnitude increases, both mutual information and accuracy decrease. (a) MI by suppression (b) MI by row-masking (c) MI by feature-masking (d) Accuracy by suppression (e) Accuracy by row-masking (f) Accuracy by feature-masking 83

5.7 Performance of the VindiCo information-based detector (in terms of false positive and false negative rates) versus different alarm thresholds. 83

6.1 Sentio architecture. The context of the driver (attention) is inferred using the data collected by phone and wearables. The context of the vehicle (vehicle speed) and vehicle environment (distance to front collision) is collected by the vehicle sensors. The adaptation actions are then personalized based on the driver's decision. 92

6.2 Modeling human-vehicular interaction using Reinforcement Learning. 97

6.3 A Markov Decision Process model for the human driver and the vehicle. The states of the MDP corresponds to the state of both the driver (attention level) and the vehicle (relative distance and relative velocity). To capture the fact that human behaviors change over time and across different individuals, the transition probabilities between these states are assumed to be unknown and time-varying. To enhance readability, the relative velocity is quantized into two states (low relative velocity and high relative velocity) and only the transitions of the state (distracted, $\Delta d > 14$, low relative velocity) are shown. 99

6.4 Multisample timescales. An action is taken by the agent every T_a samples. The reward is calculated after an action is taken by a time equals T_l. In this example, $T_a = 5T_s$ and $T_l = 10T_s$, where T_s is the state observing (sensor) rate. 104

6.5 Handling the reward function for four different cases for driver behavior. Case 1: FCW is on and driver acknowledges it by pressing the brakes. Case 2: FCW is on and driver ignores it. Case 3: FCW is off and driver acknowledges it by not pressing the brakes. Case 4: FCW is off and the driver presses the brakes. To enhance the readability of the figure we removed the subscript a and the index t from the notation of $I_a(t)$. 105

6.6 Vehicle dynamics virtualization testbed used in the evaluation study 108

6.7 False positives and negatives with different learning parameters for five driving traces across 15 minutes simulation time: (1) $\epsilon = 0.1, \alpha = 1.0, \gamma = 1.0$, (2) $\epsilon = 0.1, \alpha = 0.6, \gamma = 1.0$, (3) $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.8$, (4) $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.7$, and (5) $\epsilon = 0.5, \alpha = 0.6, \gamma = 0.8$. 110

6.8 False positives and negatives with different values for T_a and T_l for five driving traces across 15 minutes simulation time using $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.8$: (1) $T_a = 0.5s, T_l = 2.5s$, and (2) $T_a = 0.5s, T_l = 4s$, (3) $T_a = 0.5s, T_l = 5s$, (4) $T_a = 2.5s, T_l = 5s$, (5) $T_a = 4s, T_l = 5s$. 113

6.9 Relative distances and violations of two distracted drivers with change in behavior over time over a simulation time of 30 minutes using Sentio and with a fixed threshold alert. 114

6.10 Driving Safety and Experience for Driver #1. 116

6.11 Driving Safety and Experience for Driver #2. 116

7.1 Proposed architecture for IoPAT edge devices. 124

7.2 Prediction Mean Vote (PMV) for the three occupants (y-axis) across time (x-axis) using IoT system (left) and IoPAT system (right) for varying occupants' activity and stress level (relaxed/stress). 127

8.1 A pictorial architecture for the envisioned mobile-assisted, context-aware, and personalized automotive systems. Information communicated over the internal networks inside the automotive system is fused with information collected from various phone/wearables sensors to infer complex human and environment state. This complex state is then used to adapt the behavior of both the car as well as the various apps running over the phone. 135

LIST OF TABLES

2.1 On the top, examples of TRI tables for Battery capacity and Signal strength (RSSI) contexts. Each TRI associates a unique *Operation Range Identifier* (ORI) to each record. For each class, CAreDroid creates TRI for all different contexts. The association between the TRIs and the class is done later, after the optimization of the DEX files takes place. On the bottom, a Replacement Map that associates each key = (class-id, method-id) with its corresponding ORIs. CAreDroid creates a unique RM for the application. — *legend:* B: Battery Capacity, T: Battery Temperature, V: Battery Voltage, W: WiFi connectivity, S: Signal strength, Q: Signal Quality M: Mobility L: Location. 22

2.2 significant line of code (SLOC) results for case study 1 showing the SLOC for different implementations along with the percentage increase of SLOC relative to the non-context aware implementation. 29

2.3 Execution time results for case study 1 showing the profiling of different parts for all the three implementations. The overhead is computed relative to the non context-aware implementation. The results show the efficiency of both the must fit and best fit policies. It also shows the performance increase resulting from using the cache. 30

2.4 Significant lines of code (SLOC) results for case study 2 showing the SLOC for the three implementations along with the percentage increase of SLOC relative to the non-context aware implementation. 34

2.5 Execution time results for case study 2 showing the overhead for the different implementations. 34

2.6 Results of the significant line of code (SLOC) for the three implementations of the Smart Camera application used in case study 3. The results shows the SLOC along with the percentage increase relative to the non-context aware implementation. 36

4.1 List of Phone settings (PS). 60
4.2 Clustering accuracy (in percentage) of all users compared to the baseline accuracy (the accuracy based on blind guesses) by applying k-means using the settings from Table 4.1. 63

4.3 Results of scanning the developed SpyCon using signature-based malware detection packages. 64

4.4 Context-aware apps and their side-channel. 65

5.1 Timing analysis of VindiCo against increasing complexity of context-aware apps measured by number of adaptation tags in the registry file. 86

6.1 Comparison between the performance of fixed threshold FCW and multisample Q-learning in Sentio for 11 distracted drivers (A = Aggressive Driver, S = Assertive Driver, and D = Defensive Driver). Degradation in metric performance (with respect to Fixed policy) is marked in red. 119

ACKNOWLEDGMENTS

All praise be to Allah the High, "who teacheth by the pen, teacheth man that which he knew not.", Quran[96:4, 96:5]. I say what Prophet Solomon said: "$\cdots$ O my Lord! so order me that I may be grateful for Thy favours, which thou hast bestowed on me and on my parents, and that I may work the righteousness that will please Thee: And admit me, by Thy Grace, to the ranks of Thy righteous Servants.", Quran[27:19]

I wish to express my deep gratitude to my advisor Prof. Mani Srivastava for his guidance, his invaluable support, and important remarks on the developed results and the written manuscripts and his continuous encouragement. This dissertation would not have been the same without him. He has undoubtedly been the greatest intellectual influence in my life. I will always cherish the time we spent working together. Working in the Networked and Embedded Systems Lab allowed me to build a balanced view of both engineering fundamentals and practical implementation. I am also incredibly grateful to Prof. Puneet Gupta for the fantastic learning experience, the great ideas and the opportunities you have always provided me with especially under the umbrella of the multi-year and multi-institutional Variability Expedition project that was funded by the NSF.

I would like to thank Microsoft Research for supporting me through the Microsoft Research Ph.D. fellowship program (2016-2018). My work in this thesis would have not been completed without their support. I would also like to thank the the Mobility and Networking Research group at Microsoft Research especially Dr. Victor Bahl for giving me the opportunity to open new venues for my research and career paths through my internship in 2016.

I want to thank all co-authors and collaborators; Dr. Lucas Wanner(PhD'14), Dr. Mark Gottscho(PhD'17), Dr. Yasser Shoukry(PhD'15), Bo-Jhang Ho, Debbie Tsai, and Moustafa Alzantot. It has been an amazing privilege to be able to work and learn with such talented people. On top of my collaborators comes Dr. Lucas Wanner. Thank you for your support through the first years of developing this thesis. A special thanks to my former and current lab mates at NESL; Dr. Lucas Wanner(PhD'14), Dr. Supriyo Chakraborty(PhD'14),

Dr. Paul Martin(PhD'16), Dr. Haksoo Choi(PhD'15), Dr. Chenguang Shen(PhD'17), Dr. Bharathan Balaji(PhD'15), Bo-Jhang Ho, Moustafa Alzantot, Fatima Anwar, Renju Liu, Debbie Tsai, and my amazing husband Dr. Yasser Shoukry(PhD'15). You have been like a big family to me and an essential part of this experience. I would like to thank you all for the fun moments we had together. The same goes to my colleagues at NanoCad lab; Dr. Mark Gottscho(PhD'17) and Dr. Yasmine Badr(PhD'17).

This journey would not have been possible without the support of my friends. Thank you for staying close and keeping me rooted through the years of developing this thesis. Special thanks to Dina Elsissy, Hend Fares, Hawraa Salami, Mai Daftedar, Nada Abdalla, Sohaila Alanssary, and Dr. Yasmine Badr.

I would like to thank my parents (Nefessa Youssef and Hosni Elmalaki) for their unfailing help throughout my life. They offered me much advice and encouragement that was a great source of comfort. I hope I make you proud. I am grateful to my brothers (Ahmed Elmalaki and Mohamed Elmalaki) for always believing in me.

To my son Yahia and daughter Layla: Thank you for giving me the inspiration to achieve greatness. Thank you for giving me the joy of life.

Finally, To my beloved husband Yasser: Thank you for your patience, great support and encouragement during the most important stages of my life. Thank you for taking all responsibility of our life and our kids while providing me with full comfort and concentration in my work. I now have to repay you the countless nights and weekends spent in the working on this dissertation.

VITA

2003–2008	Egyptian Government Award for Excellence in Undergraduate Studies, Five years in a row, Ain Shams University, Cairo, Egypt.
2008	B.S. (Computer and Systems Engineering), Ain Shams University, Cairo, Egypt. Distinction Award, Graduation with Honors
2008–2011	R&D Engineer, Mentor Graphics, Egypt
2014	M.S. Electrical Engineering, UCLA
2015	Best Paper Award and Best Community Paper Award, MobiCom'15
2016	Nominated Best Teaching Assistant, Electrical and Computer Engineering Department, UCLA
2016	Grace Hopper (GHC) Scholar
2016	Ph.D. Intern, Microsoft Research
2016–2018	Microsoft Research Ph.D. Fellow

PUBLICATIONS

Salma Elmalaki, Bo-Jhang Ho, Moustafa Alzantot, Yasser Shoukry, and Mani Srivastava, "VindiCo: Privacy safeguard against context-aware spyware," *In preparation.*

Salma Elmalaki, Bo-Jhang Ho, Moustafa Alzantot, Yasser Shoukry, and Mani Srivastava, "SpyCon: Context-aware adaptation based spyware," *In preparation.*

Salma Elmalaki, Debbie Tsai, and Mani Srivastava, "Sentio: Driver-in-the-Loop Forward Collision Warning Using Multisample Reinforcement Learning," The 16th ACM Conference on Embedded Networked Sensor Systems (SenSys'18), Shenzhen, China, November 2018 (acceptance rate=15.6%).

Salma Elmalaki, Yasser Shoukry, and Mani Srivastava, "Internet of Personalized and Autonomous Things (IoPAT): Challenges, Architecture, and Applications," The 1st ACM International Workshop on Smart Cities and Fog Computing (CitiFog'18), Shenzhen, China, November 2018.

Salma Elmalaki, Lucas Wanner, and Mani Srivastava, "CAreDroid: Adaptation Framework for Android Context-Aware Applications," The 21st Annual International Conference on Mobile Computing and Networking (MobiCom'15), Paris, France, September 2015 (Best paper award) (Best community paper award) (acceptance rate=18%).

Salma Elmalaki, Mark Gottsho, Puneet Gupta, and Mani Srivastava, "A Case for Battery Charging-Aware Power Management and Deferrable Task Scheduling in Smartphones," USENIX 6th Workshop on Power-Aware Computing and Systems (HotPower'14), Colorado, USA, October 2014 (acceptance rate=34%).

Ankur Sharma, Joseph Sloan, **Salma Elmalaki**, Lucas Wanner, Mani B Srivastava, and Puneet Gupta, "Towards Analyzing and Improving Robustness of Software Applications to Intermittent and Permanent Faults in Hardware," ICCAD: International Conference on Computer Aided Design (ICCAD) 2013, pp:435 – 438, Asheville, USA, October 2013.

Lucas Wanner, **Salma Elmalaki**, Liangzhen Lai, Puneet Gupta, and Mani Srivastava, "VarEMU: An Emulation Testbed for Variability-Aware Software," Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Montreal, QC, Canada, September 2013 (acceptance rate=27.9%).

CHAPTER 1

Introduction

The age of autonomous systems that adapt to human behaviors has arrived. In these context-aware and pervasive systems, human reactions and behaviors are observed by edge devices and are used to adapt the behavior of the whole system. By continuously developing a cognition about the environment and the human state, and adapting accordingly, these systems can provide the user with a unique and personalized experience. Personalized systems can enable a multitude of applications in the context of smart buildings, smart cities, healthcare, and automotive systems by providing services that are not only adaptable to the human but can also intervene and maybe override human actions to ensure his safety and satisfaction.

While traditional autonomous systems interact with humans, in general, by collecting data directly from humans and their environment, a unique feature of personalized autonomous systems is their ability to assess human satisfaction and closing the loop by taking actions to adapt to the changes in his mood, needs, and expectations. This tight coupling between human behavior and computing promises a radical change in human life [Pic95].

Thanks to the recent advances in computational power, memory capacities, networking bandwidths, and battery capabilities, edge devices are now capable of performing machine learning algorithms to infer complex human states [SRN12, Muk15, SP13]. However, very few works in literature focused on assessing personalized needs and satisfaction to adapt to complex human states that vary between individuals and vary for the same individual across time. It is these challenges in building personalized and autonomous systems that we address in this thesis.

1.1 What is Pervasive Autonomy?

There is a long-standing desire to build practical pervasive computing systems that are both autonomous and personalized. The objective of these systems is to continuously develop a cognition about the environment and the human state and adapt accordingly to provide the user with a unique and personalized experience through intervening with autonomous actions. In other words, in these context-aware systems, the first step is extracting complex semantics from various sensory data to infer the state (or context) of both the human user along with her physical environment. The inferred context is then used by adaptation algorithms to take actions that change the behavior of the system and to match the user and environment state. Finally, automatically assessing the user satisfaction and changing the adaptation decisions accordingly.

1.2 Challenges

As we stand on the edge of an explosion of data from sensory devices, there has been a corresponding increase in system complexities and key challenges that need to be addressed. Building new context-awareness engines that extract complex human and environment states from high-bandwidth signals, inferring and predicting the human intent and satisfaction, understanding this human-computer interaction while taking into account power-limitation of these devices along with security and privacy concerns, are just examples to name few.

We highlighting several challenges that arise in the context of personalization of pervasive autonomy. In addition to the traditional challenges in pervasive systems which include power, memory, connectivity, and computational issues, personalized autonomous system faces the following challenges:

1.2.1 Adaptability

Personalized autonomous system needs to continuously monitor the satisfaction of the human user, and "learn" and "correct" the actions accordingly. The two types of variations namely

(i) *variations between individuals* and (ii) *variations within the same individual* are central themes in the adaptability challenge that need to be carefully addressed.

After adapting to human by deciding different actions, human users will respond to these actions. Similarly to the variations in human preferences, is the variations in the time for a human to respond to the actions taken by autonomous systems. While some users may react to every action taken by autonomous systems, others may react after the autonomous system has already performed multiple actions. To correctly learn the human preference, personalized autonomous systems must take into account this variation in the observed human response time.

A related challenge is concerned with the ability to fuse complex human states observed from different individuals. Thanks to the current advances in machine learning, several works in the literature reported significant breakthroughs in observing individual human states [OGB11, CSD15, HYT14]. However, very few works focused on the problem of fusing these individual human states to build complex states that represent the aggregate behavior of all humans interacting with the system [WGT11] .

1.2.2 Privacy

Unfortunately, the same act of adapting to the human context often leads to systems where increased sophistication comes at the expense of more privacy weaknesses. At the heart of personalized autonomy, privacy is the notion that actions taken—in according to information collected about human's state—poses a significant privacy risk on inferring user sensitive information. By simply monitoring the patterns of the actions taken, an adversarial eavesdropper may be able to reverse engineer these patterns to leak information about the human state.

Detecting the amount of information that personalized autonomous actions could leak is a challenging problem. Unlike traditional spyware where a priori information about its code signature or input-output behavior is known, this privacy leaking spyware is not identified yet. This opens the question of how to design a generic detection and mitigation algorithms

that can limit the amount of information that could be leaked using such spyware.

1.2.3 Context Engines Support

A pervasive autonomous system needs to continuously monitor the state of the user and adapt accordingly. This opens the question of how to design abstractions and support for context-aware systems. That is, just as file and socket abstractions help applications handle traditional input, output, and communication; a context-aware runtime system could help applications adapt according to user behavior and physical context. This challenge focuses on redesigning the computation stack to provide this support and allow developers to focus on how to build context-aware applications that interact and adapt to humans.

1.3 Our Contribution

The contribution of this thesis–to address the aforementioned challenges–is multi-fold. Specifically, we focused on how to **design machine-learning based systems to build adaptation and personalization services, design OS system support technologies for pervasive autonomous computing, along with addressing privacy concerns that arise as a consequence of personalized services**.

In particular, we propose an operating system support for personalized context-aware computing. We start by introducing **CAreDroid**, an adaptation framework for android context-aware applications. CAreDroid is a framework that is designed to decouple the application logic from the complex adaptation decisions in Android context-aware applications. In CAreDroid, context-aware methods are defined in application source code, the mapping of methods to context is defined in configuration files, and context-monitoring and method selection are performed by the runtime system. The CAreDroid framework is integrated as part of the Dalvik Virtual Machine (DVM) in Android OS. In particular, at runtime, CAreDroid intercepts the various sensor flows to determine the current context of the phone (where context parameters include energy, network connectivity, location, and user activity). CAreDroid uses this information along with the provided per-application configuration in

order to dynamically and transparently trigger adaptations and to find the set of methods that, at any point in time, better suit the device's context. Next, we focused on building an operating system mechanism that adapts to a context that is not largely explored named "battery charging pattern". We purpose **CAMPS**, a charging-aware power management adaptation for mobile operating system. While prior battery-aware systems research has focused on discharge power management in order to maximize the usable battery lifetime of a device, here we proposed that context-aware operating systems also need to carefully consider the process of battery charging. The power consumed by the system when plugged in can influence the rate of battery charging, and hence, the availability of the system to the user. Hence, understanding the human behavior and preference in charging his phone is essential. We show that there is potential for software schedulers to increase device availability by distributing tasks across the charging period.

The second contribution proposes a privacy firewall for personalized autonomous computing. While personalized computing opens the door for a multitude of applications while enhancing the user experience, they come with a cost. My research showed that context-aware systems open the door for side-channel to leak sensitive personal information. That is, while context-aware autonomous applications adapt their behavior based on the current context of the user, this very act of changing the behavior can be used by malicious software to reverse engineer the human context. We studied the extent to which a malicious app can monitor the adaptations triggered by authentic context-aware apps and extract user's information. In particular, we showed a concrete instantiation of a new category of spyware which we refer to as Context-Aware Adaptation Based Spyware (**SpyCon**). To circumvent this side-channel leakage, we proposed a novel OS software mechanism to detect and mitigate SpyCon apps which we called **VindiCo**. Being a new spyware, traditional spyware detection methods that are based on code signature or app behavior are not adequate to detect SpyCon. Therefore, VindiCo proposes a novel information-based detection technique and several mitigation strategies. We implemented VindiCo through extending the Android Open Source Project (AOSP) with a new layer that supports the purposed detection mechanism and mitigation policies.

Finally, the third contribution focuses on the personalization of pervasive systems. We start by targeting the advanced driver assistance systems (ADAS). We propose a Reinforcement Learning (RL) based algorithm called "multisample Q-learning" algorithm in the context of driver-in-the-loop (ADAS). We use the proposed multisample Q-learning to develop **Sentio**, an RL agent for a driver-in-the-loop Forward Collision Warning (FCW) system. This RL agent continuously monitors the state of the driver and the environment surrounding the vehicle to release the FCW early enough to match the driver attention level and preference (regarding the relative distance between the driver car and other cars). We implemented a proof-of-concept of the proposed Sentio system that demonstrates the feasibility of our algorithm. We evaluated Sentio on human drivers using a virtualized simulated environment. Afterwards, we propose an architecture for personalized and autonomous IoT systems that weaves personalization and context-awareness into the very fabric of IoT. We term this architecture as the Internet of Personalized and Autonomous Things or **IoPAT** for short By combining ideas from reinforcement learning and information theory, we show—using an example of smart and personalized home services—how the proposed IoPAT can adapt to human behaviors that are varying between individuals and vary, for the same individual, across time.

1.4 Organization Of This Dissertation

This dissertation touches three different aspects of pervasive autonomous systems.

Part 1- Operating System Support for Pervasive Autonomous Systems. This part presents **CAreDroid** in Chapter 1 and **CAMPS** in Chapter 2.

Part 2- Privacy Firewall for Personalized Autonomous Computing. This part presents **SpyCon** in Chapter 3 and **VindiCo** in Chapter 4.

Part 3- Personalization of Pervasive Autonomy. This part is presents **Sentio** in Chapter 5 and **IoPAT** in Chapter 6. We wrap up this dissertation by providing some conclusions and future directions of research in Chapter 7.

Part I

Operating System Support for Pervasive Autonomous Systems

CHAPTER 2

CAreDroid: Adaptation framework for context-aware mobile application

2.1 Introduction

Computation is becoming increasingly coupled with the physical world. Context-aware applications typically feature multiple interchangeable methods and sets of parameters, each of which is activated when the system is under a specific set of physical conditions. A music streaming application, for example, may request lower quality streams from a server when using a cellular network radio than when using WiFi. Social network applications may discover and promote interaction between users in close physical proximity. A video encoding application may delay or lower the quality of its processing to save energy when the system is running out of battery. When implementing context-aware applications, developers typically must probe sensors, derive a context from sensor information, and design an adaptation engine that activates different methods for different contexts. With adequate support from the runtime system, context monitoring could be performed efficiently in the background and adaptation could happen automatically [PZC14]. Application developers would then only be required to implement methods tailored to different contexts. Just as file and socket abstractions help applications handle traditional input, output, and communication; a context-aware runtime system could help applications adapt according to user behavior and physical context.

In this work we introduce CAreDroid, a framework for Android that makes context-aware applications *easier to develop* and *more efficient* by decoupling functionality, mapping, and monitoring and by integrating context adaptation into the runtime. In CAreDroid,

context-aware methods are defined in application source code, the mapping of methods to context is defined in configuration files, and context-monitoring, and method replacement are performed by the runtime system.

Because applications using CAreDroid do not need to monitor and handle changes in context directly, they can be written using significantly fewer lines of code than would be required if only using the standard Android APIs. Because CAreDroid introduces context-monitoring at the system level, it can avoid the indirection overhead of reading sensor data in the application layer, therefore making context-aware applications more efficient.

To allow for transparent switching between polymorphic implementations—which are alternative implementations of the same method that either provide same functionality with different performances or provide alternative functionality for the same method—the CAreDroid framework is integrated as part of the Dalvik Virtual Machine (DVM). In particular, at runtime, CAreDroid intercepts the various sensor flows in order to determine the current context of the phone (where context parameters include energy, network connectivity, location, and user activity). CAreDroid uses this information along with the provided per-application configuration in order to dynamically and transparently trigger adaptations and to find the set of methods that, at any point in time, better suit the device's context.

2.1.1 Related Work

A context-aware system requires three major elements: (1) a set of mutually replaceable polymorphic methods, (2) a context monitoring system, and (3) an adaptation engine that switches between different methods based on the monitored context. We divide the related work based on the three elements mentioned above.

2.1.1.1 Developing Context-Dependent Behavior

We can identify three main strategies in developing the context-dependent behaviors as follows.

- **Code partitioning:** Code partitioning for remote execution is based on the idea of cyber-foraging [Sat01] where mobile devices offload some of the work to a remote machine with more resources like a server [LLH13]. The server can then execute the heavy work on behalf of the mobile devices that have scarce energy. The idea of cyber foraging has been addressed in previous work with different aspects. Both Spectra and Chroma [FPS02, BSP03] do program partitioning and run part of the code on a surrogate server. They both rely on an earlier work called Odyssey [NSN97] that explored the idea of application adaptation based on network bandwidth and CPU load. Puppeteer [DWZ01] focusses on adaptation to limited bandwidth by making transcoding. Transcoding is a transformation of data to change the fidelity [NSN97] of the application to save energy.

- **Reflective techniques:** Reflection, originally noted by Smith [Smi82], is a technique that has emerged in computing languages to provide inspection and adaptation of the underlying virtual machine. Reflective techniques have been exploited in mobile computing middleware to address context change. Reflective mechanisms have been used by Capra et al. [CBM02] such that applications acquire information about the context, and then the middleware behavior and the underlined device configuration are tuned accordingly.

- **Alternate code paths:** Alternate code paths or algorithmic choice has been addressed in energy-aware software literature such as Petabricks [ACW09] and Eon [SKG07]. The choice between alternate algorithmic implementations of the same code is done dynamically based on the energy availability. Each code path has different energy consumption in a tradeoff with quality or the accuracy of the result. A code path that is chosen by a pre-determined battery life has been explored in [LMM07] in which tasks that have identical functionality are defined by developers. These tasks have different quality of service versus energy usage characteristics for embedded sensors application. In Green [BC10] a calibration phase is done at the beginning to determine the sampling rate—which eventually affects the accuracy of the result—in order to adapt to

the available energy. Algorithmic choice has been further used in software libraries to deliver the best performance based on the hardware configuration [FJ05, LGP07].

2.1.1.2 Context Monitoring

Efficient context monitoring has been studied throughly in literature. The work reported in [KLJ08, LYL10] provides frameworks for sensor-rich context monitoring. The main focus of that work is to minimize the energy consumption of the context-monitoring system. To further enhance the energy efficiency, Suman [Nat12] exploits the temporal correlation between contexts in order to infer some context from others without reading the actual sensor measurements. To avoid performance degradation due to minimizing the energy consumption in context monitoring frameworks, the work in [KSB13, CLL11] focuses on the optimization between continuous context monitoring, energy, latency and accuracy. Moreover, mobile operating systems currently support context monitoring functionalities.

For example, the recently added *getMostProbableActivity()* API can be used to return the result of the Android OS activity recognition engine (e.g. biking or walking). Other examples are the Geofencing APIs provided by both Andorid and iOS which allow listening to the entrance and exit events from particular places and therefore allow for location based context applications.

2.1.1.3 Adaptation Engine

Prior work related to adaptation engines can be classified into two broad categories (i) *application-oriented adaptation engines* and (ii) *operation system-oriented adaptation engines*. The work reported in [BGX10, ZGF11] is a representative work for the first class. In this work, an application specific adaptation engines are designed and implemented with specific focus on energy-aware context adaptation. The work in [ACK13, VC11, CKL11] lies in the second category, where adaptation engines are proposed to perform context-aware OS functionalities such as context-aware memory management, context-aware scheduling, ... etc.

2.1.2 CAreDroid Contribution

The work reported in this chapter can be categorized under the class of *application-oriented adaptation engines*. While there is a rich body of work in designing application-specific adaptation engines, a systematic support for context-awareness is still missing from contemporary mobile operating systems. The work in this chapter aims to fill this gap by providing OS support for the adaptation needed by context-aware Android applications.

In this chapter we discuss the design and implementation of CAreDroid and present application case studies demonstrating the effectiveness of the system. Technically, we make the following contributions:

- We design CAreDroid, a framework for the implementation of context-aware polymorphic methods and for the definition of application-specific rules used to map methods to contexts (Section 2.3);
- We extend the Dalvik Virtual Machine in the Android OS to provide adaptation support for context-aware application. The resulting CAreDroid can transparently switch between polymorphic versions of application methods at runtime (Section 2.5);
- We provide two levels of complexity of the mapping between contexts to methods, (i) a binary criteria (called must fit) and a relaxed criteria (called best fit) (Section 2.5.2);
- We demonstrate how application developers can leverage CAreDroid to make applications context-aware with minimal disruptions to the standard application development process (Section 6.6).

The reminder of this chapter is organized as follows: Section 2.2 introduces the system architecture of CAreDroid. Details of CAreDroid including its configuration, monitoring, and context adaptation algorithms are presented in Sections 2.3, 2.4, and 2.5 respectively. Section 6.6 shows the evaluation and case studies. Finally, we discuss some issues related to the design of CAreDroid and give conclusions in Sections 2.7 and 2.8, respectively.

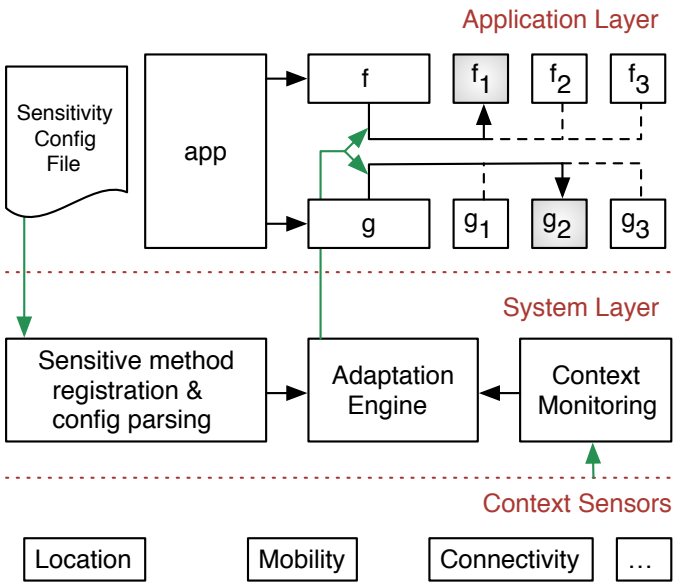


Figure 2.1: CAreDroid System Architecture. The developer provides a set of polymorphic methods and provides a configuration file describing how these methods shall be called. At runtime, CAreDroid monitors the phone context and adapts the application behavior accordingly.

2.2 System Architecture

The main objective of CAreDroid is to provide the application developer with support to easily build adaptation in context-aware applications. Hence, from a developer perspective, the design of CAreDroid needs to satisfy the following properties:

1. **Usability:** CAreDroid needs to add minimal overhead on the application developer at development time.

2. **Performance:** The adaptation engined needs to add minimal execution overhead when the application is running.

Motivated by these two design objectives, we designed CAreDroid as discussed in this section. A conceptual overview of the CAreDroid architecture is shown in Figure 2.1. Applications normally call polymorphic methods f and g. Each method is aliased to one of its

versions (f_1 and g_2, respectively, in the example). A sensitivity configuration file, defined on a per-application basis, describes rules that determine under what context each of the polymorphic versions should be used. For each version of a method, sensitivity rules define acceptable ranges of operation for different sensors of system context. Method f_1 could define, for example, two rules stating that WiFi connectivity and battery charging status should be equal to true, while f_2 could define one rule stating that remaining battery capacity should be between 0% and 20%. Rules are assigned priorities that help determine which of the versions should be used when multiple rules are valid.

In the system layer, CAreDroid parses the application configuration file to discover adaptable methods and their rules of operation. A context-monitoring module abstracts the various sensors in the system, and exposes context information to an adaptation engine. When changes in context occur, the adaptation engine changes the aliasing of the adaptable methods according to the sensitivity rules. If more than one version of a method matches the current context, the priorities of the sensitivity rules are used to choose between them. When there are no alternatives of a method that exactly matches the context, CAreDroid chooses the version that most closely conforms to the current state of the device. CAreDroid is organized in three modules:

2.2.0.1 Context Sensitivity Configuration File

For each context-aware application, a sensitivity configuration file maps methods to contexts. The file is structured as a series of *sensitive methods* and their respective context *sensitivity lists* described in XML format. In keeping with our goal of decreasing development complexity for context-aware applications, the file is a straightforward description of acceptable ranges of operation for each method under different contexts. A detailed description of the CAreDroid configuration file is presented in Section 2.3.

2.2.0.2 Context Monitor

CAreDroid has a dedicated module that continuously probes the current phone context. CAreDroid supports both *raw contexts* that can be directly known by reading the state of the hardware (e.g. WiFi connectivity, battery level) as well as higher level *inferred contexts* such as mobility status (e.g., walking, running) that require advanced processing of sensor information. As mentioned in Section 2.1.1.2, the design of context monitoring systems is a well studied topic. The main work in this chapter does *not* focus on efficient implementation of context monitoring system. However, context monitoring is yet an essential part in order to evaluate any adaptation engine. Hence, in Section 2.4 we describe a simplistic implementation of context monitoring which can be augmented by any of the previous proposed context monitoring algorithms.

2.2.0.3 Adaptation Engine

In order to choose the correct polymorphic implementation that best suits the current context, CAreDroid uses the data supplied by the developer in the configuration file. Alternative implementations of sensitive methods are connected together through a replacement map that lists all *candidates methods* that can be used for a sensitive call. Whenever more than one candidate implementation fits the current context, CAreDroid uses a conflict resolution mechanism to pick the implementation with the highest priority.

Because it frees developers from having to implement adaptation strategies in the application layer, the CAreDroid adaptation engine is the main factor in meeting our goal of decreasing development complexity for context-aware applications.

Section 2.5 shows how context-to-method matching and conflict detection are implemented efficiently to meet our goal of reducing runtime overhead.

2.3 Sensitivity Configuration File

The configuration file is an XML file that is supplied by the application developer. To fit in the Android flow, the configuration file is stored as an asset file packed with the application package file (APK). In this section, we describe the structure of this XML file along with the post-processing steps performed by CAreDroid over this file.

2.3.1 Configuration File Structure

For each *sensitive method*, the developer provides different polymorphic implementations. Each polymorphic implementation of a method is described by a name, a tag, and a priority. The name corresponds to the method name in source code. The tag associates different implementations of a method with one another. For example, if methods f_1 and f_2 are polymorphic implementations of the same method, then both of them must be associated with the same tag, for example f. Finally the priority for a method helps the system resolve ambiguities when multiple versions of a method satisfy the current context.CAreDroid

For each polymorphic implementation, the developer assigns a *sensitivity list*. This *sensitivity list* is the list of context states for which this polymorphic implementation shall be triggered. In our current implementation of CAreDroid, we focus on four context categories:

- **Battery state:** In this category, we define three contexts which are (1) the remaining battery capacity ($0\% - 100\%$) (2) the battery temperature ($-30°$ C – $100°$ C) which is an indicative of high battery load as well as elevated power consumption; and (3) operating battery voltage, which is an indicative of the battery health.
- **Connectivity state:** In this category, we define three contexts: (1) WiFi connection status (On - Off), (2) WiFi link quality ($0 - 70$ A/V°), and (3) RSSI Received signal strength indication ($0 - 4$).
- **Location:** In this category, we consider one context state, which is GPS location. In this state, the developer is allowed to provide the latitude and longitude coordinates of a square area.

```
<Method>
  <MethodName>AdjustCameraPowerAware
  </MethodName>
  <priority>1</priority>
  <tag>cameraAdjust</tag>
  <batterycapacity>
    <vstart>0</vstart>
    <vend>25</vend>
  </batterycapacity>
</Method>
<Method>
  <MethodName>AdjustCameraWhileRunning
  </MethodName>
  <priority>2</priority>
  <tag>cameraAdjust</tag>
  <batterycapacity>
    <vstart>20</vstart>
    <vend>100</vend>
  </batterycapacity>
   <mobility>run</mobility>
</Method>
```

Figure 2.2: Snippet of a CAreDroid configuration file.

- **Mobility state:** In this category, we consider only the current mobility state of the phone holder, which can take one of the following values: still, walking, running and driving.

