UNIVERSITY OF CALIFORNIA
RIVERSIDE

Cross-Layer Design of Reliable and Secure Cyber-Physical Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

by

Bowen Zheng

June 2018

Acknowledgments

I would like to first give my deepest appreciation to my advisor - Professor Qi Zhu. Without his guidance, I would not have been able to complete this dissertation. Since I joined the University of California, Riverside five years ago, Professor Qi Zhu has guided me through the gate of various research challenges in the domain of cyber-physical systems and provided me with valuable insights for not only academic research but also life and career. Therefore, I have not only learned the skills to lead a research, such as critical thinking, problem solving and academic writing, but also commitment, teamwork, and the passionate attitude. I am fortunate to obtain these valuable capabilities from my advisor.

I would like to thank Dr. Chung-Wei Lin and Dr. Huafeng Yu at Toyota InfoTechnology Center. It was a wonderful experience to intern at Toyota and it was amazing to obtain continuing collaboration with Dr. Lin and Dr. Yu afterwards. The insights from the industry provided me another way of viewing the research challenges and broadened my research towards autonomous vehicles and vehicular communication. I would like to thank Professor Fabio Pasqualetti who was in collaboration with our lab. The weekly discussion with the group of Professor Pasqualetti brought me a lot of ideas and insights from the control domain, and helped me a lot with the problem formulation. I would like to thank Professor Sandeep Gupta and his student Yue Gao for my first Ph.D. project and helped me to learn the way of doing research. I am also grateful for Professor Sheldon Tan for the project collaborated with his student Taeyoung Kim, and always being helpful for my research. There are many other brilliant researchers to thank, including but not limited to Shinichi Shiraishi, Wenchao Li, Rajasekhar Anguluri, Gianluca Bianchin, Akila Ganlath, Taeyoung Kim, Peng Deng and Hengyi Liang. I would like to give special thanks to Professor Nael Abu-Ghazaleh and Professor Hyoseung Kim who serve as my committee and provide me with precious recommendations on my research and dissertation. I would also give special thanks to all the professors who helped me during my study.

I would also acknowledge ACM/IEEE Design Automation Conference (DAC), IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), and ACM Transactions on Design Automation of Electronic Systems (TODAES) for publishing my works in 2015. I would also like to acknowledge IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems (TCAD) and IEEE Computer Society Annual Symposium on VLSI (ISVLSI) for publishing my works in 2016, IEEE International Conference on Smart Computing (SMARTCOMP) for publishing my work in 2017, and IEEE/ACM International Conference on Computer-Aided Design (ICCAD) for publishing my works in 2016 and 2017.

I am also grateful for working with my lab colleagues Peng Deng, Tianshu Wei, Hengyi Liang, Shuyue Lan, Bryan Marsh, Zhilu Wang, Hyungjong Choi and Di Chen. They not only brought brilliant work to our lab but also enlightened me through various discussions. I am fortunate to meet my friends at UC Riverside, including Xiaoxiong Ding, Kuan Zhou, Linchao Liao, Yan Zhu, Peng Wang, Shaocheng Wang, Yue Cao, Xing Zheng, Shukui Zhang, Hongsheng Yu, Luting Yang, and many others. I would like to thank my best friend Zihao Wang who encouraged me a lot when I am encountered with difficulties and we had great times composing music and traveling during our leisure time. It made my Ph.D. life more colorful and enjoyable.

In the end, I would express my deepest gratitude to my family. My parents always provide me with the best education they can afford and always support me to pursue my dreams. My grandma helped take care of me during my school years and she always concerns about my study and living. I am fortunate I have my family as the powerful backing force on my road to pursue dreams.

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Cross-Layer Design of Reliable and Secure Cyber-Physical Systems

by

Bowen Zheng

Doctor of Philosophy, Graduate Program in Electrical Engineering
University of California, Riverside, June 2018
Dr. Qi Zhu, Chairperson

The design of secure and reliable cyber-physical systems has become increasingly challenging due to the growing complexity of their software and hardware, as well as the interactions among different systems and with the physical environment. Next-generation automotive systems are representative cyber-physical systems with such challenges. They are not only capable of conducting perception, planning, and control through complex software and hardware within the vehicle, but also able to communicate with other vehicles and roadside infrastructures for safety and efficiency applications. The design, analysis, and validation of vehicular applications (such as cooperative adaptive cruise control and intersection management) involve multiple layers: the application layer, the software layer, and the hardware layer. Furthermore, various metrics and stringent requirements, such as timing, safety, security, and fault-tolerance, makes the design, analysis, and validation even more challenging.

To cope with these challenges, we present CONVINCE, a cross-layer modeling, exploration, and validation framework for the design of next-generation automotive systems. CONVINCE is a holistic framework containing mathematical models, synthesis and validation algorithms, and simulation of both inter-vehicle and intra-vehicle behaviors. Various metrics are considered across multiple layers of the framework.

At the application and software layers, we consider cooperative adaptive cruise control and intelligent intersection management to address the challenges from communication delays and possible security attacks. We present a *delay-tolerant* protocol for intelligent intersection management and conduct modeling, simulation, and verification for analyzing the safety, liveness, and performance of the protocol. We also develop a codesign approach for addressing the trade-off between security and control performance with the consideration of implementation feasibility. At the software and hardware layers, we address the software to hardware mapping considering fault-tolerance and security. We conduct fault-tolerance design to improve system-level error recovery rate by applying soft error detection and recovery mechanisms with real-time constraints. We also present the security-aware mapping for both CAN-based and TDMA-based systems with limited resources and strict timing constraints. We have conducted experiments with industrial applications and synthetic examples for our cross-layer framework and demonstrated its effectiveness.

Contents

List of Figures **xii**

List of Tables **xiv**

1 Introduction **1**
- 1.1 Design Challenges . . . 2
  - 1.1.1 In-Vehicle Design Challenges . . . 3
  - 1.1.2 Connected Vehicle Design Challenges . . . 8
- 1.2 Related Work . . . 13
  - 1.2.1 In-Vehicle Design . . . 13
  - 1.2.2 Connected Vehicle Applications . . . 18
- 1.3 Contributions . . . 20

2 CONVINCE Framework **22**
- 2.1 Overview . . . 23
  - 2.1.1 Application Level Verification and Validation . . . 25
  - 2.1.2 Software Modeling . . . 26
  - 2.1.3 Hardware Modeling . . . 26
  - 2.1.4 Analysis Models . . . 27
- 2.2 Summary . . . 32

3 Cross-Layer Design of Functional and Software Layers **34**
- 3.1 Cooperative Adaptive Cruise Control . . . 35
  - 3.1.1 Timing Attacks . . . 35
  - 3.1.2 CACC Application . . . 37
  - 3.1.3 Simulation-Based Analysis of CACC performance . . . 39
- 3.2 Delay-Tolerant Intelligent Intersection Management . . . 42
  - 3.2.1 Basic System Model . . . 44
  - 3.2.2 Problems with Communication Delay . . . 47
  - 3.2.3 Delay-tolerant Intersection Management Protocol . . . 48
  - 3.2.4 Timed Automata for UPPAAL Verification . . . 55
  - 3.2.5 Extensions to Multiple-Lane Intersections . . . 59

3.2.6 Simulation-Based Analysis of Intersection Management Performance 63
3.3 Security, Control, and Schedulability Codesign 72
3.3.1 General Codesign Formulations 72
3.3.2 Automotive Domain Codesign 85
3.3.3 Automotive Case Study Results 94
3.4 Summary 107

4 Cross-Layer Design of Software and Hardware Layers 108
4.1 Fault-Tolerance-Aware Mapping 108
4.1.1 System Model 110
4.1.2 Single-core CPU Platform 116
4.1.3 Multicore CPU Platform 123
4.1.4 Distributed Platform 127
4.1.5 Simulator for Evaluating Error Coverage 129
4.1.6 Experiments 132
4.2 Security-Aware Mapping 140
4.3 Summary 142

5 Conclusions and Future Directions 143

Bibliography 146

List of Figures

1.1 Typical autonomous driving functions. 4
1.2 A hypothetical architecture for autonomous vehicles. 5

2.1 Cross-layer design for connected vehicles. 24
2.2 CONVINCE: cross-layer modeling, exploration and validation framework for connected vehicles. 25

3.1 Packet loss rate under different strengths of flooding attack. During the simulation, 50 vehicles are uniformly distributed on a road of length 300m and normal packet sending rate is 10 Hz. 36
3.2 Spacing-time diagram of 3 vehicles in CACC under different strengths of flooding attacks. 41
3.3 Speed-time diagram of 3 vehicles in CACC under different strengths of flooding attacks. 41
3.4 The refined modeling, simulation and verification framework for the proposed delay-tolerance intersection management protocol. 43
3.5 The intelligent intersection management system. 45
3.6 Issues of the basic protocol under communication delays. 46
3.7 Vehicle state machine. 50
3.8 Intersection Manager state machine. 53
3.9 The timed automata modeled in UPPAAL. 56
3.10 An example showing the intersection model. 60
3.11 The screenshots of simulation tools in our framework. 63
3.12 The performance comparison between the proposed protocol and traditional traffic lights. K: north-south directions have K times the traffic arriving rate as the east-west directions 65
3.13 Performance of our protocol under different communication delays. 67
3.14 Setup for single-intersection simulations. 68
3.15 Performance of a single intersection under different communication delays. . 69
3.16 Simulations based on SUMO. 70
3.17 Performance of nine interconnected intersections under the same attack. . . 71

3.18 Performance of nine intersections (denoted by "A" to "I") if one intersection is under timing attack. 71
3.19 A cyber-physical systems with multiple control loops sharing an embedded platform. Attackers may eavesdrop on the communication medium and apply various attacks. 73
3.20 A motivating example: (a) no message is encrypted, (b) both messages are encrypted and the period of C_2 has to be increased, which leads to higher security level but lower control performance. 75
3.21 Illustration of the interface variables and their relations to design metrics. . 76
3.22 Fitting the control performance of the Simulink electrohydraulic servo example for different sampling periods with an exponential decay function. . . . 87
3.23 The modified automotive subsystem used in the case study. 97
3.24 Mean and standard deviation of the Kalman filter performance under different number of encrypted messages and different key lengths (simulations of control loop 3). 99
3.25 Pareto front between normalized control performance and security level for the industrial example. An example feasible region denotes all feasible solutions under requirement that control performance ≥ 0.3 and security level $\geq$ 0.3. 100
3.26 Impact of sensing noise on security level and encrypted messages for the industrial example: (a) The number of messages encrypted under different noise levels and different security requirements. (b) The noise level is fixed at 9%*op. (c) The security level set as 0.3. 102
3.27 Pareto front between normalized control performance and security level using Observability Gramian and Kalman filter for the industrial example. 103
3.28 Pareto front between normalized control performance and security level for synthetic examples with different number of control tasks (using Kalman filter based security level measurement). 105
3.29 Pareto front between normalized control performance and security level for synthetic examples with different number of control tasks (using Observability Gramian based security level measurement). 106

4.1 A fault-tolerant design example for single-core CPU platform: (a) only EOC is used, (b) only EED is used, (c) EOC/EED selection. 117
4.2 Task duplication model on a dual-core CPU 124
4.3 Task concentration model on a dual-core CPU 126
4.4 Comparison of various methods for synthetic examples on a single-core platform 137
4.5 Comparison of various methods for synthetic examples on a dual-core platform (task concentration model) 138
4.6 The percentage of EOC used vs. EED used under different EED overhead and error detection rate for modified industrial example 139
4.7 The security challenges in automotive systems 140
4.8 Security-aware mapping for TDMA-based systems 141

List of Tables

3.1 Security level of control loop 3 with different number of encrypted messages. 99
3.2 Algorithm runtime of simulated annealing under different number of control tasks 107

4.1 System error coverage under different number of cores for the industrial example 134
4.2 System error coverage under different CPU speeds and number of cores for the industrial example (multicore platform) 135
4.3 System error coverage under different end-to-end latency requirements and number of ECUs for the industrial example (distributed platform) 136

Chapter 1

Introduction

A cyber-physical system (CPS) is an integration of embedded computing, communication, and control techniques interacting with physical processes [91, 122]. Typically, embedded computing controls the physical processes and in turn, the physical processes provide information from the sensors to embedded software through communication networks, thus forming a feedback loop. Cyber-physical systems are prevalent in various domains such as automotive systems, medical devices, and energy management systems. With the advancement in these application domains, both the software and hardware become increasingly complex, making the design process more and more challenging, especially when considering various design metrics, including timing, system performance, security, reliability, etc. Furthermore, the interconnection of individual cyber-physical systems increases the complexity and may bring a number of attack surfaces. In this chapter, we use the automotive system to summarize the design challenges from individual systems to the connected applications and introduce the related works at each layer.

1.1 Design Challenges

Security and reliability have become pressing issues for cyber-physical systems. Cyber-security attacks to systems such as vehicles and industrial robots can lead to not only privacy loss but also life-threatening consequences. The soft errors caused by transient faults and the failures caused by permanent faults can also lead to catastrophic system failures in extreme cases. These issues become more challenging due to the increasing complexity of both software and hardware, the tight computation and communication constraints and the dynamically changing physical environment.

A typical cyber-physical system is the automotive system. Next-generation automotive systems will become more autonomous and connected. Significant progress has been made for autonomous driving in recent years. Although testing autonomous vehicles can already be seen on real roads in many countries, accidents have been reported for big companies like Uber, Tesla, and Google [1, 3, 8]. The design, analysis, and validation of secure and reliable CPS like automotive systems still encounter tremendous challenges, especially under stringent resource and real-time constraints.

The security and reliability issues become even more challenging when considering connected applications, such as intersection management and cooperative adaptive cruise control. The open environments lead to less robust communication, less predictive timing behaviors, and more attack surfaces.

In the rest of the section, we introduce the challenges for in-vehicle design and inter-vehicle applications, respectively.

1.1.1 In-Vehicle Design Challenges

Functionality Complexity

From the functionality perspective, complex applications like autonomous driving post more challenges on the software design. Figure. 1.1 shows the basic functions of a typical autonomous vehicle, including perception, planning, behavioral executive, and motion control [148, 74, 25, 74, 114, 161]. The perception module collects data from a variety of sensors (e.g., LIDAR, radar, GPS, cameras, and ultrasound sensors) and detects obstacles, road shape, and other critical information from the environment. The planning module makes driving decisions based on the real-time data from the perception module. The behavior executive module specifies driving behavior under different environments, such as intersection, parking lot, and high way. Based on the driving behavior, the motion control module physically adjusts actuators such as steering, acceleration, and braking. It is estimated by Morgan Stanley that the software can account for 40% of the value of an autonomous vehicle [32]. Simply from the year 2000 to 2010, the number of lines of embedded software increased from one million to more than ten million [28, 101, 131]. The increasing complexity in functionality requires the shift from traditional federated architecture to integrated architecture as described in the following subsection.