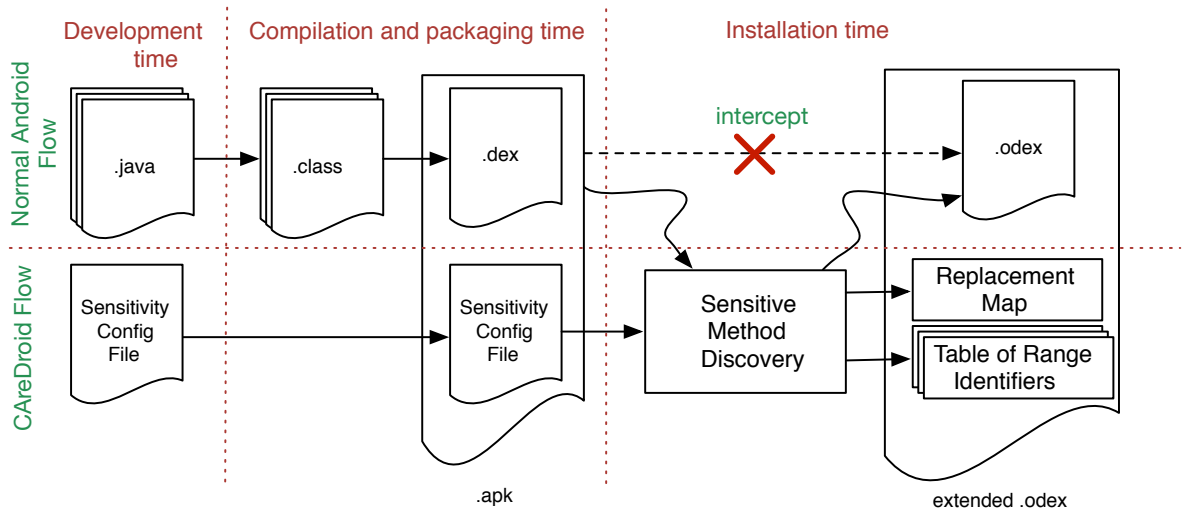


Figure 2.3: CAreDroid's extended installation flow. CAreDroid intercepts the installation process of the app on the device in order to parse the sensitivity configuration file. The final outcome of this process is two data structures named "Replacement Map" and "Table of Range Identifiers."

2.3.1.1 Example

To illustrate the construction of the configuration file, we provide a small example in Figure 2.2. In this example, we have two polymorphic methods for adjusting the camera parameters under different contexts. One method, `AdjustCameraPowerAware`, is designed to save energy. Hence, its `BatteryCapacity` range is from 0% up to 25%, and it can execute whether `wifi` is on or off. The second method is dedicated to adjusting the camera while the user of the device is running. For example, this method should adjust the focus and the scene parameters of the camera to give a better quality image. Accordingly, the `mobility` is assigned to be *run*.

2.3.2 Configuration File Processing

After the developer supplies the CAreDroid configuration file, several post-processing steps are required at the installation time of the application. In particular, the XML file needs to be parsed, and the extracted information is then used to fill specific data structures. Figure

2.3 shows how CAreDroid flow extends the normal Android compilation and installation flow. This flow diagram shows the steps needed to post-process the configuration file. Parsing of the configuration file, discovery of sensitive methods, and registration of adaptation parameters with the adaptation engine is implemented in the Dalvik Virtual Machine (DVM), as described in the remainder of this section.

2.3.2.1 Sensitive Method Discovery

Upon compilation of the Java code, the generated Dalvik Executable File (DEX) contains all compiled bytecodes of methods stacked on top of each other. A call to a method is then accomplished by pointing to the offset of the first instruction inside the DEX file. For example, let us consider a call to the `myObject.foo()` method. The following bytecode:

```
invoke-virtual {v14}, [method@101e]
```

is used, where `v14` is the reference to the object instantiated from the class `myObject`, and `0x101e` is the offset of the first instruction in `myObject.foo()` in the DEX file. Note that the textual name of the method (e.g. "foo") is still preserved in the generated DEX file and the association between the method name and the method offset can still be extracted.

The DEX file along with all asset files (including the CAreDroid configuration file) are then packaged in the application package file (APK). When the APK file is installed, Android creates a new virtual machine to host the application. During this process, the Android flow extracts the DEX file and post-processes it in order to generate the Optimized DEX (ODEX) file.

The DEX optimization consists of two main steps. The first step is executed while class loading takes place. During this step, each method is assigned with a local method ID (compared to the global method ID assigned in the DEX). The second step of the DEX optimization process takes place when object references are linked with their classes. In this step, inheritance, polymorphism, method overriding, and method overloading are resolved. In particular, a virtual table is generated for each class. Each resolved method corresponds to an entry in this virtual table. Therefore, each method is now identified with its unique

index inside it class virtual table. As a result, the call to the `myObject.foo()` method is further translated into:

```
invoke-virtual-quick {v14}, [000c]
```

where `v14` is the reference to the class object, `myObject`, and `000c` is the index for method `foo` inside that class's virtual table. Note that the association of the method name with its index in the virtual table is no longer preserved in the ODEX file.

Switching between different polymorphic implementations is equivalent to intercepting the operation of the bytecode corresponding to `invoke-virtual-quick` and supplying a different method ID. To perform this operation, CAreDroid must be able to keep track of the method IDs and relate them back to the IDs of different polymorphic implementations. Therefore, CAreDroid modifies the DEX/ODEX build process in Android to add hooks for context-awareness in sensitive method calls. This is accomplished through a *table of range identifiers* and a *replacement map*. This process is shown in Figure 2.3.

2.3.2.2 Replacement Map (RM) and Table of Range Identifiers (TRI)

The "Replacement Map" (RM) is a collection of (key, value) pairs defined for each polymorphic method.
The purpose of this map is to link each of the multiple polymorphic alternatives with their *sensitivity lists*.

The *key* of this map is a composite key that consists of the pair of class-id and method-id extracted initially from the DEX file. The *value* field of the RM is an array whose length is equal to the number of contexts (mobility, location, battery capacity, etc.). This array specifies the *sensitive* operation range for this method for all different contexts. To facilitate this association, we introduce another data structure called the "Table of Range Identifiers" (TRI).

The TRI consists of multiple *associative arrays*. For each of the context sensors, we create a corresponding associative array. To construct such an array, we extract all the operation ranges provided in the configuration file, and associate a uniquely generated *operation range*

identifier (ORI) to each of the operation ranges. An example for such an associative array for battery capacity is shown in Table 2.1. Since the ranges of operations can vary from one class to another (based on the developer's intent , as described by the configuration file), we generate a TRI per class per context. The association between the class and the corresponding set of TRIs is made after the optimization of the DEX file process takes place. Once all TRI tables are built, we connect them to the RM by copying the corresponding ORI from the TRI data structure. An example of the RM is shown in Table 2.1.

At runtime, CAreDroid uses the TRI along with the current context to retrieve all ORIs that satisfy the current context. These ORIs are then used as inputs to the RM to retrieve the corresponding method IDs. If more than one method matches the ORIs, a conflict is discovered and needs to be resolved as described in Section 2.5.

2.3.2.3 ODEX Extension

After CAreDroid constructs all the TRIs and the RM data structures, the DEX file passes through the normal Android optimization process, resulting in the generation of the ODEX file. Since the optimization process of the DEX file can result in a change in the method IDs, CAreDroid intercepts the process of optimizing the DEX files in order to update the RM, as shown in Table 2.1.

Finally, we extend the ODEX file structure by adding a reference to the RM data structure, which is generated by the described process. We extend the internal Android *class* data structure in order to associate the corresponding TRIs generated for that particular class. We also extended the internal Android *object* and *method* data structures by adding *sensitivity flags*. These flags are used later by the **CAreDroid Adaptation Engine** to facilitate the method switching.

2.3.3 Online Change of Context Ranges

While the configuration file needs to be specified by the developer at development time, the sensitive values of some sensitive contexts may not known until the code is running on

B-Range	ORI
0→100	1
20→30	2
10→20	3
30→100	4

S-Range	ORI
0→2	1
1→4	2
2→3	3
0→3	4

...

Class ID	Method ID	B	T	V	W	Q	S	M	L
0x01	0x00F	1	2	1	2	4	2	1	4
0x01	0x01E	2	3	4	2	3	3	2	2
0x02	0x02A	2	2	1	0	0	0	2	1
0x02	0x01F	2	2	1	0	0	0	8	3

Table 2.1: On the top, examples of TRI tables for Battery capacity and Signal strength (RSSI) contexts. Each TRI associates a unique *Operation Range Identifier* (ORI) to each record. For each class, CAreDroid creates TRI for all different contexts. The association between the TRIs and the class is done later, after the optimization of the DEX files takes place. On the bottom, a Replacement Map that associates each key = (class-id, method-id) with its corresponding ORIs. CAreDroid creates a unique RM for the application. — *legend:* B: Battery Capacity, T: Battery Temperature, V: Battery Voltage, W: WiFi connectivity, S: Signal strength, Q: Signal Quality M: Mobility L: Location.

the phone. For example, an application that changes its behavior whether the user is at *home* or at *work*. The location information (longitude and latitude of home and work) is not known at development time. Accordingly, CAreDroid supports online modification of the values associated with each sensitivity context. This takes place by asking the developer to write a specific file to the file system. CAreDroid parses this file whenever appropriate and re-updates the TOI accordingly. Note, that CAreDroid allows only changing the values associated with each sensitivity list but not the number of sensitive contexts associated to a polymorphic implementation.

2.4 CAreDroid context monitoring

In this section, we describe how CAreDroid acquires the current context at runtime with less overhead than Android Java APIs. Phone contexts can be numerous, and include raw values (like accelerometer data, GPS longitude and latitude, remaining battery capacity, etc.), or inferred states (like user mobility). While the in contribution of this work is *not* efficient implementation of a context monitoring system, this is an essential part of any adaptation engine. In this section, we give details on how CAreDroid acquires both raw and inferred phone contexts. The work in this section can be indeed complemented by any of the context monitoring systems that appeared currently in the literature.

2.4.1 Raw Context Monitoring

Android exposes sensor information to the software stack through a Hardware Abstraction Layer (HAL). The HAL features a set of sensor managers that work as an intermediate layer between the low-level drivers and the high-level applications.

In order to reduce the overhead, we need to bypass the HAL layer and the associated sensor managers. This can be done by snooping on the interface between the HAL and the low-level device drivers through the `sysfs` virtual file system. In particular, each sensor (e.g. accelerometer, battery sensors, and WiFi) device driver exports its data into a set of files located under `/sys/class/`. In our work, we create an internal Dalvik VM thread that continuously reads these files to determine the state of the battery sensors and WiFi availability. The WiFi signal quality and signal strength are monitored via reading `/proc/net/wireless`. Similarly, the GPS location is determined by snooping over the Android Binder that connects the `Android LocationManager` with the GPS hardware driver.

2.4.2 Inferred Context: Mobility State

Mobility state detection is calculated by processing the raw accelerometer data obtained by the internal VM thread described above. In order to infer the mobility state, we adapt the classification procedure described in [RBE08, RLR09] to detect whether the user is stationary, walking, or running. This classifier is based on the Geortzel algorithm [Goe58]. Finally, to reduce the computational delay due to running the mobility state classifier, we let the classifier run on a separate DVM internal thread.

2.5 CAreDroid Adaptation Engine

The adaptation Engine is the core of CAreDroid. It is where the method replacement happens at runtime. In this section, we explain how CAreDroid extends the execution phase of the Android flow to automatically and transparently switch between methods.

2.5.1 Dalvik Interpreter Extension

Recall that the developed application is compiled and translated into an ODEX file. Bytecode stored in the ODEX file is then interpreted at runtime. In particular, the Dalvik Virtual Machine (DVM) runtime utilizes two types of interpreters. The first is called the *portable* interpreter, which is implemented in C code and is not specific to a certain platform architecture. The second interpreter is called the *fast* interpreter which is implemented in assembly language and tailored towards a specific platform. The DVM supports switching between the two interpreters at runtime.

In our framework, we extend the *portable* interpreter to support the CAreDroid runtime engine. The extended interpreter checks the current interpreted ODEX bytecode. Whenever the bytecode corresponding to the `invoke-virtual` instruction is detected, CAreDroid intercepts the execution of the interpreter. It then checks the arguments of the `invoke-virtual` instruction — the method ID and the class ID — against the *sensitivity flags* in the extended ODEX file, described in Section 2.3.2.3. If the *sensitivity flag* is set,

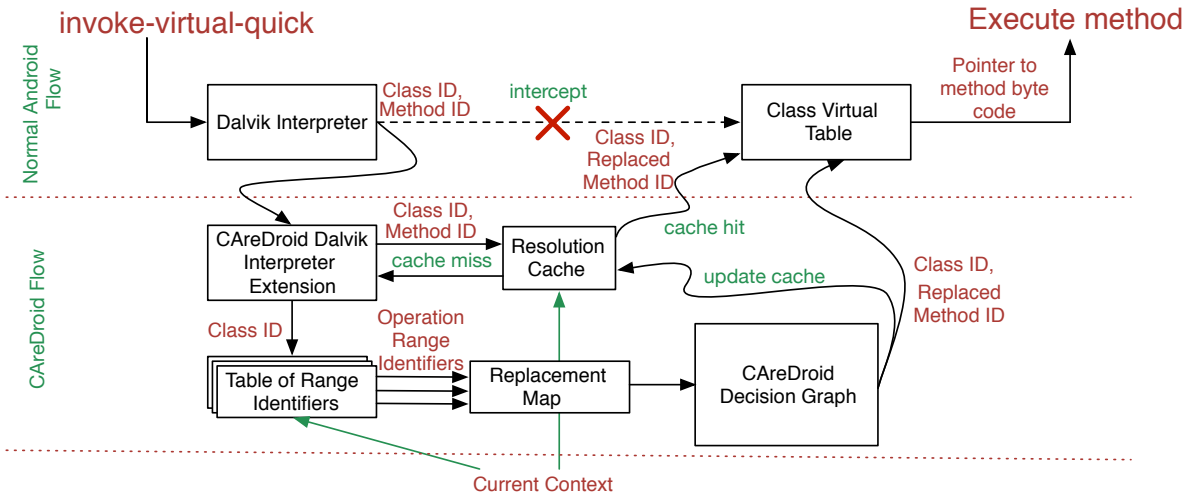


Figure 2.4: Flow of CAreDroid Adaptation Engine at runtime. The CAreDroid extended interpreter intercepts the execution of the Dalvik opcode *invoke-virtual-quick* to check whether the method invoked is sensitive or not. If the method is sensitive, then the CAreDroid adaptation engine checks the current context and picks the correct polymorphic method. This process is done through leveraging the information in the TRI and RM data structures along with the conflict resolution mechanism implemented using the decision graph. Finally, to speed up the process, CAreDroid uses a resolution cache, which exploits the temporal locality of the context.

then CAreDroid needs to pick the polymorphic method that best fits the current context.

The process of choosing the best polymorphic implementation needs to resolve the conflicts in the user configuration. This is done through the CAreDroid decision graph module (discussed later) along with the TRI and RM data structures. In order to accelerate the process of picking the correct polymorphic implementation, CAreDroid uses a *resolution cache* that exploits the temporal locality of the adaptation decisions. This process is shown in Figure 2.4 and illustrated in the CAreDroid *decision graph* and the *resolution cache* in the following subsections.

Note that the *portable* interpreter (where CAreDroid is running)has a negative effect on the execution time of the application. To address this issue, we switch between the *fast*

interpreter and the *portable* interpreter at runtime. The execution starts normally with the *fast* interpreter and, when the interpreter hits an invocation of a sensitive class, the interpreter switches to the *portable* interpreter and the CAreDroid adaptation process takes place. After executing the sensitive method, the interpreter switches back to the *fast* version.

2.5.2 Which Polymorphic Implementation to Pick?

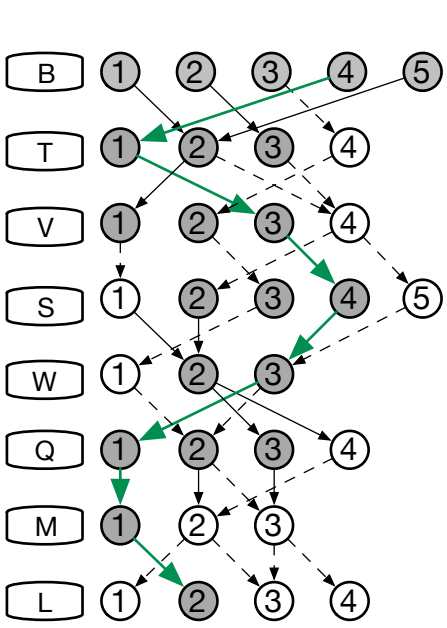
In order to choose the correct implementation that best suits the current context, our framework utilizes the data supplied by the developer in the configuration file. Note that it is possible that, for a given context, multiple methods are valid *candidates*, leading to a conflict that needs to be resolved.

2.5.2.1 Best Fit vs Must Fit

The first step is to choose a set of *candidate* methods. We allow for two policies. In the first policy, *must fit*, a method is considered a valid *candidate* if the current context satisfies *all* the operation ranges for all sensitive contexts. In the second policy, *best fit*, a method is a valid candidate if the current context satisfies *at least one* operation range of the sensitive contexts. The choice of policy is determined by the configuration file.

2.5.2.2 Decision Graph

We use a Directed Acyclic Graph (DAG) to choose the candidate method. Each level of the graph marks one sensitive context (e.g. battery capacity, mobility state). The sensitive contexts are ordered based on their priority as defined in the configuration file. For each sensitive context, we create nodes for all *operation range identifiers* (ORI)—previously discussed in Section 4— that appear in the *replacement map* (RM) data structure. In other words, to build the decision tree, we traverse the RM horizontally. For each row of the RM, we create nodes corresponding to all distinctive ORIs in that row. This process is repeated for all the rows in the RM. An example of an RM and the associated decision graph is shown in Figure 2.5.



M1	M2	M3	M4	M5
1	2	3	4	5
2	3	4	1	2
1	4	2	3	4
1	2	3	4	5
2	2	1	3	3
4	3	2	1	2
2	3	3	1	2
1	4	3	2	3

Figure 2.5: An example of a Replacement Map(RM) (right) and its associated decision graph (left). The nodes at each level correspond to the ORIs in the same level of the associated RM. The edges in the decision graph correspond to the five methods shown in the RM. The shaded nodes correspond to the ORIs that match the current context. The solid arrows correspond to the active paths that match both the RM and the current context. Finally the path marked in green corresponds to the method that satisfies all the ORIs, and therefore this method is the one picked by CAreDroid for execution.

The methods contained in the RM columns dictate the decision graph topology. Accordingly, we traverse the RM vertically and connect the ORIs that correspond to the same method by edges. This is shown for the same example, in Figure 2.5.

When the phone context is reported, we use the TRIs in order to know which ORIs are active, i.e., which ORIs match the current context. The next step is to use this information to eliminate some choices in the decision graph. For example, in Figure 2.5, we mark the active ORIs with a gray color and the corresponding active edges with solid arrows.

The final step is to compare the available active paths that start from the top level. In

the *must fit* policy, CAreDroid considers only the active paths that connect the first level all the way to the lower level. If no such path exists, then no method replacement is going to take place. On the other hand, the *best fit* policy considers the longest path that starts from first level. Referring to the example in Figure 2.5, only one active path is passing through all the DAG levels and corresponds to method M4. Therefore, CAreDroid picks this method for execution. The Class ID and Method ID of this method is reported back to the normal Android flow to be executed. If further conflict exits, we use the method priority reported in the configuration file.

2.5.3 Conflict Resolution Cache

While the adaptation strategy for CAreDroid is fairly straightforward, performing it for every individual sensitive method call in the system would incur in an unnecessary overhead. In order to decrease the overhead of the context-to-method resolution mechanism, our framework uses a resolution cache. Our heuristic assumes that the operating point does not change over short time periods. Therefore, if a method is called multiple times within a short amount of time (inside a loop for example), the same polymorphic implementation might be used for all of these calls. The cache is used to store the recently resolved candidates, that is, the recent phone context along the method ID that is chosen for each phone context. Each entry corresponds to the eight values of the phone context along with the method ID for the optimal method. The cache uses a Least Recently Used (LRU) approach to replace entries.

2.6 Evaluation

We evaluate CAreDroid with four case studies. In the first one, we focus on assessing two metrics namely, reducing the number of significant lines of code and the execution time of the context-aware application. In the remaining three case studies, we show examples of applications that can benefit from context adaptation using CAreDroid.

All case studies are carried over a Nexus 4 phone running a modified system image for platform 4.2 API 17 [aos]. The execution time is obtained using the Android SDK tracer

[And]. The size of the original system image for Android 4.2 is 234.368 MB, the modified system image that support CAreDroid is 245.26 MB. Hence, the overhead in the system image is 4.6%.

2.6.1 Case Study 1: A Simple Application

Platform	SLOC	% Increase
Non-context aware (Base)	275	-
Context-aware (Pure Java)	606	120%
CAreDroid	275 +78[a]	28.3%

[a]SLOC of the XML Configuration file

Table 2.2: significant line of code (SLOC) results for case study 1 showing the SLOC for different implementations along with the percentage increase of SLOC relative to the non–context aware implementation.

In this case study, we implement a simple application that has only one sensitive method with three polymorphic implementations. In particular, this simple application implements a numerical solver for linear equations (which is a cornerstone algorithm in many image processing algorithms used to enhance photos before posting them to social media applications). We implement three polymorphic variants of this solver named LUP-decomposition (LU), Cholesky decomposition (CHD), and Conjugate Gradient (CG). These three methods have different memory and computation time characteristics. These mathematical functions are exhaustively used in image processing applications. Each implementation corresponds to a particular tradeoff between performance and computation time. In particular, CHD gives the most accurate results while suffers from high computation time. On the other extreme, LU gives the least accurate results (compared to CHD and CG) while leads to better computation time. The purpose of this case study is to characterize the performance of CAreDroid while switching between these three polymorphic implementations.

Platform	Solver	CPU time (ms)						Overhead	
		Method time	Decision Tree		Context Monitoring (parallel) thread)	Total			
			without cache	with cache		without cache	with cache	without cache	with cache
Non-context aware(Base)	LU	8.322	-		-	8.322		-	
	CHD	16.872	-		-	16.872		-	
	CG	13.375	-		-	13.375		-	
Context aware (Pure Java)	LU	8.322	0.227		5.093	13.642		63.92%	
	CHD	16.872	0.776		5.093	25.741		52.56%	
	CG	13.375	0.351		5.093	18.819		40.70%	
CAreDroid (Must Fit)	LU	8.322	0.183	0.030	0.336	8.841	8.688	6.23%	4.39%
	CHD	16.872	0.335	0.031	0.336	17.543	17.239	3.98%	2.17%
	CG	13.375	0.198	0.030	0.336	13.909	13.741	3.99%	2.736%
CAreDroid (Best Fit)	LU	8.322	0.183	0.031	0.336	8.841	8.689	6.23%	4.41%
	CHD	16.872	0.732	0.031	0.336	17.635	17.239	4.522%	2.17%
	CG	13.375	0.489	0.030	0.336	14.2	13.741	6.17%	2.73%

Table 2.3: Execution time results for case study 1 showing the profiling of different parts for all the three implementations. The overhead is computed relative to the non context-aware implementation. The results show the efficiency of both the must fit and best fit policies. It also shows the performance increase resulting from using the cache.

In order to characterize the CAreDroid performance, we generate an arbitrary configuration file that assigns each of the three solvers to different battery and connectivity contexts. We evaluate CAreDroid against a pure Java implementation performing the same functionality. That is, the pure Java application listens to changes in battery and WiFi connectivity using the standard HAL callback mechanism provided by the Android APIs. We implement a *non-context aware* implementation that *magically* knows which polymorphic method shall be called without knowing the context (for the purpose of comparison) and we call it the

base non-context aware.

2.6.1.1 Reduction in Significant Line of Code (SLOC)

In this example, using CAreDroid reduces the SLOC for the application by a factor of 2x compared to a Java implementation using standard Android APIs. Table 2.2 shows the SLOC for each of the implementations.

2.6.1.2 Reduction in Execution Time

In this test case, we let the 4 different implementations (non context aware, pure Java, must fit and best fit) run over the phone for several hours while collecting profiling information. The profiling information are then averaged out and the result is reported in Table 2.3. To further investigate the effect of the resolution cache, we run the test with and without the cache functionality to allow for comparison. The results in Table 2.3 show that CAreDroid reduces the CPU time overhead (compared to the pure Java implementation) by a factor of 12x, on average, while adding a minimal overhead (2.5%–4.4%) compared to the non context-aware case. Furthermore, the resolution cache leads to decreasing the decision tree time by at least an order of magnitude whenever there is no change in the operating point.

Finally, with no cache (or alternatively when a cache miss occurs) *best fit* policy adds slightly more overhead compared to the *must fit* policy due to the complexity of the decision graph used by the former. The same order of overhead also appears in the pure Java implementation because of the added code for switching between contexts.

2.6.1.3 Energy Profiling

Finally, we characterize the energy consumption (and hence the battery life time) due to context monitoring and adaptation. In this experiment, we monitor the voltage and discharging current of the battery during 2.5 hours while running the application under the four platforms (best fit, must fit, pure Java, and non-context aware). The experiment starts at the same battery capacity and at room temperature. We run each experiment three times.

In the first one, we deactivate the context switching functionalities and focus only on the energy consumed by the context monitoring. These results are reported in Figure 2.6(a). In the second run, we run both context monitoring as well as context switching but calling an empty method. The energy measurements are then subtracted from the energy measurements from the previous experiment. The purpose of this experiment is to profile the effect of the decision tree and the context switching mechanism. This is shown in Figure 2.6(b). Finally, we run the full implementation to get the overall energy consumption of our system and compare it to the non-context aware one.

Overall, the results show that bypassing the HAL layer and performing the context monitoring inside the OS lead to decreasing the energy consumed by a factor of 36%. The results also show a similar decrease of energy consumption due to implementing the context switching inside the OS with a slight difference between the must fit and the best fit switching policies. Also, as seen in Figure 2.6(c), the energy consumption of pure Java implementation consumes around 69.33% more energy compared to CAreDroid. This energy consumption can be further improved by using energy-aware context monitoring techniques that previously reported in the related work (Section 2.1.1.2).

2.6.2 Case Study 2: A Context-Aware Phone Configuration

With increasing reported accidents resulting from texting while driving, we develop an application that changes the phone configuration based on the underlying context of the phone[1]. We manifest the **location**, **mobility state**, and **battery** in this application. In particular, we would like the application to change the phone configuration as follows:

- **Default**: keep the phone in its default configuration.
- **Driving**: (1) disable messaging and email notifications, (2) block certain caller num-

[1]Some applications in the market attempt to control the phone configuration like Tasker [tas] and Locale [loc] by providing hooks to the user to configure the phone based on certain rules that the user defines. However, these applications have only boolean decision. The rules must be all satisfied in order to change the configuration, while CAreDroid provides more complex formula (the best-fit policy). Moreover, Tasker and Locale do not support all the contexts supported by CAreDroid.

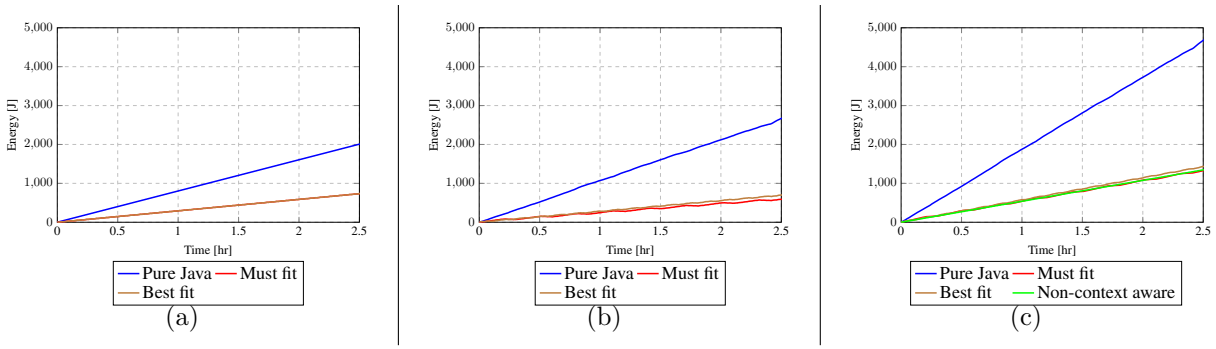


Figure 2.6: Energy consumption results for case study 1 showing (a) energy used when context monitoring is running alone (b) energy consumed by the context switching subsystem and (c) the total energy consumed. On each case, we plot the energy consumed by the pure Java implementation as well as the must fit and best fit implementations of CAreDroid. The results in (a) show that bypassing the HAL layer and implementing the context monitoring inside the OS allowed CAreDroid to use 36% less energy within the 2.5 hours lapse of the experiment. The results in (b) show that both Must fit and Best fit adaptation significant outperform the pure Java implementation in terms of energy consumed (and hence battery lifetime). The overall results (c) show that CAreDroid consumes only 6.73% energy compared with the non-context aware implementation and provides 69.33% energy saving compared to the pure Java implementation.

bers specified by a list (i.e. forward calls from this list to the voice mail) and (3) enable bluetooth (to connect the phone to car speaker).

- **Running**: (1) enable GPS (if not enabled), (2) block certain caller numbers specified by a list, and (3) mute the alarms.
- **At home**: (1) enable WiFi, (2) block certain caller numbers specified by a list, (3) raise the alarm volume, and (4) set ringer volume to normal.
- **At work**: (1) enable WiFi, (2) lower the alarm volume, (3) put the phone in vibrating mode, and (4) block certain list of caller numbers.
- **Power saving**: (1) lower the ringer volume, (2) disable bluetooth (if enabled), and (3) enable the automatic adjustment of screen brightness.

Platform	SLOC	% Increase
Non-context aware (Base)	282	-
Context-aware (Pure Java)	873	201%
CAreDroid	282 +277[a]	98.2%

[a]SLOC of the XML Configuration file

Table 2.4: Significant lines of code (SLOC) results for case study 2 showing the SLOC for the three implementations along with the percentage increase of SLOC relative to the non-context aware implementation.

Platform	CPU time[a]	Overhead
Non-context aware (Base)	1.942	-
Context aware (Pure Java)	12.14	525.12%
CAreDroid (Best Fit)	2.015	3.76%

[a]CPU Time (ms) = Method time + HAL Callback time + Inferences

Table 2.5: Execution time results for case study 2 showing the overhead for the different implementations.

For each of these configurations, a polymorphic method is implemented. The objective is to call the correct method based on the context. Similar to the previous case study, we implemented a non-context aware implementation (for the purpose of comparison), a context-aware implementation using the normal Android flow, and a context-aware implementation using CAreDroid.

2.6.2.1 Reduction in Significant Line of Code (SLOC)

CAreDroid decreases the code complexity (quantified by SLOC) by a factor of 2× (including the SLOC of the XML configuration file) compared to the implementation based on the

normal Android flow, as shown in Table 2.4.

2.6.2.2 Reduction in Execution Time

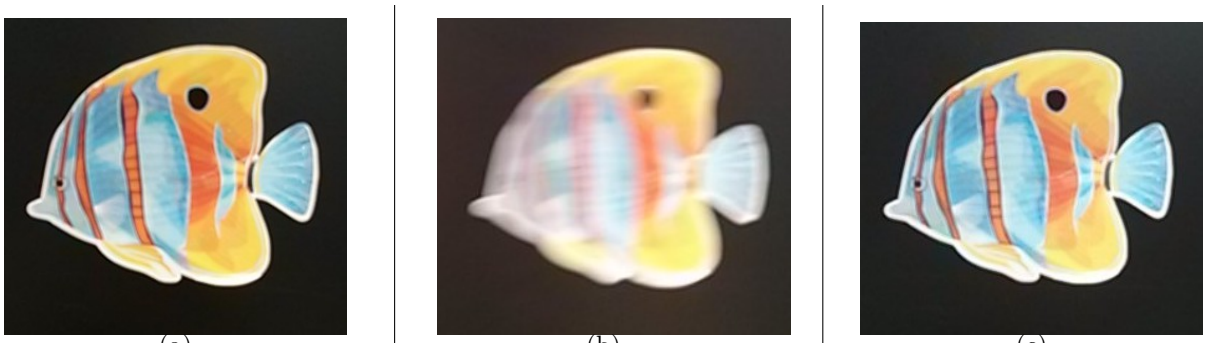
In this test case, CAreDroid is configured and the phone is allowed to change between different contexts leading to a change in the application behavior. The time profiling is done across different contexts and the one with maximum CPU overhead is reported in Table 2.5.

For this case study, the pure Java implementation adds a 525.12% CPU overhead. On the other hand, CAreDroid introduces a minimal overhead of 3.76% compared to the non-context aware implementation. The large overhead of the former can be explained by observing that there are 16 possible cases that need to be handled if the application developer were to implement the same app without using CAreDroid. Needless to say that the developer—without CAreDroid—has to implement all the Android listeners to all contexts as well as provide the high-level inferences of mobility state from the raw data. The small overhead in CAreDroid compared to the pure Java implementations again can be accounted to the fact that all the context-awareness operations (context monitoring and adaptation) are implemented natively inside the operating system.

2.6.3 Case Study 3: Context-Aware Camera

In this case study, we build a context-aware camera application. The camera adjusts its features parameters based on the phone context. We have five different methods that CAreDroid alternates between. The focus of this study is on making the *focus*, *scene mode* and *flash mode* adaptive to the context. However, this can be extended to handle all the camera features. The five modes are listed as follows:

- **Default**: Configure the focus mode to "default"
- **When runing**: adjust the focus mode to the "continuous picture" mode.
- **When walking**: adjust the scene mode to the "steady photo" mode.
- **When still**: adjust the focus mode to the "fixed" mode".



(a) (b) (c)

Figure 2.7: Photos taken by the Smart Camera application developed for case study 3: (a) the photo taken while the phone holder is standing still, (b) the photo taken while the phone holder is walking and no context-awareness is taking place, and (c) photo taken while the phone holder is walking and using the Smart Camera application built on top of CAreDroid.

- **Power saver**: (1) adjust the flash mode to "off", (2) adjust the focus mode to "fixed" mode, and (3) adjust the quality of the picture to "minimum."

Similar to previous case studies, we implemented a polymorphic method for each of these modes and the objective is to call the appropriate method based on the phone context.

Platform	SLOC	% Increase
Non-context aware (Base)	277	-
Context-aware (Pure Java)	782	182%
CAreDroid	277 +133[a]	48%

[a]XML Configuration file

Table 2.6: Results of the significant line of code (SLOC) for the three implementations of the Smart Camera application used in case study 3. The results shows the SLOC along with the percentage increase relative to the non-context aware implementation.

The test is performed as follows. First a photo is captured while the phone is held in a standstill position using the original *camera* application provided by the phone. Next, the user starts to walk/run while trying to capture the photo for the same object again

(a) Default "Canny" (b) Optimized "Sobel" (c) Fast (d) Server "Canny"

Figure 2.8: Accuracy results of the different polymorphic methods used for the context-aware image processing application used in case study 4.

using the original *camera* application. Finally, the same experiment is done while using the CAreDroid-based *context-aware camera* application.

The results of the implemented application is shown in Figure 2.7. Figure 2.7(a) shows the original photo captured from a stand still position. Figure 2.7(b) shows the captured photo while the phone holder is walking and no context-awareness processing is taking place, and finally, Figure 2.7(c) shows the captured photo with the user is walking and using the developed context-aware camera with CAreDroid. As with the previous case studies, Table 2.6 shows the reduction of the overhead in SLOC when the context-aware Camera application is developed using normal Android flow compared to the proposed CAreDroid flow. The table shows a reduction of SLOC by more than a factor of 3×.

2.6.4 Case study 4: A Context-Aware Image Processing Application

In this case study we focus on how CAreDroid can be used to offload/adapt to heavy computations. In particular, we choose image processing as an example for extensive computation. In this study, we focus on performing edge detection with variable quality based on the availability of WiFi (in order to offload the computation to a server) and on the remaining battery capacity. The application utilizes various implementations of the edge detection, described as follows:

- **Default**: This implementation gives the highest accuracy and runs locally on the

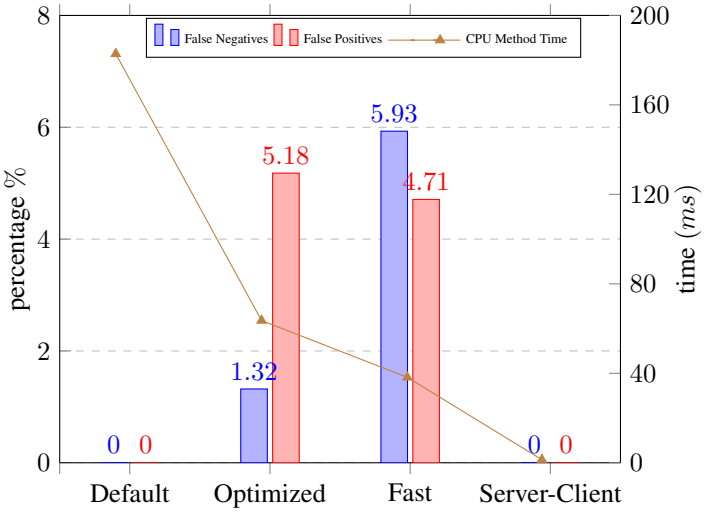


Figure 2.9: Results of different edge detection algorithms used in case study 4. This figure shows the percentage of false positives and false negatives versus CPU execution time for different algorithms.

mobile.

- **Optimized**: When the battery capacity is critical (very low), it is important to reduce the number of computations. Hence, this implementation is less accurate.
- **Fast**: High increase in the battery temperature can be interpreted as is an indication of a heavy load running on the phone. Therefore, in order to enhance the response of the app, this implementation downscales the image size.
- **Server-Client**: If the WiFi connectivity is available with good quality, the app can remove the burden of computation to a remote server.

The accuracy results of the five edge detection methods are shown in Figure 2.8.

This case study shows that context awareness can be used to explore the tradeoff between execution time and accuracy with usage of WiFi connectivity.

2.7 Discussion

The underlying idea behind CAreDroid is the ability of the system to sense and adapt to variations in the environment and available resources. In this section we discuss some issues that faced us during the design and implementation of CAreDroid.

2.7.1 Why is CAreDroid implemented inside the OS?

One possible design of CAreDroid was to design it as a library which provided context adaptation functionalities through a set of exposed APIs. Compared to the current design of CAreDroid, the library-based design falls behind in terms of the two design criteria discussed in Section 2.2 named *Usability* design and *Performance*. From the usability point of view, the library implementation of the adaptation engine forces the developer to issue subsequent calls to the library APIs. Missing calls to the library APIs may result to degradation in the context awareness of the developed application. On the other hand, the current design of CAreDroid makes the application developer completely *oblivious* from the adaptation. He is asked only to provide the adaptation *policy* in the XML configuration file. Afterwards, CAreDroid intercepts the execution of the methods while it is being interpreted by the Dalvik VM and perform the adaptation automatically. From the performance point of view, implementing the context adaptation and monitoring in the low level results into less execution time as proved by the experimental test cases shown in Section 6.6.

2.7.2 Privacy

Sensing and understanding the user's context and taking decisions accordingly can lead to various privacy leaks. Android privacy mechanism depends on providing the user with different queries in order to grant permissions to the application to use the sensory data. In our work, CAreDroid ensures that the adaptation policy specified by the developer does not use sensory data that are not permitted by the user. For this end, CAreDroid parses the

application's permissions included in the Android *manifest file*[2]. The adaptation engine in CAreDroid uses only permissible contexts as per application permissions.

2.7.3 Developer Matters

Despite the fact that the adaptation engine decision is obfuscated from the developing phase, in some scenarios—for example when the best fit policy is used—the developer may be interested in retrieving the current operating point. Therefore, CAreDroid addresses this issue by providing an API called "*read_operating_point*()" which can be used to read the current values of different contexts.

2.7.4 Limitations

The CAreDroid framework described here is not without some limitations:

- Polymorphic methods in CAreDroid must be pure functions, i.e., they cannot perform I/O and cannot change global program states, and their output must depend only on the method arguments. To allow for non-pure functions, the framework would require state-migration procedures between every possible pair of polymorphic methods.
- CAreDroid assumes that an application developer can provide multiple implementations of sensitive methods. Specifying the right constraints is not an easy task and it may be better to suggest the right constraints to the developer during a validation phase of the application. However, this is an open research point and previous work [CCF12, BGF10] has identified the importance of enforcing the developer to suggest the adaptation policy and not letting the adaptation engine automatically synthesize the adaptation policy.
- CAreDroid also expects the application programmer to be aware of suitable ranges of operations for different sensitive methods. In the future, we intend to explore automated code profilers that could suggest ranges of operation for each of the choices,

[2]an Android XML file that declares the permissions required by the application

helping users in defining suitable adaptation configuration files.

2.7.5 Broader Uses of CAreDroid

CAreDroid supports connectivity context such as Wifi connectivity, signal strength and quality of the signal as well as low level context such as battery temperature. These contexts can be manifested to decide if some intensive computation should be *offloaded* to a server or if an *approximate* computation should be used. In particular, if battery capacity is good (high enough) and there is WiFi connectivity with good strength then CAreDroid can switch to a method that offloads intensive computation to a server and remove the burden of computation from the phone. Hence, the concept of cyber-foraging discussed in Section 2.1.1.1 can be directly implemented using CAreDroid. Similarly, approximate computation (or algorithmic choice as discussed in Section 2.1.1.1) can be implemented using CAreDroid by manifesting the temperature context as well as the battery capacity context.

2.8 Conclusion

Context-aware computing is a powerful technique for physically coupled software. It can enhance functionality and improve resource usage of applications by adapting them to context. In this chapter, we present CAreDroid, an adaptation framework for context-aware applications in Android. CAreDroid allows applications developers to develop context-aware applications without having to deal directly with context monitoring and context adaptation in the application code. In CAreDroid, multiple versions of methods that are sensitive to context are dynamically and transparently replaced with each other according to application-specific configuration file.

By pushing the context monitoring and adaptation functionalities to the Android runtime, CAreDroid is able to provide context-awareness more efficiently and with significantly fewer lines of code compared to current Android development flow. In particular, using different case studies, we show how CAreDroid can be used to develop context-aware appli-

cations. Results from these case studies show that CAreDroid reduces the code complexity by at least half while decreasing the computation overhead by at least a factor of 10× compared to non-CAreDroid applications.

CHAPTER 3

CAMPS: Charging-aware Adaptation for Power Management in Mobile Operating System

3.1 Introduction

It has been shown that user behavior has a major influence on overall battery life [RZ09, FMK10]. In the ideal case, users should never need to explicitly manage battery life; rather, devices should work in a perpetual manner with minimal user intervention. We propose that a metric to best capture this ideal is the availability of the system to meet user needs with the highest possible quality of service (QoS). We define *availability* in this context as the proportion of time the system can deliver the subjective user-desired functionality. At a high level, device availability is a function of the holistic battery charge/discharge processes over time. We consider the net energy stored in the battery as a proxy for availability.

In general, the energy gained from a charging event depends on (1) user's behavior (e.g., how long they stay plugged in), (2) the battery-related hardware (e.g., the power *supply*, charge controller, and battery characteristics), and (3) the non-battery system hardware and software comprising the power *load*, whose energy consumption is also directly influenced by the user, e.g., by running applications. We propose new threads of research in *battery charging-aware* (i) *power management* and (ii) *deferrable task scheduling*. The latter – which we call CAMPS – is the ability to defer, split, or otherwise reschedule a non-critical and/or non-real-time task at a macro time scale to prioritize power delivery to the battery while charging. We attempt to answer the following questions about hypothetical charging-aware software solutions:

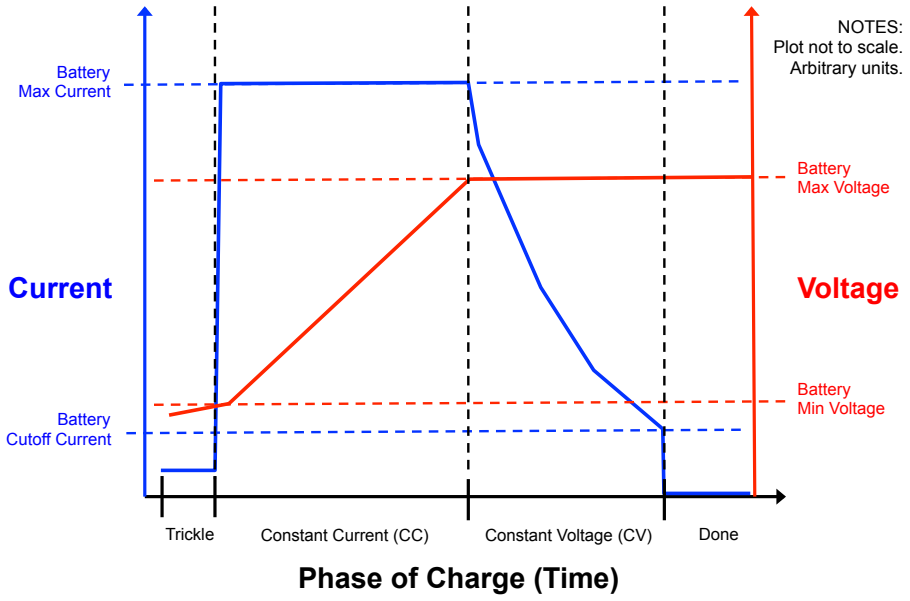
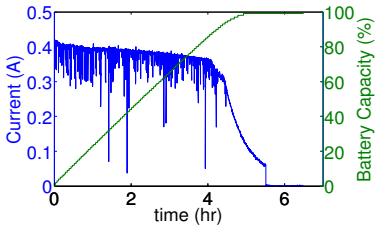


Figure 3.1: Illustration of typical battery charging current and voltage characteristics.

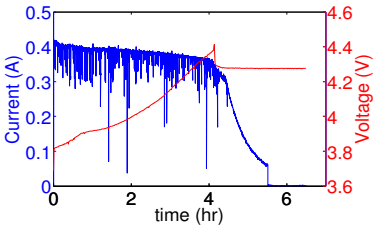
- What opportunities exist to improve overall mobile system availability?
- Under what conditions would a user benefit?
- How many users would benefit?

3.2 Smartphone Charging Profile

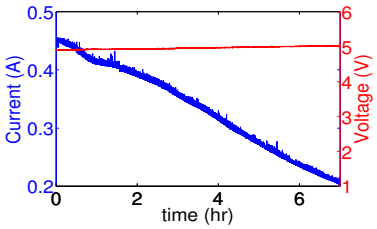
As discussed in the previous section, the energy gained from charging depends on factors such as the power supply capability, the battery characteristics, etc. A representative illustration for the typical lithium-ion (Li-ion) battery charging process is shown in Figure 3.1 [Pan07, Son14]. The charging process is typically divided into two primary phases from fully depleted 0% state-of-charge (SOC) to 100% SOC. In the first phase, the device's charge controller circuit outputs a high constant current (CC) which delivers high power to the battery. Once the battery voltage has reached a certain threshold, typically 4.2 V, the charge management controller circuit transitions to the constant voltage (CV) phase, which maintains the voltage threshold and allows current drawn by the battery to fall off gradually. Once the drawn battery current has reached a minimum level, the charge controller terminates charging to



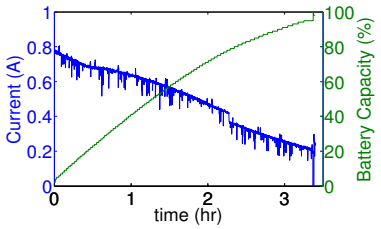
(a) Battery current and SOC (USB charger)



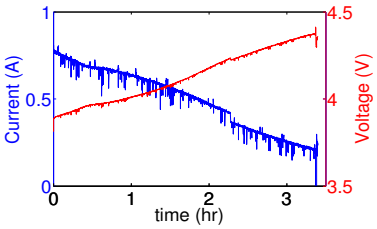
(b) Battery current and voltage (USB charger)



(c) USB charger current and voltage



(d) Battery current and SOC (AC charger)



(e) Battery current and voltage (AC charger)

Figure 3.2: Nexus 4 charging characteristics from 0% to 100% SOC. The smartphone was powered on and idle for each test.

prevent battery damage from over-charging. Note that the power delivered to the battery is typically highest in the CC phase, and drops off during the CV phase.

3.2.1 Effect of Charger Type

To quantify the effect of different charger capabilities, we set up a testbed consisting of a Nexus 4 smartphone along with a programmable source measure unit (SMU). The power path from the supply (AC adapter or standard USB) to the battery consists of two parts. The first part is a 5 VDC path from the supply output to the power management integrated circuit (PMIC), which typically includes both the charge management controller circuit and the voltage regulators for the system's VCC rails. The second part is a path from the charge controller to the battery. In our testbed, we utilize the SMU to measure the power in the first power path, while we query the internal charge controller circuit (Qualcomm PM8921) in the Nexus 4 to measure the power in the second power path. The chip reports the battery

voltage and current to the operating system (OS) using the standard Hardware Abstraction Layer (HAL) in Android.

In our experiments, we charge the device from 0% state-of-charge (SOC) to 100% SOC via a standard 5 VDC USB plug (whose standard imposes a 500 mA current limit) and the AC to USB adapter included with the Nexus 4, which allows up to 1.2 A output at 5 VDC. The measured results are shown in Figure 3.2.

3.2.1.1 Power Headroom

As can be seen from Figure 3.1 and Figure 3.2, the power drawn by the battery while charging depends on the phase of charge. Note that the maximum power of the 5 VDC supply is not drawn throughout the entire charging process. We define the instantaneous *power headroom* as the maximum power that the supply *can deliver* minus the maximum power that the battery *can absorb*. This headroom can be used to do useful work for the system load without impacting the energy gained by the battery during charging.

3.2.1.2 Charging Time

In order to understand how the charging process and duration depends on charger capability, we analyze the amount of time spent in the CC and the CV phases as well as the SOC at which the phase transition occurs. As shown in Figure 3.2(a), by charging the phone via USB, the current drawn is approximately 400 mA during the CC phase, and is fairly constant, being limited by the USB 500 mA restriction. When the CV phase starts after about 4.2 hours, the SOC is approximately 85%. The time spent in the CV phase is approximately 1.3 hours. In the AC charging experiment, there is no CC phase, due to the fact that the drawn power is limited by the battery's ability to absorb current, not a limitation of the charger. In this case, the battery starts by drawing 800 mA and then the current decays to maintain a smooth rise in battery voltage. The battery is fully charged in 3.4 hours via AC compared to 5.5 hours in case of USB.

3.3 User Charging Behavior

The power headroom observed in Sec. 3.2 will play a significant role in the task deferral opportunities we explore in Sec. 3.4. Before we explore these opportunities, we now quantify how many users could benefit from charging-aware software techniques, which depends on user charging behavior. For example, if a specific user tends to charge their device such that it progresses through both the CC and CV phases during a single charging event, then this user might benefit from a task scheduler that accommodates time-varying power headroom. However, if a user tends to unplug the phone before entering the CV phase, then this user might not benefit from the proposed types of charging-aware techniques proposed in Sec. 3.4.

A user's charging behavior can be quantified as the answer to the following statistical questions:

1. What is the SOC when the device is plugged into the supply, irrespective of when it is unplugged?
2. What is the SOC when the device is unplugged, irrespective of when it was plugged?
3. What is the charging duration for each unique plug-to-unplug charging event?

To answer these questions, we study the user charging behavior of 40 randomly chosen and anonymous Nexus 4 users over a period of roughly six months using the Device Analyzer [WRB13] dataset.

3.3.1 SOC at Plug-In Event

We start by calculating the arithmetic mean value for the SOC at plug-in for each individual user's aggregated charging events, depicted in Figure 3.3. We then calculate the global arithmetic mean (the mean of the individual user means) of SOC at plug-in, shown as a horizontal line in Figure 3.3. For the 40 users under consideration, the global arithmetic mean for SOC when plug-in events occur is 47%. We use this global mean to classify the users into three groups: users who tend to plug-in (1) at high SOC (60-100%), (2) around the mean SOC (40-60%), and (3) at low SOC (0-40%). Figure 3.4 shows a representative

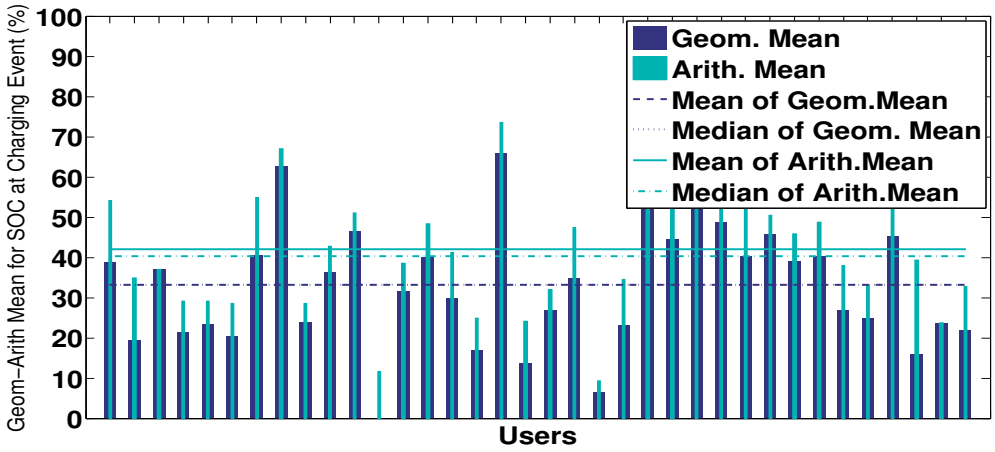


Figure 3.3: Mean SOC at plug-in events for each user.

user from each of the respective categories from top to bottom. In particular, Figure 3.4(a), Figure 3.4(d), and Figure 3.4(g) show the histograms of the SOC when plugged-in for each individual user. It is clear that these three users represent the three classes of users we defined: below, around and above the global mean SOC when plug-in events occur.

3.3.2 Charging Duration

The data shows that the global arithmetic mean of the charging durations across all users is 120 minutes (regardless of the SOC at plug-in time), but the median charging duration is less than that. The histogram of the charging duration for all the charging events aggregated across all users is shown in Figure 3.5.

The correlation between the charging duration and the SOC level at plug-in could affect our choice of how to categorize user behaviors. Thus, we aggregate all the user data for charging events into one set, and calculate the correlation coefficient between the SOC at plug-in with the charging duration. We perform the same computation for each category of the users separately. We conclude that in general, the charging duration is weakly correlated to SOC (coefficient is below 0.06) at the time the user plugs-in the phone. Thus, we do not take charging duration into consideration for categorizing users based on their charging behaviors.

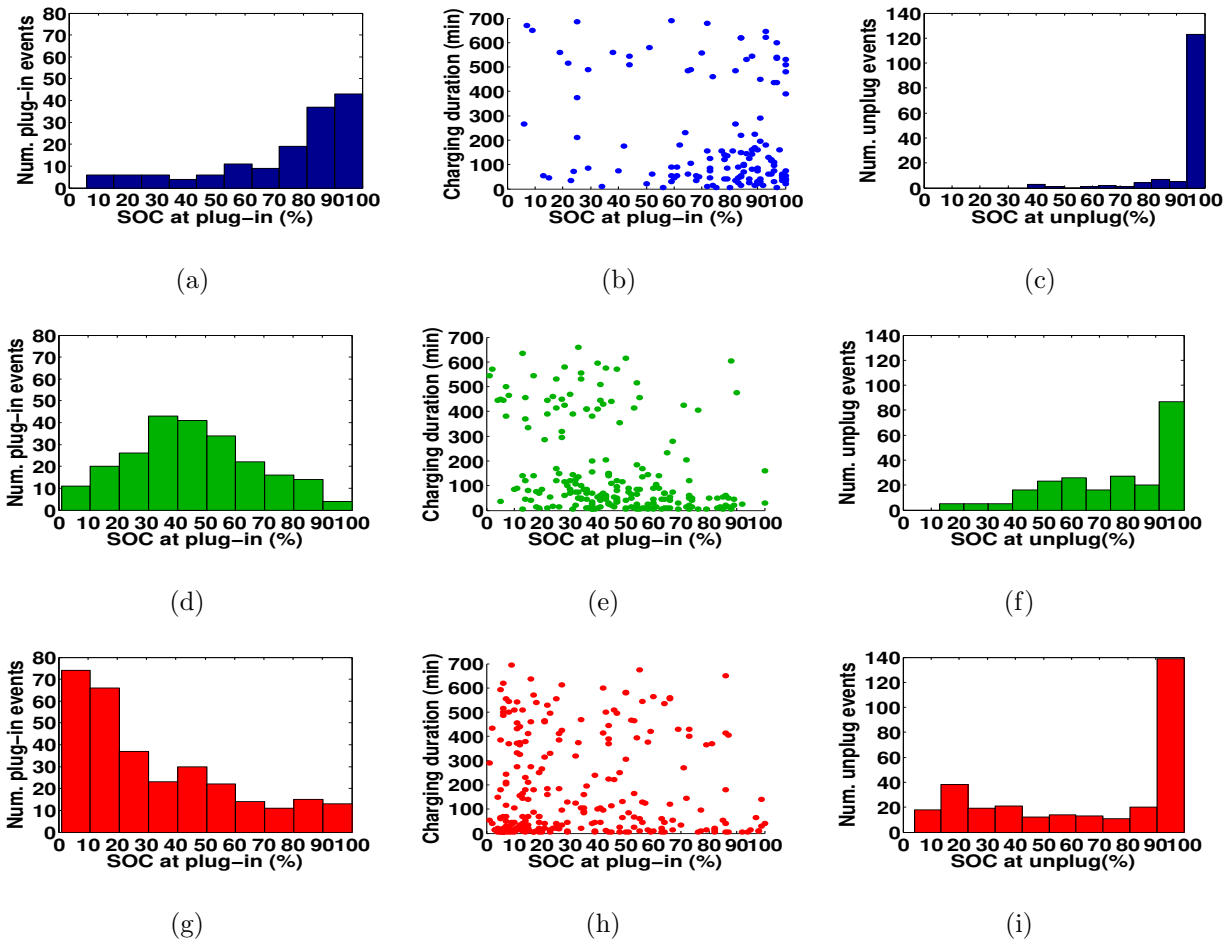


Figure 3.4: Charging behavior for three distinct users. Each row represents an exemplar user whose behavior follows different charging behavior trends from Classes 1, 2, and 3, which are colored accordingly as blue, green, and red.

The previous conclusion can also be drawn by considering the representative users from the three classes. Figure 3.4(b), Figure 3.4(e), and Figure 3.4(h) show the charging durations for these specific users versus the SOC level when the phone was plugged-in. We observe that across all users, charging duration tends to be low. The same trend can be observed when considering all users across the three categories as shown in Figure 3.6.

3.3.3 SOC at Un-plug Event

We extend our analysis by considering the SOC when the phone is un-plugged, irrespective of the SOC when it was plugged-in. The data for the representative users are shown in

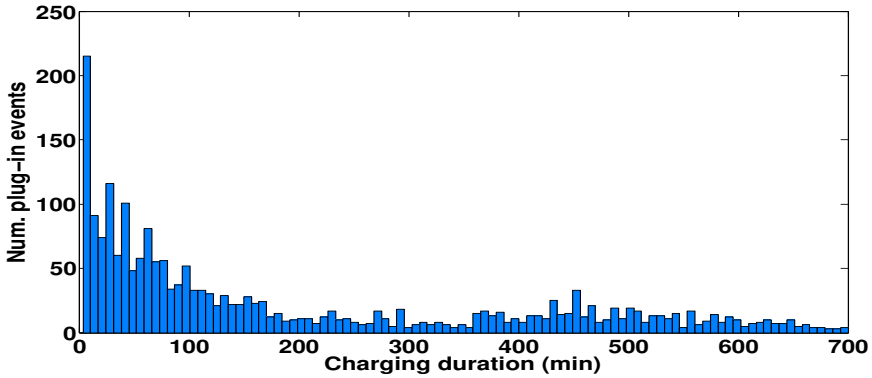


Figure 3.5: Histogram of charging duration for all charging events across all users.

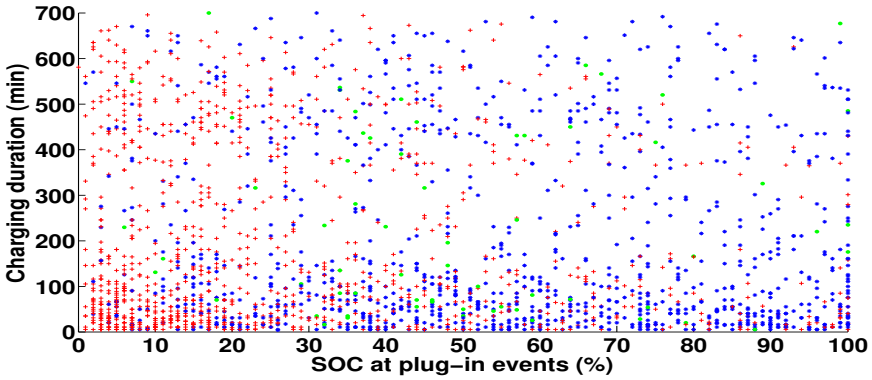


Figure 3.6: Charging duration vs. SOC when plugged in for all charging events across all users. Blue $*$ represents users of Class 1. Green $\bullet$ represents users of Class 2. Red $+$ represents users of Class 3.

Figure 3.4(c), Figure 3.4(f), and Figure 3.4(i). We observe that typically either the users let their phone charge until complete, or it coincidentally completes because the charging duration happens to be long enough (as we observed in Sec. 3.3.2, the charging duration is not correlated with SOC when plugged-in, which implies that charge completion is not necessarily the primary goal for users). This observation can be generalized using the same correlation calculations done in the previous experiment from Sec. 3.3.2. We find that in general, all three user types have similar unplugging behavior. Hence, we conclude that using the SOC when un-plugged as a parameter does not affect the charging behavior classification of users.

3.3.4 User Distribution

From the previous discussions, we are able to classify the users based on their charging behavior. The main criterion is to consider their average SOC when they plug in their smartphone. It is important to understand how the users are distributed across the three categories.

From the 40 users in our data set, we observed that 44% of the users tend to charge their phone when the SOC is above 60% (above the global mean, i.e., Class 1), and 47% of the users charge their phone when the SOC level is below 40% (below the global mean, i.e., Class 3). Finally, 9% of the users charge their phone in the mid range between 40% and 60% (around the global mean, i.e., Class 2).

We apply this user classification to determine the proportion of users that tend to plug in their devices at medium or high SOC. The behavior of these users tends to progress through both the CC and CV phases. Those users could benefit from deferring some tasks to CV phase where greater power headroom typically exists. According to the previous user distribution, 53% from the examined users fall into this category.

3.4 Opportunities for Task Deferral

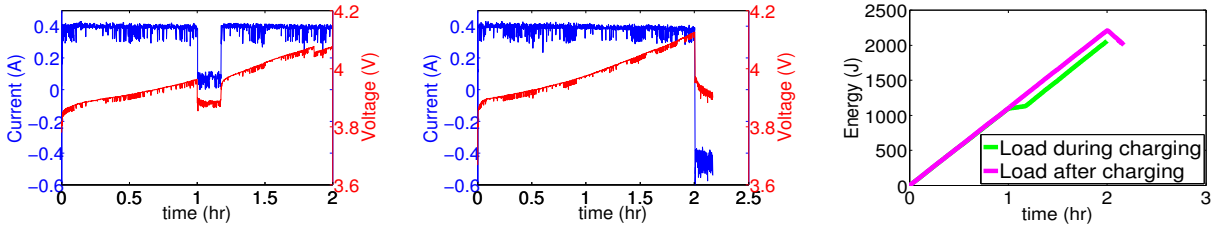
From our experiments described thus far, we demonstrated the existence of power headroom during certain phases of the charging process, and concluded that 53% of users likely have their devices experience significant time-varying power headroom while plugged in, but are currently not able to exploit it. In this section, we propose simple task deferral policies that exploits this power headroom in order to enhance the device availability. This is done by attempting to increase the net energy stored in the battery at the end of the charging event and task completion, whichever occurs later. It is given that the task must begin running after plug-in occurs. The charger used in this experiments is USB cable for Nexus 4 phone running Android 4.2.

We evaluate the simple proposed policies by manually launching an application during

different phases of charge, emulating the ability of the OS to defer the task automatically. We assume that the OS would know which tasks are deferrable, and which are not, without affecting the user experience. Tasks that are interactive or otherwise time-sensitive should generally not be deferred, as this would severely affect the highly subjective device availability to the user. Modification of the OS scheduler to bin tasks based on their tolerance for deferral or other rescheduling techniques is part of our future work.

3.4.1 Schedule Tasks After Unplugging

One simple scheduling possibility is to run the task just after the phone is unplugged. This leaves the phone to charge at the maximum rate while plugged in, without being affected by the power consumption from the task.



(a) App runs while phone is charging.

(b) App runs after unplugging phone.

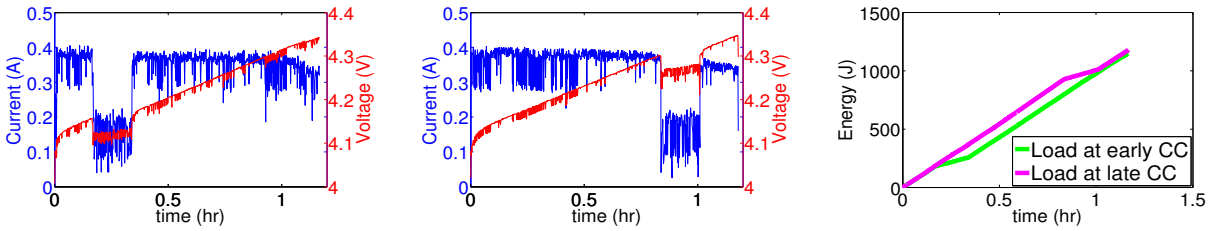
(c) Energy gained by the battery.

Figure 3.7: Effect of running an app during vs. after the charging period.

To evaluate this policy, we fix the charging duration to 120 minutes (which we found as the global arithmetic mean for charging duration in Sec. 3.3.2) and the initial SOC to 25%. The phone runs an app that uses power-hungry GPS and Wi-Fi for ten minutes. In the first sub-experiment, the app is run while the phone is charging, while in the second sub-experiment, we manually defer the same app until the phone is unplugged. For both cases, we calculate the energy gained by the battery by observing the current and the voltage of the battery over time. A representative run of the experiments are shown in Figure 3.7(a) and Figure 3.7(b). Figure 3.7(c) shows the net energy gained by the battery in both cases. We observe that deferring the task to after the phone is un-plugged performs worse compared to running the task while it is charging.

3.4.2 Schedule Tasks Within the Constant Current Phase

In this experiment, we schedule the same task at different stages of the CC phase in order to determine whether it will affect the net energy gained by the battery by the end of the charging period.



(a) Early stage of CC phase. (b) Late stage of CC phase. (c) Energy gained by battery.

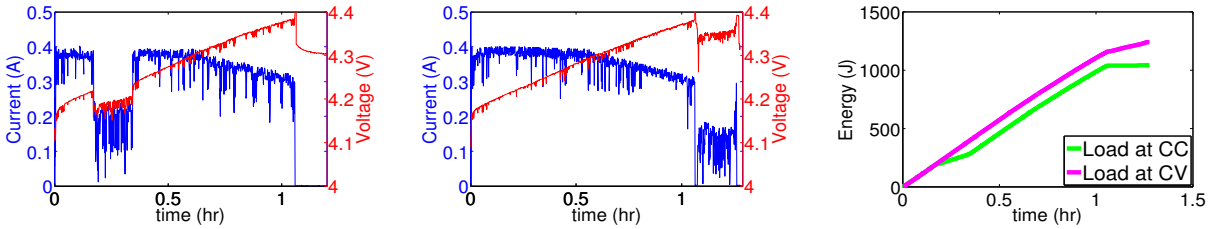
Figure 3.8: Effect of running an app early vs. late in the CC phase.

As in the previous experiment, we run sub-experiments to compare between the two scenarios. We fix the initial SOC to 70% SOC and the charging duration to 70 minutes. In the first sub-experiment, we run the app beginning at ten minutes after the plug-in event as shown in Figure 3.8(a). In the second sub-experiment, we start the same app ten minutes before the beginning of the CV phase as shown in Figure 3.8(b). We find a small improvement in net energy gain that occurs due to deferring the task to later in the CC phase, as shown in Figure 3.8(c).

3.4.3 Schedule Tasks in the Power Headroom

A third possibility is to defer the task to the CV phase, where the greatest power headroom is present. In this experiment, we fix the initial SOC to 78 (CC phase) and the charging duration to 120 minutes (which ensures that the phone hits the CV phase). We run the app once in the CC phase (Figure 3.9(b)) and once on the CV phase (Figure 3.9(b)). The CV phase starts once the peak battery voltage is reached. From Figure 3.9(c), we observe an 18.9% increase in the energy gained by the battery in the latter case.

This can be explained if we look carefully in Figure 3.9(b) and Figure 3.9(b). In the former



(a) During CC phase. (b) During CV phase. (c) Energy gained by battery.

Figure 3.9: Effect of running an app in the CC vs. CV phases.

case the current drops approximately from an average of 400 mA to 150 mA. However, in the latter case the current drops from an average of 300 mA to 100 mA. The difference in the drop between the two cases can be explained by the power headroom present in the CV phase at this time. Assuming that the maximum deliverable power from the supply and the app power demand are constant, this means that a greater part of the load current was drawn from the supply and not from the battery during the CV phase.

3.5 Conclusion and Future Work

We present a case for battery charging-aware power management and deferrable task scheduling to improve overall device availability. In particular, we propose to utilize the power headroom during certain phases of battery charging to run these tasks, rather than starve the battery of energy during its most power-intensive charging time. Increasing the energy delivered to the battery during the charging period, or conversely, decreasing the required charging duration to reach full SOC would improve overall device availability to the user.

Our study on Nexus 4 smartphone user charging behavior shows that most users tend to charge their phone for less than 120 minutes, and that the charging duration is largely independent of the SOC when the smartphone is plugged in or unplugged. We estimate that around 53% of users could benefit from battery charging-aware software policies that maximize energy delivered to the battery while plugged in.

We observe mixed results for the different simple proposed charging-aware task deferral policies. Positively, we find that deferring tasks to the CV phase can improve the net energy

gained by the battery by approximately 18.9%. In contrast, we observe that deferring tasks until the end of the charging period or deferring the tasks within the CC phase leads to no significant net battery energy increase. All of the proposed scheduling schemes could be implemented in a smartphone OS via online monitoring of the power headroom, and they represent only a small portion of the possible scheduling policies.

Our future work seeks to pursue the ideas around power headroom and user charging behavior further. Quantifying power headroom based on the battery characteristics and the stage of the charging process is essential to determine the number and type of tasks to be deferred based on their predicted energy requirements. A task could also be split between two phases based on the amount of headroom available and the energy requirement of the task. Furthermore, charging-aware software might have even greater potential to improve availability in systems with heterogeneous energy storage architectures that require careful energy management. Finally, it would be useful to predict whether a given user at during some charging event is likely to reach a period with greater power headroom, using factors beyond those explored in this chapter. Accordingly, our future work may include a study of user-specific models in this direction.

Part II

Privacy Firewall for Personalized Autonomous Computing

CHAPTER 4

SpyCon: Context-aware Adaptation Based Spyware

4.1 Introduction

Context-awareness is the ability of software systems to sense and adapt to the surrounding environment. Many contemporary wearable/mobile applications adapt to users in different ways based on context such as locations, connectivity states, energy resources, and proximity to other users to name a few. As we stand on the edge of an explosion of data from these sensory devices, there has been a corresponding increase in applications and dedicated sensing frameworks targeting the integrity between context (sensing) and corresponding adaptations (actions)[EWS15].Unfortunately, the same act of adapting to user context often leads to systems where increased sophistication comes at the expense of more privacy weaknesses. At the heart of privacy is the notion that information collected from the physical world through sensors poses a significant privacy risk on *inferring* user sensitive information like behavior and location. While there exists a recent body of work on identifying malicious apps, which if granted access to sensory data can perform unwanted *inferences*, the question of whether a malicious app can still perform inferences *without having direct access to sensory data* remains unanswered. In other words, we ask the following question: do context-based actions taken by authentic context-aware applications—which are granted access to sensory data—open side-channels for malicious apps which do not have direct access to such data? That is, by monitoring actions triggered by authentic context-aware apps, can a malicious app still be able to perform unwanted inferences about user like behavior and location. In this chapter, we introduce a new type of spyware that exploits the privacy leaks in context-aware adaptation which we call **SpyCon**.