Architecture Complexity

Traditionally, automotive manufacturers or OEMs (original equipment manufacturers) obtain parts from Tier-1 suppliers and assemble the vehicle. Many of the subsystems are specified by the OEMs but built by the Tier-1 suppliers and provided as a black-box

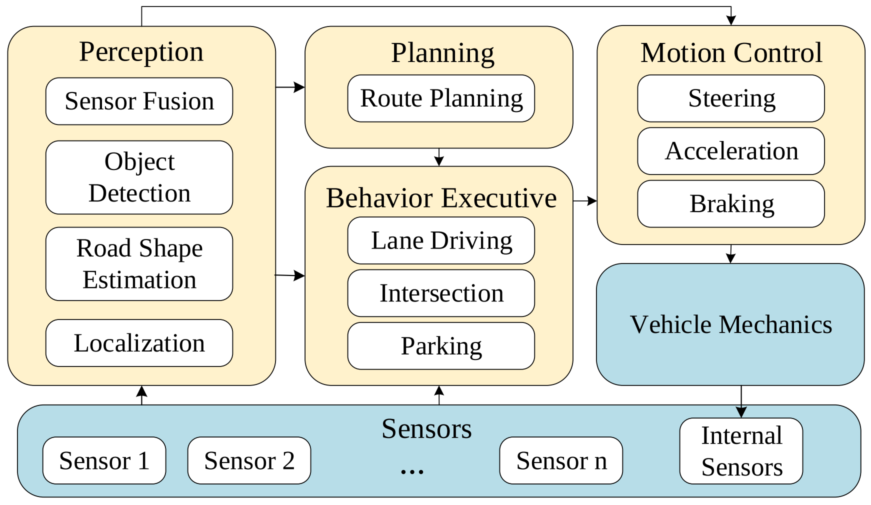


Figure 1.1: Typical autonomous driving functions.

(a functionality deployed on one ECU with interfaces). The designers connect multiple ECUs through buses like CAN (controller area network) and build the electronic system. This architecture is called *federated architecture* and it leads to 50 to 100 ECUs in current automotive systems [11, 27]. The trend is to reduce the number of ECUs and increase the computation power of the computation units [24]. As a result, multiple functions can share one ECU and one function can be distributed over multiple ECUs [38]. This architecture is called *integrated architecture*. However, the sharing and contention among multi-core and distributed systems become another challenge. In addition, new computational components such as Field Programmable Gate Array (FPGA) [57, 133] and Graphical Processing Unit (GPU) [65, 90] can be adopted for computationally-intensive applications, such as video and image processing for autonomous vehicles. This forms a heterogeneous system as shown in Figure. 1.2, where various computation units (ECU, FPGA, and GPU) are connected to

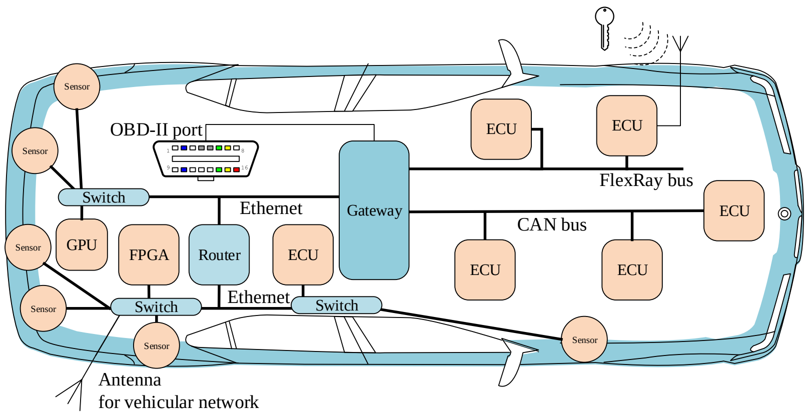


Figure 1.2: A hypothetical architecture for autonomous vehicles.

the bus system. The bus system is also heterogeneous as different modules require different bus speed and reliability. For example, autonomous driving applications require the bus to transmit a large amount of data which beyond the limits of buses like CAN, LIN, or FlexRay. Protocols based on the Ethernet [69, 70, 128, 129], have emerged to be good candidates, such as Time-Sensitive Networking (formerly known as Ethernet AVB [124, 41]) and Time-Triggered Ethernet [58]. In the heterogeneous bus system, a gateway is adopted to interconnect the buses and thus the on-board diagnostics-II (OBD-II) ports are also connected through the gateway for engineers and technicians to monitor the intra-vehicle traffic and diagnose problems.

High Volume and Dynamic Data

A large amount of data generated from the sensors need to be transmitted and processed in real time in future automotive systems. The experimental autonomous vehicle from Google (Waymo) is reported to generate 750MB data from sensors [2]. The internal bus system is required to transmit these data in real time and various components also need to process them in real time. The real-time constraints make it challenging to design an efficient and reliable architecture for such volume of data. As cyber-physical systems interact closely with the environment, the workload may vary significantly in different surroundings [33]. For instance, the workload in urban areas is typically higher than in rural areas as the perception module need to detect more objects and deal with more complex situations in urban areas. The dynamic workload from the environment uncertainty brings in uncertainty in software design.

Various Design Requirements

A variety of objectives and metrics need to be addressed across the design, analysis and validation stages of the automotive systems. These metrics and objectives include timing, performance, fault tolerance, and security. As shown in [36, 37, 53, 160, 158], these metrics often conflict with each other due to the timing and resource requirements of the system. For example, shorter sampling periods and end-to-end latencies of control loops usually increase the control performance [37], but may have negative impacts on schedulability [37] and security [158] due to the decrease of timing slacks in the system. In this thesis, we mainly focus on timing, security, and fault-tolerance and the possible trade-off among them.

- *Timing.* Timing is essential in cyber-physical systems as CPS requires real-time interaction with the physical environment. For example, when you press the brake pedal of a vehicle, the brake system needs to physically brake the vehicle within a deadline to guarantee safety. When taking into account the sharing and contention of various processes over the multi-core or distributed systems, the timing correctness becomes more critical. Furthermore, many design metrics are directly related to timing. For example, control performance is increased with shorter sampling period, but fault-tolerance and security may then be limited without enough timing slack to adopt these techniques.

- *Security.* The heterogeneous architecture provides a variety of *cyber and physical* interfaces for attackers to utilize. The in-vehicle bus system (CAN, FlexRay, LIN, etc.), OBD-II port, Bluetooth, multi-media system, and key-less entry system have been discussed or shown in the literature to be vulnerable [153, 30, 23, 64]. Although traditional cybersecurity approaches [143] (like encryption and authentication) and control-theoretic approaches [118] can be utilized to protect CPS, they introduce overhead on computation and communication. These overheads may affect control performance, schedulability, and other timing-related metrics. It is crucial to quantitatively model, analyze and validate the system in a cross-layer framework.

- *Fault-Tolerance.* Among these objectives and metrics, fault tolerance is one of our focuses as soft errors have become a major concern in CPS. Due to the continuous scaling of technology, high energy cosmic particles and radiation from the application environment [22, 152], the number of soft errors is rapidly increasing. While for CPS like automotive systems, the safety is closely related to the number of soft errors that can be detected and recovered. Efforts are taken across multiple levels (e.g., manufacturing, circuit, and architecture level [152]). However, these approaches are not able to entirely eliminate errors and may not be suitable for the commercially off-the-shelf processors within embedded systems.

System Integration

Due to the complexity of software and hardware with various design metrics and objectives, the system integration becomes more challenging. When the software modules are mapped to the heterogeneous platform, the designer needs to decide the way to allocate the software modules to different computation units, the scheduling on each component, and the packing and transmission of signals, considering various design metrics and objectives. For example, with the contention from other software tasks on the same core, whether it is able to finish the current task within its deadline. Another example is that with asynchronous communication in a distributed system, whether the end-to-end latencies from sensors to actuators are within the safe range. At the system level, whether the system satisfies requirements on timing, control performance, security, and fault-tolerance needs to be studied in a cross-layer framework.

1.1.2 Connected Vehicle Design Challenges

Connected environments and applications make the problems even more challenging. The purpose of connected applications is to overcome the limitations of single vehicle and increase safety and transportation efficiency. For instance, a left-turning vehicle may fail to detect an obstacle coming from the left due to the blind spots of its sensors, the precision limitations, or the shadings in the environment. With information exchanged among vehicles, they can maintain a shorter safe distance and cooperatively cross the intersection, and thus increasing the transportation efficiency. As a result, vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications (generalized as V2X communications) are proposed for connected applications. Among the protocols of V2X communications, Dedicated Short Range Communication (DSRC) is under the development of the United States [61, 80]. In this standard, the PHY and MAC layers are defined by IEEE 802.11p and IEEE 1609.4, which work as the basis for both safety and non-safety applications. The upper layers are specifically defined for safety and non-safety applications. For non-safety applications, the typical TCP/IP protocol stack is used for transportation and network layers. For safety applications, the transportation and network layers are replaced with WSMP (Wave Short Message Protocol) developed to avoid excessive overhead. There is a message sublayer which defined the message sets for the safety application layer. Among the messages, basic safety messages (BSMs) are exchanged among vehicles and infrastructures in a routine about the vehicle state information (such as speed, acceleration and location). Based on the received information, the safety application can estimate the position and movement of the surrounding vehicles and take early actions to avoid potential collisions or cooperatively improve transportation efficiency. However, the design of connected applications for CPS still faces many challenges besides the increasing complexity as shown below.

Dynamic and Uncertain Environment

The openness of the connected environment leads to the adoption of wireless communication which is less predictive and robust.

- *Timing.* The timing behaviors of connected applications are extremely critical to guarantee safety. However, the timing behavior of V2X communication is less predictive than that of in-vehicle networks (like CAN or FlexRay) due to various factors in the physical environment. In the literature, the transmission delay of DSRC in terms of factors like vehicle density and packet sending rate is studied in [155, 76, 100]. It is shown in [155] that the mean delay of safety message transmission can be as high as 50 ms with vehicle density at 0.1 vehicle/m and a typical packet arriving rate of 10 pkts/s, and the max delay can be as high as 325.57 ms. Therefore, it is more difficult to model and analyze the performance of connected applications and guarantee the correctness of timing behaviors.
- *Robustness.* Due to the nature of wireless communication and the application environment of moving vehicles, the connections among vehicles and infrastructures are less robust. In [154, 21, 46], it is shown that significant packet delays and losses could happen in the vehicular network under dense traffic. Furthermore, malicious jamming or flooding attacks [20, 159] can create more severe packet delays and losses. A connected application should be robust enough to tolerate faults and deal with changing environments.

Security Challenges

The open environment of V2X applications further broadens the potential attack surfaces. The security issues of V2V and V2I communications have been studied at multiple levels, from the underlying DSRC communication protocol to application-level attacks and defenses.

DSRC provides several security measures. In the physical layer, safety messages are designated to be transmitted through frequency channel 172. For the WSMP layer, the format and means to secure message exchange are defined in IEEE 1609.2 protocol, including the Elliptic Curve Digital Signature Algorithm (ECDSA) asymmetric cryptographic algorithm for authentication and the combination of AES-CCM and Elliptic Curve Integrated Encryption Scheme for encryption [80]. However, these techniques are not enough to protect the connected applications against attackers maliciously increasing the delays and losses of packets by jamming or flooding, as well as insider attacks.

At the application layer, various attacks are studied in the literature for connected applications [126, 14, 45, 77, 125]. In summary, the commonly seen attacks can be categorized as follows.

- *Message falsification.* A vehicle sends false safety critical information to mislead victim vehicles to wrong decisions.
- *Impersonation.* A vehicle fakes its identity for malicious purposes or escaping liability.
- *Message tempering.* A vehicle intercepts and modifies packets passing through it.
- *Denial of service attack.* An attacker sends messages to jam the communication channels to prevent regular services.
- *Sybil attacks.* A vehicle pretends to have multiple fake identities to gain large influence.
- *Privacy issues.* An attacker may conduct malicious acts by access victims' location information or electronic ID.

Consensus Challenges

Connected transportation systems are naturally distributed systems. As the communication in distributed systems is not always reliable, Michael Fischer, Nancy Lynch, and Michael Paterson proved that fault-tolerant agreement is impossible in asynchronous distributed systems in 1985 [50]. This leads to the trade-off between safety (the agreement is correctly reached) and liveness (the termination will eventually happen)[1]. However, agreement is critical for many connected applications to avoid collisions. Quorum systems are usually used in asynchronous systems to address the trade-off. In Quorum systems, the decision is made based on the majority votes from the participating nodes. In [29], Soma Chaudhuri introduced the concept of set agreement, where k-set agreement means there can be up to k different output values in the system. In this way, 1-set agreement denotes total consensus and n-set agreement denotes no consensus at all for n nodes. The authors from UC Berkeley introduced PBS (Probabilistically Bounded Staleness) to quantitatively measure a probability called PBS$< k, t >$: the probability of the read is within the k recent versions after t seconds [19]. The Paxos Algorithms [88, 89] proposed by Leslie Lamport can be used to deal with fail-stop failures and message losses. This algorithm can guarantee safety and eventual liveness in asynchronous distributed systems. However, this algorithm is not easy to implement in practical connected applications. It is challenging to propose simplified versions of these techniques specifically designed for connected applications to guarantee safety, eventual liveness, and no deadlocks. This demands the verification and validation of the protocols for connected applications. With the distributed nature, less pre-

[1]The safety and liveness properties are used in the distributed systems communities. In this thesis, the safety and liveness properties are specifically defined for intelligent intersection management in Chapter 3.

dictive communication, and various attack surfaces, the verification and validation become extremely challenging.

In summary, the modeling, exploration, and validation of connected automotive systems should be considered *across system layers* including applications, software implementations, architecture platform, and the communication among them. In this thesis, we present CONVINCE, a cross-layer modeling, exploration, and validation framework for connected vehicles. CONVINCE is a holistic framework containing mathematical models, synthesis and validation algorithms, and simulation of both inter-vehicle and intra-vehicle behaviors. Various metrics are considered across multiple layers. We will introduce the details of the framework in Chapter 2.

1.2 Related Work

In this section, we introduce the related works from in-vehicle software and hardware design to vehicular applications. For in-vehicle software to hardware mapping, we focus on two metrics: security and fault-tolerance. For vehicular applications, we mainly focus on cooperative adaptive cruise control and intelligent intersection management.

1.2.1 In-Vehicle Design

System Integration

Several automated design space exploration techniques have been proposed to facilitate the integration process in the literature. In [132], the authors propose a methodology that adopts constraint-based formalization for automotive software and hardware design, considering the safety requirements of the ISO26262 standard. In [111], the authors discuss the advantages of virtual prototypes to virtually integrate automotive software and hardware through design space exploration and system verification. In [44], the authors further combine firmware-related functionalities (like diagnostic tests) into a holistic design space exploration framework for the design of automotive electronics. In [156], the authors propose a model-based formal integration framework for automotive software considering interoperability.