4.1.1 Related Work

While mobile users benefit from sensing technologies, there are increasing privacy and security concerns. The permission systems on both Android and iOS become the first line of defense to protect users from leaking their information. However, the traditional grant-all-or-none policy allows third-party apps to have all permissions [**?**, HHJ11]. Even worse, most users do not realize the potential privacy hazards after granting such permissions. For example, though seems innocuous, `ACCESS_WIFI_STATE` becomes a privacy intrusive permission since a local MAC address can serve as a unique device identifier [ACR14]. It is reported that as little as 17% of users pay attention to the permissions during app installation [FHE12]. Different side-channel attacks have been proposed. For example, inertial sensory data and touch screen trajectories can reveal user passwords [HON12, MVB12, OHD12, MVC11]. Besides, we witnessed how to exploit cellular signal strengths, air pressure, or power consumption for locations [MSV15], gyroscope for eavesdropping conversations [MBN14], system-level aggregate statistics for user's real-world identity [ZDH13], and activity information leakage through music [**?**] and the state of shared memory for foreground apps, and even, `activity` transition sequences [NYY15]. Moreover, there is a trend that malicious apps are also adapting to wearable devices [RGK11, WLR15]. For example, MoLe [WLR15] exploits the wrist motion derived from smartwatches to infer keystroke inputs. So far we have provided examples showing "Your apps are watching you" [KT10] which a majority of users will never realize and for sure "These aren't the droid you're looking for" [HHJ11]. Contrary to the aforementioned side-channel attacks, we consider a spyware *which does not have access to sensor information* like inertial or gyroscope sensors, a spyware which can monitor only the actions that are triggered—by other apps—based on changes in these sensory data. Similar to the spyware considered in our work, [ZDH13] shows some information leaking channels in Android (e.g., phone speaker status) that a malicious app can monitor without permission.

4.1.2 Chapter Contribution

- We exploit a new side-channel attack vector arising from monitoring changes of phone adaptations by context-aware applications. We call this new set of attacks a *context-*

aware adaptation based spyware, or in short, SpyCon.

- We show a concrete instantiation of a SpyCon which can maliciously infer user's behavior by monitoring context-based adaptions. We assess the performance of the developed SpyCon through a one-month user study.

4.2 Context-aware Adaptation based Spyware

We introduce the Context-aware Adaptation based Spyware (SpyCon) and illustrate how it works and its potential negative outcomes by showing an example of a SpyCon that stealthily learns the semantics of the user locations inferred by those adaptions made by other context-aware apps.

4.2.1 Popular Phone Manager Apps

Location-based phone settings management is one of the most popular context-aware applications[1]. Due to their capability to adapt to user context, apps like Tasker [tas], and Locale [loc] gained more than one million downloads from Google Play Store. These context-aware apps allow users to define their profiles. A *profile* contains a context-based trigger and a set of actions. Once the phone context matches the trigger, the corresponding actions to change the *phone settings* are performed. For instance, a common profile configuration is muting the ringer volume when a user is in class. Motivated by the popularity of these location-based context-aware apps, we choose user location as the information that our SpyCon leaks.

4.2.2 Spyware Description

We are interested in designing a SpyCon that monitors changes in phone settings—which are triggered by a location-based context-aware app—and uses these changes to leak user's location. We start by making two important remarks: (1) **No user permissions:** Many phone settings can be monitored without user permissions, such as current screen brightness.

[1]By the time this chapter was written, context-aware phone settings management applications had ranked 3rd in the Productivity category in the Android Developer Challenge [tas].

PS	Description	PS	Description
R	Ringer mode	P	Wallpaper
H	Touch sound	D	Dialpad sound
W	Enable WiFi	A	Alarm volume
I	Ringer volume	M	Media volume
T	Display timeout	B	Screen brightness
V	Vibration on touch	L	Screen locking sound

Table 4.1: List of Phone settings (PS).

(2) **Ambiguity on setting changes:** Manual adjustment can make changes in phone settings through physical buttons. Although SpyCon cannot discriminate *a priori* between the changes in the phone settings done by a location change or by manual adjustment, machine learning algorithms can be handy in discovering repetitive patterns in the data. The operation of SpyCon is divided into two phases: (1) **Logging:** SpyCon monitors all the changes in phone settings and records a timestamped value upon a change is detected. A list of phone settings that we consider is given in Table 4.1. (2) **Data mining:** SpyCon analyzes the data to discover repeated patterns.

4.2.3 SpyCon User Study

We study how SpyCon can invade users' privacy, by developing two applications, a shadow logging app, and a SpyCon.

4.2.3.1 Shadow Logging Application

As mentioned in Section 4.2.1, users define *profiles* in apps like Tasker and Locale. Users have to enter a fixed-radius circular geofence as a *context* trigger, as well as a set of *actions* (e.g., adjusting screen brightness or changing ringer mode to vibration) that will be activated when the user enters these geofences. We need to collect the ground truth for the context and the triggered actions to compare later and to understand how much information is leaked by context-aware apps. Hence, we developed a shadow app that resembles the functionality

of Tasker and Locale. The full phone settings we considered are listed in Table 4.1. To keep track of the golden output (ground truth) for later evaluation, the shadow app keeps a timestamped record whenever the active profile is changed, implying that the user moves to a different location.

4.2.3.2 SpyCon Application

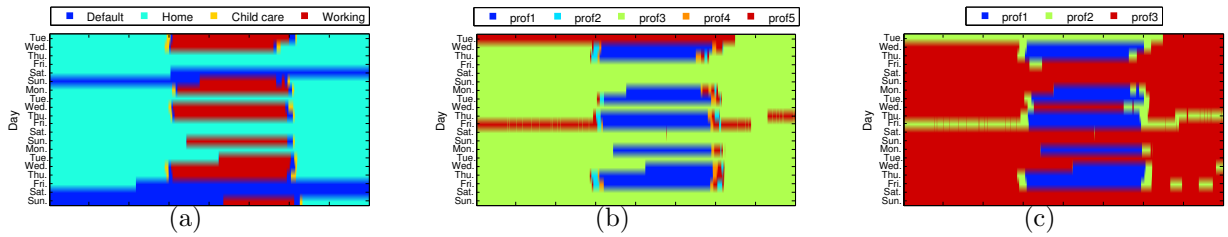
We developed a SpyCon that logs the phone settings in the background *without any interaction* with all the other apps, including the context-aware app[2]. All the settings collected by the SpyCon can be accessed *without permissions* in Android OS. SpyCon may require the knowledge of other applications installed on the phone to know whether the phone settings—or any other context-based adaptations—are altered by a context change. For example, the SpyCon we describe here can reason that the change in phone settings is caused by a change in user's location by knowing a priori that there is an app that adapts the settings based on location. This information can be retrieved in two steps; (1) The malicious app needs to know what other apps are running on the same phone, which can be acquired by calling the Android API `getInstalledApplications()` [3]. (2) The malicious app needs to know what context adaptations are involved in the context-aware apps. This can be fetched from the app store (e.g., "Google Play") page. For example, Tasker app specifies that phone settings are adapted based on user location.

4.2.3.3 User Study

We implemented both apps mentioned above on Android 5.0.1 running on Nexus 5. Seven participants are recruited in our user study. Each participant carries the phone for four weeks. Users can choose the settings/profiles based on their personal preferences, and they are allowed to change the phone settings manually. Based on the data we collected we explore

[2]In the real world, SpyCon can provide some functionality but collect data stealthily, which is a typical way a spyware hides its real intention.

[3]Even though Android may protect this API by adding a permission in the future, studies showed that it is hard for most users to relate the side-channel privacy implications to the granted permissions [FHE12].



(a) (b) (c)

Figure 4.1: One example of profile timeline from user #2. (a) The golden output. (b) Clustering result based on full observation. (c) Clustering result based on dominant features derived by feature selection algorithm.

what information can be maliciously mined.

4.2.4 Experiment 1: Data Mining by Clustering

Revealing the semantics of the user location trace, or equivalently, the active profile sequence from phone settings is challenging since there is not *always* a one-to-one mapping between a profile and a phone setting. This is because: (1) Users configure only a subset of the settings listed in Table 4.1 and it is not known a priori which settings are used. (2) Users can manually override the phone settings by pressing the volume buttons or adjusting the brightness through the built-in *Settings App*. Hence, we use k-means algorithm to approach the user data mining problem. Deciding the number of clusters in the k-means algorithm is hard in general because it is application dependent. Since our SpyCon does not know how many profiles are defined by users, we brute-forcedly set k to be any value between 2 through 7 (selected based on the maximum number of profiles defined by our participants). The clustering result with the highest silhouette score is returned.

4.2.4.1 Critical Phone Settings

Inspired by how most unsupervised machine learning algorithms work, we implement a greedy algorithm to find dominant phone settings as follows:

(i) Initialize the selected feature set $S = \phi$.

UID	# clusters using all features				Dominant features
	base	2	3	$2-7$	
1	75.2	+18.9	+22.9	+19.1	+21.8 W,R,V
2	56	+17.2	+24	+18.3	+24.1 R,B,W
3	80.5	+12.9	+14.4	+13.6	+16.7 R
4	45.6	+37.3	+34.2	+35.6	+35.9 W,R,L
5	42	+24	+35.2	+24	+41.8 T,R,A
6	57.9	+4.4	+36	+4.4	+40.7 A,R,B,W
7	78	+15	+15.5	+15	+15.6 R,O
Avg.	62.2	+18.5	+25.8	+18.2	+28.1

Table 4.2: Clustering accuracy (in percentage) of all users compared to the baseline accuracy (the accuracy based on blind guesses) by applying k-means using the settings from Table 4.1.

(ii) We examine every other setting f not in S by performing k-means with feature set $S \cup \{f\}$. The silhouette score h_f is computed accordingly.

(iii) Denote $\hat{h}$ as the maximum h_f from the previous step. If $\hat{h}$ is larger than previous silhouette score, then $S = S \cup \{f\}$ and go back to step 2. Otherwise, terminate.

4.2.4.2 Privacy Implications

The clustering result of one participant is shown in Figure 4.1. Figure 4.1(a) shows the actual user profile changes across the day (the golden output as explained in Section 4.2.3.1). Figure 4.1(b) shows the k-means clustering result (using an adaptive number of clusters) and demonstrates similar patterns with the golden output.Our algorithm can successfully capture events even with short periods. For example, it learns that user #2 regularly goes to a particular place after leaving or before returning home, which turns out to be the childcare from our post-interview. Clustering result derived from dominant features using our feature selection algorithm is shown in Figure 4.1(c). Figures 4.1(b) and 4.1(c) clearly indicate the ability of the developed SpyCon to reconstruct user context (switching profiles in this case)

Anti-virus Package Name	Scanning Result
AVG AntiVirus	no threat
Symantec Norton Security & AntiVirus	no threat
AVAST Mobile Security & Antivirus	no threat
McAfee Security & Power Booster	no threat
Kaspersky Internet Security for Android	no threat

Table 4.3: Results of scanning the developed SpyCon using signature-based malware detection packages.

by just monitoring its side effect (changes in phone settings)[4]. The overall accuracy of the clustering algorithm is reported in Table 4.2. We define *baseline accuracy* by using blind guesses, that is, the SpyCon always reports a user is at home without observing any phone settings. The results in the rest of the columns are the additional information (the increase in accuracy) the SpyCon gains over the baseline accuracy if an inference is used based on the number of clusters. The accuracy derived from dominant features is slightly higher because the feature selection algorithm excludes noisy features. We report dominant features for each user in the last column of Table 4.2. This study shows that the designed SpyCon can infer with an average accuracy of 90.3% the user behavior, in particular: (1) the average commuting time between home and work, (2) the average time spent at work and home, and (3) the weekend behavior, such as if a specific place is frequently visited on Sundays and average time spent at home.

4.2.5 Experiment 2: Detection Using Current Antivirus Apps

We used five well-known anti-virus applications (shown in Table 4.3) on the developed SpyCon. None of them reported this app as malware. This follows from the fact that the proposed SpyCon does not have any suspicious code signature.

[4]If the user specifies two profiles with the same settings, SpyCon will recognize them as the same profile. However, the incentive of the user to define the same settings for multiple profiles defies the idea of the context-aware app.

Context-aware App	Context	Side-channel
Tasker [tas]	location	phone settings
Locale [loc]	location	phone settings
Silence [sil]	calendar events	phone settings
RockMyRun [roc]	biometrics	music played
HABU music [hab]	mood	music played

Table 4.4: Context-aware apps and their side-channel.

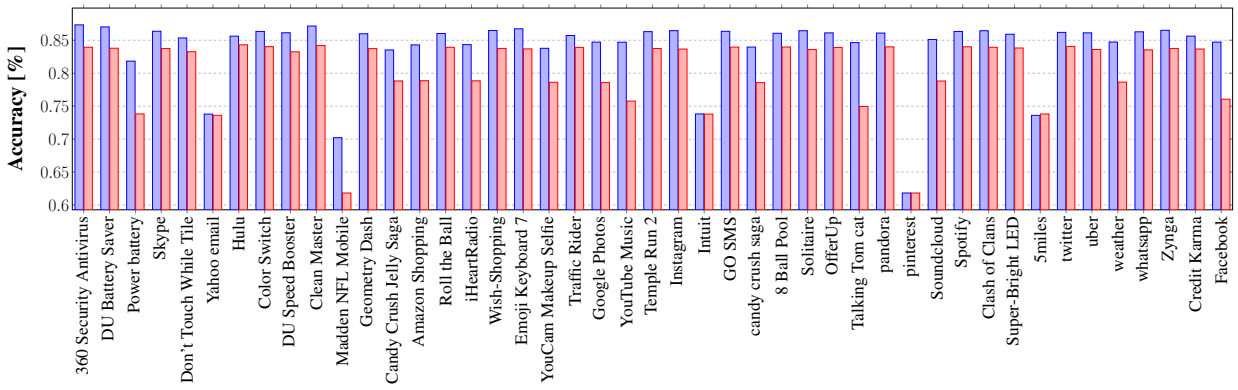


Figure 4.2: Accuracy of leaking information about user from data collected by 45 of the most downloaded free apps; (blue) accuracy of identifying the semantics of the user location when a location-based context-aware app (Tasker/Locale) is used, and (red) accuracy of identifying user calendar profile when a calendar-based context-aware app (Silence 2.0) is used.

4.2.6 Experiment 3: Beyond Location SpyCon

While the previous experiment aims at studying how the proposed SpyCon can leak the semantics of user location, we further explore how exploiting this side-channel can reveal other user information. We study several context-aware apps in the Android market and report the monitored *context* and the corresponding *actions* taken by these apps in Table 4.4. Since other apps can observe these actions (even without asking for user permissions), these actions open a side-channel that leaks information about the user. For example, if a SpyCon knows a priori the presence of Silence App [sil] (an app which changes your phone settings

based on the calendar events), it can reveal the timing or repetition of calendar events based on the side-channel of phone settings.

4.2.7 Experiment 4: How many SpyCons in the market?

Identifying whether an app is exploiting the proposed context-adaptation side-channel is hard without the knowledge of the app behavior. Hence, we focus in this experiment, instead, on identifying apps in the market that possess enough information to leak, however, we do not make the claim that these apps are SpyCon. We performed static analysis on 45 of the most downloaded free apps from the Google Play store to check which APIs they use. We intercepted the APIs that retrieve the information possessed by these apps. Finally, we used this collected information to leak user context. In particular, we focus on the setup when Tasker/Locale (a location-based context adaptation app) and Silence (a calendar events-based context adaptation app) are changing the phone setups based on location and calendar events, respectively. Figure 4.2 shows the accuracy of retrieving the user context from the data retrieved by the 45 apps. Our results show that 86% of the top downloaded free apps have enough information to leak user context with some of the apps scoring more than 80% accuracy when a location-based context-aware app is used. Similarly, for calendar-based context, 64% of these apps can leak information with an accuracy more than 80% showing the significance of this side-channel information.

4.3 Conclusion

We introduced a new class of privacy-threatening spyware that is designed to snoop around adaptations made by context-aware apps which we called SpyCon. We showed through the user study that by monitoring the context-based adaptations triggered by context-aware apps, SpyCon could infer user behavior. To exacerbate the situation, our experiments showed that this new spyware is undetectable using off-the-shelf antivirus and moreover many of the top 45 downloadable free apps have enough information to reveal about user. Future work includes studying the case when multiple SpyCons collect different data and collaboratively

leak user information by fusing these partial data. As well as, studying different detection methods and mitigation strategies for SpyCon.

CHAPTER 5

VindiCo: Privacy Safeguard Against Context-aware Adaptation Based Spyware

5.1 Introduction

In this chapter, we purpose **VindiCo**, a novel detection and mitigation engines that are designed to protect against privacy leaks due to context-aware adaptations which SpyCon exploits. Contrary to traditional malware detection approaches, VindiCo does not rely on prior information about code signature or run-time behavior of similar malware. Instead, VindiCo is designed to be generic and agnostic to the implementation of SpyCon.

5.1.1 Related Work

Malware Detection Techniques: Several techniques have been proposed for malware detection and can be broadly categorized into two groups. (1) *Code signature-based approach* [EKK11, GZZ12, EOM11, LLW12, ARF14] detects stealthy behavior based on the code flow. (2) *Behavior-based approach* [YY12, ZJS09, KMP11, EGH14] performs information leakage detection in execution time but tackling the issue from different layers of an operating system. DroidRanger [ZWZ12] points out that an app can download binary payload in the runtime, which code-signature based approach can never diagnose its intention but raise a warning of a potential hazard.

Nevertheless, VindiCo is distinct from the above techniques because a malicious app can learn sensitive information based on the adaptations made by another app.

Malware Mitigation Techniques: Different mitigation techniques have been proposed,

including sensory value perturbation [BRS11, HHJ11, CSR14], finer-grained permission control [JMV12, NKZ10], and permission recommendation systems [AH13]. πBox [LWG13] and SemaDroid [XZ15] introduce a notion of *privacy budget* and seek a balance between utilization and privacy sacrifice. VindiCo shares the similar goal to quantify the degree of information being leaked and choose an appropriate data perturbation method and accordingly a mitigation magnitude based on the desired degree of data distortion.

5.1.2 Contribution

In this chapter we purpose VindiCo a safeguard against SpyCon. In particular, our contribution can be summarized as follows:

- We design, implement, and demonstrate VindiCo, a safeguard against SpyCon. We introduce a novel detection mechanism (we named it information-based detection) along with two genres of mitigation policies. We re-examine the proposed mitigation policies against the developed SpyCon to assess it and show how its performance decreases after applying mitigation policies.
- We evaluate VindiCo through extending Android Open Source Project (AOSP)[aos] with a new layer that supports the purposed detection mechanism and mitigation policies.

The remainder of this chapter is organized as follows, Section 5.2 illustrates the architecture of the proposed VindiCo safeguard. Implementation details of VindiCo are shown in Section 5.3. Finally, we provide an experimental evaluation of VindiCo in Section 5.4.

5.2 VindiCo System Architecture

In VindiCo, we focus on the general question of how to design software mechanisms that can detect and mitigate SpyCon.

5.2.1 Threat Model and Design Objectives

Motivated by the observations from the previous section, we define our threat model as follows.

[**T1**] **No prior signature or behavior**: Since SpyCon is a new class of spyware, we assume neither prior knowledge of code signatures nor prior recorded suspicious behaviors exist.

[**T2**] **Unknown inference algorithm**: We assume no knowledge of the algorithms used by SpyCon to infer the sensitive information.

[**T3**] **Access to APIs**: We assume that SpyCon has access to the APIs needed to monitor user behavior. Our study in the previous section shows that many of such Android APIs do not require permissions.

[**T4**] **Knowledge of Existence of VindiCo**: Finally, we assume that SpyCon is aware of the existence of VindiCo. That is, SpyCon has full knowledge of VindiCo detection, and mitigation algorithms and SpyCon may adapt its behavior accordingly.

T1 and T2 imply that traditional malware detection schemes (e.g., [XZS10]) are not adequate in our problem setup. Hence, a new SpyCon detection scheme has to be designed without possessing any prior information (signature or behavior) of such spyware. In particular, we propose the following design objectives in VindiCo:

[**O1**] **Generic Detection**: VindiCo should be able to detect any possible context-aware adaptation based spyware without prior knowledge of spyware signatures or behavior.

[**O2**] **Mitigation**: VindiCo should be able to mitigate the impact of a possible SpyCon without affecting the authentic context-aware application performance.

[**O3**] **Performance**: VindiCo should be lightweight and add minimal execution overhead.

Motivated by these three design objectives, we designed VindiCo as discussed in this section in three main blocks: Context-adaptation registration, Detection Engine, and Mitigation Engine. The details of each building block are given in the subsequent sections.

5.2.2 Context-adaptation Registration

To protect adaptions made by context-aware apps, VindiCo needs to know the context-action relations within the apps.

One action in an adaption can be made based on several context, and one context can be shared by different actions. For example, a user may use a camera app and another app for listening to music in the background while running. Upon detecting that the user is running, the camera app adjusts the hardware camera focus while the music app changes the music playlist. Both actions from these two apps, i.e., changing camera properties and altering music playlist, are made based on the same context, i.e., user's physical activity. Providing the context-action relations is essential for VindiCo to monitor how likely a SpyCon is inferring associated context.

To automatically retrieve the context-action relations, we use FlowDroid [ARF14] (which augments the Java code analysis tool Soot [soo]) to analyze the call flow graph of the context-aware app and extract all relations between the setter APIs and getter APIs. The final output is an XML file that maps context (getter APIs) to actions (setter APIs). We call this XML file a *registry file*. The developer can then view the automatically generated XML file to verify or modify it before being used by VindiCo.

In the registry file example shown in Figure 5.1, there are two `adaptation` policies. Each *adaptation* represents one context-action relation. The first adaptation makes decisions based on GPS location and updates ringer mode and alarm volume accordingly. In the second adaptation, the application adjusts the camera settings to accommodate battery capacity, GPS location, and transportation modality.

During application installation phase, VindiCo checks the existence of the registry file. If the file exists, VindiCo considers the associated application as a context-aware app and will offer detection and mitigation mechanisms to protect this app from potential context exposures against any other suspicious SpyCon.

```
<adaptation>
        <context> getGPSlocation() </context>
        <action>  setRingerMode()  </action>
        <action>  setAlarmVolume() </action>
</adaptation>
<adaptation>
        <context> getTransportationModality() </context>
        <context> getBatteryCapacity()        </context>
        <context> getGPSlocation()            </context>
        <action>  setFocusMode()              </action>
        <action>  setJpegQuality()            </action>
        <action>  setSceneMode()              </action>
</adaptation>
```

Figure 5.1: Snippet of a VindiCo registry file.

5.2.3 Information-Based Detection Engine

It follows from assumptions T1 and T2 in our threat model that existing signature-based detection mechanisms are not able to detect the developed SpyCon. Similarly, behavior-based detection techniques are not suitable to detect such spyware since no prior behavior is known. Exacerbating the situation, the behavior of SpyCon is coupled to the behavior of the authentic context-aware apps. That is, if the authentic context-aware app triggers actions more frequently, SpyCon will monitor the actions by calling the getter values APIs for these actions more frequently. This behavior coupling hinders the usability of the behavior-based detection techniques.

The idea behind the information-based detection is to keep track of the ability of SpyCon to infer the context through monitoring actions triggered by this context. Recall that we do not have any prior knowledge or assumption on how SpyCon performs its inference (T2). To this end, we draw on the literature of information theory and leverage *mutual information* to quantify the amount of correlation (or dependence) between two random variables. In

our scenario, we use the mutual information between *context* and *action* as a metric to measure how certain a SpyCon can infer context from observed actions. Mutual information provides a theoretical bound on the inference capability of *any* learning algorithm. Generally speaking, the lower the mutual information between context and actions is, the smaller the accuracy *any* inference algorithm can get. Push into one extreme, if the mutual information is zero, then *no* algorithm can infer context from monitored actions.

Based on this intuition, our detection engine continually tracks what actions have been monitored by each app, computes an estimate of mutual information between the actual context and those actions monitored by this app, and assigns a *suspicion score* accordingly. This score is then passed to our Mitigation Engine, as an indicator of the magnitude of countermeasures, which aims at reducing the amount of information possessed by suspicious apps.

5.2.4 Mitigation Engine

Once the *suspicion score*, assigned to each possibly running SpyCon, increases beyond a certain threshold (named alarm threshold), the Mitigation Engine informs the user with the possibility of having a SpyCon and asks the user permission to apply countermeasures against the suspicious app. Upon the user consent, the Mitigation Engine seeks a general way to hinder the SpyCon's capability of revealing user context. While completely blocking all side-channels may not be practical, our goal is to reduce its bandwidth drastically. This process needs to be done without any prior assumption on the type of inference algorithms used by SpyCon (T2). There are two solution regimes to achieve this: 1) imposing delays so that a SpyCon cannot get the latest action updates in real time, and 2) tearing down the correlation between the actions monitored by SpyCon and the associated context. We call the mitigation method in the first regime *delay* and introduce three different mitigation methods in the second regime: *suppression*, *row-masking*, and *feature-masking*.

Before delving into the details of various mitigation algorithms, we would like to stress the following facts:

- Complementary and adaptive mitigation: The two regimes of mitigation methods achieve different goals and can complement each other. In practical scenarios, we combine both delay mitigation with one in the second regime which reduces the mutual information. VindiCo does not assume any mitigation can always outperform the others. In fact, a mitigation technique may be effective only in a particular situation. Hence, VindiCo starts by selecting a mitigation method and applies it accordingly. Next, VindiCo tracks the effectiveness of the imposed mitigation technique by constant monitoring of changes in the suspicion score. If a SpyCon can still survive under current mitigation treatment, VindiCo increases the mitigation magnitude first and eventually switches to a different method.
- The Scope of mitigation: The mitigation is applied on a per-app basis instead of a system-wide configuration. Hence, well-behaved apps (including the protected context-aware app) can receive correct action values. Therefore, mitigations cause no negative impact on context-aware apps.

The rest of this section we explain the details of the mitigation techniques we have in VindiCo.

5.2.4.1 Mitigate by Delay

As shown in Section 4.2.4, monitoring the phone setting changes updated by the context-aware app can reveal the semantics of the user location and thus infer user's schedule. Our first strategy is to let VindiCo *delay* the action values so that a SpyCon can only get an outdated context instead of the latest one. VindiCo chooses a delay of d minutes which is selected randomly to prevent the SpyCon from revealing the delay.

5.2.4.2 Mitigate by Suppression

With the previous mitigation method, even though SpyCon may not be able to get the real-time update from delay mitigation, it can retrofit users' daily routine by aligning with a *priori* knowledge, for example, when a person usually goes to school or work. Once the personal schedule is derived, this mitigation will have no effect in the future. The fundamental

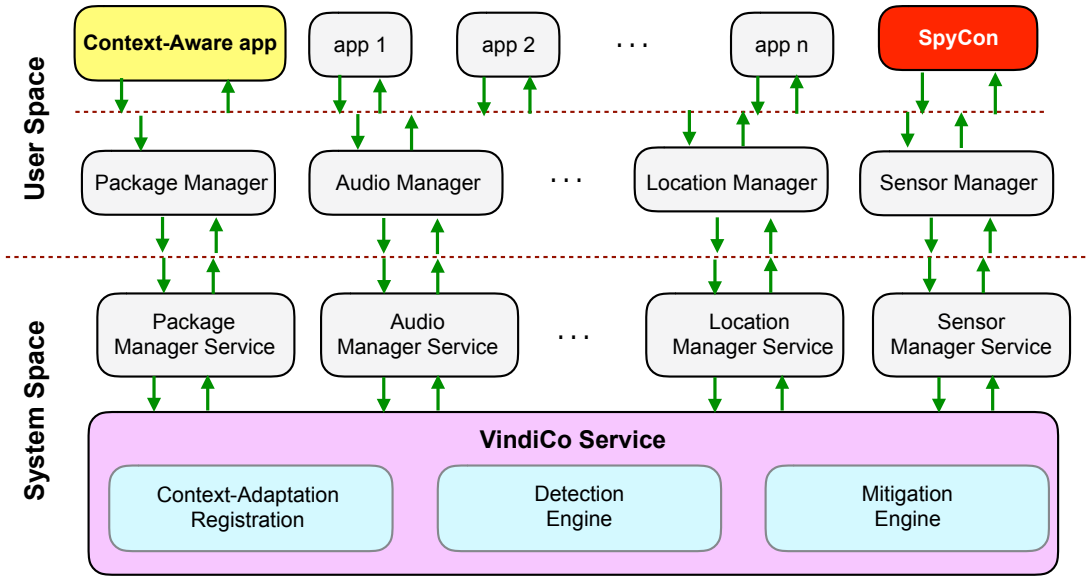


Figure 5.2: VindiCo architecture. The context-aware application is registered in VindiCo by context-adaptation registration module. The behavior of the context-aware app is monitored, and a possible SpyCon is detected by the Detection Engine. Adequate mitigation technique is then taken by the Mitigation Engine.

reason is that merely imposing a delay will not reduce the mutual information. Suppression mitigation, on the other hand, is designed to decrease the information a SpyCon can harvest. The main idea of this mitigation technique is to give SpyCon false adaptation actions by returning action values associated with another context.

VindiCo suppression mitigation module randomly chooses one of the latest k recorded values for the action. The number k is the mitigation magnitude of this method.

5.2.4.3 Mitigate by Masking

Another mitigation to increase obfuscation is to mask some action values, i.e., returning zeros. We explore two variants of the masking approach: *row-masking* and *feature-masking*. In row-masking, our system returns correct action values, but with a specific probability p, our system returns 0 to SpyCon for all the action values after context changes. The masking effect takes place until the next adaptation is made. In consequence, SpyCon

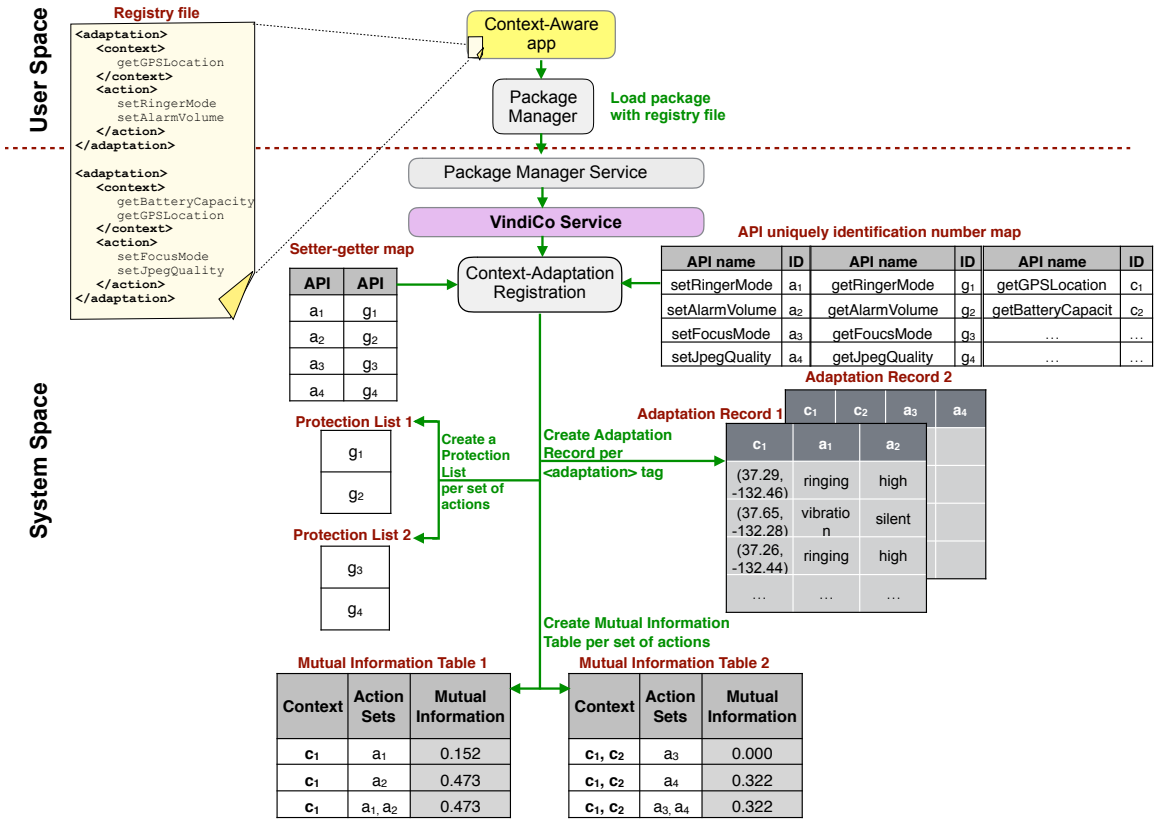


Figure 5.3: Context-adaptation registration. At installation time, VindiCo checks the existence of a registry file and starts processing it accordingly. Registry file processing constructs all necessary data structures that are needed by the detection and mitigation engines.

cannot infer anything during the masked periods, but can still make inferences based on the unmasked observations. The feature-masking approach, in contrast, considers each action value individually. Upon an adaptation is made, each action has a certain probability to be masked. As a result, at any given context change, the SpyCon can only observe partial action values after an adaptation occurs. The decision of masking all actions in row-masking or which action values to be masked in feature-masking depends on flipping a biased coin with a selected probability p, which serves as the parameter of mitigation effectiveness.

The implementation details of the mitigation module are given in Section 5.3.3, and the evaluation of the mitigation techniques is shown in Section 5.4.2.

5.3 Implementation

We extended the Android system layer to add the three main parts of VindiCo as shown in Figure 5.2. In this section, we explain the implementation details of all the three modules.

5.3.1 VindiCo Context-adaptation Registration

In this section, we explain, with an example in Figure 5.3, the data structures used by VindiCo to track the information possessed by possible SpyCon apps.

5.3.1.1 Adaptation Record

The purpose of this data structure is to keep track of the actions taken by the authentic context-aware apps and the changes in the context that triggered such actions. An adaptation record table is created for each *adaptation* in the registry file. Upon changes in either context or actions, VindiCo appends a record to these tables consisting of both the context and the actions along with a timestamp. Figure 5.3 demonstrates two adaptation record tables corresponding to two *adaptation* tags. Initially, these tables are empty and are filled as the context-aware app decides to perform a new adaptation. These tables are then used by the *Detection Engine* to estimate the amount of *information* that can be potentially retrieved if some actions are monitored.

5.3.1.2 Protection Lists

Most of the setter APIs in the Android Framework managers have corresponding getter APIs, and a malicious app can use these getter APIs to monitor any change in actions triggered by authentic context-aware apps. For example, if an authentic context-aware app uses `setRingerMode()`, a SpyCon can use `getRingerMode()` to track the action triggered by this authentic app.

VindiCo summons all getter APIs (which can potentially leak user context) into a list called *Protection List*. Take the registry file example in Figure 5.1, the corresponding Pro-

tection Lists after parsing the registry file are shown in Figure 5.3. In particular, two lists are constructed for the two adaptation tags. The first Protection List includes two functions, `getRingerMode()` and `getAlarmVolume()`. Similarly, the second list contains the corresponding getters of the three mentioned functions.

5.3.1.3 Mutual Information Tables

As explained in Section 5.2, the Detection Engine depends on information-based technique. VindiCo generates one *Mutual Information Table* per *adaptation* tag in the registry file. The table consists of the power set of all the actions, and each row stores the mutual information of the subset of the actions and the context. Figure 5.3 gives two examples of Mutual Information Tables. All the entries are initialized to zeros and will be updated by the Detection Engine.