In-Vehicle Security

In [85], the authors successfully compromise a real vehicle and demonstrate the vulnerability of vehicle electronic systems. They take advantage of the internal CAN buses and use packet sniffing, targeted probing, fuzzing and reverse engineering to complete the attacks. The security issues of in-vehicle bus systems have been observed earlier [153]. In particular, the security issues of CAN have been studied in a number of works [153, 85, 93, 145] due to its prevalence in automotive systems. First, the broadcasting nature of CAN makes it susceptible to eavesdropping. Second, the static priority used for scheduling may lead to Denial-of-Service (DoS) attacks. Third, the lack of authentication field in the frame requires more complex protocol design. For buses like FlexRay and MOST which allocates time slots for different messages, the threats of flooding and DoS can be reduced. However, the difficulty of timing analysis increases with security mechanisms added [130, 153, 109]. An attacker can also utilize the vulnerabilities of Low-cost buses (like LIN which is typically used for non-critical modules such as power windows and rain sensors) to gain access to safety-critical systems [153]. Another way to access the internal bus system is to exploit the wireless communication interfaces such as Bluetooth and keyless entry systems [23]. If the On-Board Diagnostics (OBD) port is accessed by the attacker, the packets transmitted through the internal bus system are completely accessed [30]. Once the attackers have gained access to in-vehicle architecture platform, they can then exploit the vulnerabilities of aforementioned bus systems to deploy sensor spoofing attacks [64], replay attacks, masquerade attacks, DoS attacks, etc.

There are a number of studies in the automotive domain to enhance automotive security. The authors in [151] introduce aspect-oriented modeling to model attacks as aspects and evaluate the system under attack based on the model-based design methodology. In [106], the authors introduce security analysis for automotive architectures using probabilistic model checking. In [140], the authors present a security-aware network controller to enhance the security of the gateway. In [95, 97], the authors model the impact of message authentication techniques on real-time constraints in automotive systems. However, these works do not consider the impact of security enhancing mechanisms on sampling periods and control performance, and do not model their relation with system security (rather it is assumed that authentication requirements are directly given at the message level, which may not be practical in many design processes).

In the cyber-physical system community, various control-oriented approaches are proposed against attacks on cyber-physical systems [118, 54, 116, 139]. In [118], the authors design attack detection and identification monitors from a control-theoretic perspective. In [54], an optimal control approach is proposed to address jamming in the communication channel between the controller and the plant. In [116], a recursive networked predictive control method is proposed to deal with denial of service attacks. In [139], a minimax control approach is presented to address network packet scheduling attacks. However, resource and real-time constraints are not considered in these approaches, and there is no guarantee of schedulability and control performance.

In-Vehicle Fault-Tolerance

Errors may manifest as application crashes, control flow violations (illegal branches) or silent data corruptions. In [51], the authors categorize the online error detection techniques into embedded error detection (EED) and explicit output comparison (EOC). EED refers to the broad collection of error detection techniques that detect and recover part of the errors with built-in techniques. EED-based techniques do not rely on redundant execution but come with computation overheads (e.g., the state-of-the-art control flow checking techniques take 30% of computation overhead but with about 70% recovery rate). In practice, the performance overhead of EED may vary greatly depending on the EED implementation and application [51]. For example, control-intensive programs naturally incur a higher performance penalty when adopting CFC compared to computation intensive programs. Examples of EED techniques include, but are not limited to watchdog timers [105], control flow checking (CFC) [112], and instruction signature checking. EOC refers to the techniques relying on explicit redundancy to detect and recovery errors. For instance, executing the same task multiple times with the same inputs and comparing their outputs. Examples of EOC include the classic triple modular redundancy (TMR) architecture and a scaled down version that executes the same task twice to detect the error and re-execute the task in case of output mismatch [51].

In the literature, there have been a number of studies on fault tolerance for real-time systems. In [72], Izosimov et al. use process re-execution and replication to recover from transient faults with built-in EED-type detection techniques. In [123], they further utilize checkpointing which allows re-execution from the checkpoint instead of the entire process. In [71], they trade off between hardware hardening and software re-execution. Huang et al. in [66] explore spatial and temporal redundancy in a time-triggered architecture. Pinello et al. in [121] explore replication to tolerant errors in controlled plant and the execution platform. Burns et al. in [35, 92] explore priority assignment and conduct schedulability analysis for fault-tolerant hard real-time systems, with focus on EED-type techniques. Kim et al. in [83, 82] categorize tasks to hard recovery, soft recovery, and best-effort recovery types based on their fault tolerance requirements, and apply various replication strategies. There are also several studies on fault burst model where multiple transient faults may occur during a time interval [102, 138, 59]. In [115], the authors present a model to analyze transient errors for automotive safety-critical applications. In [84], the authors study and discuss the faults during the startup and operation of a FlexRay network, and propose a bus guardian. In [79], the authors introduce a fault-tolerant control strategy which adjusts control input at runtime based on the occurrence of faults. The authors in [12, 127] address software-based self-tests and built-in self-tests to enhance automotive fault diagnosis.

1.2.2 Connected Vehicle Applications

Cooperative Adaptive Cruise Control

Cooperative Adaptive Cruise Control (CACC) is the technology that utilizes V2V wireless communication to enhance the traditional single-vehicle adaptive cruise control (ACC) by communicating with other vehicles to cooperatively maintain a safe gap which is relatively shorter. In 1997, National Automated Highway Systems Consortium (NAHSC) demonstrated eight vehicles cooperatively forming a platoon to increase transportation capacity on I-15 in San Diego. Since then, many CACC architectures [147, 78, 42] and simulation engines [49, 103, 56, 55, 134] are proposed. In recently work [15], the authors propose the detailed protocol for CACC with Finite State Machines and simulation engines integrated with network simulators which helps the study of timing delays and packet losses in such systems.

Intelligent Intersection Management

In transportation systems, intersection management plays a critical role as intersections are associated with a significant percentage of traffic accidents and essential for transportation efficiency. According to the Fatality Analysis Reporting System (FARS) in the United States, 40% of crashes and 21.5% of fatal traffic accidents are related to intersections [31, 107]. Traditional intersection management is difficult to adapt to real-time traffic with pre-defined traffic signals or stop signs. Although “smart” traffic lights are proposed to dynamically adjust traffic signals by estimating traffic conditions with information provided by surrounding sensors [141, 67, 104], this strategy also faces bottlenecks as it is difficult to adjust controlling period with different traffic patterns.

With the rapid advancement of autonomous driving and vehicular communication technologies, *intelligent intersection management* techniques have shown great promise in improving intersection safety and transportation efficiency. In an intelligent intersection, autonomous vehicles with vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication capabilities exchange information of current driving states *with each other or with roadside infrastructures* to cooperatively cross the intersection. Typically, the intelligent intersection systems fall into two categories: centralized and distributed as we discuss below.

- *Centralized.* In centralized intersection management, the traffic lights are replaced with an intersection manager. Vehicles send requests to the intersection manager and the intersection manager grants access according to its scheduling policy [43, 73, 86, 162, 13, 108]. In the literature, the intersection is discretized into grids and the intersection manager assigns vehicles to grids for each time step and avoids potential conflicts. In [43, 73, 63], the scheduling is fine-grained as it allows vehicles with potentially conflicting routes to enter as long as no grid is assigned to more than one vehicle at the same time step. The authors in [43] extend the proposed protocol with both V2I communication and virtual traffic lights for transition period where autonomous vehicles and regular vehicles co-exist. The work in [73] studies fuel consumption and vehicle emission compared with traditional traffic lights. The authors in [86] use control theories to prove system safety and liveness through hybrid architectures without message delays. In [162], the problem is formulated as a linear programming problem where traffic flows are modeled with conflict points as constraints.

- *Distributed.* Distributed intersection management requires vehicles to negotiate the order of their crossings. The authors in [17, 18, 16] have a series of works for distributed intersection management where every vehicle broadcasts enter, cross and exit messages with their current grid. They prove their protocol is deadlock-free through wait-for graph and mathematical reasoning. In [108], their system can be proven to be deadlock-free using Petri Net models. In [13], Timed Petri Nets models are again used to decide the sequence of vehicles entering intersection for traffic smoothness.

However, these studies assume the wireless vehicular communication to be perfect with no explicit consideration of packet delays or losses. In practice, vehicular networks may suffer significant delays and losses with dense traffic [154, 21, 46] and may be even worse under malicious jamming or flooding attacks [20, 159]. Previous works *lack the consideration of communication message delays and losses*, and consequently cannot ensure the proposed protocols to be safe, deadlock-free and efficient in practical conditions.

1.3 Contributions

The main contributions of this work include:

- A cross-layer modeling, exploration, and validation framework CONVINCE considering various design objectives and metrics, among which timing, security, and fault-tolerance are our focuses.

- A delay-tolerant protocol for intelligent intersection management. The protocol guarantees safety and assures that as long as the communication delays are bounded, deadlock-free and liveness can be guaranteed.

- Approaches for codesigning the functional and software layers that quantitatively models the impact of security techniques on control performance and platform schedulability, and explores trade-offs between security level and control performance while guaranteeing real-time constraints for cyber-physical systems.

- Approaches for codesigning the software and hardware layers that formulates the impact on system timing for different error tolerance mechanisms including both EOC and EED based techniques, and optimizes the task-level selections of tolerance mechanisms, for various fault models and task execution models on representative single-core, multi-core, and distributed platforms.

- Approaches for codesigning the software and hardware layers that formulates the impact of applying security mechanisms for both CAN-based and TDMA-based systems with limited resource and stringent timing constraints.

The rest of the thesis is organized as follows. In Chapter 2, we introduce the cross-layer modeling, exploration, and validation framework CONVINCE. In Chapter 3, we introduce the design of functional and software layers. The applications include CACC and intelligent intersection management and the co-design of control, security, and schedulability for cyber-physical systems. In Chapter 4, we study the design of software and hardware layers with fault-tolerance and security as objectives and various real-time requirements as constraints. Chapter 5 concludes the thesis.

Chapter 2

CONVINCE Framework

With the challenges described in the Section 1.1, the modeling, exploration, and validation of connected automotive systems should be considered *across system layers* including applications, software implementations, and architecture platform. The concept of such cross-layer design is illustrated in Figure 2.1. The top layer is the application layer and typical applications include V2X and autonomous driving applications. In the application layer, functional verification and validation are done and constraints on timing, robustness, and security are decomposed to individual vehicles. For example, if the application is Cooperative Adaptive Cruise Control (CACC), where every vehicle in the group communicates with other vehicles to adaptively maintain a safe distance from its preceding vehicle (referred to as *gap* in the rest of the thesis), the performance mainly depends on timing, the error rate of the messages, and the security level of the system [47]. Through verification and validation, constraints are decomposed to individual vehicles. For example, the constraints on end-to-end latency, the constraints on error correction ability and the constraints on security level. Inside each individual vehicle, these constraints guide the task generation and task to platform mapping at the software implementation layer. In this vision of cross-layer design, if the constraints obtained from the high-level application layer cannot be fulfilled, the software implementation layer can provide feedback to the application layer to relax some constraints. The software implementation layer and the hardware architecture layer inside one individual vehicle also communicate with each other through the constraints on timing, communication bandwidth, computation resource, etc. Similarly, if the constraints cannot be fulfilled after hardware exploration, the higher levels can trade-off among design metrics and relax some constraints.

2.1 Overview

To achieve the vision in Figure 2.1, we present CONVINCE, a cross-layer modeling, exploration, and validation framework for connected vehicles. The framework includes mathematical models, synthesis and validation algorithms, and a heterogeneous simulator for addressing inter-vehicle communications and intra-vehicle software and hardware in a holistic environment. The overview of CONVINCE is shown in Figure 2.2, including modeling, mathematical analysis, exploration, verification and validation components. Modeling is conducted across multiple layers, including the high-level V2X application modeling, the software modeling inside one vehicle, and the hardware modeling. These models can be abstracted and used by the *analysis and exploration engine* to optimize the design and verify whether design constraints are met. The models can also be leveraged by the *simulation engine* to validate system designs, identify potential issues and provide design insights. In

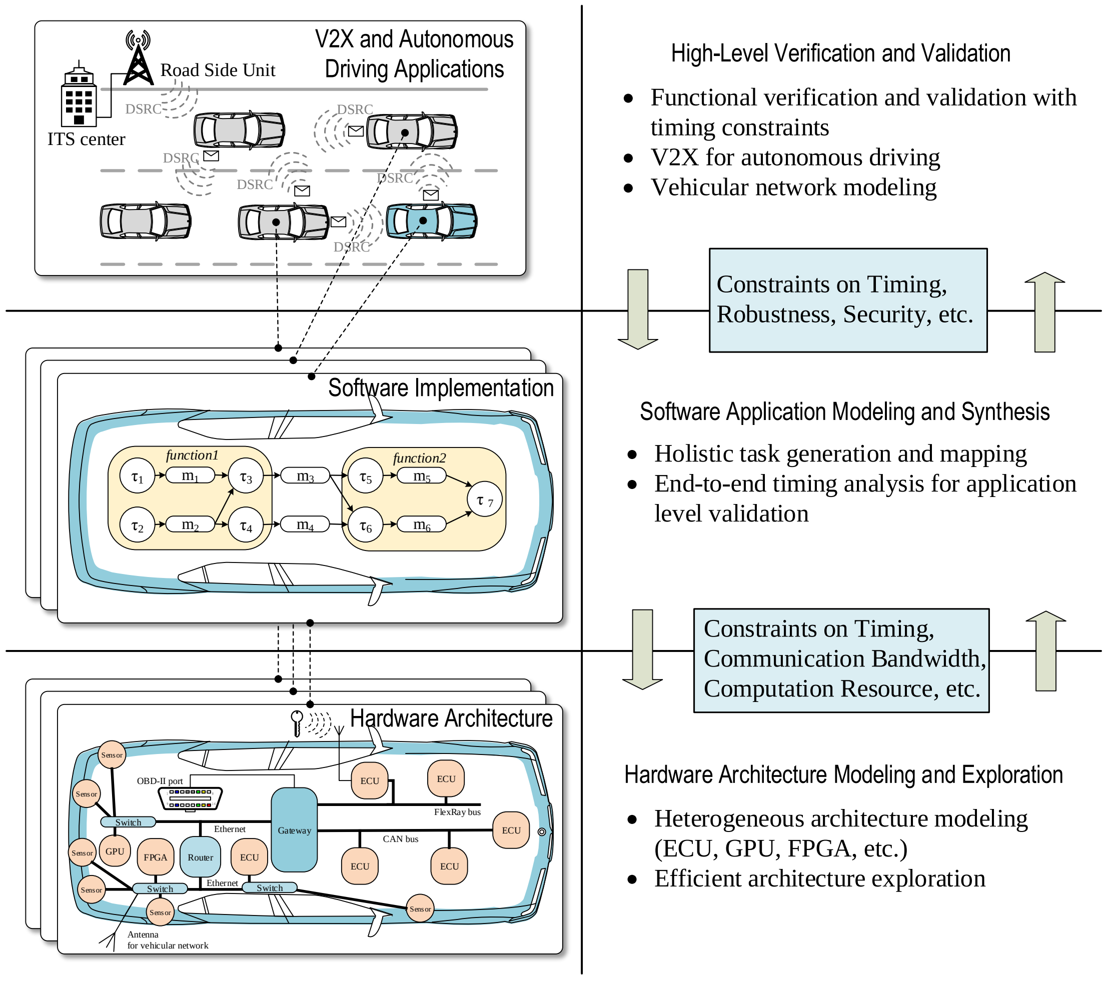

V2X and Autonomous Driving Applications
Road Side Unit
ITS center
DSRC
Software Implementation
function1
function2
τ_1 m_1 τ_3 m_3 τ_5 m_5 τ_7
τ_2 m_2 τ_4 m_4 τ_6 m_6
Hardware Architecture
Sensor
OBD-II port
Switch
Ethernet
GPU
FPGA
Router
ECU
Gateway
FlexRay bus
CAN bus
Antenna for vehicular network
High-Level Verification and Validation
Functional verification and validation with timing constraints
V2X for autonomous driving
Vehicular network modeling
Constraints on Timing, Robustness, Security, etc.
Software Application Modeling and Synthesis
Holistic task generation and mapping
End-to-end timing analysis for application level validation
Constraints on Timing, Communication Bandwidth, Computation Resource, etc.
Hardware Architecture Modeling and Exploration
Heterogeneous architecture modeling (ECU, GPU, FPGA, etc.)
Efficient architecture exploration

Figure 2.1: Cross-layer design for connected vehicles.

the rest of the section, timing property will be used as an example to demonstrate the modeling, analysis and exploration of the CONVINCE framework.

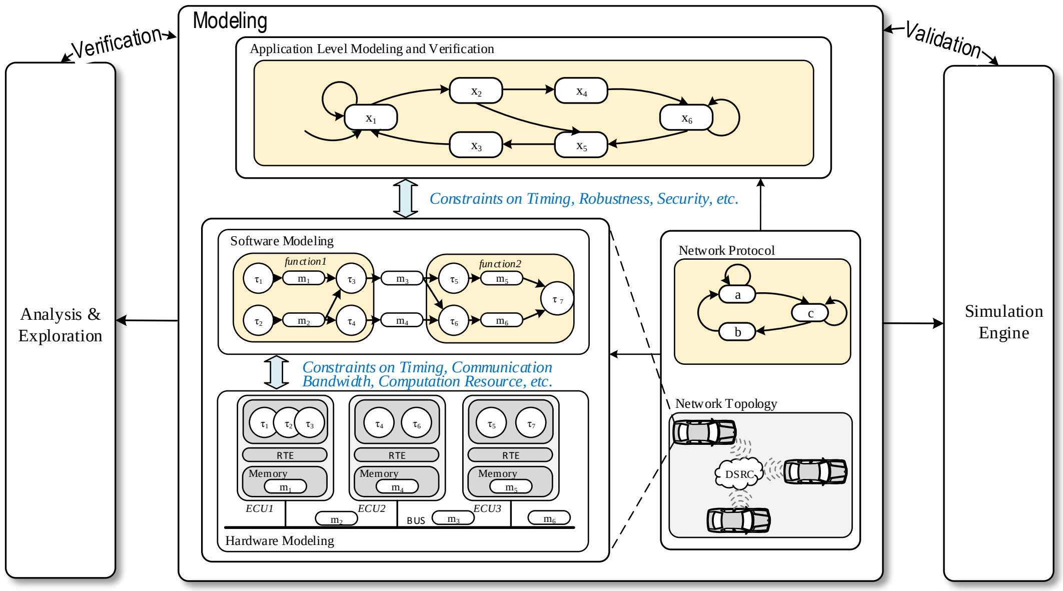


Figure 2.2: CONVINCE: cross-layer modeling, exploration and validation framework for connected vehicles.

2.1.1 Application Level Verification and Validation

The application protocol can be abstracted by verification models. For example, in our delay-tolerant intelligent intersection management, we use timed-automata to model the protocol and use the tool UPPAAL [10] to verify properties like deadlock-free, liveness and safety. The verification decides some timing-related parameters, such as resending period, which serve as the timing constraints for in-vehicle design.

2.1.2 Software Modeling

The software model can be captured by a synchronous reactive task graph. Synchronous reactive models are prevalent practice for modeling control-centric cyber-physical systems in automotive and avionics domain, and used in popular tools such as Simulink [6]. Synchronous reactive system contains a fixed set of synchronized communicating processes, as shown in the software model in Fig 2.2.

For timing analysis, many timing-related parameters need to be abstracted from the synchronous reactive model. For tasks, the most important parameters are the worst-case execution time C_{τ_i} for task τ_i on certain ECU, and the activation period T_{τ_i}. For messages, the most import parameters are message length l_{m_i} for message m_i, the message period T_{m_i}, and the source and destination tasks of the message m_i.

2.1.3 Hardware Modeling

The hardware model can be captured through architecture description languages, for example, architecture modeling and description language (AADL). AADL language captures the system through components, and each component is characterized by its identity, interfaces, properties and subcomponents. AADL also provides extensions in two ways, the user-defined properties and the language annexes [48].

For timing analysis, some user-defined properties need to be added. For example, the computation speed of the CPU (may be captured by frequency), the number of cores for the CPU, and the scheduling policy on it. Similarly, the bus speed, the bus protocol and the scheduling policy for the bus also need to be added as user-defined properties.

2.1.4 Analysis Models

The analysis models refer to the mathematical models used to quantitatively study various metrics of automotive system. Here we use timing models to illustrate how timing properties are analyzed through this framework.

Computation Model (Task Model)

The software layer is captured by a set of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The tasks can be mapped to multiple computation units as shown in Figure 2.2. The timing property of each task τ_i is captured by a worst-case execution time C_{τ_i} (for a specific platform) and an activation period T_{τ_i}. Every task is required to finish its execution before its deadline (e.g., sometimes set as its period). Fixed-priority preemptive scheduling is modeled in the framework as every task is assigned a priority offline and lower priority tasks can be preempted by higher priority tasks. The worst-case response time r_{τ_i} (the longest time it may take to complete task τ_i) can be formulated as the following equation [163]:

$$r_{\tau_i} = C_{\tau_i} + \sum_{\tau_k \in hp(\tau_i)} \left\lceil \frac{r_{\tau_i}}{T_{\tau_k}} \right\rceil C_{\tau_k}. \tag{2.1}$$

The first term of the equation denotes the worst-case execution time C_{τ_i} and the second term represents the preemption time from higher priority tasks in set $hp(\tau_i)$ on the same computation unit.

Communication Model

In V2X and autonomous driving applications, messages are exchanged at different levels. At application level, messages are transmitted through wireless channels from one vehicle to others. Inside the vehicle, the messages can be transmitted on bus or exchanged through memory among modules and tasks. In autonomous driving system, the in-vehicle bus system can also be heterogeneous like CAN and Ethernet [69, 70, 128]. The analysis of the communication latency is essential as automotive systems are timing-critical systems and failures to fulfill the timing requirements may lead to catastrophic outcome.

Intra-Vehicle Communication: In our model, the message access delay for memories is modeled as a small constant, and the mathematical models to capture CAN bus and Ethernet are discussed below.

- *CAN Bus*: CAN bus is prevalent in current automotive systems. The protocol is priority based and non-preemptive. The worst-case response time r_{m_i} for message m_i is as follows [163]:

$$r_{m_i} = C_{m_i} + B_{max} + \sum_{m_j \in hp(m_i)} \left\lceil \frac{r_{m_i} - C_{m_i}}{T_{m_j}} \right\rceil C_{m_j}. \tag{2.2}$$

  The timing property of each message m_i is captured by worst-case transmission time C_{m_i} and period T_{m_i}. As CAN protocol is non-preemptive, the message may have to wait for the longest transmission time of any lower priority messages, denoted as B_{max}. The third term denotes the waiting time due to higher priority messages in set $hp(m_i)$.

- *Ethernet*: Ethernet is discussed to be the potential candidate for autonomous driving, including Time-Sensitive Networking (Ethernet AVB) and Time-Triggered Ethernet (TTEthernet) [144]. Time-Sensitive Networking extends traditional full-switched network by adding eight priorities (three bits) for priority scheduling, and Credit-Based

Shaping (CBS) algorithm is used to select transmission schemes for different classes. The Time-Sensitive Networking classifies traffic into Class-A, Class-B, and best-effort class. Class-A has the highest priority and typically with 2 ms latency and Class-B has the second highest priority and typically with 50 ms latency [144]. The best-effort class assigns its traffic with the rest lower priorities. The packets with the same priority are queued in a FIFO in the corresponding class. We adopt the Compositional Performance Analysis (CPA) as shown in [41] to quantitatively analyze the worst-case timing behavior of Time-Sensitive Networking. Time-Triggered Ethernet is also extended from the switched Ethernet by assigning the transmission of messages to time slots following the time division multiple access (TDMA) fashion. As TDMA scheme assigns time slots offline, it makes the message delay deterministic and predictable. However, synchronization protocol is needed to deal with clock jitter. Besides time-triggered communication, Time-Triggered Ethernet also provides rate-constrained messages and best effort messages that are event triggered. The rate-constrained messages are those with less strict timing requirement and best effort messages are for traditional Ethernet applications with less or no timing constraints. We adopt the TDMA analysis in [97] to quantitatively analyze the timing behavior of TDMA-based network.

Inter-Vehicle Communication (DSRC): DSRC is the standard for vehicular communication in the United States. The protocol stack of Wireless Access in Vehicular Environments (WAVE) is developed for DSRC. The WAVE protocol stack supports two kinds of applications at the application layer: safety applications and Internet applications. For Internet applications, the transportation layer and network layer cover the traditional TCP/IP stack. For safety applications, the WAVE Short Message Protocol (WSMP) replaces TCP/IP stack for transportation layer and network layer. All applications share the same data link layer protocol and physical layer protocol.

As safety messages are time-critical, the IEEE 802.11p protocol that covers the data link layer and physical layer has been studied in [155]. The IEEE 802.11p protocol allocates seven 10 MHz wide channels for multi-channel operation, among which one control channel (CCH) is for safety communication only, and six service channels (SCH) for regular communication. To deal with the media access contention, the Enhanced Distributed Channel Access (EDCA) is utilized to classify the messages into four priority categories and set corresponding contention window and arbitration inter-frame spaces for the back-off procedure CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance). In [155], the authors establish two Markov chains for two different priority groups to analyze the delay distribution in the broadcast mode. We adopt the probability density function of message latency in [155] to analyze the latency for V2X communication.

Besides latency, packet loss is another major concern when designing safety-critical systems [47]. Although CSMA/CA has been adopted in the IEEE 802.11p protocol, it can only reduce the collisions instead of eliminating them. Furthermore, its performance can saturate if large amount of requests are generated or jamming is performed. According to the standard, if two broadcasting messages collide, both messages are lost and no retransmission will be scheduled. If vehicle-to-vehicle messages collide, the messages will be retransmitted within the limit of the retransmission times. If the maximum retransmission times have reached, the message will be discarded. Packet loss may also happen when a wireless communication channel is affected by fading and shadowing. Packet loss may significantly affect the performance of V2X safety applications as vehicles need timely information to predict danger and take actions.

End-to-End Latency: In automotive system, deadlines can also be set on functional paths. For example, the path latency from the action of pressing brake to the action of the corresponding actuator should be bounded within a preset value to ensure safety. The worst-case end-to-end latency for path p can be calculated as Equation (3.26). Because of the asynchronous nature of the communication, task and message periods may need to be added. The details of calculating path latency can be found in [34].

$$l_p = \sum_{\tau_i \in p}(r_{\tau_i} + T_{\tau_i}) + \sum_{m_i \in p}(r_{m_i} + T_{m_i}) \tag{2.3}$$

Security Model

The emerging of autonomous driving and vehicular communication provides the attacker with a variety of attacking surfaces, including the On Board Diagnostics-II (OBD-II) port [85], the various sensors and the wireless communication interfaces such as DSRC, Bluetooth and keyless entry system [23]. The authors in [85] successfully compromised a real vehicle by hacking into its engine control system, brake control system, and other electronic components. The security-aware design for CAN-based and TDMA-based intra-vehicle network has been studied in [95, 97]. Besides intra-vehicle security, inter-vehicle communication brings in more concerns for safety applications.

General Optimization Formulation

By applying the quantitative models, the exploration can be done by solving the optimization problem or simply finding a feasible solution to the problem. Besides the timing properties discussed above, we can also set constraints such as reliability level rel must be higher than a preset value REL_0 and security level s must be higher than a preset value S_0.

$$\text{Optimize design objective} \tag{2.4}$$

$$\text{s.t. } r_{\tau_i} \leq T_{\tau_i} \quad \text{(timing)} \tag{2.5}$$

$$r_{m_i} \leq T_{m_i} \quad \text{(timing)} \tag{2.6}$$

$$l_p \leq D_p \quad \text{(timing)} \tag{2.7}$$

$$rel \geq REL_0 \quad \text{(reliability)} \tag{2.8}$$

$$s \geq S_0 \quad \text{(security)} \tag{2.9}$$

$$\text{other constraints} \tag{2.10}$$

By solving such problem, we can obtain design decisions such as task to ECU allocation, task scheduling, message allocation, message scheduling, security techniques assignment and fault tolerance techniques assignment.

2.2 Summary

In this Chapter, we introduce CONVINCE, a cross-layer modeling, exploration and validation framework for cyber-physical systems. In the framework, computation, communication (including both intra-vehicle and inter-vehicle communication), and system metrics such as timing, security, and fault-tolerance are quantitatively modeled for design space exploration, validation and verification. Such analysis sets the foundation for our work on exploring and validating security and fault-tolerance designs of cyber-physical systems.

Chapter 3

Cross-Layer Design of Functional and Software Layers

In this chapter, we first use the design of cooperative adaptive cruise control (CACC) and delay-tolerant protocol to address the timing security at the functional and software layers through the CONVINCE framework. We demonstrate the impact of malicious jamming or flooding attacks on CACC and intersection management. We then conduct functional verification and validation for the delay-tolerant intersection management protocol against timing attacks. The results of the verification contain various timing-related parameters and these parameters propagate to lower layers as constraints, for example, constraints on end-to-end latency. Second, we demonstrate the cross-layer in-vehicle design through the CONVINCE framework. The control performance, security, and schedulability codesign combines control performance and system security level in the control application layer with the schedulability and security constraints from the embedded platform layer.

3.1 Cooperative Adaptive Cruise Control

In this section, we use Cooperative Adaptive Cruise Control (CACC) as a case study to demonstrate the effectiveness of CONVINCE in analyzing the impact of security attacks in vehicular communication and ultimately the application performance.