5.3.2 VindiCo Detection Engine

The Detection Engine is a core module in VindiCo that implements the information-based detection mechanism. We provide the details of our implementation with an illustration of the detection flow in Figure 5.4. In particular, the information-based detection engine works as follows:

5.3.2.1 Step 1: Tracking Context-Aware App Behavior

The first step is to keep track of all the changes in context. Whenever a change occurs in a context, VindiCo timestamp such change and records the timestamp along with all the actions (taken by context-aware apps) and the corresponding context that caused these actions. This step is performed through filling in these values in the *Adaptation Record* table.

Note that we have a separate Adaptation Record for each map between context and actions. Therefore, even if a particular action may have been triggered by multiple changes in different context, we will treat each context separately, and all the corresponding Adaptation Record tables will be updated

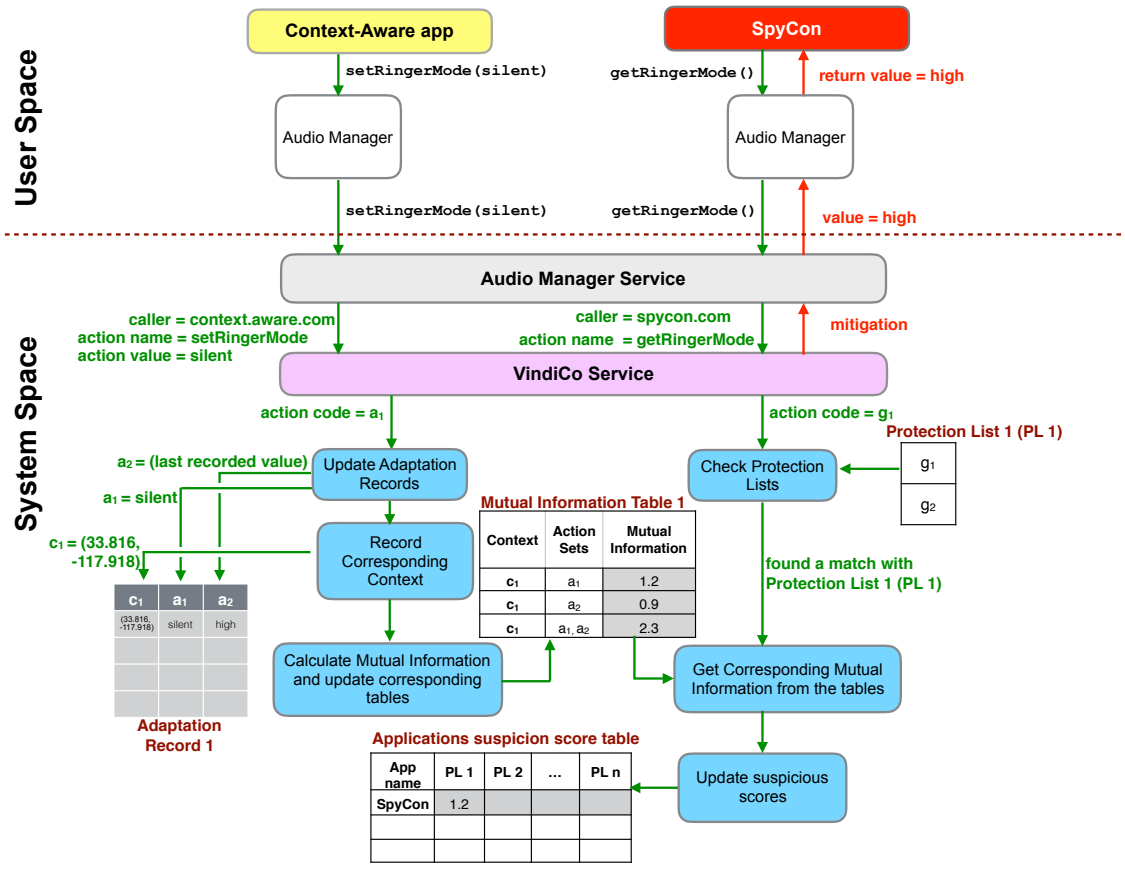


Figure 5.4: VindiCo Detection Engine. When a context-aware app calls a *setter* API to adapt to user context, VindiCo Service intercepts the call and checks if this API call is in the *Adaptation Record*. Next, the mutual information algorithm updates the corresponding Mutual Information Table based on the new data recorded in the Adaptation Record. Whenever an app calls a *getter* API that matches one of the API in the *Protection Lists*, the mutual information corresponding to the *getter* API is retrieved and assigned to this app as a suspicion score.

5.3.2.2 Step 2: Computing Mutual Information

The next step is to calculate the mutual information based on the time series stored in the *Adaptation Record* tables. Since we do not know a priori which (set of) actions are going to be used by a SpyCon for adversarial inferences, we compute and store the mutual information between the context and the power set (all different combination) of all actions.

These values are then stored in the *Mutual Information Tables* and are updated whenever new entries are added to the *Adaptation Record*.

5.3.2.3 Step 3: Monitoring Suspicious App Behavior

The last step is to use all the *Protection Lists* to track the behavior of suspicious apps. Whenever an app invokes a *getter* API that can be used to monitor an action (i.e., an API in the *Protection List*), VindiCo starts to suspect this app by assigning a *suspicion score* to this app. The value of this *suspicion score* is equal to the mutual information associated with the *getter* API (invoked by the app) which is stored in the *mutual information tables*.

Since the same app may have called several *getter* APIs belonging to different *Protection Lists*, VindiCo associates a set of *suspicion scores* (instead of just one *suspicion score*), one per *Protection List* in a table that is called *applications suspicion scores* table (see Figure 5.4).

5.3.2.4 Convergence of the estimated mutual information

An important aspect that is related to the use of mutual information is the convergence of estimated mutual information to the actual mutual information from the data samples accessed by VindiCo. That is, it is well known that calculating an estimate of mutual information with enough precision requires the access to multiple samples from both context and triggered actions[1]. Hence, one can argue that a malicious spyware—which is aware of the existence of VindiCo as per assumption T4 in our threat model—may be tempted to reduce the frequency for which it monitors the actions with the goal of decreasing the number of samples used to estimate his mutual information and hence *deceives* VindiCo by reducing the associated *suspicion score*[2]. To avoid such situation, we associate mutual information to actions instead of apps. That is, whenever an action takes place, we update the amount of mutual information between this particular action and all the context associated with it. At

[1]An estimate of the mutual information converges to actual mutual information asymptotically.

[2]On average the actual mutual information is the upper bound for the estimate mutual information.

any time point, once an app monitors this action, we copy the mutual information associated with this action to the app. Therefore, regardless of how frequent an app monitors actions, we associate to it the mutual information based on all samples in history—recall that the number of these samples are not controlled by the malicious app—not only on those to which it has access.

5.3.3 VindiCo Mitigation Engine

When an API is called in Android, an application ID is passed along with the API call to the system layer. The mitigation engine checks if this application ID has an entry in the applications suspicion scores table. Afterward, the Mitigation Engine checks which *Protection List* this API call is part of to know the corresponding suspicion score. As mentioned in Section 5.2.4, VindiCo does not presume any mitigation treatment outperforms the others, and opportunistically selects a method and see the effect and gradually increases the mitigation. If the suspicion score cannot be effectively decreased, VindiCo will switch to another mitigation method.

In order not to affect the performance of the API call, we cache the decision taken by the mitigation engine on this API call for this application ID for future calls, and in a separate thread, the mitigation engine checks the effect of mitigation on the mutual information to update this cache. The overhead on the API call is shown in Section 5.4.3.

5.4 Evaluation

We implemented the proposed VindiCo by modifying the Android system image for platform 5.0.1 API 22 [aos]. The developed system is installed on Nexus 5 phones. VindiCo adds 44 MByte to the original Android 5.1 system image resulting in an increase of 4.5% of the image size (original image size is 998 MByte). In this section, we show the performance of the proposed VindiCo.

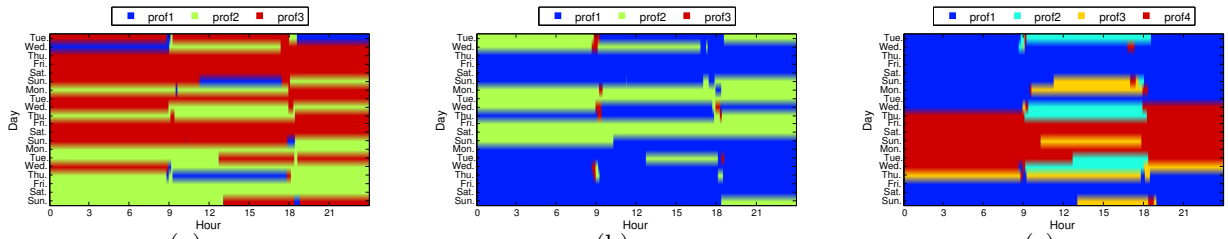


Figure 5.5: Profile timeline of user #2 after VindiCo applies the mitigation techniques. (a)Suppression mitigation *(3 rows)* (b)Row-masking mitigation *(p=0.4)* (c)Feature-masking mitigation *(p=0.4)*

5.4.1 Experiment 5: Performance of Information-Based Detection

5.4.1.1 Detection Accuracy

In this experiment, we investigate the effect of choosing the *alarm threshold* (the threshold on the *suspicion score* above which an app is considered malicious) on the performance of the proposed information-based detection algorithm. In particular, we evaluate the detection performance regarding both the false positive rate and false negative rate. In this context, an application is considered *malicious* whenever its clustering accuracy exceeds the *baseline accuracy* by blind guesses defined in Section 4.2.4.2. A false positive flags the case when the information-based detector claims that an application possesses enough information to accurately identify the user behavior with accuracy more than the *baseline accuracy* while the real accuracy of the app is indeed lower than the *baseline accuracy*. A false negative is defined similarly. Similar to the previous experiment, we focus again on the case when SpyCon is monitoring phone settings changed by a location-based context-aware app (Tasker/Locale) and when these settings are changed by a calendar events-based context-aware app (Silence).

Figure 5.7 reports the false positive and false negative rates obtained from 8000 data points collected from our user studies versus different *alarm thresholds*. For the extreme case, when the alarm threshold is set to zero, any application that possesses any *slight* information is marked to be malicious by the information-based detector. This leads to detecting any SpyCon (i.e., false negative rate = 0) on the cost of an excessive false positive rate (where

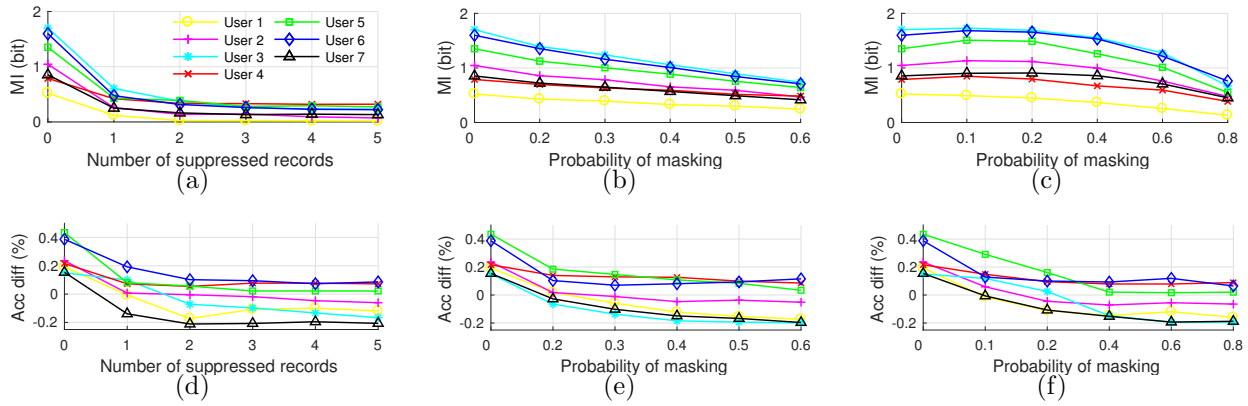


Figure 5.6: Mutual information (MI) and clustering accuracy difference (Acc diff)—with respect to the baseline accuracy—after applying different mitigation methods. When mitigation magnitude increases, both mutual information and accuracy decrease. (a) MI by suppression (b) MI by row-masking (c) MI by feature-masking (d) Accuracy by suppression (e) Accuracy by row-masking (f) Accuracy by feature-masking

all benign applications are marked malicious as well). As the alarm threshold increases, the information-based detection becomes less aggressive leading to a significant decrease in the false positive rates while sacrificing the ability to detect malicious apps (reflected by the increase in the false negative rates). The results reported in Figure 5.7 suggest that an alarm threshold of 0.65 leads to a compromise between false positive and false negative rates (false positive rate = 0.1 and false negative rate = 0.15).

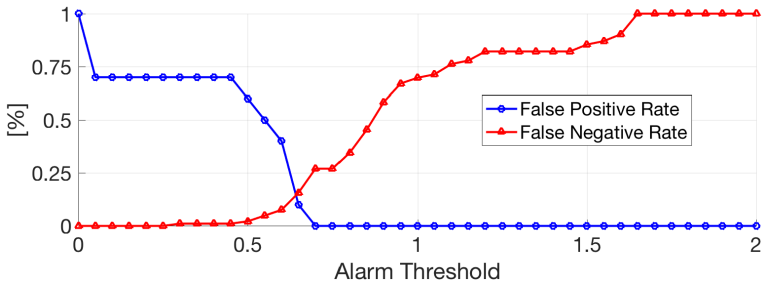


Figure 5.7: Performance of the VindiCo information-based detector (in terms of false positive and false negative rates) versus different alarm thresholds.

5.4.1.2 Detection of SpyCon in the Market

Next, we ran the proposed information-based detection algorithm (with alarm threshold set to 0.65) against the 45 real applications from the market used in Experiment 4. Recall that Experiment 4 asserts most apps can identify the user context with accuracy more than 80% which in turn indicates that each app possesses high amount of information. Running the proposed information-based detection algorithm against these applications results in suspicion scores that are greater than the alarm threshold for *all* the 45 apps. Therefore, VindiCo marks *all* these apps as malicious reflecting the fact that these apps possess enough information to leak sensitive details about the user behavior. For space limits, we report here the suspicion score of the extreme cases (the top two and bottom two apps in terms of accuracy in Experiment 4) for the SpyCon that monitors changes based on location as follows: 360 Security Antivirus (0.974), Clean Master (0.971), Pinterest (0.722), and Madden NFL Mobile (0.784). Similar suspicion scores are obtained when SpyCon is monitoring changes based on calendar events.

5.4.2 Experiment 6: Performance of Mitigation Algorithms

To evaluate the performance of the proposed mitigation technique, we applied the proposed mitigation methods on SpyCon. We plot the profile timeline of the same user (user 2) after applying the mitigation with a one parameter value in Figure 5.5. Comparing the results in Figure 4.1(c) and Figure 5.5, we notice the distortion in the patterns of user profile compared to the case when no mitigation is applied.

To better judge the effect of the proposed mitigation algorithms, we plot the accuracy of the clustering algorithm developed when different mitigations are applied along with the corresponding mutual information in Figure 5.6. These results show how the gained accuracy by SpyCon above the baseline accuracy—to identify the user profile—drops in general to the baseline accuracy (as shown in the accuracy difference in Figure 5.6), and for some users, it even drops to less than the baseline accuracy. Moreover, and as expected from the theoretical underpinnings of mutual information, whenever mutual information decreases—

as a consequence of the mitigation parameter (number of records k in suppression case and the probability of masking p in the row-masking and feature masking case)—the accuracy of the developed SpyCon must decrease (on average).

We also observe that the suppression method leads to a sharp decrease in the mutual information with a fast saturation whenever $k > 2$ as shown in Figure 5.6(a). In contrast, the other two proposed mask mitigation methods (row-masking and feature-masking) lead to a gradual decrease in the mutual information as the probability of masking p increases (Figure 5.6(b), 5.6(c)). This gradual degradation provides more flexibility for VindiCo to adjust to a desired mutual information.

Finally, though we expect mutual information should monotonically decrease when the mitigation is stronger (higher masking probability), 5.6(c) shows that mutual information goes higher when masking probability $p = 0.1$. One possible explanation is that noisy settings are cleared with this configuration, causing higher correlation between mitigated data and user profile. However, the mutual information decreases sufficiently when larger masking probability is applied (i.e., $p >= 0.4$).

5.4.3 Experiment 7: Timing Analysis of VindiCo

The execution time is obtained using Android traceview [And]. Table 5.1 shows the CPU execution time when the complexity of the context-aware app increases, which is majorly reflected by the number of adaptation tags in the registry file. Parsing and processing the registry file take approximately $6ms$. Fortunately, this overhead takes place only during the installation of a new package on the phone. On the other hand, at runtime, VindiCo adds negligible overhead (less than $0.1ms$) which is approximately 3% increase from the average execution time of the original API call.

5.4.4 Experiment 8: Effect on Benign Applications

Our results in Experiment 4 show that false positives (mistakenly assigning a suspicion score to a non-malicious app) are indeed possible. In this experiment, we study the effect of

	Overhead in CPU Time (ms)			Description
	2 adapt	**3 adapt**	**4 adapt**	
Parsing registry file	3.186	3.186	3.187	Overhead takes place during the installation time of the application.
Registry file processing	1.48	2.22	2.96	Construction of the necessary data structures. Overhead takes place while loading the application for the first time after the application is launched.
API call in context-aware apps	0.066	0.076	0.076	Tracking the values of context and action from the context-aware application and filling the *Adaptation Records* at runtime. Overhead takes place whenever an action API is called by the authentic context-aware application. On average, the original API call consumes 2.56 ms, and hence the overhead is 2.9%.
API call in other apps	0.056	0.090	0.095	Overhead when a `get` API is called from one of the *Protection List* by other apps. On average, the original API call consumes 2.56 ms, and hence the overhead is 3%.

Table 5.1: Timing analysis of VindiCo against increasing complexity of context-aware apps measured by number of adaptation tags in the registry file.

applying the mitigation techniques on the functionality of different apps. We emphasize the fact that VindiCo applies mitigation *only* after notifying the user and getting his consent (as discussed in Section 5.2.4).

We examine the effect of changing the return value of phone settings on Skype and Facebook when mitigation is applied. In particular, we split our study into (i) effect on the app's main functionality and (ii) effect on user experience. To that end, we recruited

five participants and asked them to use Skype and Facebook while VindiCo is applying different mitigation techniques and the users are asked to compare their experience with the case when no mitigation is applied. We note that all the participants could not notice any changes in Skype and Facebook main functionality (audio/video calls in case of Skype and sharing/reading posts and videos in the case of Facebook). This is a direct reflection of the fact that VindiCo only modifies the phone setting values shown in Table 4.1 which do not directly affect these functionalities. On the other hand, several secondary effects were reported by the user study participants. For example, while Facebook was configured to stream videos whenever connected to WiFi automatically, it was noticed that, in many time instances, Facebook does *not* stream videos even with WiFi connectivity is present. This is a consequence of VindiCo suppressing the values of the WiFi connectivity and misleading Facebook about the current WiFi connectivity. Another noticed secondary effect occurs when the user taps the "Like" button. The expected behavior from Facebook is to use the current volume setting to play the notification sound associated with the "Like" button. However, some of the users noticed that Facebook plays the notification sound using an incorrect volume which leads to user discomfort especially when some music is also playing in the background. This is accounted for the fact that VindiCo suppresses the values of the volume settings before being read by Facebook disallowing Facebook from using the right setting.

5.5 Conclusion

We designed VindiCo, a safeguard which protects authentic context-aware applications against leaking private information via this side-channel. VindiCo employs a general detection technique based on mutual information algorithm which is agnostic to implementation details of context-based spyware and uses three mitigation techniques to hinder the performance of SpyCon, which are delaying, suppressing, and masking. An end-to-end use case has been shown to demonstrate the effectiveness of the proposed VindiCo architecture by having a SpyCon monitoring an authentic context-aware phone setting application. Our mitiga-

tion techniques have shown a degradation of SpyCon inference accuracy from 90.3% to the baseline accuracy and by only adding negligible overhead (3%) on the API call performance.

Future work includes studying the effect of collaborative SpyCon. In such scenario, multiple apps collect different (or partial) information from the proposed side-channel and fuse them to leak sensitive user information while having small individual mutual information and hence bypass the information-based detection algorithm. Another direction is to extend VindiCo beyond context-aware applications by learning the context-actions relations at runtime without the need of a registry file that is generated at installation time.

Part III

Personalization of Pervasive Autonomy

CHAPTER 6

Sentio: Driver-in-the-Loop Forward Collision Warning Using Multisample Reinforcement Learning

6.1 Introduction

The dramatic increase in sensing capabilities has opened the door for a multitude of new applications in the context of Advanced Driver Assistance Systems (ADAS). ADAS are developed to adapt the vehicle systems according to the vehicle and/or human context in an attempt to enhance both the vehicle safety and the driving experience. In these context-aware systems, sensor information collected from various sensors are fused together to infer the current context (or state) of the system and adapt the system functionality to match the system's and the user's state. Lane departure warning, front collision warning, and driver drowsiness detection are just examples to name a few [LCG15, LY15, YZL15, CZZ15].

While ADAS is one of the fastest-growing segments in automotive electronics [The], most ADAS systems—except for driver drowsiness detection—focus on adapting to the state of the environment surrounding the vehicle without taking the driver and passenger state into consideration. Automatic lighting, adaptive cruise control, automatic braking, lane departure warning, and front collision warning are all among the most utilized ADAS systems that fall into this category. However, due to the increase in sensing capabilities, mobile systems and wearables have shown substantial success in inferring different human contexts [SRN12, MJV13, Muk15]. Given the state of the environment surrounding the vehicle (inferred using the vehicle sensors) along with the state of the driver (inferred using mobile/wearables systems), we ask the question of how to design algorithms that can make use of this information and provide a personalized driving experience.

In this chapter, we focus on the Forward Collision Warning (FCW) system. Standard FCW system uses radars to detect vehicles or obstacles in front of the car. The system measures the time-to-crash based on the distance and the relative velocity of the front object and, if the time-to-crash is below a certain threshold (signaling a possible risk of collision), it sounds an alarm and displays a visual alert, prompting the driver to apply the brakes.

Unfortunately, to design FCW systems, automotive makers fix a certain threshold based on multiple experiments capturing the average human behavior and response. However, as studied by the US Department of Motor Vehicles, emotions (both negative and positive) can cause distraction and delay driver's response by a few seconds [DMV]. Moreover, arguing with a passenger could be even more distracting than talking on a cellphone while driving [LS13]. Significant findings showed that contentious conversations to be more emotionally taxing, and the ability to drive was significantly compromised [LS13].

These observations motivate the need to design a driver-in-the-loop FCW system which adapts to individual drivers' responses based on their cognition context. We argue in this chapter that by carefully designing the FCW system, a driver-in-the-loop FCW could adapt to the driver state and trigger the visual alarm early enough to accommodate the latency in the driver response.

Aided by the current developments in building machine learning based agents, recent literature in designing and building context-aware systems focused on using labeled data to train a machine learning classifier that can "infer" the human state. A smart FCW system not only needs to "infer" or "learn" the human preference but also continuously "adapts" and "takes actions" in the form of visual warnings.

This feedback property opens the door to design a "Reinforcement Learning (RL)" based agent to build the proposed driver-in-the-loop FCW system. Unfortunately, applying standard RL algorithms like Q-learning, in the context of FCW, faces several challenges. In this chapter, we identify these challenges and propose a modified Q-learning algorithm, named the "multisample Q-learning", to address these challenges. The proposed algorithm is then used to build a driver-in-the-loop FCW that adapts its behavior, at runtime, to the atten-

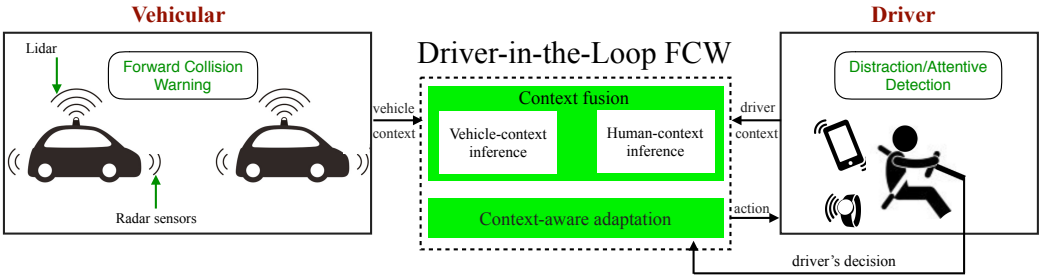


Figure 6.1: Sentio architecture. The context of the driver (attention) is inferred using the data collected by phone and wearables. The context of the vehicle (vehicle speed) and vehicle environment (distance to front collision) is collected by the vehicle sensors. The adaptation actions are then personalized based on the driver's decision.

tion level, preference, and response time of human drivers. This online adaptation leads to a personalized driving experience as reported by several human drivers.

6.1.1 Related Work

6.1.1.1 Personalized Forward Collision Warning

Learning the driver behavior model is an essential step in personalized FCW. However, collecting a sufficient data has been the primary setback of these models [MDE13]. Using statistical modeling to reduce the required data set has been reported in [MDE13].

Unfortunately, learning fixed parameters for a driver model, using offline data, can increase the false alarms rate if the driver behavior changes at runtime, which was not addressed by the proposed statistical modeling framework [MDE13]. Real-time parameter identification of driver behavior is proposed in [WYL16]. By assuming a fixed parametric model for the driver behavior, the work reported in [WYL16] uses online data to update this model and adapts the FCW threshold according to the learned parametric model. This adaptation reduces the false warning rate if the driver behavior changes. Other tunings for the warning threshold have been addressed, by using GPS data [NGC16], drivers' expected response decelerations (ERDs) [WCZ16], and drivers' longitudinal braking behavior using GMM [SDS17].

Our work differs from previously reported work in the following sense. First, unlike [MDE13], we avoid designing offline thresholds and focus on the problem of continuous adaptation to the driver behavior. Unlike the other work on online adaptation [MDE13, WCZ16, SDS17], we do not restrict our algorithm to particular parametric models. Instead, we use an entirely data-driven approach to adapt to the human behavior. Second, all previously related work focuses on using only the information collected by the vehicle sensors (e.g., Radars) and ignore the fact that a rich set of sensory data obtained from mobile/wearable systems that can help in identifying the state of the driver.

6.1.1.2 Human-in-the-loop Context-Aware Automotive Systems

The role of humans in automotive systems can be divided into three domains [OT16], (1) inside the vehicle cabin, (2) around the vehicle, and (3) inside the surrounding cars. While there has been a vast amount of work targeting each domain, we focus in this section on those related to the first category. We classify the work in this domain into two subcategories namely (i) intent detection and (ii) state detection. The work reported by [LCG15] falls in the first subcategory, intent detection, in which a hidden Markov model is used as a probabilistic model that captures the intent of the driver. The second category is concerned with the attention of the human agent. The work reported by [AKK13, TST14] falls in this category. Other reported results in this category include studying the driver behavior to measure his level of fatigue through analyzing images of the driver [ZJ04, RLB04, DB08], distraction of the driver [RLB04] and inattentiveness/distraction detection [AKK13, TST14]. Similarly, the work reported in [SJP15] actively monitors the driver to detect his cell phone use. Detecting drunk drivers and calling the police for help has also been studied in [DTB10].

Our work builds on top of recent successes reported in the second subcategory. We aim to design algorithms that make use of the inferred human state to design a personalized FCW that adapts to the driver state.

6.1.2 Contribution

Sentio aims at putting the human state and preference into the loop of computation, more specifically, we make the following contributions:

- Designing a modified Q-learning named the "multisample Q-learning" algorithm that addresses inherent challenges of using the standard Q-learning algorithm in the context of FCW.
- We use the proposed multisample Q-learning to develop an RL agent for Sentio; a driver-in-the-loop FCW system. This RL agent continuously monitors the state of the driver and the environment surrounding the vehicle to release the FCW early enough to match the driver attention level and preference (regarding the relative distance between the driver car and other cars).
- We implemented a proof-of-concept of the proposed Sentio system that demonstrates the feasibility of our algorithm. We evaluated Sentio on human drivers using a virtualized simulated environment.

6.2 Sentio System Architecture

A conceptual overview of the proposed driver-in-the-loop FCW (Sentio) architecture is shown in Figure 7.1. The proposed Sentio is divided into three main modules as follows:

6.2.1 Human Context-Inference

The first step towards a driver-in-the-loop FCW is to infer the current state of the human driver. While recently reported work in the literature showed how to infer complex human states [WBS11, JBN12, MKC13], the objective of this chapter is *not* to develop a new human context-inference engine. Instead, and to facilitate verifying the central concepts of the proposed Sentio, we will focus on a single facet of inferring human distraction. In particular, we will make use of recent studies that show how being engaged in a conversation can lead to a high distraction as well [MVP04, LS13].

Therefore, the current implementation of the human context-inference module outputs a binary signal (attentive/distracted) based on the available audio streams collected by the driver's phone. Since the design of such inference engine is *not* the main focus of the chapter, we postpone the implementation details of this module until the Evaluation section (Section 6.6). Indeed, the proposed Sentio can be generalized directly to any other human context-inference engine that may use more complex data streams (e.g., in-vehicle camera streams, ECG, heart rate) collected from various phone/wearables sensors.

6.2.2 Vehicle and Environment Context-Inference

The objective of this module is to utilize the car sensors to infer the context of the car and its surrounding environment. In particular, we are interested in two sensory information: (i) relative distance and (ii) relative velocity which can be directly inferred from the information collected by the car front-facing radars and LiDARs along with the car speedometer. Raw information from these two sensors can be captured by scanning the internal CAN bus of the car through the standard OBD-based CAN scanners.

The final output of this module is a quantized version of the relative distance. All relative distances below 7m are considered as one "dangerous" state[1] (very near to collision), and all relative distances above 14m are considered as one "very far" state[2] in which the system does not need to react. Similarly, the relative velocity is quantized, where all negative relative velocities are considered as one state, signaling the case when the velocity of the driving car is smaller than the leading car and hence a collision will never occur.

6.2.3 Context-Aware Adaptation Engine

The context-aware adaptation engine is the main contribution of the chapter. It is responsible for (i) using both the driver and the vehicle context to—implicitly—infer the likelihood of

[1]Using the 2 seconds rule [New] for safe driving with an average relative velocity of 15 kph (typical deceleration rate for a typical passenger car per second) between the driving car and the leading car.

[2]Maximum recommended safe distance using an average relative velocity of 15kph between the driving car and the leading car [New].

approaching a dangerous state based on the driver attention level, his response time, and his driving preference, and (ii) alerting the driver early enough to avoid a possible crash.

Learning the best timing that best suits the human state is subjective to the human interaction and response to this alert which vary from one human to another. This variation gives rise to two clear challenges:

- **Variation between individuals:** Every human driver has his preference regarding the relative distance he likes to keep between his car and the leading car [WYL16]. When an alert is signaled, the human observes the relative distance and decides, based on his preference, whether to comply with the alert signal by decelerating the car or ignore it [WYL16].
- **Variation within the same individual:** Human driver, in general, can change his preferences over time [WYL16]. In other words, a driver can prefer a different relative distance across time. This variation comes from the fact that other outside daily factors change the driver behavior which can not be entirely comprehended. Similarly, when the human attention level varies, his response time in observing the relative distance and taking action changes as well [CDW03].

To address these challenges, the adaptation-engine continuously monitors the driver reactions to the issued FCW. If the driver ignores the warnings, our engine "learns" that the driver finds this warning false (earlier than the driver preference). If the driver reacts to the issued warning, our engine learns that this warning matches the driver preference. This continuous feedback loop of issuing warnings (actions) and monitoring the driver decision (response) fits naturally within the Reinforcement Learning (RL) paradigm.

In the RL problem, a software agent tries to learn the behavior of an environment by issuing actions and observing the change in the state of the environment with the purpose of maximizing a notion of a total reward. In the context of Sentio, as shown in Figure 6.2, the environment is both the human driver and the vehicle. Hence, the environment state consists of both the human state (attention level) and the vehicle state (relative distance and the relative velocity). The only action that is decided by the RL agent is when to issue

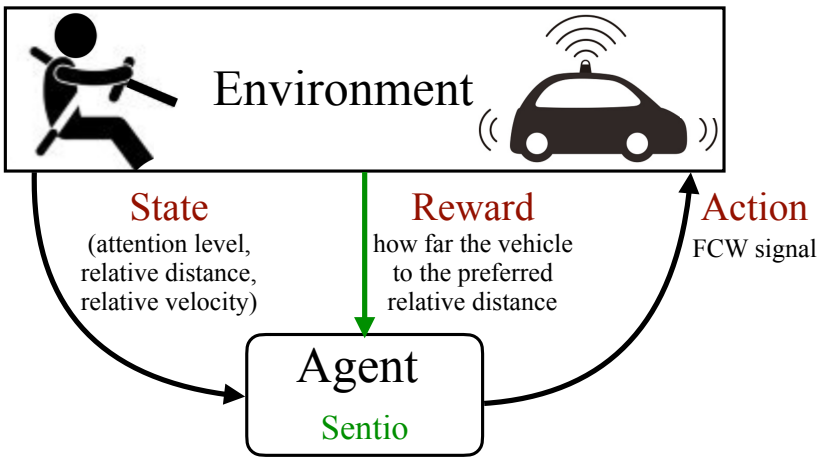


Figure 6.2: Modeling human-vehicular interaction using Reinforcement Learning.

the FCW while the reward of the system is how far the current relative distance is from the driver preferred relative distance.

Therefore, to correctly design an RL algorithm we need to define its three main components namely (i) model for the environment, (ii) reward function, and (iii) learning algorithm. In the context of Sentio, and compared with the standard RL setup we face several challenges in defining these three components that we illustrate as follows:

CH1 Dynamic and time-varying rewards: The standard RL setup assumes the reward function to be time-invariant. That is, applying the same action at the same environment state shall lead to receiving the same reward. Unfortunately, in the context of Sentio, the reward received depends on the driver preference (the safe distance that he prefers to keep between the two cars) which varies over time and is unknown to the RL agent. This time-varying aspect precludes using techniques like Inverse Reinforcement Learning (IRL) which aims to construct the reward function from previously collected data.

CH2 Ignoring actions generated by the RL agent: In the standard RL setup, the actions produced by the RL agent are assumed to have a direct effect on the environment. In other words, it assumes that the environment always obeys the actions taken by the agent which implies that the rewards received reflect the actions decided by

the RL agent. This assumption is violated in our setup since the human driver may or may not comply with the warnings triggered by the RL agent. That is, while the agent may decide to issue the FCW, the driver may choose to ignore the warning and accelerate the car. Hence, the reward that is received by the RL agent does not always reflect the RL agent actions but the decisions taken by the driver.

CH3 Reward time horizon: Finally, the standard online RL setup assumes that rewards received at each step (or execution) of the algorithm are due to the taken actions in this step. Again, such assumption is violated in our setup since the human response (which is in the order of seconds) is not always instantaneous. Therefore, the environment may take several steps (or executions) until the effect of the action is observed and the corresponding reward is received. Moreover, the human response time is unknown, depends on his attention level, and varies across time [CDW03]. While prior work addressed this challenge by adding the extra assumption that a delayed reward follows a Poisson distribution [CGS14], in the case of Sentio, there is no evidence that the driver delay follows such distribution.

In the subsequent sections, we illustrate our solution to each of these challenges.

6.3 Human Driver as a Markov Decision Process

Solving the reinforcement learning problem starts by carefully modeling the environment—the human driver and the vehicle in our case—in a way that captures how the environment state changes in response to applied actions. Moreover, modeling the human driver has to take into account the two challenges mentioned earlier namely the variations between individuals and the variations within the same individual.

Accordingly, in Sentio, we model the change in the human and vehicle state as a Markov

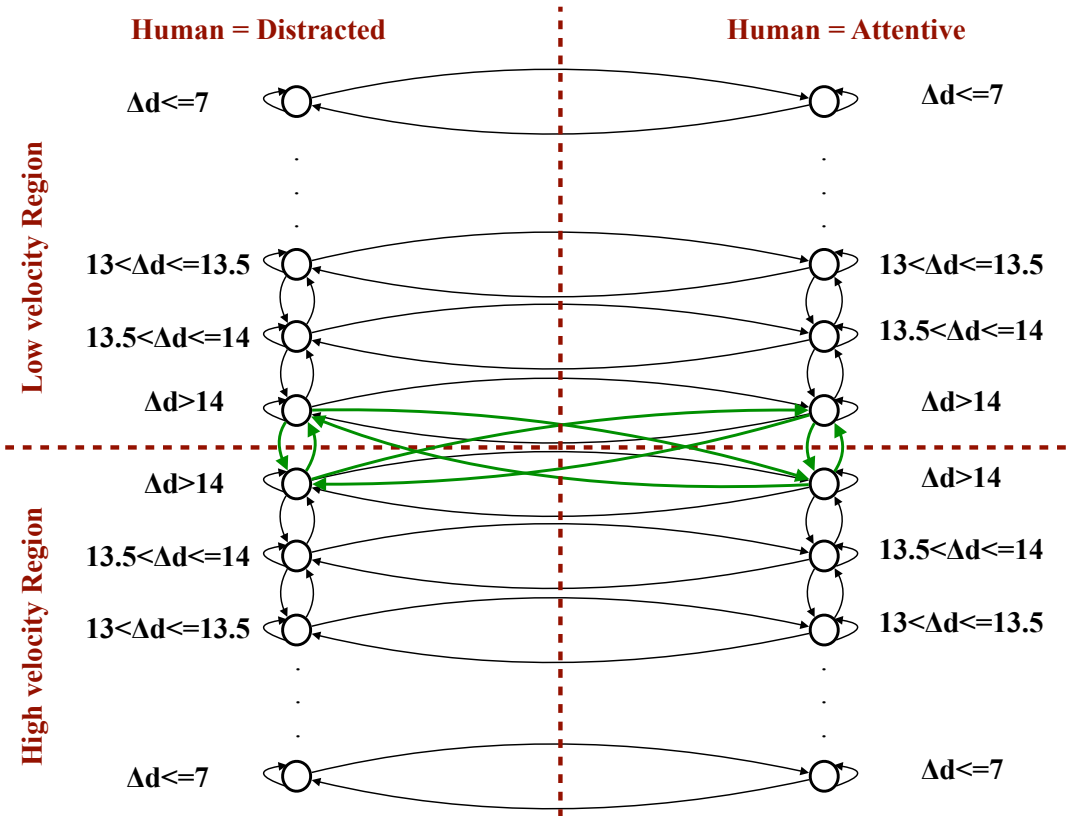


Figure 6.3: A Markov Decision Process model for the human driver and the vehicle. The states of the MDP corresponds to the state of both the driver (attention level) and the vehicle (relative distance and relative velocity). To capture the fact that human behaviors change over time and across different individuals, the transition probabilities between these states are assumed to be unknown and time-varying. To enhance readability, the relative velocity is quantized into two states (low relative velocity and high relative velocity) and only the transitions of the state (distracted, $\Delta d > 14$, low relative velocity) are shown.

Decision Process (MDP) with unknown transition probabilities[3]. The states of the MDP are based on the context of both the driver (attention level) and the vehicle (relative distance and relative velocity), i.e., each state is a tuple $s = (a, \Delta d, \Delta v)$ where a denotes the human

[3]Prior work in the literature prefers to model the human as Partially Observed Markov Decision Process (POMDP) [RV11]. POMDP models reflect the fact that sensor data are not capable of measuring the actual human state but measure only a function of the human state. However, POMDP based RL algorithms are computationally intractable hindering their practical use [Mur00]. To overcome these limitations, we model the human as an MDP, and we rely on the increasing success of sophisticated machine learning inference algorithms in estimating the human state.

attention ($a = 0$ means the human is distracted and $a = 1$ otherwise), Δd denotes the relative distance, and Δv denotes the relative velocity. As shown in Figure 6.3, for the same attention level and relative velocity, the relative distance can only increase or decrease by one level at a time (distance can not jump suddenly from $3m$ to $5m$ without being $4m$ in between[4]) and hence any two states $s = (a, \Delta d, \Delta v)$ and $s' = (a, \Delta d \pm q, \Delta v)$ (where q is the quantization of the relative distance) are connected. Similarly, the relative velocity can only increase or decrease by one level at a time and we connect the states accordingly. We also assume that the human attention level can change at anytime and hence any two states $s = (a, \Delta d, \Delta v)$ and $s' = (a', \Delta d, \Delta v)$ where $a \neq a'$, for the same $\Delta d, \Delta v$, are connected. To capture the fact that human attention may change simultaneously with relative distance and/or relative velocity, any two states $s = (a, \Delta d, \Delta v)$ and $s' = (a', \Delta d \pm q, \Delta v \pm q)$ with $a \neq a'$ are also connected. Indeed, the choice of the quantization of both the relative velocity and relative distance affects both performance and the complexity of the algorithm of Sentio.

This MDP takes as an input the actions taken by the RL agent. The RL action is whether to issue a warning (warning = 1) or not (warning = 0). Upon receiving any of these actions, the MDP changes its state with some probability. To capture the variation between individuals and the variation within the same individual, the transition probabilities of this MDP are assumed to be unknown and can change over time. In Section 6.5, we will make use of a reinforcement learning algorithm that continuously learns and updates these unknown transition probabilities and takes actions accordingly.

6.4 Dynamic & Time-Varying Rewards

The objective of any RL agent is to generate actions that steer the environment from "bad" states into "good" states. The definition of "bad" and "good" is captured by assigning a reward value $R_a(s)$ to each action a in all the states s. Learning algorithms are then used at runtime to monitor the current reward and generate actions that maximize the total reward. In the standard definition of the RL problem, it is assumed that the reward $R_a(s)$

[4]This assumption is valid given a high enough sampling rate.

is time-invariant (static) and is given as an input to the RL agent. However, as we argued before (recall challenge **CH1** in Section 6.2.3), the reward function depends on the driver preference (relative distance between his car and the leading car) which varies across time and is unknown *a priori*.

6.4.1 Reward Function Definition

To address this challenge (**CH1**), we define the reward function to consist of two components. One component that is static and can be described offline $R'(s)$, and the other component changes at runtime $I_a(t)$, i.e., we define the reward function as:

$$R_a(s,t) = I_a(t) \times R'(s)$$

where the index t is used to denote the fact that the reward function is time-varying. The fixed component $R'(s)$ reflects the prior bias that lower relative distances are more dangerous than higher relative distances and attentive states are more favorable than distracted states. The time-varying component $I_a(t)$ is binary indicator signal, i.e., $I(t) \in \{-1, 1\}$ reflects the driver acknowledgment to the actions (warnings) triggered by the agent (recall challenge **CH2** in Section 6.2.3). That is, the proposed Sentio will start first by taking action based on the offline portion of the reward function $R'(s)$. Once the action is taken (trigger an FCW), Sentio monitors the reaction of the driver. If the driver "acknowledges" the warning by applying the brakes, we conclude that the action taken by the RL agent is correct, and hence the final reward is $R_a(s,t) = 1 \times R'(s)$. On the other hand, if the driver does not acknowledge the action taken by the agent, we set $I_a(t)$ to be negative, and the final reward is equal to $R_a(s,t) = -1 \times R'(s)$. While the driver reaction may not be instantaneous (due to different driver response time, recall challenge **CH3** in Section 6.2.3), the proposed multisample Q-learning algorithm (discussed in Section 6.5) is designed to address this issue.

In our problem setup, we set the reward component $R'(s)$ according to how far s is to the safest state s^*. The safest state s^* is the one where the relative distance is the furthest, the car speed is less than or equal to the leading car and the human is in attentive state.

The reward $R'(s)$ is then calculated as:

$$R'(s) = \frac{1}{\text{shortest path between } s \text{ and } s^* + 1} \times 100.$$

6.4.2 Random Human Actions and Erroneous Rewards

A driver may apply brakes for various reasons other than collision avoidance, e.g., before taking a turn or changing lanes or just a random press on the brakes due to other distractions. If such random braking took place when the car state was within the modeled set of states (i.e., the relative velocity is positive, and the relative distance is below 14m), such human actions might affect the reward received by the RL agent. However, thanks to the continuous monitoring and online learning, this erroneously received rewards will be overridden by the subsequently received rewards. Indeed, between the instance in which the RL agent erroneous rewards and right rewards, the RL agent may cause false positive events (issuing FCW where none is needed) or false negative events (not issuing FCW where one is required). In Section 6.6, we evaluate the proposed system on human drivers which exhibits such erroneous behavior.

6.5 Multisample Q-Learning

Learning the optimal policy—action per state that maximizes the total reward—when the transition probabilities of the MDP model are unknown can be solved using RL. By applying an action in a particular state and observing the next state, the RL converges to the optimal policy that maximizes the reward function. This type of RL technique is called Q-learning algorithm.

6.5.1 Standard Q-Learning

The Q-learning algorithm assigns a value for every state-action pair. For each state s, the Q-learning algorithm chooses an action a (among the set of allowable actions) according to a particular policy. After an action a is chosen and applied on the environment, the Q-learning

algorithm observes the next state s' of the environment and updates the q-value of the pair (s, a) based on the reward of the observed next state as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R_a(s) + \gamma \max_a Q(s',a) - Q(s,a)]$$

The hyperparameters γ and α are known as the discount factor and the learning step size, respectively. To choose an action a at each state s, an ϵ-greedy algorithm is adopted. In the ϵ-greedy algorithm, the RL agent chooses the action that it believes has the best long-term effect with probability $1 - \epsilon$, and it picks an action uniformly at random, otherwise. In other words, at each iteration, the RL agent flips a biased coin and chooses the action with the maximum q-value with probability $1 - \epsilon$ or a random action with probability ϵ. This hyperparameter ϵ (also known as the exploration versus exploitation parameter) controls how much the RL agent is willing to explore new actions, that were not taken before, versus relying on the best action that is learned so far.

Note that the standard Q-learning algorithm assumes that the environment reacts instantaneously to the RL actions, changes its state before the next iteration of the RL algorithm is executed, and the reward corresponding to this action is observed. In the context of Sentio, these assumptions are not valid. In particular, due to delays stemming from the human response as well as the mechanical response of the vehicle, the environment state does not change instantaneously in response to actions taken by the RL agent (recall challenge **CH3** in Section 6.2.3). To address this point, we propose a modified version of the Q-learning algorithm named "Multisample Q-Learning" algorithm.

6.5.2 Multisample Q-Learning Algorithm

A direct approach to address the challenge mentioned above is to collect multiple environment states over a fixed window (or horizon) after each action taken by the RL agent. Meaning, after the RL agent decides as action (e.g., trigger the FCW), the RL agent shall wait for a fixed horizon that captures the delay stemming from the environment response and treat all the information collected within that horizon as one change in the environment. Unfortunately, such approach will force the RL agent to wait for the whole horizon to pass

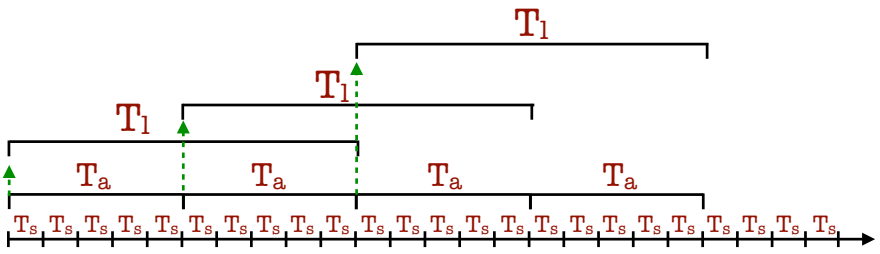


Figure 6.4: Multisample timescales. An action is taken by the agent every T_a samples. The reward is calculated after an action is taken by a time equals T_l. In this example, $T_a = 5T_s$ and $T_l = 10T_s$, where T_s is the state observing (sensor) rate.

before taking the next action. Taking into account that human response, and hence the fixed horizon, is in the order of seconds [Sum00], we conclude that such approach will force the RL agent to take action every few seconds degrading its ability to react to various scenarios. Another approach is to discard all the out-dated information and take into consideration only the fresh information [Tsi94]. While this algorithm is guaranteed to converge to the optimal policy, discarding the information will lead to a significant increase in the convergence time [Tsi94] which will occur whenever the driver behavior changes over time.

Therefore, the proposed multisample Q-learning algorithm divides the Q-learning into three different, overlapping, timescales (see Figure 6.4), namely:

- **State observing rate (T_s):** the state of the environment should be monitored whenever new measurements from the sensors are available.
- **Actuation rate (T_a):** to ensure fast reactivity from the agent, the RL agent should decide an action and update (learn) the Q value at a relatively fast rate.
- **Learning rate (T_l):** the reward corresponding to a particular taken action should be evaluated at a relatively slow rate to take into account the delay in the environment change. Once a reward is calculated, the RL agent can update the Q value to reflect the learned rewards so far.

These distinctive rates raise other challenges that we address in the next subsections.

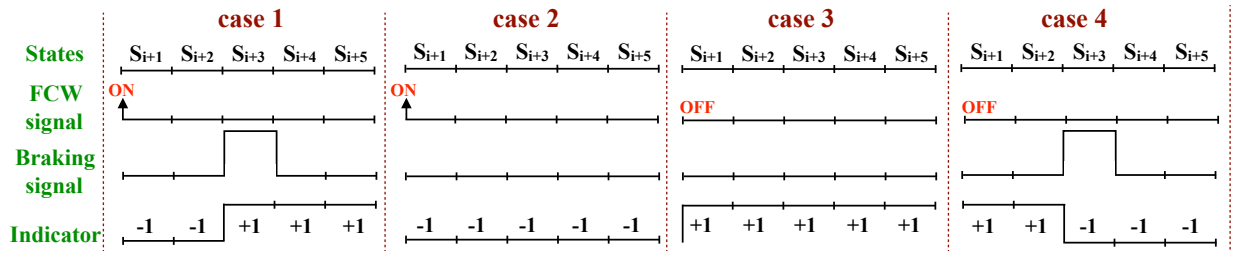


Figure 6.5: Handling the reward function for four different cases for driver behavior. Case 1: FCW is on and driver acknowledges it by pressing the brakes. Case 2: FCW is on and driver ignores it. Case 3: FCW is off and driver acknowledges it by not pressing the brakes. Case 4: FCW is off and the driver presses the brakes. To enhance the readability of the figure we removed the subscript a and the index t from the notation of $I_a(t)$.

6.5.2.1 Finite Horizon Reward

Recall that the reward is computed as $R_a(s,t) = I_a(t) \times R'(s)$ where $R'(s)$ represents how far the current state is from the safest state, and $I_a(t)$ represents how happy the driver is with the taken action. Computing the reward across a horizon (instead of single state) raises the question of how to aggregate all the monitored states within that horizon. For example, consider the case when the RL agent decides that at time t_1 and state s_1 to trigger the FCW. Starting from the time t_1 and until the end of the reward horizon, the driver chooses to acknowledge the warning by pressing the brakes for some period followed by releasing the brake pedal and accelerating the car. Multiple schemes can be proposed to aggregate the reward in such scenario. In the proposed multisample algorithm, we use the heuristic in which the indicator variable $I_a(t)$ latches to the change in the driver acknowledgment, i.e., once the driver acknowledges the RL warning, $I_a(t)$ will remain equal to one for the remainder of the reward horizon. This method is explained in Figure 6.5. Finally, the aggregate reward is computed as:

$$R_a(s_1, t_1) = \sum_{i=t_1}^{t_1+n} I_a(i) \times R'(s_i)$$

where n is the reward horizon. Note that, unlike standard Q-learning where the reward is not a function of time, the reward now is assigned to a particular action that is taken at

time t_1 at the beginning of the reward horizon.

6.5.2.2 Learning Update Rate and Overlapping Rewards

To enhance the reactivity of the RL agent, the RL agent decides the next action even before the reward horizon of the previous action is collected and aggregated leading to an overlap between the effects of different actions. For example, consider the case when the RL agent decides that at time t_1 *not* to trigger the FCW. Before the reward is calculated for this action, the RL agent decides at time t_2 to trigger the FCW. Both the actions will affect the environment during the time $t_1 \leq t \leq t_1 + n$ in which the reward for the first action is being collected. This raises the question of how to accommodate overlapping effects on the collected reward. To address this question, we use multiple indicators $I_a(t)$ one per each action (the subscript a is to emphasize the fact that multiple indicator variables are used, one per each action) taken in the previous $t-n$ horizon. Each indicator is handled differently according to the rules depicted in Figure 6.5. We assume that the inaccuracy that may arise while calculating the rewards due to different actions in an overlapped reward horizon will have a minimal effect over a long time due to the feedback nature we have in Sentio. The effect of choosing the learning update rate is studied in the evaluation shown in Section 6.6. Algorithm 6.5.2.2 summarizes all the details discussed in Sections 6.4 and 6.5. An illustration for the three proposed timescales is shown in Figure 6.4.

6.6 Experimental Results

We evaluate the proposed Sentio by recruiting a total of eleven human drivers (six males and five females) with ages that vary between 22 and 35 years. To ensure the safety of the recruited drivers, reduce liability, and meet the ethical standards of experimenting on human subjects, we decided to use a simulation environment (see Figure 6.6). In this environment, an industry-level physics simulator (named CarSim [Mec]) is used to simulate the physics of both the driving car and the leading car. Human drivers use a driving wheel and a pedal to interface with the physics-level simulator. Our evaluation uses the following metrics:

Algorithm 1 Sentio Multisample Q-learning algorithm

Hyper parameters: Learning parameters: α, γ, ϵ

Time scales: $T_s \leq T_a \leq T_l$

Initialization routine:

$Q(s, a_{\text{no warning}}) = 1 \qquad \forall s \in S;$

$Q(s, a_{\text{trigger warning}}) = 0 \qquad \forall s \in S;$

Push the state s^* into the queue M;

numberOfHops $= 0$;

while M is not empty **do**

Dequeue state s from M;

$R'(s) = \frac{1}{\text{numberOfHops}+1} \times 100$;

Enqueue all states that can reach state s in M';

if M is empty **then**

Copy M' into M;

numberOfHops $=$ numberOfHops $+ 1$;

return $R'(s), Q(s, a)$

State Observation Routine (activated every T_s)

Collect data from car and phone sensors;

Run inference to produce environment state s;

Push current state to the state stack S;

Actuation Routine (activated every T_a)

Pull the current state s from the state stack S;

Choose action a using ε-greedy policy;

Save action to the action list A;

Apply action to the driver;

Learning Routine (activated every T_l)

For all actions a in the stack A **do:**

$\tau =$ current time;

s' = current environment state;

s = the state at which the action a was taken;

$t = \tau - T_l$ (time at which the action a was taken);

$R_a(s,t) = 0$;

For all times t_i between $\tau - T_l$ and τ **do**:

Compute the indicator variable $I_a(t_i)$;

Get the state s_i at time t_i;

$R_a(s,t) = R_a(s,t) + I_a(t_i) \times R'(s_i)$

Update the Q matrix accordingly

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R_a(s,t) + \gamma \max_a Q(s',a) - Q(s,a)]$$

Figure 6.6: Vehicle dynamics virtualization testbed used in the evaluation study .

1. **Learning performance:** which is quantified by:

   - **False Negative (FN) Rate:** False negatives occur whenever the car violates the safety distance (relative distance below 7m) as a result of the FCW decides not to warn the driver early enough. Lower FN rate reflects better performance.
   - **False Positive (FP) Rate:** This metric quantifies the rate of false alarms. That is, the rate of which the FCW is triggered unnecessarily. Lower FPR reflects better performance.

2. **Driver safety:** While different violations of the safety distance have different severity (in terms of the relative distance and the period for which the violation lasted), we

quantify the driver safety using the **Violation Severity (VS)** which is computed as a weighted sum of the violated distance over the violation time. Lower SV reflects higher driver safety.

3. **Driving experience:** A better driving experience is the one in which the driver does not use high braking intensity more often. Therefore, we use the histogram of the **Braking Intensity (BI)** as an indication of the driving experience.

We divide our evaluation into two sections. In the first section, we aim to tune the parameters of the RL algorithm (e.g., explore the tradeoff between exploration/exploitation, learning rate, and learning step size). In the second section, we use the proposed RL algorithm (with the parameters tuned from the first set of experiments) to evaluate the driver safety and experience and compare it against the fixed-threshold FCW that is used typically in modern cars.

6.6.1 Parameter Tuning

To study the effect of each of the hyperparameters and correctly tune them, we need a controlled environment in which we can repeat the same exact experiment multiple times using different tuning parameters. Indeed, human behavior varies between each experiment and is not reproducible. For that end, we started by using the physics simulator along with the interfacing wheel and pedal to collect several driving traces from all the recruited drivers aiming to build a human simulator that can be used for parameter tuning. Analyzing the 11 human driver traces, we can categorize the sample of drivers obtained into three categories—based on their reaction to the FCW—as follows:

- **Defensive drivers (2 out of 11 drivers)**: These drivers press the brakes directly once they hear the FCW. Afterwards, they check the relative distance to the leading car and decide whether the braking action they took was a right decision or it was a false alarm.
- **Aggressive drivers (1 out of 11 drivers)**: After hearing the FCW, these drivers do not react to the warning until they first check the environment and react accordingly.
- **Assertive drivers (8 out of 11 drivers)**: After hearing the FCW, these drivers release

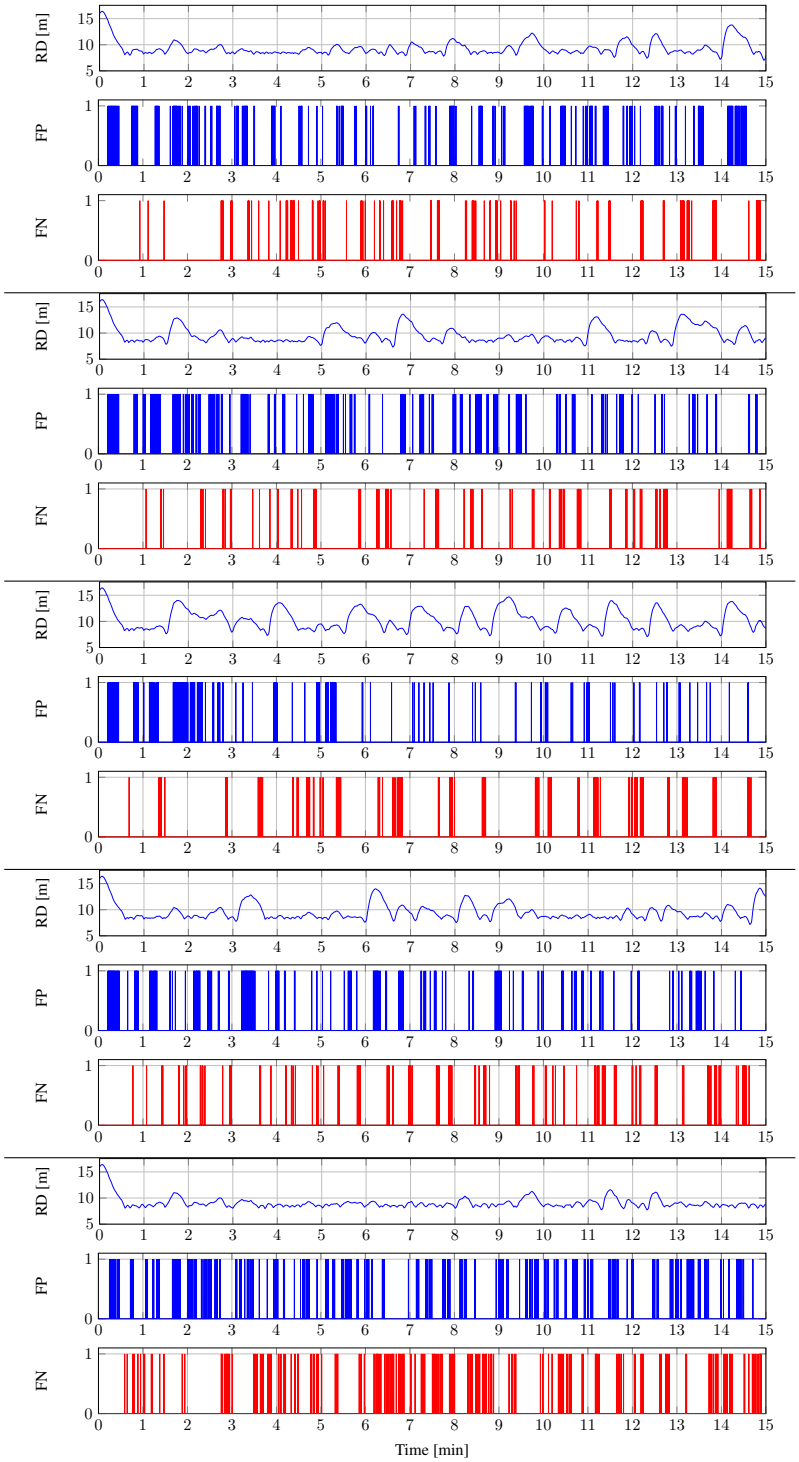


Figure 6.7: False positives and negatives with different learning parameters for five driving traces across 15 minutes simulation time: (1) $\epsilon = 0.1, \alpha = 1.0, \gamma = 1.0$, (2) $\epsilon = 0.1, \alpha = 0.6, \gamma = 1.0$, (3) $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.8$, (4) $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.7$, and (5) $\epsilon = 0.5, \alpha = 0.6, \gamma = 0.8$.

the acceleration pedal (as a precaution), check the relative distance with the leading car, and then decide whether to press the brake or not.

Moreover, our data show four distinguishing aspects between the assertive drivers namely:

1. **Braking intensity:** which defines how intense the driver tends to press the brake pedal.
2. **Acceleration intensity:** which describes how fast/slow the driver prefers to drive based on the intensity of pressing the acceleration pedal.
3. **Comfort braking distance:** which specifies the preferable relative distance that the driver likes to keep between his car and the leading car.
4. **Response time:** which defines the time taken by the driver to observe the relative distance to the leading car and take action (whether presses the brake or ignores the warning).

According to these observations, we built a human driver simulator to mimic the behavior of the assertive drivers (since the majority of the driver sample we got was assertive). The simulator continuously picks—at random—the values for the braking intensity, acceleration intensity, and response time.

6.6.1.1 Experiment 1a: Parameter Tuning

We examine the tradeoff between exploration and exploitation (ϵ) in choosing the appropriate action for the current context (vehicle context and human context) along with the learning parameters (the discount factor (γ) and the learning step size (α)). Figure 6.7 shows the traces of the relative distance across the different choice of parameters[5] with their corresponding false positives and false negatives pattern.

- **Effect of the learning step α:** First we fix the values of ϵ and γ and sweep the value of the learning step α between $\alpha = 1$ and $\alpha = 0.6$. We observe that for higher values of α (Figure 6.7 - case 1) the number of false positives does not decrease with time which signals that the RL agent is not able to learn the human preference and not able to produce the warnings in a way that satisfies the driver. However, by decreasing the learning step

[5]Due to space limit, we show only 5 choices of parameters.

α—which has the effect of asking the RL agent to use multiple data for learning instead of depending entirely on the latest learned data—we observe that the number of false positives decreases over time (Figure 6.7 - case 2). We conclude that a low value of the learning step α is needed.

- **Effect of the discount factor γ:** Fixing the value of α at the best value from the previous experiment $\alpha = 0.6$, we start to sweep the values of the discount factor γ and compare the results (Figure 6.7 - case 2, case 3, and case 4). We observe again that lowering the discount factor γ leads to enhancing the number of false positives as seen by comparing the results in case 2 ($\gamma = 1.0$) and case 3 ($\gamma = 0.8$) of Figure 6.7. However, by decreasing the discount factor further, we note that the number of false positives starts to increase again as evident by case 4 ($\gamma = 0.7$) in Figure 6.7. We conclude that a value of $\gamma = 0.8$ achieves the best results.
- **Effect of the exploration/exploitation factor ϵ:** Finally we study the impact of the exploration/exploitation factor ϵ. This factor controls the confidence of the RL agent on the current Q values. Lower values of ϵ reduce the probability of picking a random action and tend to decide actions based on the learned Q values. A higher value of ϵ asks the RL agent to explore more actions at random aiming to test actions not taken before. Comparing the results in Figure 6.7 for case 3 ($\epsilon = 0.1$) and case 5 ($\epsilon = 0.5$) we observe that increasing ϵ leads to an increase in both false positives and false negatives. We conclude that $\epsilon = 0.1$ leads to the best results.

6.6.1.2 Experiment 1b: Timescales Tuning

After fixing the values of the learning step α, the discount factor γ, and the exploration/-exploitation factor ϵ, we examine the other hyper-parameters in the multisample Q-learning algorithm (the state observation rate T_s, the actuation rate T_a, and the learning rate T_l). Since T_s is constrained by the availability of the sensory information and the state interference algorithm, we fixed it at 0.25 seconds and we examined different combination for T_a and T_l. In Figure 6.8 we show the traces for five of these combinations. First, we fixed the value of the T_a at 0.5 seconds and changed the value of T_l. We noticed that the higher the

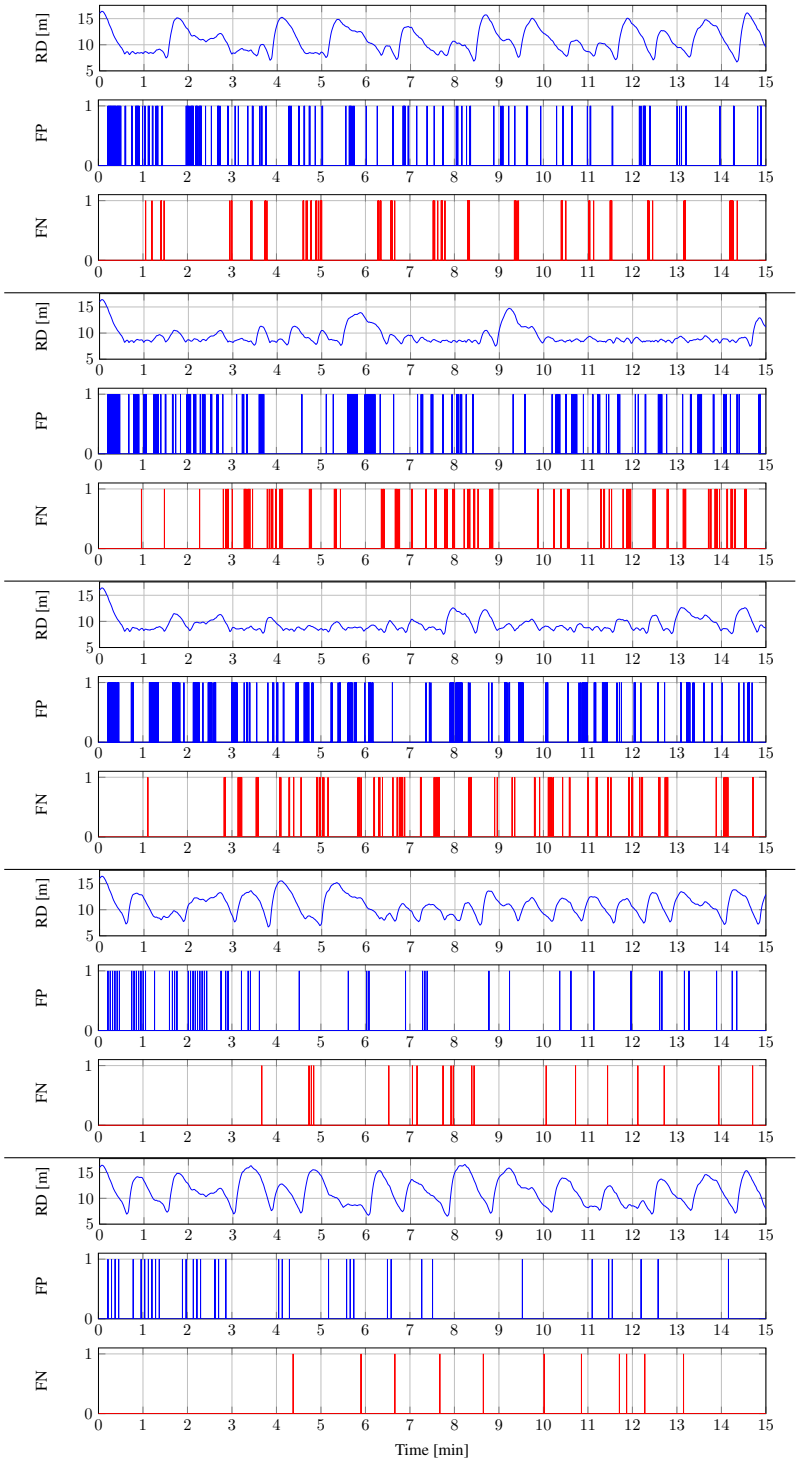


Figure 6.8: False positives and negatives with different values for T_a and T_l for five driving traces across 15 minutes simulation time using $\epsilon = 0.1, \alpha = 0.6, \gamma = 0.8$: (1) $T_a = 0.5s, T_l = 2.5s$, and (2) $T_a = 0.5s, T_l = 4s$, (3) $T_a = 0.5s, T_l = 5s$, (4) $T_a = 2.5s, T_l = 5s$, (5) $T_a = 4s, T_l = 5s$.

value of T_l, the smoother the trace of the relative distance which indicates a better driving experience. This entails that the more we incorporate the time response of the human in T_l, the more the human enjoys a better driving experience. Next, we fixed T_l to 5 seconds and examined different values for T_a. Although FN and FP appeared to be less as we increased the value of T_a, the FNR increased as we increased T_a. In the case when T_a is 2.5 seconds the FNR increased to 3.11% compared to 2.44% in the case when T_a is 0.5 seconds. On the other hand, the FPR decreased as T_a increased. In particular, when T_a is 0.5 seconds the FPR is 4.05% compared to 3.25% when T_a is 2.5 seconds. This shows a tradeoff between smooth driving experience and low values for FPR/FNR. Hence, we chose the values of T_a and T_l to be 0.5 seconds and 5 seconds respectively which achieve the smoothest driving trace with low FNR on the expense of a little higher FPR.

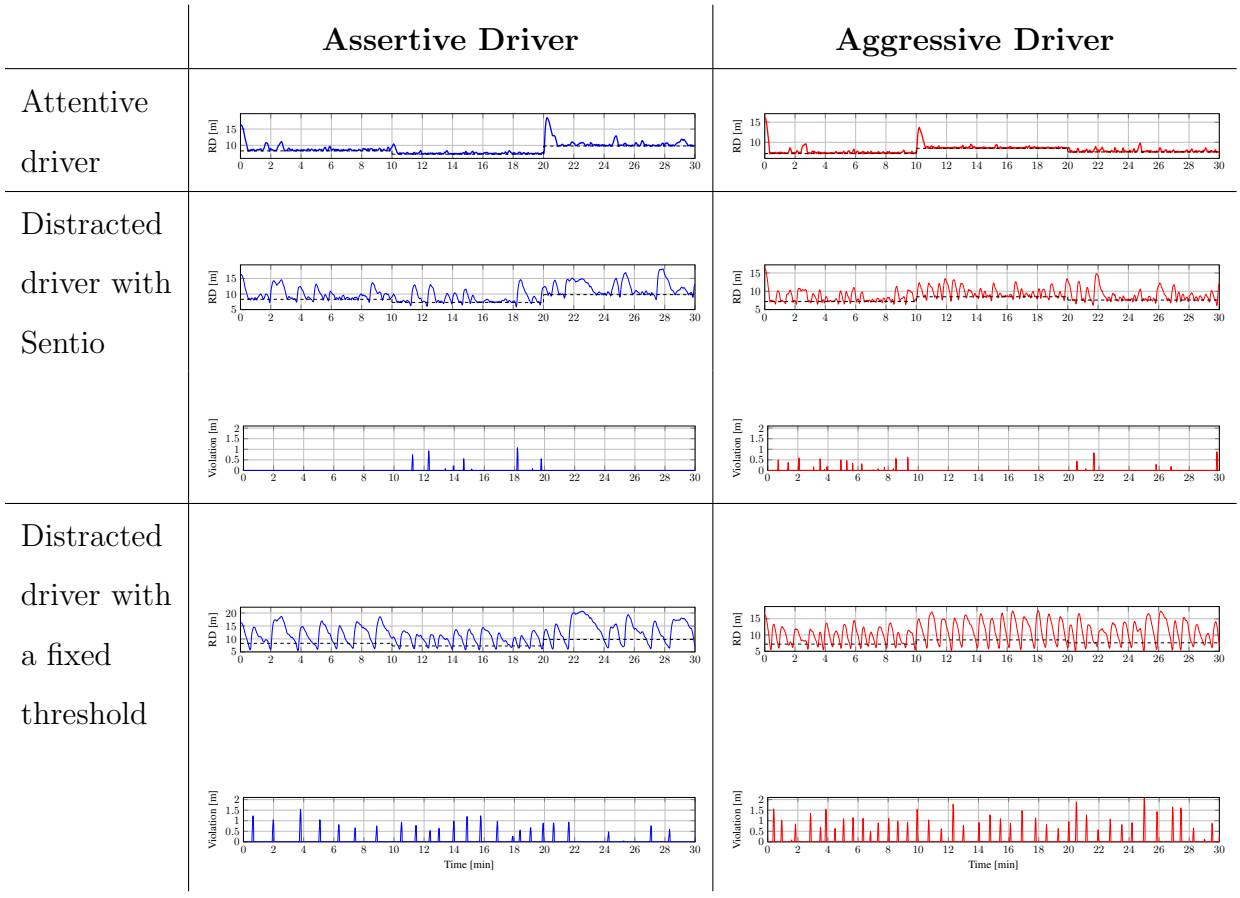


Figure 6.9: Relative distances and violations of two distracted drivers with change in behavior over time over a simulation time of 30 minutes using Sentio and with a fixed threshold alert.