3.1.1 Timing Attacks

As shown in [154], significant packet delays are possible with dense traffic for DSRC. With the density of 0.1 vehicle/m, the maximum delay for routine messages can be 325.57 ms according to [154]. An attacker can maliciously flooding the channels to increase packet delays and losses. In this thesis, we assume the attacker floods the wireless channels to impair the vehicular communication. The attacker can be a participant of the vehicular network or a malicious attacker from the road side. We first study the relationship between the strength of the flooding attack and the packet loss rate. In this study, we assume there are 50 vehicles distributed on a road of length 300m. The transmission power of the DSRC module is 26dBm. The EDCA related parameters are set as follows (CWmin: minimum contention window size; CWmax: maximum contention window size; AIFSN: arbitration inter-frame spaces): CWmin = 15, CWmax = 1023, and AIFSN = 3. The flooding message is of length 500 bytes.

We assume some of the vehicles within the 50-vehicle group are malicious attackers. We classify the simulations into three scenarios: 1 attacker, 10 attackers and 20 attackers. In each scenario, every attacker applies the flooding attack with the same strength that varies from 100 Hz (i.e., sending flooding packets at a rate of 100 Hz) to 1 KHz to 10

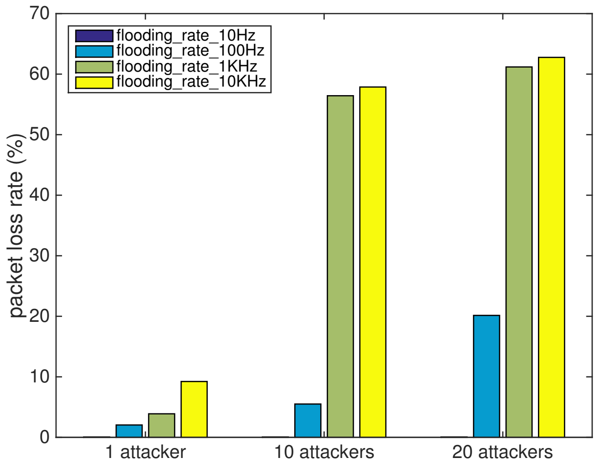


Figure 3.1: Packet loss rate under different strengths of flooding attack. During the simulation, 50 vehicles are uniformly distributed on a road of length 300m and normal packet sending rate is 10 Hz.

KHz. The normal vehicles send packets at a rate of 10Hz (set as 10 Hz in the experiments following [15, 80]).

We use NS-3 to simulate these scenarios, and the results of flooding attack regarding packet loss rate are shown in Figure 3.1. From the figure we can see that for normal traffic, the packet loss rate is around zero. When malicious flooding attack is conducted, the packet loss rate can reach 63% in this case study. We can observe that as the number of attackers increase and/or the attacking strength increases, the packet loss rate also increases (and could be even higher than 63%).

For the CACC applications, the V2X messages are exchanged at a certain rate (set as 10 Hz in the experiments following [15, 80]), and packet loss may lead to outdated information for the following vehicle. If a message is not received within the time window (set as 0.1s in the experiments), the following vehicle has to rely on its own sensors for deciding the safe gap, and CACC is in fact downgraded to ACC (introduced below). We also assume that CACC will be restarted once messages can be successfully received during the time window. Next, we quantitatively study how packet loss ultimately affects the CACC performance.

3.1.2 CACC Application

CACC is the technology that utilizes V2V wireless comsmunication to enhance the traditional single-vehicle adaptive cruise control (ACC) by communicating with other vehicles to cooperatively maintain a safe gap. Platooning, where a leading vehicle leads a group of closely-following vehicles to move like a train, can be formed with CACC enabled vehicles. As platooning can maintain a shorter gap between vehicles and reduce speed variations, it may enhance traffic efficiency and reduce emission. In [15], the authors have designed and implemented a CACC platooning management protocol. In this case study, we will leverage this protocol to study the security issue across multiple layers.

In the protocol designed in [15], every CACC-enabled vehicle receives the acceleration of its preceding vehicle through V2V messages, and obtains the location and speed of the preceding vehicle from sensors such as radar. With these information, each vehicle can maintain a safe gap to its preceding vehicle. As in [15], the equation to calculate the safe gap g_{safe} is

$$g_{safe} = 0.1v_f + \frac{v_f^2}{2D_f^{max}} - \frac{v_p^2}{2D_p^{max}} + 1.0, \tag{3.1}$$

where v_f denotes the speed of the following vehicle and D_f^{max} denotes the maximum deceleration of the following vehicle, and similarly, v_p denotes the speed of the preceding vehicle and D_p^{max} denotes the maximum deceleration of the preceding vehicle. The minimum gap required is 1.0m.

After receiving the location of the preceding vehicle d_p, the current gap between two vehicles can be calculated as $g = d_p - d_f - l_p$, where d_f is the location of the following vehicle, and l_p is the length of the preceding vehicle. Depending on g, the following vehicle may enter different modes and decide its acceleration.

1) Collision Avoidance Mode: If $g < g_{safe}$, the following vehicle will enter the collision avoidance mode. In this mode, the vehicle will decelerate with its maximum deceleration D_f^{max} until the gap becomes safe again. Therefore, in this mode, the new acceleration for the following vehicle is $a_{control} = D_f^{max}$.

2) Gap Control Mode: If $g \geq g_{safe}$, the following vehicle will enter the gap control mode. In this mode, the following vehicle follows the preceding vehicle to maintain a time gap T_{gap}. The desired acceleration a_{des} of the following vehicle can be calculated as following [15]:

$$a_{des} = 0.66a_p + 0.99(v_p - v_f) + 4.08(g - v_f T_{gap} - 2.0), \tag{3.2}$$

where a_p denotes the acceleration of the preceding vehicle and g is the current gap $g = d_p - d_f - l_p$. The actual acceleration to control the vehicle can be calculated as below [15]:

$$a_{control} = \frac{a_{des} - a_f}{\tau} \Delta t + a_f, \tag{3.3}$$

where τ is the controller delay and set as 0.4s. As in [15], $a_{control}$ is bounded by [-3, 3], and Δt is set as the sending rate 0.1s.

ACC Application

As stated in Chapter 1, packet loss and delay can happen in vehicular network. Upon packet loss or delay, the following vehicle cannot obtain the latest acceleration information of the preceding vehicle (a_p), and can only depend on the speed and location information of the preceding vehicle (obtained from its own sensors) to maintain a safe gap, i.e., entering the ACC mode. In the extreme cases, the preceding vehicle may fully brake with the maximum deceleration D_p^{max}. Therefore, we conservatively assume $a_p = -D_p^{max}$ when calculating the desired acceleration as in Equation (3.4). As ACC mode lacks acceleration information, the desired gap between vehicles should be larger. According to [15], the time gap T_{gap} is set to 0.55s for CACC gap control model and set to 1.2s for ACC gap control model. As a result, the safe gap of ACC becomes larger.

$$a_{des} = -0.66D_p^{max} + 0.99(v_p - v_f) + 4.08(g - v_f T_{gap} - 2.0) \tag{3.4}$$

3.1.3 Simulation-Based Analysis of CACC performance

We leverage VENTOS (VEhicular NeTwork Open Simulator) [15] for our simulation, which itself is an integration of several tools with CACC platooning implemented. VENTOS is based on the structure of Veins [9], an simulator that combines the open source traffic simulator SUMO [7] and open source network simulator OMNeT++ [5] with WAVE protocol stack implemented. In addition to OMNeT++, we also leverage NS-3 [4] for packet level simulation of V2X communication networks.

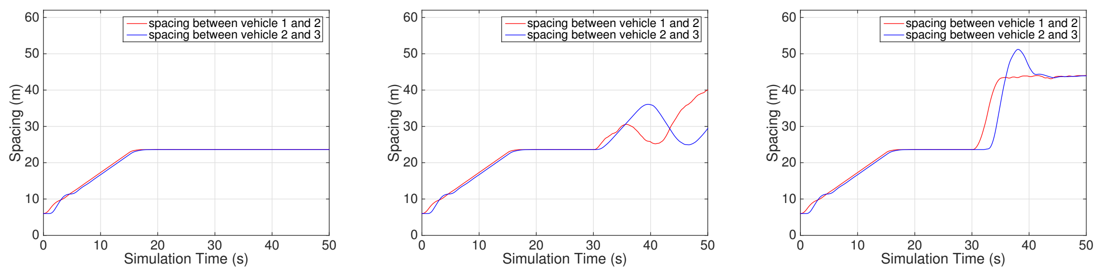
CACC Performance Deterioration under Attack

We assume there are three vehicles joining the CACC application, namely Vehicle 1, Vehicle 2 and Vehicle 3. At time zero, the vehicles are aligned in a line with a gap of 1m between each two consecutive vehicles. Vehicle 1 is set as the leading vehicle. Vehicle 2 and Vehicle 3 will automatically follow Vehicle 1 and maintain safe gaps. The simulation has two phases:

- *Warming up*: From 0s to 15s, Vehicle 1 constantly accelerates with an acceleration of $2m/s^2$, and reaches $30m/s$ at time 15s. Vehicle 2 and Vehicle 3 also accelerate according to the CACC protocol.
- *Keeping speed*: From 15s to 50s, Vehicle 1 stops accelerating and keeps the speed $30m/s$. Vehicle 2 and Vehicle 3 can catch up with Vehicle 1 during this phase.

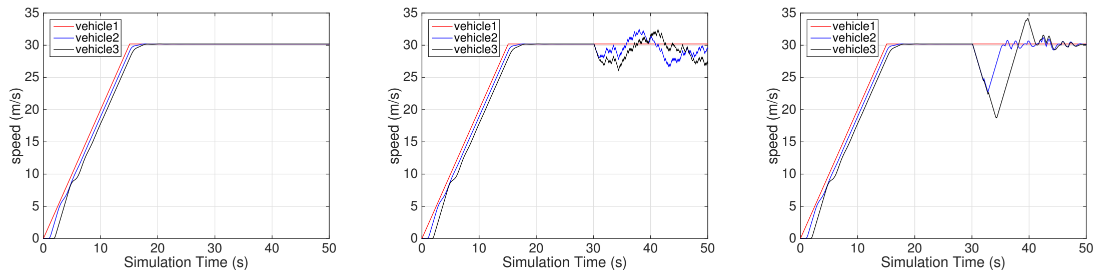
Then, flooding attack is scheduled at time 30s during the keeping speed phase with different strengths. The performance deterioration due to flooding is demonstrated in Figure 3.2 and Figure 3.3.

Figure 3.2 (a) and Figure 3.3 (a) demonstrate the normal behavior without any flooding attack. Figure 3.2 is a spacing-time diagram, where y-axis denotes the spacing between each two consecutive vehicles on the road. The spacing includes the gap between vehicles and the length of one vehicle (set as 5m in our experiments), i.e., it is the distance from the preceding vehicle's front bumper to the following vehicle's front bumper. Figure 3.3 demonstrates the vehicle speeds as the simulation time increases. In Figure 3.2 (a), the spacing gradually increases to around 24m, indicating the CACC protocol is functioning well. In Figure 3.3 (a), we can observe that during the first 15s, Vehicle 2 and Vehicle 3 are



(a) no attack (b) 40% packet lost (30s to 50s) (c) 90% packet lost (30s to 50s)

Figure 3.2: Spacing-time diagram of 3 vehicles in CACC under different strengths of flooding attacks.



(a) no attack (b) 40% packet lost (30s to 50s) (c) 90% packet lost (30s to 50s)

Figure 3.3: Speed-time diagram of 3 vehicles in CACC under different strengths of flooding attacks.

catching up with Vehicle 1. During 15s to 50s, Vehicle 2 and Vehicle 3 also reach the cruise speed of Vehicle 1 and maintain the spacing around 24m and a speed at $30m/s$.

Figure 3.2 (b) and Figure 3.3 (b) demonstrate the CACC performance with flooding attack that causes 40% packet loss. Since flooding attack starts from time 30s, the curves are the same as the normal behavior case from 0s to 30s. We can observe that after flooding attack, the vehicle spacing in Figure 3.2 (b) oscillates around 30m. From Figure 3.3 (b) we can observe that after the attacking at time 30s, Vehicle 2 and Vehicle 3 can not follow Vehicle 1 smoothly. Instead, they have to speed up or slow down constantly. This is because some packets are lost, and thus Vehicle 2 and Vehicle 3 have to switch between the CACC safe gap and ACC safe gap. They can correct their acceleration when latest packets arrive, however the driving efficiency of the individual vehicles and the entire system has already been affected.

Figure 3.2 (c) and Figure 3.3 (c) demonstrate the CACC performance with flooding attack that causes 90% packet loss. Similar to the 40% packet loss case, the curves are the same as the normal behavior case for the first 30s. When flooding starts at 30s, the vehicle spacing significantly increases to around 44m, as most of the packets are lost. Vehicle 2 and Vehicle 3 can not follow Vehicle 1 smoothly. In this case, the vehicles are in ACC mode most of the time and the driving efficiency has been severely reduced.

3.2 Delay-Tolerant Intelligent Intersection Management

In this Section, we present a delay-tolerant centralized intersection management protocol, which takes into account the possible communication delays and losses (as shown in Section 3.1.1) between vehicles and the central intersection manager. We refine CONVINCE to a modeling, simulation, and verification framework for analyzing the safety, liveness and performance of the specific protocol, as shown in Fig. 3.4. We also model and implement our protocol in the SUMO traffic simulation suite [7], with the extension of modeling communication delays. We verify the safety and liveness properties of our protocol by building more abstract timed automata models and leveraging the UPPAAL environment[1] [10].

[1]UPPAAL is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of timed automata.