6.6.1.3 Experiment 2: Tracking driver changing behavior

While the previous parameter tuning is studied by using the same type of driving behavior (assertive driving) and for a fixed driver preference, the next step is to test the performance of the tuned RL agent against different driving behaviors and for changing driver preference (with respect to the relative distance to the leading car). We use our human simulator to synthesize two driver behaviors; one simulates an assertive driver while the second simulates an aggressive driver over a 30 minutes simulation time. During this time, the behavior of the driver (with respect to the preferred relative distance) changes. We imposed distraction over these two synthesized drivers (by increasing the response time and braking intensity) [CDW03] and observed the difference between the proposed Sentio and the classical FCW (which uses a fixed ratio of relative distance and relative velocity to signal the alert) typically used in modern cars.

Ultimately, we want the RL agent to learn to issue the alert ahead of time when the driver is distracted preventing the vehicle to violate the dangerous $< 7m$ safety distance.

To compare the performance of the two systems (fixed vs. Sentio), we plot both the driving trace (relative distance) along with the violation of the safety constraint (i.e., the amount of time and distance below $7m$) for each case in Figure 6.9. In the case of a fixed alert threshold, we observe that the violation in the case of aggressive drivers is higher compared to the assertive drivers. Comparing the fixed threshold policy of both the assertive and aggressive drivers to the proposed Sentio, we observe that (1) the relative distance is fluctuating more whenever the driver is distracted. This reflects the fact that the simulated driver brakes only after hearing the alarm due to the distraction effect. This is more severe in the case of the aggressive driver due to using higher braking intensity (2) a reduction in the violation of the driver safety, thanks to the fact that the RL agent was able to learn the driver behavior and issue the warning early enough even if the driver preference changes over time. In particular, the VS metric is reduced by 85.8% for the assertive driver and reduced by 90.25% for the aggressive driver.

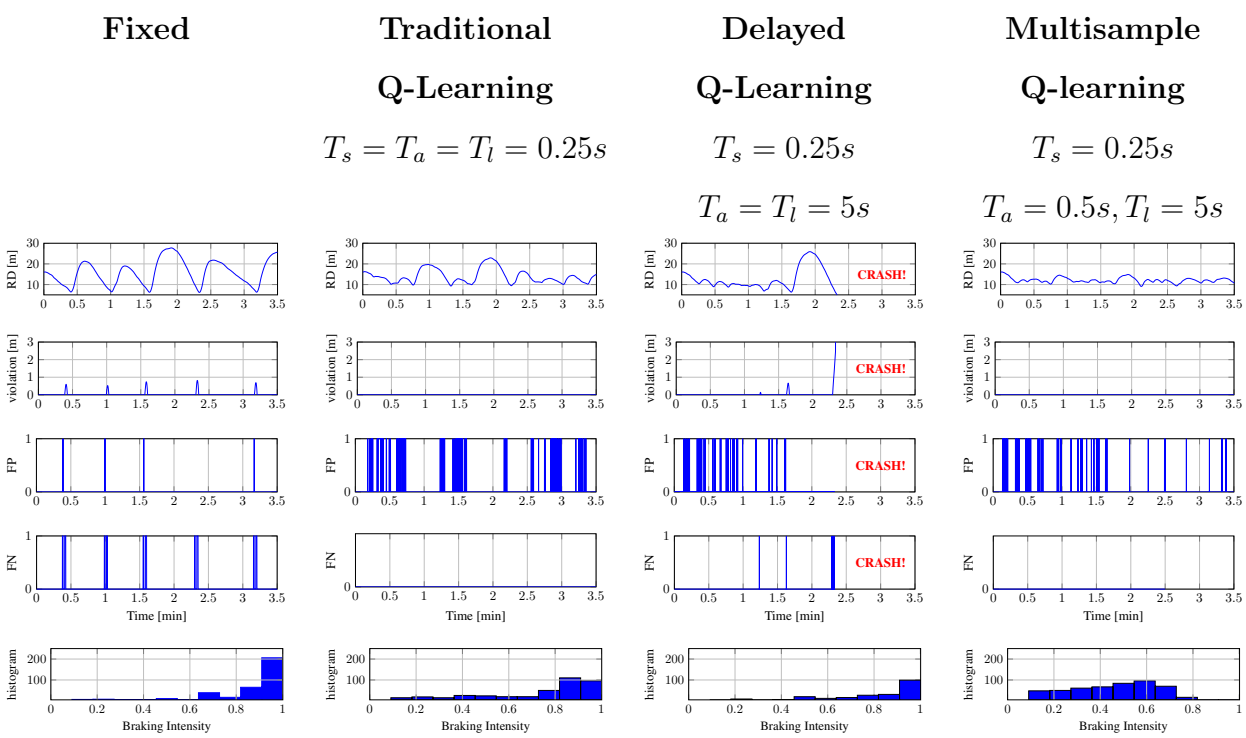


Figure 6.10: Driving Safety and Experience for Driver #1.

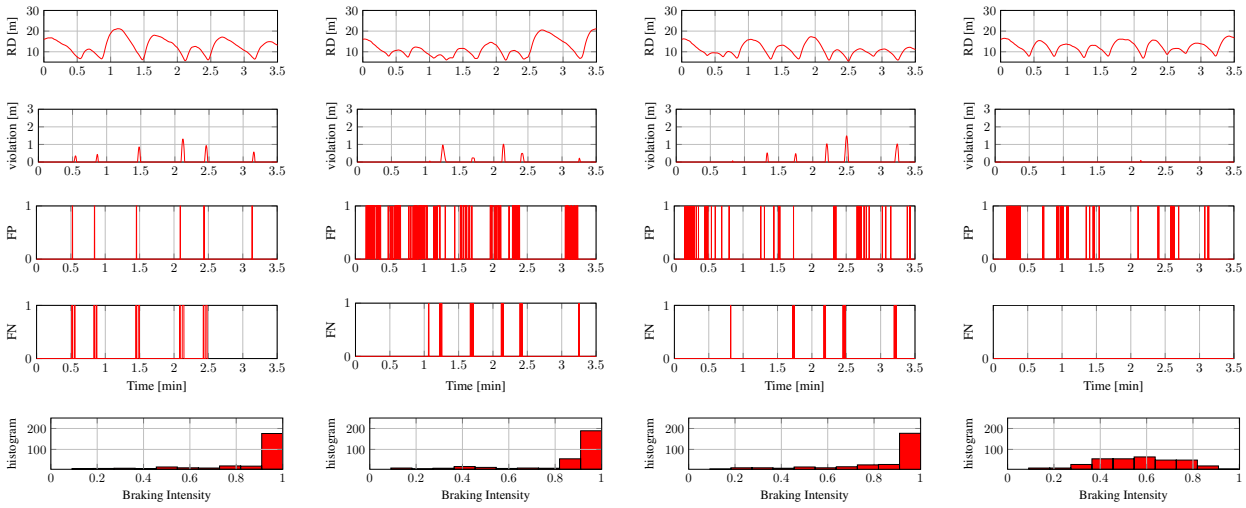


Figure 6.11: Driving Safety and Experience for Driver #2.

6.6.2 Human Driving Experience

After gaining confidence in the proposed Sentio in a controlled environment, the next step is to ask the same recruited human drivers to use the proposed Sentio. We compare their driving experience against (i) the fixed-threshold system, (ii) the standard Q-learning (where observing state, learning, and taking actions occur every sample), (iii) the delayed Q-learning (where observing state, learning, and taking actions occur after a fixed delay to take into account the human response time), and (iv) the proposed multisample Q-learning. Every driver is asked to drive for 30 minutes (average commute time in the US is 25.4 minutes [US]) in each experiment while one of the four algorithms is chosen randomly at runtime (to remove driver bias). To examine the effect of random behavior as discussed in Section 6.4.2, they are free to brake at anytime regardless whether or not they hear an FCW.

Drivers are asked to engage in conversations with others while driving. A machine learning classifier is used to distinguish between attentive/distracted states. We trained a logistic regression model using recordings from MUSCAN [SCP15], CallFriend [CZ96], CallHome [CGZ97], and Librivox [Lib]. We used Mel Frequency Cepstral Coefficient (MFCC) as the main feature in our model as it is a fair approximation of human perceptions of pitch [GFK05]. Our trained model could predict if the person is engaged in a conversation with 97.2% accuracy on a 5-fold cross-validation within the training set and 97% accuracy on the test set.

We show in Figure 6.10 and Figure 6.11 the traces for the first 3.5 minutes (showing the learning transients) of two drivers along with the corresponding traces for the four metrics (VS, FP, FN, and BI histogram). We then report the results of the four metrics for all the 11 human drivers in Table 6.1.

Below are the observations and conclusions drawn from these experiments.

- **Fixed threshold:** Fixed threshold leads to the worst driving experience whenever the driver is distracted. This is evident by the histogram of the braking intensity where we observe that drivers are obliged to use high braking intensity (pressing the braking pedal all the way to its maximum limit) to avoid a collision. Even with such driving experience,

we observe that violations are not entirely removed, but yet drivers violate the safety constraint.

- **Traditional Q-learning:** This method leads to no learning and a random behavior of issuing the warning. This is best captured by the high amount of false positives across time. While this method reduced the violation compared to the fixed threshold, this is mainly due to a large number of warnings produced by this method (with most of them being false). This behavior can be accounted to the fact that rewards are computed based on the instantaneous values of the environment state without taking into account the delay in the human response.
- **Delayed Q-learning:** While the learning behavior (and hence the FP rate) are enhanced compared with the traditional Q-learning, we observe that it was not able to warn the first driver early enough to avoid the crash. This stems from the fact that this method learns and takes action only after the whole reward horizon has ended leading to missing critical events and hence caused a crash.
- **Multisample Q-learning:** Similar to the delayed Q-learning, we observe that the proposed multisample Q-learning can learn the preference of the driver leading to a decrease in both FN and VS metrics with the cost of an increase in the FP especially while learning. However, thanks to the multisampling of the proposed method, we observe an increased safety compared to the delayed Q-learning (measured by the VS metric). Moreover, we observe that the histogram of the braking intensity (and hence the driving experience) has improved significantly where the drivers tend to use smaller barking intensities thanks to the fact that warnings are produced early enough leaving the driver with enough time to brake slowly.

Table 6.1 shows the results for the same metrics for all the 11 humans across the entire 30 minutes experiment time. These results reflect similar conclusions. On average, Sentio reduced the violation severity by 94.28%, reduced the mean of the braking intensity by 20.97%, reduced the false negative rate from 55.90% down to 3.26%. These improvements come at the cost of increasing the false positives (false alarms) from 2.71% to 5.15%, on average. Indeed, most of these false alarms occur during the initial learning phase and their

	Fixed			Sentio		
	VS	BI (mean)	FNR [%]/FPR [%]	VS	BI (mean)	FNR [%]/FPR [%]
H1 (A)	85.379	0.852	54.5/0.22	0.501	0.419	0.58/1.88
H2 (S)	127.11	0.704	52.0/0.78	0.844	0.542	0.61/2.71
H3 (D)	54.294	0.41	61.61/0.49	0	0.496	0.66/3.45
H4 (S)	136.212	0.713	42.53/0.95	6.087	0.593	0.52/9.79
H5 (S)	295.411	0.702	99.0/2.46	11.03	0.536	18.69/8.37
H6 (S)	258.412	0.803	98.1/9.83	67.82	0.735	7.94/9.72
H7 (S)	105.205	0.84	52.9/6.33	0.707	0.749	1.85/7.81
H8 (D)	28.35	0.561	57.3/1.52	0.134	0.516	0.61/0.81
H9 (S)	185.755	0.739	19.7/1.28	1.560	0.604	0.39/4.53
H10 (S)	221.086	0.747	49.2/5.41	2.496	0.510	3.51/6.25
H11 (S)	121.158	0.796	28.1/0.62	1.289	0.516	0.51/1.28
AVG	147.124	0.715	55.90/2.71	8.406	0.565	3.26/5.145

Table 6.1: Comparison between the performance of fixed threshold FCW and multisample Q-learning in Sentio for 11 distracted drivers (A = Aggressive Driver, S = Assertive Driver, and D = Defensive Driver). Degradation in metric performance (with respect to Fixed policy) is marked in red.

rate decreases afterwards.

Finally, we informally asked the drivers about their experience; all drivers reported the excessive false alarms at the beginning of the experiments. However, they felt that the false alarms rate decreased significantly while driving. Moreover, nine drivers assured that, in case of Sentio, the alert was given at a proper time to examine the environment when they were distracted. However, the two defensive drivers reported that the alert was given too early and they had to brake unnecessarily.

6.6.3 Execution Time Analysis

We report the execution time of the proposed method. Recall that Algorithm 6.5.2.2 consists of different routines that are executed at different rates. Averaging out the execution time across all drivers, we observe that the initialization routine consumes 61.86 seconds. Fortunately, this routine is called offline, and hence it does not affect the online execution time. The state observation (and inference) routine consumes 20.4 ms, the actuation routine consumes 2.59 ms, while the learning routine consumes 104 ms which is negligible compared to the human response time.

6.7 Discussions

Despite the fact that Q-learning based algorithms enjoy strong mathematical guarantees in terms of convergence to the optimal policy (action per state), the mathematical guarantees for its safety are still lacking. Recently, there have been new studies in the literature that extends RL algorithms to safe RL in which safety constraints are incorporated in the learning and deployment processes. One example for such work is constraining the exploration strategy using some prior knowledge to avoid going into risky situations [**?**, **?**]. Nevertheless, the relation between the optimality of the policy and the safety of the policy is indeed an interesting area to explore and study in future work.

Indeed, Sentio does not come without limitations. As shown in the experimental results, the false alarm rate is higher on average compared to the fixed threshold FCW. A possible solution is to change the hyper-parameters over time. In particular, the exploration versus exploitation parameter ϵ plays a significant role in the increase of the false alarms. However, ϵ also plays a significant role in adapting to the changes in the driver behavior. One possible track is to change ϵ across time based on our confidence of how much the human behavior will be fixed. Moreover, the human state is complex and it is not merely binary (distracted/attentive) but a continuous range of values. A next step would be to explore more elaborate human models that take into account different human behavior and study

the effect of such elaborate models on the performance of Sentio. Moreover, industrial-level testing and verification of Sentio across a wider range of drivers' states, weather conditions, and road conditions along with real-life deployment is the next step in this research.

6.8 Conclusion

While FCW is one of the highly adopted ADAS in vehicles, its primary focus is still dependent on the environment around the vehicle (relative distance and relative velocity). However, by taking the driver state into the loop of computation, the FCW can be personalized and provide a better experience. In this chapter, we purposed Sentio, a driver-in-the-loop FCW that learns the driver preference as well as his attention level to signal the FCW at the right time. We proposed a variant of the Q-learning algorithm to solve our problem and enhance the learning rate of the driver preference. Our multisample Q-learning algorithm could track the change in the driving behavior across time and adapt the timing of the FCW accordingly without the need for offline training. By examining the results of driving traces on a car simulator for multiple drivers, Sentio shows a better performance than the fixed threshold FCW that is widely used in commercial vehicles.

CHAPTER 7

IoPAT: Internet of Personalized and Autonomous Things

While traditional IoT systems interact with humans, in general, by collecting data directly from humans and their environment, a unique feature of IoPAT is its ability to assess human satisfaction and closing the loop by taking actions to adapt to the changes in his mood, needs, and expectations. This tight coupling between human behavior and computing promises a radical change in human life [Pic95]. To emphasize the difference between the proposed IoPAT and traditional IoT, we consider the example of smart thermostats in the context of smart homes. Current state-of-art smart thermostats are capable of adapting the home temperature based on room occupation using fixed schedules and policies [LSS10]. For example, homeowners are required to define a preset of configurations, and the IoT system makes sure to follow these configurations. Unfortunately, human needs vary across time. In the context of smart home, since body's temperature needs to drop to sleep [Har07] and since body's temperature is affected by multiple factors like excitement, anxiety, body activity, and health issues, the same human may prefer a cooler or warmer temperature during sleep time. Even more, in the same mood, health, and activity situations, the same person may have a different preference for the best room temperature. Unfortunately, due to this variation in human needs and behavior, current IoT are heteronomous and incapable of providing the necessary level of personalization.

7.1 IoPAT Systems

In IoPAT systems, the first step is extracting complex semantics from various sensory data to infer the state (or context) of both the human users along with their physical environment. Such information can be collected from various edge devices including thermostat sensors, mobile phones, and wearables. These raw sensory data can be used to infer complex human states including human activity (e.g., exercising, running, walking) [LPL12] and mood [LLL13]. Thanks to the increasing computational capabilities at the edge devices and sophisticated machine learning models, such state and context inferences become ubiquitous.

The next step is to fuse the individual states inferred from various edge devices along with those relayed from the cloud. This global state is then utilized by adaptation algorithms to close the loop and take actions in an attempt to match the user and environment state. Unfortunately, the human preferences vary between different humans and even for the same human across time. Therefore, a unique aspect of IoPAT is their ability to assess the human satisfaction and "learn" how to correct their actions to enhance the user experience.

To better illustrate the potential of IoPAT systems over traditional IoT, we consider the example of a smart home. Current state-of-the-art smart IoT-based thermostats are capable of regulating home temperature based on weather forecasts and occupancy [CMH13]. That is, these smart thermostats can respond to real-time changes in the outdoor temperature based on the number of the occupants of the home. However, these systems ignore a fundamental fact. Human comfort temperature varies across individuals and, even more, it varies across time for the same individual. A home occupant may prefer a particular home temperature while reading and prefer another temperature while working out. Even for the same activity (e.g., reading), the same individual may prefer different temperatures across time. This lack of adapting to human variations in human preferences and responses is the major drawback in the current IoT systems.

To circumvent this performance gap and provide every human with a personalized experience, the proposed IoPAT utilizes sensor information available at different edge devices like mobile phone and wearable sensors to continuously infer complex human states (e.g., activ-

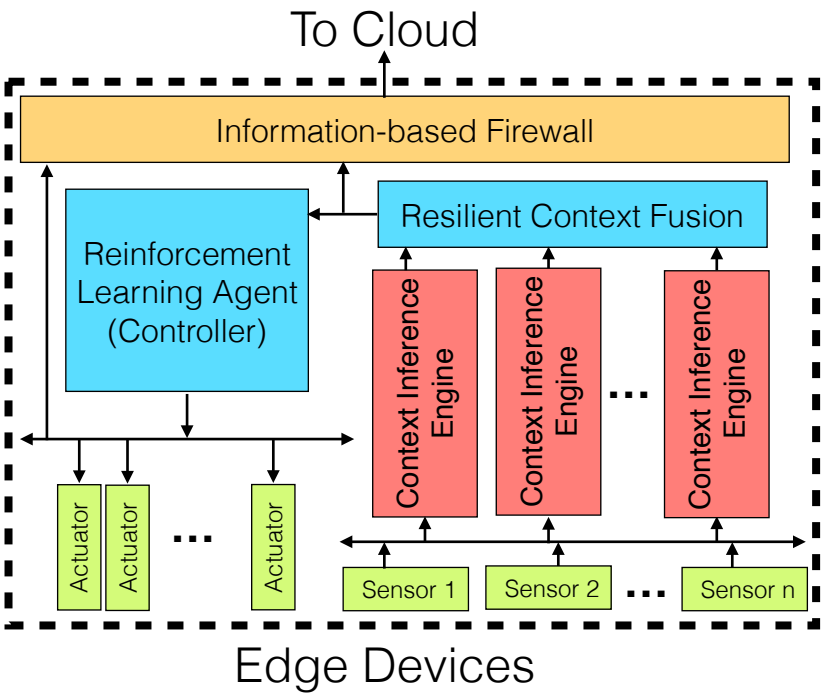


Figure 7.1: Proposed architecture for IoPAT edge devices.

ity and mood). Upon changes in any of these states, the IoPAT-based thermostat utilizes the history of this particular individual to take an action and change the home temperature. However, as discussed before, individual humans may change their comfort zone, the IoPAT-based thermostat monitors the comfort of the user using a black-globe thermometer and re-adapt accordingly. That is, the IoPAT system needs to continuously "learn" the new human preference to close the loop and provide a unique experience for each human.

7.2 Architecture for IoPAT Edge Devices

Our architecture for IoPAT edge devices consists of three main subsystems. Below we give more details about each of these subsystems.

7.2.1 Resilient Context Fusion

The first step in an IoPAT system is to utilize the available sensors to infer the state of the human and his environment. Such inference can take place directly from multiple raw

sensor measurements or by fusing different low level inferred states. While context inferences have been studied thoroughly in the literature [OGB11, CSD15, HYT14], IoPAT systems are susceptible to a unique threat vector namely the human himself. An individual human may try to maliciously influence the inference engine to in an attempt to lead the IoPAT system to decide wrong actions which may trigger several liability concerns. To alleviate this challenge, a commonly explored idea is to exploit the redundancy in the collected sensor data in order to identify and isolate the malicious data leading to a new set of resilient learning-based context inference and fusion algorithms [SCW18].

7.2.2 Reinforcement Learning Controller

Once the human and the environment state (or context) are determined, this state is then used to close the loop and take adaptation actions. As mentioned in Section 1.2.1, adapting to variations between humans and variations within the same human are the central challenge in designing an IoPAT system. To address these challenges, the IoPAT controller continuously monitors the human reactions. If the human dislikes the actions taken by the controller (as reflected by changes in his state), then the IoPAT controller "learns" the human preference otherwise the controller "learns" that this action matches the human preference. This continuous feedback loop of taking actions and monitoring the human decision (response) fits naturally within the Reinforcement Learning (RL) paradigm.

In the RL problem, a software agent tries to learn the behavior of an environment by issuing actions and observing the change in the state of the environment with the purpose of maximizing a notion of total reward. To model variations between the state of the individuals and the variation within the same individual, we model the human as a Markov Decision Process (MDP), where the states in the MDP correspond to the human state. Each action by the IoPAT controller will lead to a transition in this MDP. However, the transition probabilities between these states are unknown in advance due to the intrinsic variation in the human preference. In order to solve the MDP to get the best action for each state without knowing the transitional probabilities, our IoPAT controller is equipped with an RL

algorithm, named Q-learning algorithm, that best fits this problem setup. In the Q-learning algorithm, an agent applies an action on the environment and observes the effect of this action per state. The agent chooses an action which maximizes a notion of reward value. The reward value is a quantification of how *good* the taken action by the agent.

7.2.3 Information-Based Firewall

As motivated in Chapter 4, sharing information in the context of IoPAT systems may lead to privacy leaks stemming from the tight coupling between human behavior and actions produced by IoPAT controllers. Spyware exploiting these privacy leaks cannot be detected by the current state-of-the-art signature-based and behavior-based detection techniques. Our architecture proposes a novel information-based detection and mitigation firewall. The basic idea behind this firewall is to keep track of the ability of *any* Spyware to infer the human state through monitoring actions triggered by changes in these states. To this end, we draw on the literature of information theory and leverage *mutual information* to quantify the amount of correlation (or dependence) between two random variables. In our scenario, we use the mutual information between *state* and *action* as a metric to measure how certain a Spyware may infer the human state from observed actions. Mutual information provides a theoretical bound on the inference capability of *any* learning algorithm. Generally speaking, the lower the mutual information between context and actions is, the smaller the accuracy *any* inference algorithm can get. Push into one extreme; if the mutual information is zero, then *no* algorithm can infer context from monitored actions. Once the mutual information (and hence the correlation) between actions and human states are above a certain threshold, the firewall starts to carefully corrupt the information before being shared with other edge devices in the IoPAT system in an attempt to lower this correlation and prevent *any* inference algorithm from discovering patterns in the data.

7.3 Case Study

In this section, we study the effect of the IoPAT architecture proposed in Section 7.2. In particular, we focus on how the proposed RL-based controller could address the *adaptability*

challenges. To that end, we conducted numerical simulation using the thermal model of a house [The12] with three occupants. We simulated a period of 8 hours in total with a sample time of 30 second. We implemented the proposed IoPAT RL-based controller and a traditional IoT-based thermostat. Details about the high-fidelity mathematical model for the house, humans, and the IoPAT controller are given below.

7.3.1 Thermal Model of a House

In our simulations, we utilize a thermodynamic model of the house that takes into consideration the geometry of the house, the number of windows, the roof pitch angle, and the type of the insulation used. The house is being heated by a heater with a flow of air with temperature $50°c$. A thermostat is used to allow a fluctuation of $2.5°c$ above and below the desired set-point which specifies the temperature that must be maintained indoors. We consider two external controllers that set the desired set-point. A classical IoT thermostat that determines the set-point based on the number of occupants in the house. The second controller is the IoPAT controller explained in Section 7.3.3.

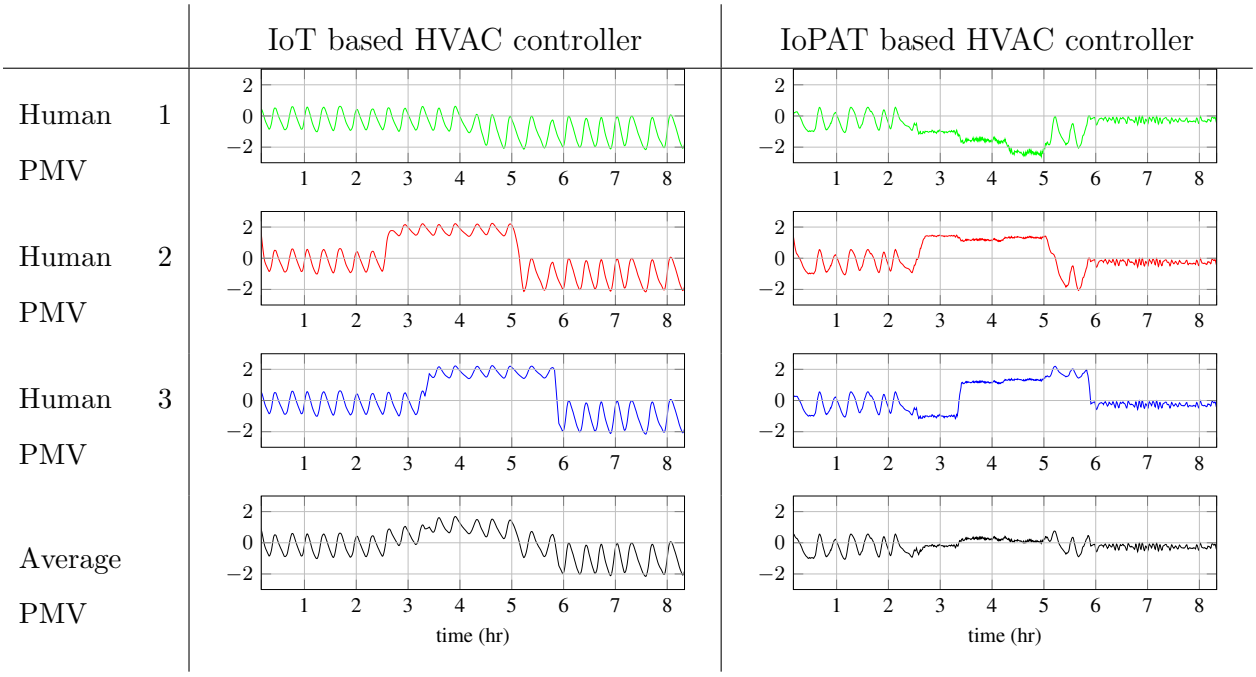


Figure 7.2: Prediction Mean Vote (PMV) for the three occupants (y-axis) across time (x-axis) using IoT system (left) and IoPAT system (right) for varying occupants' activity and stress level (relaxed/stress).

To take into account the personal behavior, we model each human as a heat source with heat flow that depends on the average exhale breath temperature (EBT) and the respiratory minute volume (RMV) [GLC10].These two parameters are highly dependent on the human activity. For example, $RMV \approx 6$ l/m when the human is resting while $RMV \approx 12$ l/m represents a human performing moderate exercise [Car07].

7.3.2 Human Thermal Comfort

We use the Prediction Mean Vote (PMV) [Fan70] as a measure for the human thermal comfort. PMV gives a score on how warm/cold a person feels. It depends mainly on the human activity, metabolic rate, clothing, environmental variables (air speed, air temperature, mean radiant temperature, and vapor pressure of air).

The PMV score ranges from -3 to 3 which is the range of thermal sensation from very cold (-3) to very hot (3). According to ISO standard ASHRAE 55 [ASH10], a PMV in the range of -0.5 and $+0.5$ for an interior space is recommended to achieve thermal comfort. PMV score for each human can be estimated using the knowledge of the clothing factor, the clothing insulation, the metabolic rate, the air temperature, the air vapor pressure, and the mean radiant temperature [Fan70].

In our IoPAT controller, we assume the knowledge of the weather forecast which affects the clothing factor and clothing insulations. For example, in our experiments, we assume a winter season and hence the IoPAT controller fixes an average for the clothing insulation of 0.9 clo [ASH10]. The metabolic rate is determined by the occupant activity [Enga] and stress level (relaxed/stressed) [SBT05]. This can be inferred using the heart rate and respiration rate either using wearable sensors [CG09] or using non-invasive technologies [AMK15]. Air temperature can be directly measured using thermostat sensors, while air vapor pressure is a one-to-one correspondent to the ambient temperature [Engb] which can be directly measured using outside thermostat as well. Finally, the mean radiant temperature can be estimated using a black-globe thermometer [BW34, KSW70].

7.3.3 RL-based Controller for IoPAT

While human stress level (relaxed/stressed) and human activity (resting, watching TV, eating, ...) can affect their thermal comfort, the IoPAT RL-based controller needs to track their PMV score and take correcting actions to maximize their comfort. Accordingly, we model each human as an MDP. States in the MDP corresponds to different PMV scores (with 0.5 granularity) resulting in 14 states for each human. We design the reward function for the RL-based controller on the difference between the average PMV (across all humans) and the nominal value -0.5 to 0.5. The RL-based controller then changes the temperature set-point accordingly. While the relation between the actions (changes in the set-point) and the human comfort is unknown to the controller, we use a Q-learning algorithm to continuously "learn" the human response to different set-points and take corrective actions.

7.3.4 Numerical Results

Using the simulation environment discussed in Sections 7.3.1 and 7.3.2, we consider the scenario in which the state of the three occupants change over time and report the thermal comfort for each human when both the IoT based controller (which changes the set-point according to the occupancy and weather forecast) and the proposed IoPAT controller are used to controlling the house temperature. During the simulation time, we change the state of the three occupants. We summarize the scenarios in our simulation as follows:

- Human 1: is sitting and relaxed for the first 4 hours. He then goes to sleep for the remaining 4 hours.
- Human 2: is sitting and stressed for 2.5 hours, performing domestic work while still being stressed for another 2.5 hours, and finally going to sleep for the remaining 3 hours.
- Human 3: is sitting and relaxed for 3 hours, performing domestic work for 2.5 hours while still being relaxed, and then finally going to sleep for 2.5 hours.

Figure 7.2 reports the comparison between the IoT controller and the IoPAT controller with respect to the individual thermal comfort and the average thermal comfort. During the first 2.5 hours, when the three occupants are seated, the IoT controller shows a good range of PMV (between -0.5 to 0.5) at a set-point of $23°c$. Similarly, the RL-based controller chooses

an action of setting the set-point at $23°c$.

However, once human 2 and human 3 start to perform domestic work, their average exhale breath temperature increases which affects their thermal comfort. While the IoT controller is only affected by the occupancy and the air temperature, it decides to still provide a set-point of $23°c$ leading to a higher PMV, almost reaching at value 2. In contrast, thanks to the feedback from the different sensors, the reward for the RL-based controller starts to decrease forcing the RL-based controller to take corrective actions. Accordingly, the RL-based controller starts to gradually decrease the set-point to $21°c$ followed by a further decrease to $20°c$ to compensate the rise in PMV for human 2 and human 3 leading to a decrease in the individual and average PMV.

Similarly, in the last 3 hours, all occupants are sleeping and hence their average exhale breath drops and their comfort score also decreases. Again, the IoT controller uses the occupancy based policy leading to the same set-point of $23°c$ which in turn leads to a drop in the PMV score reaching a value of -2. On the other side, and thanks again to the RL-based controller, it decides to increase the set-point gradually until it reaches $25°c$ to compensate the decrease in the PMV.

These results show the effectiveness of the proposed RL-based controller for IoPAT systems to track the variations in the human behavior and provide a personalized experience as compared to traditional IoT systems.

7.4 Conclusion

Our everyday life activities are becoming more dependent on edge devices. Since most of these devices interact with humans, the IoT system should include the human factor into the loop of computation. Accordingly, we propose IoPAT system. IoPAT considers the human preference and current state as an integral part to the IoT system. In this chapter, we proposed an architecture for IoPAT and discussed the challenges that face the personalization of IoT. We showed a case study of a smart house using IoPAT, in which we compared to the regular IoT with respect to the thermal comfort of the occupants.

CHAPTER 8

Conclusion and Future Research

We conclude the contributions of this thesis in this chapter by summarizing the conclusion of each of the three parts of the thesis while giving outlines for a future research direction in the area of personalized and autonomous systems.

8.1 Conclusion

This thesis was divided into three main parts which target the three main challenges of personalized autonomy. The first challenge was context-engines support (Part 1) in which CAreDroid and CAMPS were presented as operating systems support for context-aware applications. The second challenge was the privacy concern that arise from pervasive and personalized systems (Part 2). In this part SpyCon was presented to exploit the information leakage and VindiCo provided detection algorithm and mitigation techniques to address this new spyware. The third challenge was the adaptability of the personalized systems and the problem of taking the human variation into the loop of computation (Part 3). A reinforcement learning algorithm was proposed by Sentio to address this challenge in the area of ADAS while IoPAT provided a generic personalization architecture for the IoT systems.

8.1.1 CAreDroid

Context-aware computing is a powerful technique for physically coupled software. It can enhance functionality and improve resource usage of applications by adapting them to context. In CAreDroid, we present an adaptation framework for context-aware applications in Android. CAreDroid allows applications developers to develop context-aware applications

without having to deal directly with context monitoring and context adaptation in the application code. In CAreDroid, multiple versions of methods that are sensitive to context are dynamically and transparently replaced with each other according to application-specific configuration file. By pushing the context monitoring and adaptation functionalities to the Android runtime, CAreDroid was able to provide context-awareness more efficiently and with significantly fewer lines of code compared to current Android development flow. In particular, using different case studies, we show how CAreDroid can be used to develop context-aware applications. Results from these case studies show that CAreDroid reduces the code complexity by at least half while decreasing the computation overhead by at least a factor of

8.1.2 CAMPS

Humans differ in behavior and preference with respect to charging their phones. Hence, CAMPS started by analyzing the human data in charging his phone. CAMPS provided a new OS mechanisms of charging-aware power management and deferrable task scheduling that could improve overall availability for a significant portion of smartphone users. In particular, CAMPS proposed to utilize the power headroom during certain phases of battery charging to run these tasks, rather than starve the battery of energy during its most power-intensive charging time. Increasing the energy delivered to the battery during the charging period, or conversely, decreasing the required charging duration to reach full SOC would improve overall device availability to the user.

8.1.3 SpyCon

A new class of privacy-threatening spyware that is designed to snoop around adaptations made by context- aware apps was presented as SpyCon. SpyCon showed that through the user study, monitoring the context-based adaptations triggered by context-aware apps, SpyCon could infer user behavior. To exacerbate the situation, experiments showed that this new spyware is undetectable using off-the-shelf antivirus and moreover many of the top

45 downloadable free apps have enough information to reveal about user.

8.1.4 VindiCo

VindiCo was designed as a safeguard which protects authentic context-aware applications against leaking private information via this side-channel. VindiCo employs a general detection technique based on mutual information algorithm which is agnostic to implementation details of context-based spyware and uses three mitigation techniques to hinder the performance of SpyCon, which are delaying, sup- pressing, and masking. An end-to-end use case has been shown to demonstrate the effectiveness of the proposed VindiCo architecture by having a SpyCon monitoring an authentic context-aware phone setting application. Our mitigation techniques have shown a degradation of SpyCon inference accuracy from 90:3% to the baseline accu- racy and by only adding negligible overhead (3%) on the API call performance.

8.1.5 Sentio

In Sentio we focused on the adaptation challenge of ADAS. In particular, we targeted the adaptation to humans to weave personalization into the fabric of ADAS. While FCW is one of the highly adopted ADAS in vehicles, its primary focus is still dependent on the environment around the vehicle (relative distance and relative velocity). However, by taking the driver state into the loop of computation, the FCW can be personalized and provide a better experience. In this chapter, we purposed Sentio, a driver-in-the-loop FCW that learns the driver preference as well as his attention level to signal the FCW at the right time. We proposed a variant of the Q-learning algorithm to solve our problem and enhance the learning rate of the driver preference. Our multisample Q-learning algorithm could track the change in the driving behavior across time and adapt the timing of the FCW accordingly without the need for offline training. By examining the results of driving traces on a car simulator for multiple drivers, Sentio showed a better performance than the fixed threshold FCW that is widely used in commercial vehicles.

8.1.6 IoPAT

We studied the personalization on IoT systems in IoPAT. Our everyday life activities are becoming more dependent on pervasive devices. Since most of these devices interact with humans, the IoT system should include the human factor into the loop of computation. Accordingly, we proposed IoPAT system. IoPAT considered the human preference and current state as an integral part to the IoT system. We showed a case study of a smart house using IoPAT, in which we compared to the regular IoT with respect to the thermal comfort of the occupants.

8.2 Future Research

With the increased prominence of personalized computing in the context of smart cities, healthcare, and automotive systems, a future direction is to build on the research work presented in this thesis for effectively tackling new problems in the area of personalization and autonomy of ubiquitous computing. Similar to the prior research presented in this thesis, the common thread here is a two-pronged approach; end-to-end applications along with systems support for building these applications. In particular, Future work will focus on investigating three applications of personalized computing namely (i) automotive systems, (ii) differentiated education, and (iii) healthcare.

8.2.1 Mobile-Assisted, Context-aware, and Personalized Automotive Systems

As mentioned in Chapter 6, very few pervasive automotive systems take the driver state (or context) into consideration. Thanks to the recent advances in mobile and wearables, complex human states can be inferred with high precision. However, and regardless the recent advances in both mobile/wearables and automotive context-aware systems, there is a little work focusing on the integration between context-aware systems from both sides. In such integrative context-aware systems, information collected from the sensors at both the mobile/wearables and the automotive systems are combined to infer a richer context that

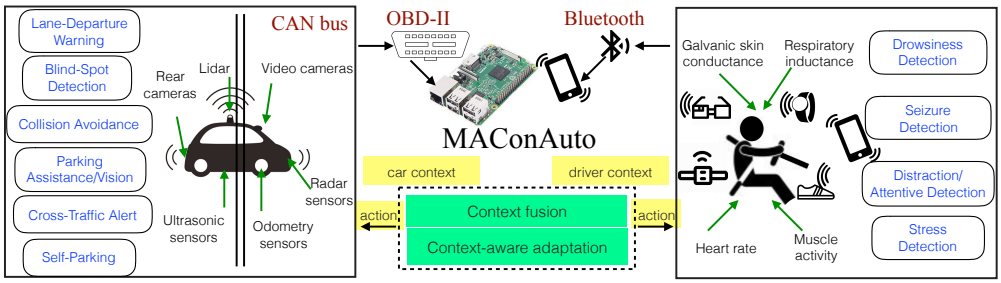


Figure 8.1: A pictorial architecture for the envisioned mobile-assisted, context-aware, and personalized automotive systems. Information communicated over the internal networks inside the automotive system is fused with information collected from various phone/wearables sensors to infer complex human and environment state. This complex state is then used to adapt the behavior of both the car as well as the various apps running over the phone.

can not be determined by any of the two systems (mobile/wearable and automotive) alone. This richer context is then used to adapt the functionality of both systems and provide recommendations and actions to both the user and the car simultaneously. We refer to such systems as Mobile-Assisted, Context-aware, Automotive Systems (a pictorial architecture for these systems is given in Figure 8.1). While such integration between mobile/wearables with automotive systems, if correctly designed, are daunting, several challenges need to be addressed to achieve such system.

The long view of this project is multifold. First, the future research agenda will examine the computing stack on both automotive and mobile/wearable sides to provide systematic support at the OS level and the networking level. Second, develop novel machine-learning algorithms that use the information collected from both the automotive and mobile/wearables to infer complex human contexts. Third, develop novel machine-learning algorithms to handle complex driver-computer interactions. Since the final objective of building such systems is to provide a personalized driving experience, accordingly, an overarching objective of this project is to build a testbed which can be used to test and verify such systems.

8.2.2 Context-Aware Internet-of-Things for Personalized Healthcare

The practice of medicine stands at the threshold of a transformation from its current focus on the treatment of disease events to an emphasis on enhancing health, preventing disease and personalizing care to meet each individual's specific health needs. It is no wonder that this transformation is ignited by the advances in both machine learning and ubiquitous computing that paved the way to developing new technologies capable of monitoring different biological aspects of the human body. Regardless the recent technological advances, there is a need to advance the current status-quo in two directions namely:

- **Turning data into actions:** a common thread in the recent advances in healthcare is to utilize supervised learning techniques for monitoring, diagnosis, and risk stratification leaving small work focusing on decision making. A future direction is exploring various problems in which a personalized system could take actions (e.g., recommending physical exercises, adjusting medication dozes, and designing optimal treatment policies) to enhance human health.

- **Systematic Support for Internet-of-Medical Things:** The widespread access to real-time, high fidelity data on each individual's health is paving the road for personalized healthcare. Unlike the rapid increase of smart sensors in the context of mobile phones and wearables, there is very few work targeting the computing stack for theses systems. Similar to the research presented in this thesis, a future direction is building end-to-end practical systems, along with developing systematic support that helps developers build customized healthcare IoT.

8.2.3 Context-Aware Personalized Differentiated Learning

A third application lies in differentiated instruction and assessment (also known as differentiated learning). Differentiated learning is a teaching philosophy that treats different students differently. In particular, it transforms the one-fits-all teaching process into a process in which different students are given different avenues to learning in terms of: acquiring con-

tent; processing, constructing, or making sense of ideas; and developing teaching materials and assessment measures so that all students within a classroom can learn effectively, regardless of differences in ability. For example, some students may learn better from a video illustration, whereas some students may learn more from a well-organized handout. Or, some students may perform better on a project, but others are good at exams. Therefore, a better strategy would be to develop a personalized educational scheme that takes into account the inherent differences between students, and the scheme should be able to change dynamically according to feedback from the student.

The objective of this research is to build context-aware instruction and assessment systems that adapt to the current state of students and adapt the pedagogical policies that fit the different student needs. Akin to the research presented in this thesis, **this project entails investigating how to build novel reinforcement learning algorithms that take into account the human (student) state and history to provide him with a personalized education material** that best suits his current state. A future direction can be **performing field studies to understand to what extent a personalized differentiated learning systems would affect the pedagogical process for different age levels**. These developed systems will lead to a better and deeper understanding of how to maximize the student learning outcomes while addressing different student needs.

REFERENCES

[ACK13] K. Ariyapala, M. Conti, and C. Keppitiyagama. "ContextOS: A Context Aware Operating System for Mobile Devices." In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pp. 976–984, Aug 2013.

[ACR14] Jagdish Prasad Achara, Mathieu Cunche, Vincent Roca, and Aurélien Francillon. "Short paper: Wifileaks: Underestimated privacy implications of the ACCESS_-WIFI_STATE Android permission." In *Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks*, pp. 231–236. ACM, 2014.

[ACW09] Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. "PetaBricks: a language and compiler for algorithmic choice." *SIGPLAN Notices*, **44**:38–49, June 2009.

[AH13] Yuvraj Agarwal and Malcolm Hall. "ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing." In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 97–110. ACM, 2013.

[AKK13] Christer Ahlstrom, Katja Kircher, and Albert Kircher. "A gaze-based driver distraction warning system and its effect on visual behavior." *IEEE Transactions on Intelligent Transportation Systems*, **14**(2):965–973, 2013.

[AMK15] Fadel Adib, Hongzi Mao, Zachary Kabelac, Dina Katabi, and Robert C Miller. "Smart homes that monitor breathing and heart rate." In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pp. 837–846. ACM, 2015.

[And] Android SDK. "Profiling with Traceview." http://developer.android.com/tools/debugging/.

[aos] "Android Open Source Project." `https://source.android.com/`. [Online; accessed 9-Mar-2016].

[ARF14] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." In *ACM SIGPLAN Notices*, volume 49, pp. 259–269. ACM, 2014.

[ASH10] ASHRAE/ANSI Standard 55-2010 American Society of Heating, Refrigerating, and Air-Conditioning Engineers. "Thermal environmental conditions for human occupancy." *Inc.Atlanta, GA, USA*, 2010.

[BC10] Woongki Baek and Trishul M. Chilimbi. "Green: a framework for supporting energy-conscious programming using controlled approximation." *SIGPLAN Notices*, **45**:198–209, June 2010.

[BGF10] Guangdong Bai, Liang Gu, Tao Feng, Yao Guo, and Xiangqun Chen. "Context-aware usage control for android." In *Security and Privacy in Communication Networks*, pp. 326–343. Springer, 2010.

[BGX10] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. "Fusing mobile, sensor, and social data to fully enable context-aware computing." In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, pp. 60–65. ACM, 2010.

[BRS11] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. "MockDroid: Trading Privacy for Application Functionality on Smartphones." In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pp. 49–54, New York, NY, USA, 2011. ACM.

[BSP03] Rajesh Krishna Balan, Mahadev Satyanarayanan, So Young Park, and Tadashi Okoshi. "Tactics-based remote execution for mobile computing." In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pp. 273–286. ACM, 2003.

[BW34] Th Bedford and CG Warner. "The globe thermometer in studies of heating and ventilation." *Epidemiology & Infection*, **34**(4):458–473, 1934.

[Car07] Robert G. Carroll. "Pulmonary System." In *Elsevier's Integrated Physiology*, chapter 10, pp. 99–115. Elsevier, 2007.

[CBM02] Licia Capra, Gordon S. Blair, Cecilia Mascolo, Wolfgang Emmerich, and Paul Grace. "Exploiting Reflection in Mobile Computing Middleware." *ACM SIGMOBILE Mobile Computing and Communications Review*, **6**(4):34–44, October 2002.

[CCF12] Mauro Conti, Bruno Crispo, Earlence Fernandes, and Yury Zhauniarovich. "Crêpe: A system for enforcing fine-grained context-related policies on android." *Information Forensics and Security, IEEE Transactions on*, **7**(5):1426–1438, 2012.

[CDW03] William Consiglio, Peter Driscoll, Matthew Witte, and William P Berg. "Effect of cellular telephone conversations and other potential interference on reaction time in a braking response." *Accident Analysis & Prevention*, **35**(4):495–500, 2003.

[CG09] Jongyoon Choi and Ricardo Gutierrez-Osuna. "Using heart rate monitors to detect mental stress." In *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, pp. 219–223. IEEE, 2009.

[CGS14] Jeffrey S Campbell, Sidney N Givigi, and Howard M Schwartz. "Multiple-model Q-learning for stochastic reinforcement delays." In *IEEE International Conference on Systems, Man and Cybernetics (SMC), 2014*, pp. 1611–1617. IEEE, 2014.

[CGZ97] Alexandra Canavan, David Graff, and George Zipperlen. "Callhome american english speech." *Linguistic Data Consortium*, 1997.

[CKL11] David Chu, Aman Kansal, Jie Liu, and Feng Zhao. "Mobile Apps: It's Time to Move Up to CondOS." In *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*. USENIX, May 2011.

[CLL11] David Chu, Nicholas D. Lane, Ted Tsung-Te Lai, Cong Pang, Xiangying Meng, Qing Guo, Fan Li, and Feng Zhao. "Balancing Energy, Latency and Accuracy for Mobile Sensor Data Classification." In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pp. 54–67, New York, NY, USA, 2011. ACM.

[CMH13] Adrian K Clear, Janine Morley, Mike Hazas, Adrian Friday, and Oliver Bates. "Understanding adaptive thermal comfort: new directions for UbiComp." In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pp. 113–122. ACM, 2013.

[CSD15] Diane J Cook, Maureen Schmitter-Edgecombe, and Prafulla Dawadi. "Analyzing activity behavior and movement in a naturalistic environment using smart home techniques." *IEEE journal of biomedical and health informatics*, **19**(6):1882–1892, 2015.

[CSR14] Supriyo Chakraborty, Chenguang Shen, Kasturi Rangan Raghavan, Yasser Shoukry, Matt Millar, and Mani Srivastava. "ipShield: a framework for enforcing context-aware privacy." In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp. 143–156, 2014.

[CZ96] Alexandra Canavan and George Zipperlen. "Callfriend american english-non-southern dialect." *Linguistic Data Consortium, Philadelphia*, **10**:1, 1996.

[CZZ15] Lan-lan Chen, Yu Zhao, Jian Zhang, and Jun-zhong Zou. "Automatic detection of alertness/drowsiness from physiological signals using wavelet-based nonlinear features and machine learning." *Expert Systems with Applications*, **42**(21):7344–7355, 2015.

[DB08] Mandalapu Sarada Devi and Preeti R Bajaj. "Driver fatigue detection based on eye tracking." In *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on*, pp. 649–652. IEEE, 2008.

[DMV] DMV. "How Emotions Affect Driving." `https://www.dmv.org/how-to-guides/driving-and-emotions.php`.

[DTB10] Jiangpeng Dai, Jin Teng, Xiaole Bai, Zhaohui Shen, and Dong Xuan. "Mobile phone based drunk driving detection." In *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS*, pp. 1–8. IEEE, 2010.

[DWZ01] Eyal De Lara, Dan S Wallach, and Willy Zwaenepoel. "Puppeteer: Component-based Adaptation for Mobile Computing." In *USENIX Symposium on Internet Technologies and Systems - USITS*, volume 1, pp. 14–14, 2001.

[EGH14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones." *ACM Transactions on Computer Systems (TOCS)*, **32**(2):5, 2014.

[EKK11] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. "PiOS: Detecting Privacy Leaks in iOS Applications." In *NDSS*, 2011.

[Enga] Engineering ToolBox. ""Met - Metabolic Rate"." `https://www.engineeringtoolbox.com/met-metabolic-rate-d_733.html`.

[Engb] Engineering ToolBox. ""Relative Humidity in Air"." `https://www.engineeringtoolbox.com/relative-humidity-air-d_687.html`.

[EOM11] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. "A Study of Android Application Security." In *USENIX security symposium*, volume 2, p. 2, 2011.

[EWS15] Salma Elmalaki, Lucas Wanner, and Mani Srivastava. "CAreDroid: Adaptation Framework for Android Context-Aware Applications." In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 386–399. ACM, 2015.

[Fan70] Poul O Fanger et al. "Thermal comfort. Analysis and applications in environmental engineering." *Thermal comfort. Analysis and applications in environmental engineering.*, 1970.

[FHE12] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. "Android permissions: User attention, comprehension, and behavior." In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, p. 3. ACM, 2012.

[FJ05] Matteo Frigo and Steven G. Johnson. "The Design and Implementation of FFTW3." *Proceedings of the IEEE*, **93**(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[FMK10] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. "Diversity in Smartphone Usage." In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, p. 179, New York, New York, USA, June 2010. ACM Press.

[FPS02] Jason Flinn, SoYoung Park, and Mahadev Satyanarayanan. "Balancing performance, energy, and quality in pervasive computing." In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pp. 217–226. IEEE, 2002.

[GFK05] Todor Ganchev, Nikos Fakotakis, and George Kokkinakis. "Comparative evaluation of various MFCC implementations on the speaker verification task." In *Proceedings of the SPECOM*, volume 1, pp. 191–194, 2005.

[GLC10] Jitendra K Gupta, Chao-Hsin Lin, and Qingyan Chen. "Characterizing exhaled airflow from breathing and talking." *Indoor air*, **20**(1):31–39, 2010.

[Goe58] Gerald Goertzel. "An algorithm for the evaluation of finite trigonometric series." *American mathematical monthly*, pp. 34–35, 1958.

[GZZ12] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. "Riskranker: scalable and accurate zero-day android malware detection." In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 281–294. ACM, 2012.

[hab] "HABU music." `https://play.google.com/store/apps/details?id=com.gravitymobile.habumusic&hl=en`. Online; accessed March 11, 2017.

[Har07] Harvard Medical School. ""The Characteristics of Sleep."." `http://healthysleep.med.harvard.edu/healthy/science/what/characteristics`, 2007.

[HHJ11] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications." In *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 639–652. ACM, 2011.

[HON12] Jun Han, Emmanuel Owusu, Le T Nguyen, Adrian Perrig, and Joy Zhang. "Accomplice: Location inference using accelerometers on smartphones." In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pp. 1–9. IEEE, 2012.

[HYT14] Kun Han, Dong Yu, and Ivan Tashev. "Speech emotion recognition using deep neural network and extreme learning machine." In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[JBN12] Pedro Jiménez, Luis M Bergasa, Jesús Nuevo, Noelia Hernández, and Ivan G Daza. "Gaze fixation system for the evaluation of driver distractions induced by IVIS." *IEEE Transactions on Intelligent Transportation Systems*, **13**(3):1167–1178, 2012.

[JMV12] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications." In *Proceedings of the Second ACM*

Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '12, pp. 3–14, New York, NY, USA, 2012. ACM.

[KLJ08] Seungwoo Kang, Jinwon Lee, Hyukjae Jang, Hyonik Lee, Youngki Lee, Souneil Park, Taiwoo Park, and Junehwa Song. "SeeMon: Scalable and Energy-efficient Context Monitoring Framework for Sensor-rich Mobile Environments." In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, MobiSys '08, pp. 267–280, New York, NY, USA, 2008. ACM.

[KMP11] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. "DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation." In *NDSS*, 2011.

[KSB13] Aman Kansal, Scott Saponas, A.J. Bernheim Brush, Kathryn S. McKinley, Todd Mytkowicz, and Ryder Ziola. "The Latency, Accuracy, and Battery (LAB) Abstraction: Programmer Productivity and Energy Efficiency for Continuous Mobile Context Sensing." In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pp. 661–676, New York, NY, USA, 2013. ACM.

[KSW70] LA Kuehn, RA Stubbs, and RS Weaver. "Theory of the globe thermometer." *Journal of applied physiology*, **29**(5):750–757, 1970.

[KT10] SCOTT THURMand YUKARI IWATANI Kane and Scott Thurm. "Your Apps Are Watching You." *WALL ST. J.(Dec. 17, 2010), http://on. wsj. com/wq7Wiw*, 2010.

[LCG15] Stéphanie Lefèvre, Ashwin Carvalho, Yiqi Gao, H Eric Tseng, and Francesco Borrelli. "Driver models for personalised driving assistance." *Vehicle System Dynamics*, **53**(12):1705–1720, 2015.

[LGP07] X. Li, M.J. Garzaran, and D. Padua. "Optimizing Sorting with Machine Learning Algorithms." In *Proceedings of Parallel and Distributed Processing Symposium*, March 2007.

[Lib] LibriVox. "librivox-free public domain audiobooks." `https://librivox.org`.

[LLH13] Ting-Yi Lin, Ting-An Lin, Cheng-Hsin Hsu, and Chung-Ta King. "Context-aware decision engine for mobile cloud offloading." In *Wireless Communications and Networking Conference Workshops (WCNCW), 2013 IEEE*, pp. 111–116, April 2013.

[LLL13] Robert LiKamWa, Yunxin Liu, Nicholas D Lane, and Lin Zhong. "Moodscope: Building a mood sensor from smartphone usage patterns." In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 389–402. ACM, 2013.

[LLW12] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. "Chex: statically vetting android apps for component hijacking vulnerabilities." In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 229–240. ACM, 2012.

[LMM07] Andreas Lachenmann, Pedro José Marrón, Daniel Minder, and Kurt Rothermel. "Meeting lifetime goals with energy levels." In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pp. 131–144, New York, NY, USA, 2007. ACM.

[loc] "Locale Application." `https://play.google.com/store/apps/details?id=com.twofortyfouram.locale&hl=en`. [Online; accessed 9-Mar-2016].

[LPL12] Oscar D Lara, Alfredo J Pérez, Miguel A Labrador, and José D Posada. "Centinela: A human activity recognition system based on acceleration and vital sign data." *Pervasive and mobile computing*, **8**(5):717–729, 2012.

[LS13] Terry C Lansdown and Amanda N Stephens. "Couples, contentious conversations, mobile telephone use and driving." *Accident Analysis & Prevention*, **50**:416–422, 2013.

[LSS10] Jiakang Lu, Tamim Sookoor, Vijay Srinivasan, Ge Gao, Brian Holben, John Stankovic, Eric Field, and Kamin Whitehouse. "The smart thermostat: using occupancy sensors to save energy in homes." In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pp. 211–224. ACM, 2010.

[LWG13] Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov. "πBox: A Platform for Privacy-Preserving Apps." In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 501–514, Lombard, IL, 2013. USENIX.

[LY15] Donghoon Lee and Hwasoo Yeo. "A study on the rear-end collision warning system by considering different perception-reaction time using multi-layer perceptron neural network." In *Intelligent Vehicles Symposium (IV)*, pp. 24–30. IEEE, 2015.

[LYL10] Hong Lu, Jun Yang, Zhigang Liu, Nicholas D. Lane, Tanzeem Choudhury, and Andrew T. Campbell. "The Jigsaw Continuous Sensing Engine for Mobile Phone Applications." In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pp. 71–84, New York, NY, USA, 2010. ACM.

[MBN14] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. "Gyrophone: Recognizing speech from gyroscope signals." In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 1053–1067, 2014.

[MDE13] F. Muehlfeld, I. Doric, R. Ertlmeier, and T. Brandmeier. "Statistical Behavior Modeling for Driver-Adaptive Precrash Systems." *IEEE Transactions on Intelligent Transportation Systems*, **14**(4):1764–1772, Dec 2013.

[Mec] Mechanical Simulation Corporation. "Mechanical Simulation." https://www.carsim.com.

[MJV13] Benoit Mariani, Mayté Castro Jiménez, François JG Vingerhoets, and Kamiar Aminian. "On-shoe wearable sensors for gait and turning assessment of patients with Parkinson's disease." *IEEE transactions on biomedical engineering*, **60**(1):155–158, 2013.

[MKC13] Ralph Oyini Mbouna, Seong G Kong, and Myung-Geun Chun. "Visual analysis of eye state and head pose for driver alertness monitoring." *IEEE transactions on intelligent transportation systems*, **14**(3):1462–1469, 2013.

[MSV15] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. "PowerSpy: Location Tracking Using Mobile Device Power Analysis." In *USENIX Security*, pp. 785–800, 2015.

[Muk15] Subhas Chandra Mukhopadhyay. "Wearable sensors for human activity monitoring: A review." *IEEE sensors journal*, **15**(3):1321–1330, 2015.

[Mur00] Kevin P Murphy. "A survey of POMDP solution techniques." *environment*, **2**:X3, 2000.

[MVB12] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. "Tapprints: your finger taps have fingerprints." In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 323–336. ACM, 2012.

[MVC11] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. "(sp) iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers." In *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 551–562. ACM, 2011.

[MVP04] Jason S McCarley, Margaret J Vais, Heather Pringle, Arthur F Kramer, David E Irwin, and David L Strayer. "Conversation disrupts change detection in complex traffic scenes." *Human factors*, **46**(3):424–436, 2004.

[Nat12] Suman Nath. "ACE: Exploiting Correlation for Energy-efficient and Continuous Context Sensing." In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pp. 29–42, New York, NY, USA, 2012. ACM.

[New] New York State Department of Motor Vehicles. "Defensive Driving." https://dmv.ny.gov/about-dmv/chapter-8-defensive-driving.

[NGC16] Joao B Pinto Neto, Lucas C Gomes, Eduardo M Castanho, Miguel Elias M Campista, Luís Henrique MK Costa, and Paulo Cezar M Ribeiro. "An error correction algorithm for forward collision warning applications." In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pp. 1926–1931. IEEE, 2016.

[NKZ10] Mohammad Nauman, Sohail Khan, and Xinwen Zhang. "Apex: extending android permission model and enforcement with user-defined runtime constraints." In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 328–332. ACM, 2010.

[NSN97] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. "Agile Application-aware Adaptation for Mobility." *SIGOPS Oper. Syst. Rev.*, **31**(5):276–287, October 1997.

[NYY15] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. "UIPicker: User-Input Privacy Identification in Mobile Applications." In *24th USENIX Security Symposium (USENIX Security 15)*, pp. 993–1008, Washington, D.C., August 2015. USENIX Association.

[OGB11] Olukunle Ojetola, Elena I Gaura, and James Brusey. "Fall detection with wearable sensors–safe (Smart Fall Detection)." In *Seventh International Conference on Intelligent Environments (IE), 2011*, pp. 318–321. IEEE, 2011.

[OHD12] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. "ACCessory: password inference using accelerometers on smartphones." In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, p. 9. ACM, 2012.

[OT16] Eshed Ohn-Bar and Mohan Manubhai Trivedi. "Looking at humans in the age of self-driving and highly automated vehicles." *IEEE Transactions on Intelligent Vehicles*, **1**(1):90–104, 2016.

[Pan07] Panasonic Corporation. "Lithium Ion Batteries Technical Handbook.", 2007.

[Pic95] Rosalind W Picard. "Affective Computing-MIT Media Laboratory Perceptual Computing Section Technical Report No. 321." *Cambridge, MA*, **2139**, 1995.

[PZC14] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. "Context Aware Computing for The Internet of Things: A Survey." *Communications Surveys Tutorials, IEEE*, **16**(1):414–454, First 2014.

[RBE08] Sasank Reddy, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. "Determining transportation mode on mobile phones." In *Wearable Computers, 2008. ISWC 2008. 12th IEEE International Symposium on*, pp. 25–28. IEEE, 2008.

[RGK11] Andrew Raij, Animikh Ghosh, Santosh Kumar, and Mani Srivastava. "Privacy risks emerging from the adoption of innocuous wearable sensors in the mobile environment." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 11–20. ACM, 2011.

[RLB04] Wang Rongben, Guo Lie, Tong Bingliang, and Jin Lisheng. "Monitoring mouth movement for driver fatigue or distraction with one camera." In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pp. 314–319. IEEE, 2004.

[RLR09] Jason Ryder, Brent Longstaff, Sasank Reddy, and Deborah Estrin. "Ambulation: A tool for monitoring mobility patterns over time using mobile phones." In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 4, pp. 927–931. IEEE, 2009.

[roc] "Rock My Run." `www.rockmyrun.com`. Online; accessed March 11, 2017.

[RV11] Stephanie Rosenthal and Manuela Veloso. "Modeling humans as observation providers using pomdps." In *RO-MAN, 2011 IEEE*, pp. 53–58. IEEE, 2011.

[RZ09] Ahmad Rahmati and Lin Zhong. "Human-Battery Interaction on Mobile Phones." *Pervasive and Mobile Computing*, **5**(5):465–477, October 2009.

[Sat01] Mahadev Satyanarayanan. "Pervasive computing: Vision and challenges." *Personal Communications, IEEE*, **8**(4):10–17, 2001.

[SBT05] G Seematter, C Binnert, and L Tappy. "Stress and metabolism." *Metabolic syndrome and related disorders*, **3**(1):8–13, 2005.

[SCP15] David Snyder, Guoguo Chen, and Daniel Povey. "Musan: A music, speech, and noise corpus." *arXiv preprint arXiv:1510.08484*, 2015.

[SCW18] Yasser Shoukry, Michelle Chong, Masashi Wakaiki, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A Seshia, Joao P Hespanha, and Paulo Tabuada. "SMT-based observer design for cyber-physical systems under sensor attacks." *ACM Transactions on Cyber-Physical Systems*, **2**(1):5, 2018.

[SDS17] Chen Su, Weiwen Deng, Hao Sun, Jian Wu, Bohua Sun, and Shun Yang. "Forward collision avoidance systems considering driver's driving behavior recognized by Gaussian Mixture Model." In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pp. 535–540. IEEE, 2017.

[sil] "Silence 2.0." `http://downloads.tomsguide.com/Silence,0301-52850.html`. Online; accessed March 11, 2017.

[SJP15] Keshav Seshadri, Felix Juefei-Xu, Dipan K Pal, Marios Savvides, and Craig P Thor. "Driver cell phone usage detection on strategic highway research program (SHRP2) face view videos." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 35–43, 2015.

[SKG07] Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. "Eon: a language and runtime system for perpetual systems." In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pp. 161–174, New York, NY, USA, 2007. ACM.

[Smi82] Brian Cantwell Smith. *Procedural reflection in programming languages*. PhD thesis, Massachusetts Institute of Technology, 1982.

[Son14] Sony Corporation. "Lithium Ion Rechargeable Batteries: Technical Handbook.", 2014.

[soo] "Soot - A framework for analyzing and transforming Java and Android Applications." https://sable.github.io/soot/. Online; accessed March 11, 2017.

[SP13] Akane Sano and Rosalind W Picard. "Stress recognition using wearable sensors and mobile phones." In *Humaine Association Conference on Affective Computing and Intelligent Interaction (ACII)*, pp. 671–676. IEEE, 2013.

[SRN12] Tal Shany, Stephen J Redmond, Michael R Narayanan, and Nigel H Lovell. "Sensors-based wearable systems for monitoring of human movement and falls." *IEEE Sensors Journal*, **12**(3):658–670, 2012.

[Sum00] Heikki Summala. "Brake reaction times and driver behavior analysis." *Transportation Human Factors*, **2**(3):217–226, 2000.

[tas] "Tasker app." https://play.google.com/store/apps/details?id=net.dinglisch.android.taskerm&hl=en. [Online; accessed 9-Mar-2016].

[The] The Statistics Portal. "Projected global ADAS revenue growth trend from 2012 to 2020." https://www.statista.com/statistics/442726/global-revenue-growth-trend-of-advanced-driver-assistance-systems/ .

[The12] The MathWorks, Inc. ""Thermal Model of a House"." https://www.mathworks.com/help/simulink/examples/thermal-model-of-a-house.html, 2012.

[Tsi94] John N Tsitsiklis. "Asynchronous stochastic approximation and Q-learning." *Machine learning*, **16**(3):185–202, 1994.

[TST14] Ashish Tawari, Sayanan Sivaraman, Mohan Manubhai Trivedi, Trevor Shannon, and Mario Tippelhofer. "Looking-in and looking-out vision for urban intelligent assistance: Estimation of driver attentive state and dynamic surround for safe merging and braking." In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pp. 115–120. IEEE, 2014.

[US] U.S Census Bureau. "Commuting Times, Median Rents and Language other than English Use in the Home on the Rise." https://www.census.gov/newsroom/press-releases/2017/acs-5yr.html.

[VC11] Narseo Vallina-Rodriguez and Jon Crowcroft. "ErdOS: Achieving Energy Savings in Mobile OS." In *Proceedings of the Sixth International Workshop on MobiArch*, MobiArch '11, pp. 37–42, New York, NY, USA, 2011. ACM.

[WBS11] Martin Wollmer, Christoph Blaschke, Thomas Schindl, Björn Schuller, Berthold Farber, Stefan Mayer, and Benjamin Trefflich. "Online driver distraction detection using long short-term memory." *IEEE Transactions on Intelligent Transportation Systems*, **12**(2):574–582, 2011.

[WCZ16] Xuesong Wang, Ming Chen, Meixin Zhu, and Paul Tremont. "Development of a Kinematic-Based Forward Collision Warning Algorithm Using an Advanced Driving Simulator." *IEEE Transactions on Intelligent Transportation Systems*, **17**(9):2583–2591, 2016.

[WGT11] Liang Wang, Tao Gu, Xianping Tao, Hanhua Chen, and Jian Lu. "Recognizing multi-user activities using wearable sensors in a smart home." *Pervasive and Mobile Computing*, **7**(3):287–298, 2011.

[WLR15] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. "Mole: Motion leaks through smartwatch sensors." In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 155–166. ACM, 2015.

[WRB13] Daniel T. Wagner, Andrew Rice, and Alastair R. Beresford. "Device Analyzer: Understanding smartphone usage." In *Proceedings of the International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Tokyo, Japan, 2013. ACM.

[WYL16] Jianqiang Wang, Chenfei Yu, Shengbo Eben Li, and Likun Wang. "A forward collision warning algorithm with adaptation to driver behaviors." *IEEE Transactions on Intelligent Transportation Systems*, **17**(4):1157–1167, 2016.

[XZ15] Zhi Xu and Sencun Zhu. "SemaDroid: A Privacy-Aware Sensor Management Framework for Smartphones." In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 61–72. ACM, 2015.

[XZS10] Liang Xie, Xinwen Zhang, Jean-Pierre Seifert, and Sencun Zhu. "pBMDS: a behavior-based malware detection system for cellphone devices." In *Proceedings of the third ACM conference on Wireless network security*, pp. 37–48. ACM, 2010.

[YY12] Lok Kwong Yan and Heng Yin. "Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis." In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 569–584, 2012.

[YZL15] Feng You, Ronghui Zhang, Guo Lie, Haiwei Wang, Huiyin Wen, and Jianmin Xu. "Trajectory planning and tracking control for autonomous lane change maneuver based on the cooperative vehicle infrastructure system." *Expert Systems with Applications*, **42**(14):5932–5946, 2015.

[ZDH13] Xiaoyong Zhou, Soteris Demetriou, Dongjing He, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, Carl A Gunter, and Klara Nahrstedt. "Identity, location,

disease and more: Inferring your secrets from android public resources." In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1017–1028. ACM, 2013.

[ZGF11] Xia Zhao, Yao Guo, Qing Feng, and Xiangqun Chen. "A System Context-aware Approach for Battery Lifetime Prediction in Smart Phones." In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pp. 641–646, New York, NY, USA, 2011. ACM.

[ZJ04] Zhiwei Zhu and Qiang Ji. "Real time and non-intrusive driver fatigue monitoring." In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pp. 657–662. IEEE, 2004.

[ZJS09] Yu Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall. *Privacy scope: A precise information flow tracking system for finding application leaks*. PhD thesis, Citeseer, 2009.

[ZWZ12] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets." In *NDSS*, 2012.