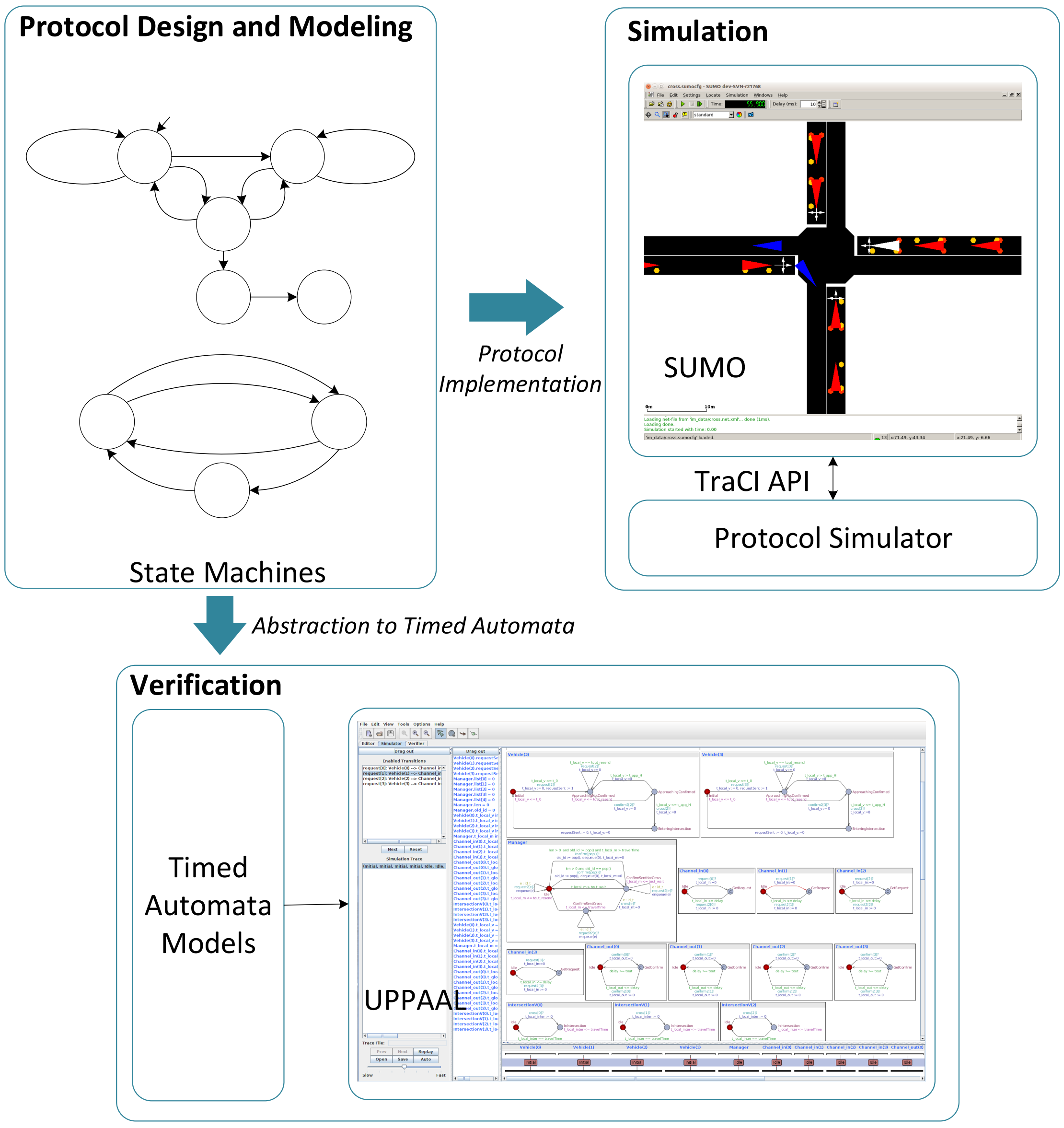


Figure 3.4: The refined modeling, simulation and verification framework for the proposed delay-tolerance intersection management protocol.

3.2.1 Basic System Model

The basic intersection management system model is illustrated in Fig. 3.5. In this system, a central *Intersection Manager* communicates with every vehicle via V2I communication channels to schedule the traffic crossing the intersection. A basic version of the protocol is as follows (a more formal and detailed description with delay consideration is presented in Section 3.2.3).

- Vehicle: 1) sends a *Request* message to the intersection manager, 2) enters the intersection only after it receives a *Confirm* message from the manager, otherwise stops before the intersection, and 3) resends a *Request* message when *Confirm* is not received within a pre-defined timeout bound.
- Intersection Manager: 1) receives *Request* messages from vehicles, and 2) schedules vehicles to enter the intersection based on a scheduling policy, e.g., first come, first served (FCFS).

As stated before, this work explicitly considers communication delays and losses between Intersection Manager and vehicles, as shown in Fig. 3.5. The goal of this work is to design a delay-tolerant protocol that can improve intersection performance/efficiency (measured by average traveling time for vehicles to cross the intersection) and satisfy the following properties:

- *Safety*: vehicles with conflicting routes (i.e., routes that may cross each other within the intersection) may never enter the intersection at the same time.[2]

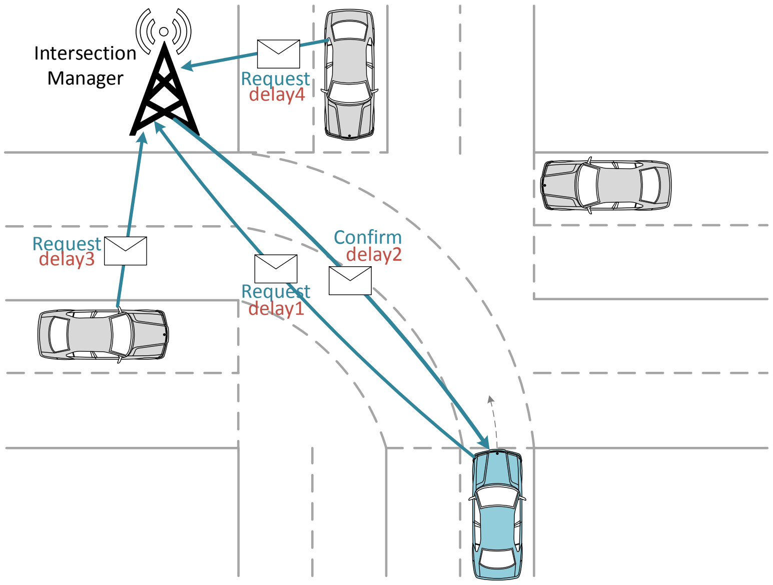


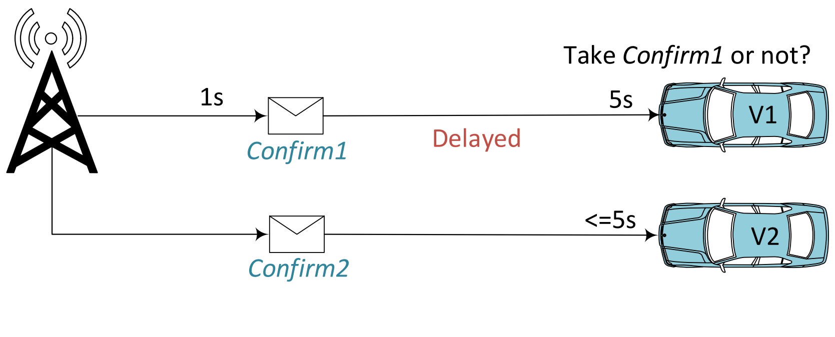
Figure 3.5: The intelligent intersection management system.

- *Liveness:* every vehicle that sends request will eventually cross the intersection, as long as the communication delays are bounded by a timeout bound.

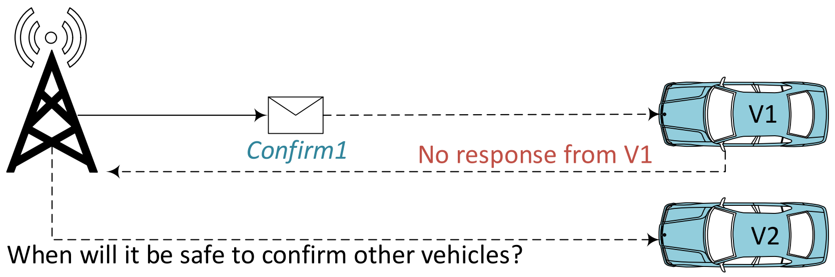
To guarantee the two properties and provide high performance, we assume that the Intersection Manager is capable of detecting whether the vehicles have entered or left the intersection, which can be provided through sensors such as cameras, traffic loop detectors, etc. We assume all vehicles are autonomous and can detect whether there is any vehicle between its current location and the intersection.

In the rest of the section, we first discuss the potential issues of the basic protocol, and then introduce the design of our delay-tolerant intersection management protocol.

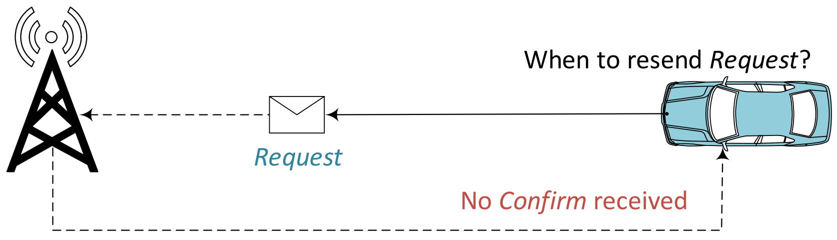
[2]It should be noted that the vehicles are assumed to have autonomous driving capabilities and may detect or even avoid incoming collisions in many cases. Nevertheless, conflicting routes could still lead to unsafe situations given the limitations of autonomous driving, and are likely to cause deadlocks even without accidents.



(a) Message living period



(b) When to safely schedule another vehicle



(c) When to resend request

Figure 3.6: Issues of the basic protocol under communication delays.

3.2.2 Problems with Communication Delay

When taking into account of communication delays in practical systems, the basic request-confirm protocol faces many issues that may lead to system deadlocks or unsafe situations. We use the following three example scenarios to demonstrate the issues the basic protocol may encounter.

The first problematic situation is shown in Fig. 3.6 (a). In this example, the intersection manager first sends *Confirm1* to vehicle V1 at time 1s, however the message is delayed by 4s. The intersection manager has then confirmed another vehicle V2 during the delay of *Confirm1* message. In this case, it might be dangerous for V1 to take corresponding actions.

Some may argue that the intersection manager should not schedule another vehicle until it gets a response from the previously confirmed vehicle. However, this may lead to the second problematic situation shown in Fig. 3.6 (b). In this example, the intersection manager sends *Confirm1* to vehicle V1, but V1 does not enter the intersection because of a long delay or possible loss of *Confirm1*. In this case, the intersection manager should not wait forever for V1 to respond, however the question is how long the intersection manager should wait before it can safely confirm another vehicle V2.

The third issue is shown in Fig. 3.6 (c) where a vehicle sends *Request* to the intersection manager but gets no response from the intersection manager. The question is when the vehicle should resend the *Request* message to avoid possible deadlock.

3.2.3 Delay-tolerant Intersection Management Protocol

Timeouts

To address the issues caused by communication delays, we introduce three types of timeouts in our protocol: 1) timeout for each message transmission, denoted as T^m_{out}; 2) timeout for a vehicle to wait before resending the request, denoted as T^r_{out}; and 3) timeout for the Intersection Manager to wait for a vehicle to enter the intersection, denoted as T^w_{out}. More specifically, T^m_{out} represents the living period of that message, i.e., the message becomes invalid and should not be used after the timeout. T^r_{out} represents how long a vehicle should wait, when no *Confirm* is received, before resending the request. T^w_{out} represents how long the Intersection Manager should wait for the currently scheduled vehicle to enter the intersection, before it schedules another vehicle.

Messages

Three types of messages are defined in our protocol for communication between the vehicle and the Intersection Manager, as shown below.

- *Request.* A request message is sent by a vehicle to acquire permission for entering the intersection. It contains *requestID*, *sender*, *sending time*, *timeout* (T^m_{out}), and *estimated arriving time* (t_{exp}). In particular, the estimated arriving time is used by the Intersection Manager to schedule the time for each vehicle to enter the intersection. As we assume the vehicles are autonomous, the estimated arriving time can be calculated using the location, speed and acceleration information collected from their sensors. The accuracy of the estimation only affects efficiency in our protocol.

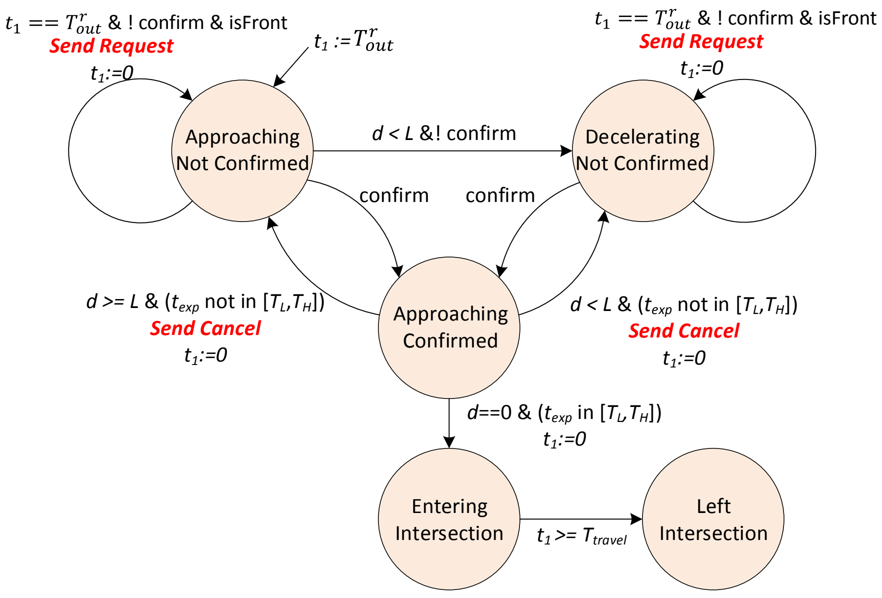
- *Confirm.* A confirm message is sent by the Intersection Manager to give permission to a vehicle for entering the intersection. It contains *confirmID*, *sending time*, *timeout* (T^{m}_{out}), and *arriving time range* ($[T_L, T_H]$). If the vehicle enters the intersection during the arriving time range, it is guaranteed to be safe according to our protocol. A vehicle cannot enter the intersection if no *Confirm* is received. If the vehicle cannot enter the intersection within the time range, it must not enter the intersection, either; instead, the vehicle can send a cancel message as discussed below.

- *Cancel.* A cancel message is sent by a vehicle to notify the Intersection Manager that a previous *Confirm* is "cancelled" by the vehicle and it will not enter the intersection. The *Cancel* message is used for improving the performance and is in fact optional. Once receiving the *Cancel* message, the Intersection Manager can schedule other vehicles immediately and does not need to wait for the vehicle to cross the intersection. Without receiving the *Cancel* message, the Intersection Manager will wait for the timeout T^{w}_{out} before scheduling another vehicle (note that the Intersection Manager knows whether the vehicle enters the intersection through sensors). The fields in a *Cancel* message include *cancelID*, *corresponding confirmID*, *sending time*, and *timeout* T^{m}_{out}.

Based on the above definitions, our protocol is described by state machines in below.

State Machine for Vehicles

The state machine for a vehicle is shown in Fig. 3.7. In the state machine, there are two variables for denoting time. Variable t_1 denotes the local time for each state and variable t_{exp} denotes the expected arriving time in global time. There are five states for the vehicle: *approaching not confirmed, decelerating not confirmed, approaching confirmed, entering intersection* and *left intersection*. The details of each state and the transitions are described below.



$t_1 == T^r_{out}$ & ! confirm & isFront
Send Request
t_1:=0
$t_1 := T^r_{out}$
$t_1 == T^r_{out}$ & ! confirm & isFront
Send Request
t_1:=0
Approaching Not Confirmed
$d < L$ &! confirm
Decelerating Not Confirmed
confirm
confirm
$d >= L$ & (t_{exp} not in [T_L,T_H])
Send Cancel
t_1:=0
Approaching Confirmed
$d < L$ & (t_{exp} not in [T_L,T_H])
Send Cancel
t_1:=0
d==0 & (t_{exp} in [T_L,T_H])
t_1:=0
Entering Intersection
$t_1 >= T_{travel}$
Left Intersection

Figure 3.7: Vehicle state machine.

<u>*Approaching not Confirmed*</u>: This is the starting state for every vehicle approaching an intersection. In this state, once the vehicle becomes the front vehicle (i.e., there is no other vehicle between it and the intersection), it sends a *Request* message and waits for the corresponding *Confirm* message from the intersection manager. Inside the request

message, the field t_{exp} includes estimated arriving time to the intersection based on current vehicle location, speed and acceleration. The vehicle will resend the *Request* if no *Confirm* is received within the timeout bound T^r_{out}. As the vehicle is approaching the intersection, if no *Confirm* is receive, it may need to decelerate and stop before the intersection waiting line. In our case, if the distance to the intersection d (or to the last vehicle waiting at the intersection) is less than a safe value L, the vehicle will enter the state *Decelerating not Confirmed*. If a *Confirm* is received within the timeout bound and before decelerating, the vehicle will directly enter the state of *Approaching Confirmed*.

Decelerating not Confirmed: In this state, the vehicle decelerates and ensures that it can fully stop before the waiting line of the intersection. The vehicle will send a *Request* if it becomes the front vehicle, and if no *Confirm* is received within the timeout bound T^r_{out}, it will resend the *Request*. The field t_{exp} in the request message will be based on the new location, speed and deceleration information. If *Confirm* is received within T^r_{out} at this state, the vehicle will enter the *Approaching Confirmed* state.

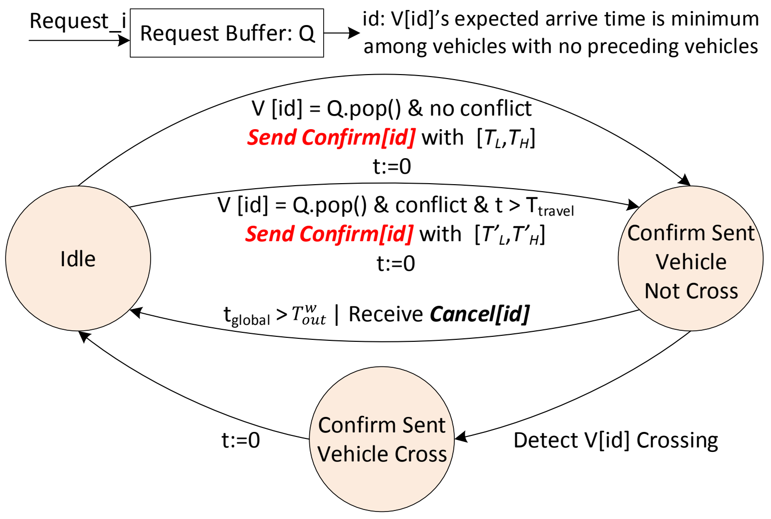
Approaching Confirmed: In this state, the vehicle has received *Confirm* from the Intersection Manager with a time range $[T_L, T_H]$ assigned for it to enter the intersection. The vehicle will continuously check whether it can arrive at the intersection within the assigned time range. If the vehicle finds it cannot enter the intersection in time, it will send a *Cancel* message to notify the Intersection Manager and switch back to the one of the states waiting for the *Confirm* message: If the distance to the intersection is still larger than the safe value ($d \geq L$), the vehicle will switch back to the *Approaching not Confirmed* state; otherwise to the *Decelerating not Confirmed* state. If the vehicle can arrive at the intersection within time range $[T_L, T_H]$, it will enter the intersection and switch to the state *Entering Intersection.* It should be noted that our protocol will still function safely (but less efficient) if there is no *Cancel* message (or it is lost or delayed too long), since the Intersection Manager can sense whether the vehicle has entered the intersection and will schedule another vehicle after timeout T_{out}^{w}.

Entering Intersection: In this state, the vehicle enters the intersection with a preset speed. Once the vehicle has left the intersection, it will enter the *Left Intersection* state. As we assume the Intersection Manager can sense whether the vehicle has entered or left the intersection, no action is needed for a vehicle in this state.

Left Intersection: In this state, the vehicle has left the intersection. No action is needed as explained above.

State Machine for Intersection Manager

The state machine for the Intersection Manager is shown in Fig. 3.8. There are three states: *Idle, confirm sent vehicle not cross* and *confirm sent vehicle cross.* Before discussing the details of each state, we first introduce the routine that handles the messages received from the vehicles, i.e., the *Request* and *Cancel* messages. All the messages received will be put into a buffer, and the messages exceeding timeout will be deleted. The message handling routine is activated during all states. The *Idle* state is the one that the Intersection Manager schedules vehicles. In our current implementation, we adopt the First Come First Served (FCFS) scheduling policy, and this can be easily changed to other policies. In FCFS, the Intersection Manager will first schedule the request from the vehicle that 1) has no other



Request_i
Request Buffer: Q
id: V[id]'s expected arrive time is minimum among vehicles with no preceding vehicles
V [id] = Q.pop() & no conflict
Send Confirm[id] with [T_L,T_H]
t:=0
V [id] = Q.pop() & conflict & t > T_{travel}
Send Confirm[id] with [T'_L,T'_H]
t:=0
Idle
Confirm Sent Vehicle Not Cross
$t_{global} > T^{w}_{out}$ | Receive Cancel[id]
Confirm Sent Vehicle Cross
t:=0
Detect V[id] Crossing

Figure 3.8: Intersection Manager state machine.

vehicle between it and the intersection (this is in fact guaranteed as only the front vehicle can send request in current model), and 2) has an estimated arriving time t_{exp} that is the smallest among all front vehicles.

Idle: In this state, the intersection manager checks whether the buffer storing messages from vehicles is empty. If it is not empty, it will select a *Request* (hence a vehicle) based on the scheduling policy. If the route of the selected vehicle conflicts with the routes of the vehicles currently inside the intersection, the Intersection Manager will wait for the current vehicles inside the intersection to finish crossing before it sends the *Confirm* message to the selected vehicle; otherwise it sends the *Confirm* message immediately. The Intersection Manager will also assign the time range for the selected vehicle to enter. In our current implementation, the Intersection Manager will first compare current time with the expected arriving time t_{exp} from the *Request*. If $currentTime >= t_{exp}$, the upper bound for the time range T_H is set to $T_H = currentTime + T^m_{out}$; otherwise it is set to $T_H = t_{exp} + T^m_{out}$. The lower bound is set as *currentTime*. After sending the *Confirm* message, the Intersection Manager will enter the state of *Confirm Sent Vehicle not Cross.*

Confirm Sent Vehicle not Cross: As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a *Cancel* message is received, the Intersection Manager will enter the *Idle* state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the *Confirm Sent Vehicle Cross* state; otherwise, it will enter the *Idle* state and schedule another vehicle.

Confirm Sent Vehicle Cross: In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the *Idle* state immediately.

Issues Revisit

Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period T^m_{out}, so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it[3]. The intersection manager will not schedule another vehicle until T^w_{out} (T^w_{out} should be set larger than T_H as T_H is the last valid time for the vehicle to enter, and this is also shown in the verification results in Section 3.2.4). It is therefore safe to enter the intersection as long as the vehicle arrives within the scheduled

[3]It is possible that after a vehicle is confirmed, the intersection manager receives another request with earlier estimated arrival time (such request probably got delayed by bad communication condition). To mitigate (but not fully prevent) such scenario, the intersection manager can put constraints such as only confirming a vehicles request if its arrival time is within a bound of current time (which was in fact implemented in our simulator)

period $[T_L, T_H]$. The third problem can be solved with the resending period T^r_{out}. Intuitively, T^r_{out} should be larger than $2 * T^m_{out}$, as this is the living period for the round trip communication between the vehicle and the manager, and both messages will be invalid after this time period. This is also shown in the verification results in Section 3.2.4. In the following section, we will use formal methods to verify the the safety and liveness properties, and study the relationship between the timeouts to avoid deadlock.

3.2.4 Timed Automata for UPPAAL Verification

In order to verify the safety and liveness properties discussed in Section 3.2.1, we abstract timed automata models from the state machines described in Section 3.2.3, and leverage the UPPAAL tool for verification. The key idea is to convert all the variables in the state machines (e.g., distances) to variables directly related to time. We are able to verify a restrictive case where the four-way intersection has a single lane from each direction. We assume the vehicles from the same direction will autonomously use car-following models, and thus will not collide.

Instead of modeling each single vehicle, we use one automata to model the vehicles from the same direction, as shown in Fig. 3.9 (a). There are four automata in total corresponding to the four directions. Each direction has a different "id" from $[0, 1, 2, 3]$. The time variable in the automata is *t_local_v*. Since only the front vehicle can send request to the Intersection Manager in our protocol, we only need to deal with one vehicle from each direction at the same time. Each time the automata goes back to the initial state, it can be considered as a new vehicle coming from the same direction.

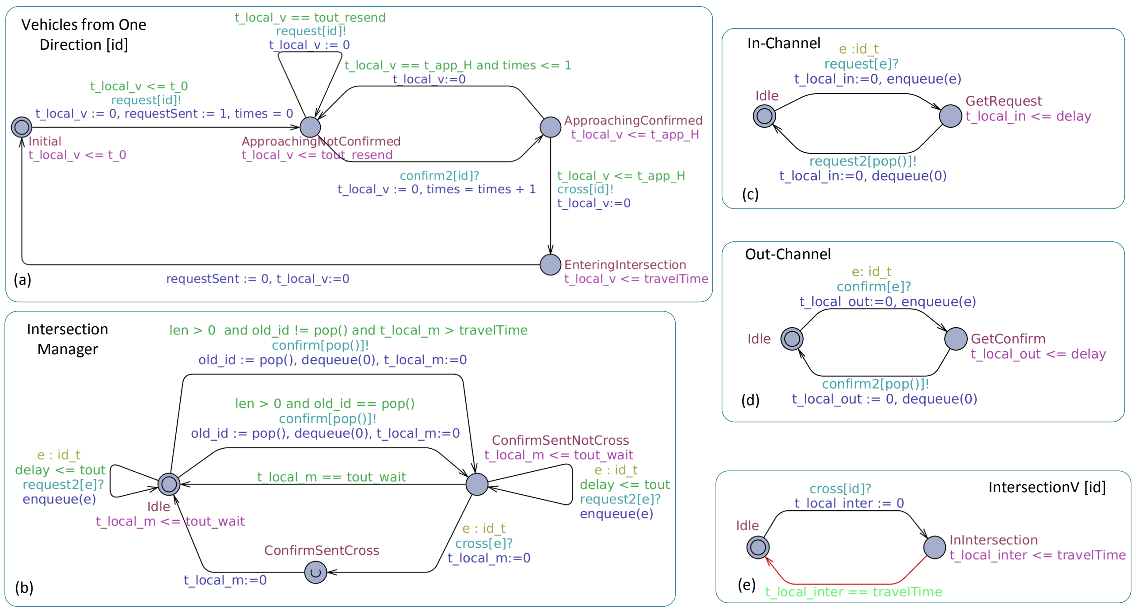


Figure 3.9: The timed automata modeled in UPPAAL.

Each front vehicle from a direction will first enter the initial state and then randomly choose a time within range $[0, t_0]$ to enter the *Approaching not Confirmed* state. This models the *uncertainty* of the time that vehicles coming to the intersection. Note that we combine the state *Approaching not Confirmed* and the state *Decelerating not Confirmed* in the vehicle state machine to one state named *Approaching not Confirmed* in the automata. This is because the only difference of the two states is how soon the vehicle will arrive at the intersection. Such difference can be abstracted through a time variable t_app, which represents the time to arrive at the intersection after receiving the *Confirm* message.

In the *Approaching not Confirmed* state, the vehicle will periodically send request if *Confirm* is not received. Once *Confirm* is received, the state becomes *Approaching Confirmed*. In this confirmed state, if the time is less than the time to approach the intersection *t_app*, the state becomes *Entering Intersection*; otherwise it will go back to *Approaching not Confirmed*. Once the vehicle enters the intersection, the vehicle behind it becomes the front vehicle. The automata state will go back to *Initial*, indicating a new vehicle arriving.

In order to model the message delays, we introduce two automata as In-Channel and Out-Channel, as shown in Fig. 3.9 (c) and (d). For all directions, there is an In-Channel for messages to be transmitted from the front vehicle to the Intersection Manager, and an Out-Channel for messages to be transmitted in the other direction. We remove the *Cancel* message in the verification, which is equivalent to the case where all the *Cancel* messages are lost. The In-Channel automata is associated with the corresponding "id"s of corresponding directions. The automata can sense the trigger of the synchronizer "request[id]" and move to the *GetRequest* state, which represents the sending of the message from the vehicle. The automata will then wait for the trigger of another synchronizer "request2[id]", which represents the receiving of the message at the Intersection Manager. Such transition is bounded by a timeout. The Out-Channel is similarly modeled.

Finally, the automata representing the Intersection Manager is shown in Fig. 3.9 (b). We first implement a queue to store the request from the vehicles, with functions as enqueue() and dequeue(). The queue is first-in-first-out, and the new request will overwrite the old request from the same direction. The enqueue() routine runs on all states. Once the queue becomes non-empty, the Intersection Manager will select a request from the buffer with an "id" number. The following scheduling is similar to the state machine case.

Verification Results

Using the timed automata models from Section 3.2.4, we have successfully proved the following properties in UPPAAL:

- **A[] not deadlock imply** $delay <= T^m_{out}$. The message delay must be smaller than the message timeout T^m_{out} to ensure that the system does not deadlock[4]. We have observed counter examples where delays longer than T^m_{out} caused deadlocks (similarly for the next two properties).

- **A[] not deadlock imply** $T^r_{out} >= 2 * T^m_{out}$. The timeout for resending the request must be at least two times larger than the timeout of the message to ensure the system does not deadlock.

- $T^w_{out} >= T_H$ **and Vehicle(i).requestSent** $\rightarrow$ **Vehicle(i).EnteringIntersection**. When the first two properties are guaranteed by setting the proper timeout bounds, and the time the Intersection Manager should wait for the currently scheduled vehicle to enter the intersection T^w_{out} is greater than the upper bound of the time range assigned to the corresponding vehicle T_H, this liveness property is proved. That is, once the vehicle sends a request, it will eventually cross the intersection.

- **A[] IntersectionV(0).InIntersection + IntersectionV(1).Intersection + IntersectionV(2).InIntersection + IntersectionV(3).InIntersection <= 1**. When the first three properties are guaranteed, this safety property is proved. That is, no vehicles from different directions can enter the intersection at the same time (note that this is a stronger condition than the safety properties discussed in Section 3.2.1).

[4]In UPPAAL, A[] p indicates p is true for all reachable states.

3.2.5 Extensions to Multiple-Lane Intersections

There are some fundamental assumptions in our extensions to multiple-lane intersections:

- There is an intersection manager in the intersection. It receives requests from vehicles, schedules them, and sends confirmations to vehicles.
- All vehicles and the intersection manager are connected (if some vehicles are non-connected, roadside sensors and lane-specific traffic lights are required).
- All vehicles should follow instructions from the intersection manager, regardless of whether a vehicle is controlled autonomously or by a human driver.
- All vehicles have basic safety systems such as pre-collision systems or lane departure alerts as the final safety features. However, the intersection manager should still schedule vehicles to avoid collisions for better safety and efficiency.

Then, we define the system model for single-intersection multi-lane systems as follows:

- An intersection has a set of ways, $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_{|\mathcal{W}|}\}$ and a set of feasible paths, $\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_{|\mathcal{P}|}\}$.
- Each way ω_i has a set of lanes, $\mathcal{L}_i = \{\lambda_{i1}, \lambda_{i2}, \ldots, \lambda_{i|\mathcal{L}_i|}\}$.
- Each feasible path π_k is an ordered pair of lanes $(\lambda_{ij}, \lambda_{i'j'})$.
- Each pair of feasible paths is either non-conflicted or conflicted, which is pre-defined based on the physical design of the intersection.

- Each system of multiple intersections is defined by a graph $(\mathcal{I}, \mathcal{E})$, where $\mathcal{I}$ is the vertex set and $\mathcal{E}$ is the edge set.
- Each vertex in $\mathcal{I}$ is an intersection.
- Each edge in $\mathcal{E}$ is between two intersections.
- Each request of a vehicle includes its current lane, a set of destination lanes, and other information that may include estimated arrival time, earliest arrival time, desired arrival time, and whether the vehicle is the first vehicle in its current lane.
- The manager will process a set of requests, send a set of confirmations, and keep unprocessed requests for the next time.
- Each confirmation is associated with a request. It includes a set of feasible paths and a time window for the corresponding vehicle to enter the intersection.

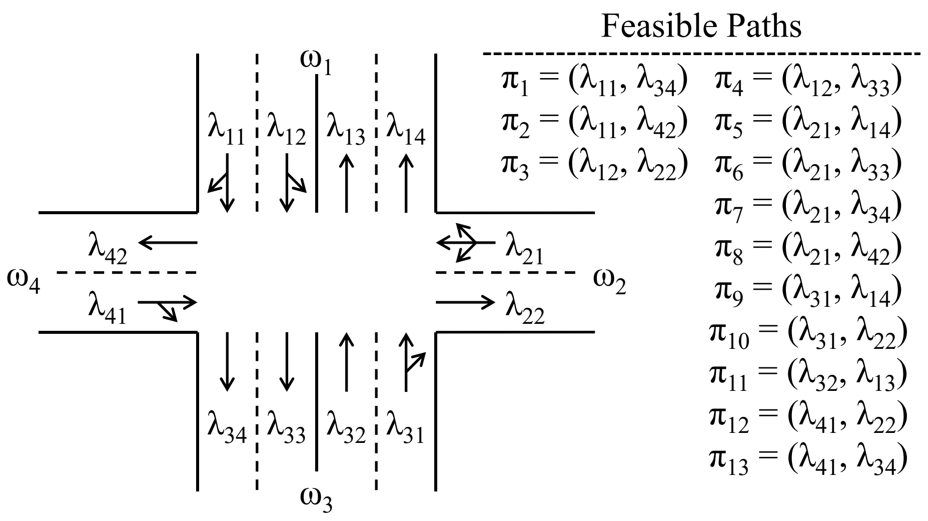

Feasible Paths
$\pi_1 = (\lambda_{11}, \lambda_{34})$
$\pi_2 = (\lambda_{11}, \lambda_{42})$
$\pi_3 = (\lambda_{12}, \lambda_{22})$
$\pi_4 = (\lambda_{12}, \lambda_{33})$
$\pi_5 = (\lambda_{21}, \lambda_{14})$
$\pi_6 = (\lambda_{21}, \lambda_{33})$
$\pi_7 = (\lambda_{21}, \lambda_{34})$
$\pi_8 = (\lambda_{21}, \lambda_{42})$
$\pi_9 = (\lambda_{31}, \lambda_{14})$
$\pi_{10} = (\lambda_{31}, \lambda_{22})$
$\pi_{11} = (\lambda_{32}, \lambda_{13})$
$\pi_{12} = (\lambda_{41}, \lambda_{22})$
$\pi_{13} = (\lambda_{41}, \lambda_{34})$
ω_1
ω_2
ω_3
ω_4
λ_{11} λ_{12} λ_{13} λ_{14}
λ_{21} λ_{22}
λ_{34} λ_{33} λ_{32} λ_{31}
λ_{42} λ_{41}

Figure 3.10: An example showing the intersection model.

An example is shown in Figure 3.10 where π_3 and π_{13} are non-conflicted and π_3 and π_{10} are conflicted. There are some hard constraints:

- The destination lane of each feasible path in a confirmation must be in the set of destination lanes in the corresponding request.
- The time windows in a confirmation must be after the earliest arrival time in the corresponding request.
- Each pair of confirmations must satisfy at least one of the following conditions: (1) the time window of one confirmation does not overlap with the time window of the other confirmation; (2) all of the feasible paths of one confirmation are non-conflicted with all feasible paths of the other confirmation.

Note that the last constraint is to provide either temporal or spatial separation for safety. As we stated before, here we use a less aggressive approach under the consideration of communication delays and losses (unlike previous methods where vehicles with conflicting paths may enter the intersection at the same time and only get separated with fine-grained scheduling that is vulnerable to timing attacks). We also make this assumption based on practical consideration of vehicle passengers' mental acceptance.

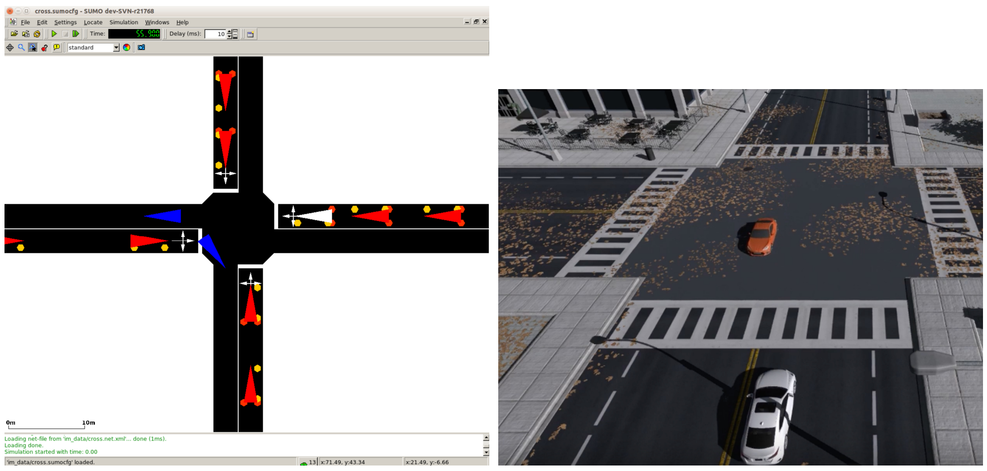
The changes to the original single-lane protocol are shown below.

- Vehicles:
  - A vehicle sends *Request* once it enters the communication range of the intersection manager.

  - Once a vehicle receives a *Confirm* message, it can proceed as long as its estimated arriving time is within the time window in *Confirm*. The *Confirm* does not have to correspond to the latest *Request* the vehicle sent.

- Intersection Manager:
  - Following a controlling period, the manager periodically checks traffic conditions and decides a set of vehicles with non-conflicting routes to send *Confirms*.
  - The manager stores every *Request* in its inbox until a corresponding *Confirm* is sent. A new *Request* from the same vehicle overwrites the old one.
  - Non-front vehicles are only confirmed with the front vehicle in the same lane.
  - The manager confirms multiple vehicles aligned in a queue by sending a single time window ($[T_L, T_H]$) to them. Note that not all vehicles in that lane have to be confirmed, but the front vehicle has to be.

- Timeouts:
  - Instead of the three timeouts in the single-lane protocol, we now only set a resending timeout for vehicles, denoted as t^r_{out}. A vehicle will resend *Request* if *Confirm* is not received after t^r_{out}. We remove the message timeout (i.e., living period). We also remove the timeout for the intersection manager to wait for currently-scheduled vehicle. Instead, T_H in the time window denotes the longest time the intersection manager will wait for.



```
Loading net-file from 'im_data/cross.net.xml'... done (1ms).
Loading done.
Simulation started with time: 0.00
'im_data/cross.sumocfg' loaded.
```

(a) SUMO simulation suite (b) Unity visualization tool

Figure 3.11: The screenshots of simulation tools in our framework.

3.2.6 Simulation-Based Analysis of Intersection Management Performance

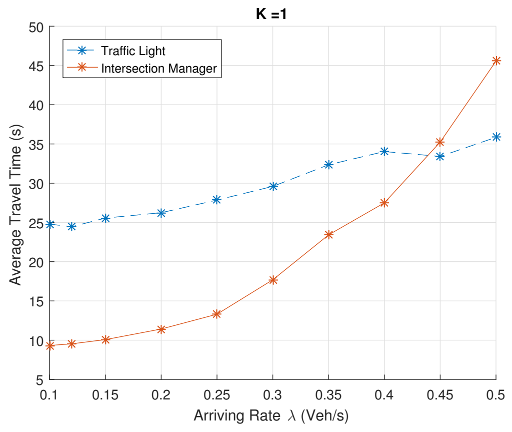
Simulator Implementation

We implement our simulation environment based on the widely-used traffic simulator SUMO [7]. Specifically, we implement the state machines for the vehicles and the Intersection Manager, following the state machines defined in our protocol. We control the movement of the vehicles by leveraging the TraCI API provided by the SUMO simulation suite. Most importantly, we added the explicit modeling of communication delays in SUMO. During simulation, at each time step, we halt the SUMO engine and obtain the location, speed and acceleration information of vehicles for facilitating our protocol simulation. In this experiment we model a four-way single-lane intersection and vehicles are arriving based on Poisson distributions. The screenshot for simulation in SUMO and our visualization tool Unity is shown in Fig. 3.11.

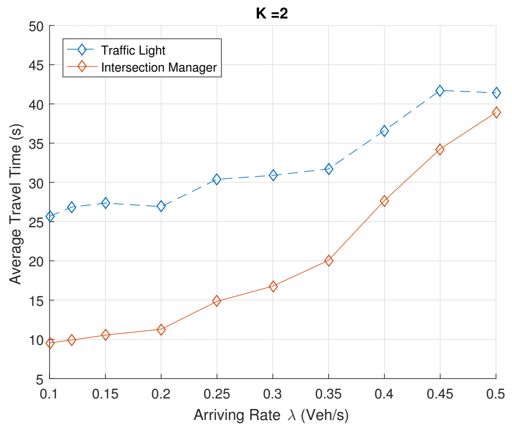
Single Intersection with Single Lane

Delay-Tolerant Protocol v.s. Traffic Lights. We first compare the performance of our protocol with traditional traffic lights. The performance is evaluated as the average traveling time of each vehicle, i.e., the time difference of entering the intersection range and leaving the intersection range. The range is a radius of 50 meters from the intersection center. The arriving rate of the Poisson distribution is within a range of $[0.1, 0.5]$ (the unit is vehicles/second). We also introduce a factor K to denote the ratio of traffic arriving rates from different directions. $K = 1$ represents that the north-south directions has the same traffic arriving rate as the east-west directions. $K = 2$ and $K = 3$ represent that the north-south direction has twice and three times of traffic arriving rate than the east-west direction, respectively. The traditional traffic light is set with a red light phase of 36 seconds, a green light phase of 31 seconds, and a yellow light phase of 5 seconds, which are the default values in SUMO. The timeout values in our protocol are set as $T_{out}^{m} = 4$, $T_{out}^{r} = 8$ and $T_{out}^{w} \geq T_H$ (all units in seconds).

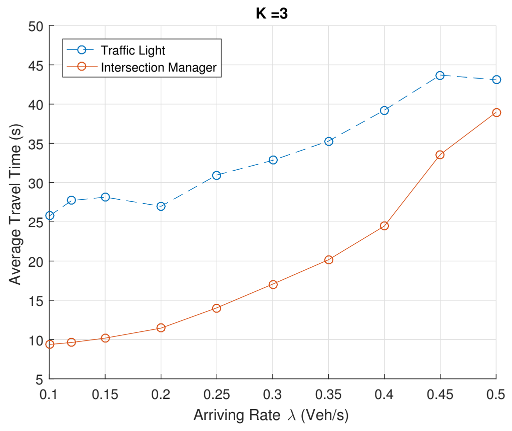
The simulation results are shown in Fig. 3.12, where the x-axis denotes the total traffic arriving rate from all directions, and y-axis denotes the ratio of the average traveling time between our protocol and the traffic lights (i.e., setting the traffic lights performance as baseline). Each data point is the average of 6 randomly generated traffic patterns following Poisson distribution. We can find out that *our proposed protocol provides significantly better performance than the traditional traffic lights* when the traffic is not too heavy or when the traffic arriving rates from different directions are *asymmetric*. When the traffic is heavy and symmetric from different directions, the traditional traffic lights achieve their



(a) $K = 1$.



(b) $K = 2$.



(c) $K = 3$.

Figure 3.12: The performance comparison between the proposed protocol and traditional traffic lights. K: north-south directions have K times the traffic arriving rate as the east-west directions

best performance and can be better than our solution (although our solution can be further improved with more finer-granularity control as planned in the future work).

Impact of Communication Delays on Performance. We further evaluate the performance (average traveling time of each vehicle) under different communication delays, as shown in Fig. 3.13. We can see that the performance significantly decreases (longer traveling time) with the increase of communication delays, in particular when the traffic is heavy. This again demonstrates the *importance of modeling and analyzing the impact of delays in intersection management*, not only for the safety and liveness properties, but also for the system performance. It should be noted that in normal traffic conditions, the communication delays are typically under one second (in the range of dozens of milliseconds and can reach hundreds of milliseconds when considering end-to-end delays [155]). Under security attacks such as jamming, the delays can be much longer. Note that if we remove the delay consideration in the protocol, deadlocks are observed during simulation.

Single Intersection with Multiple Lanes

In this experiment, we study the communication delay and its impact on the performance of a single intersection with multiple lanes. The system setup is shown in Figure 3.14. The intersection has four ways ω_1, ω_2, ω_3 and ω_4, and each direction is associated with three lanes, namely, one lane for left turn, one lane for going straight and one lane for right turn. Therefore, the feasible paths for the system are listed as follows: $\pi_1 = (\lambda_{11}, \lambda_{46})$, $\pi_2 = (\lambda_{12}, \lambda_{35})$, $\pi_3 = (\lambda_{13}, \lambda_{24})$, $\pi_4 = (\lambda_{21}, \lambda_{16})$, $\pi_5 = (\lambda_{22}, \lambda_{45})$, $\pi_6 = (\lambda_{23}, \lambda_{34})$, $\pi_7 = (\lambda_{31}, \lambda_{26})$, $\pi_8 = (\lambda_{32}, \lambda_{15})$, $\pi_9 = (\lambda_{33}, \lambda_{44})$, $\pi_{10} = (\lambda_{41}, \lambda_{36})$, $\pi_{11} =$

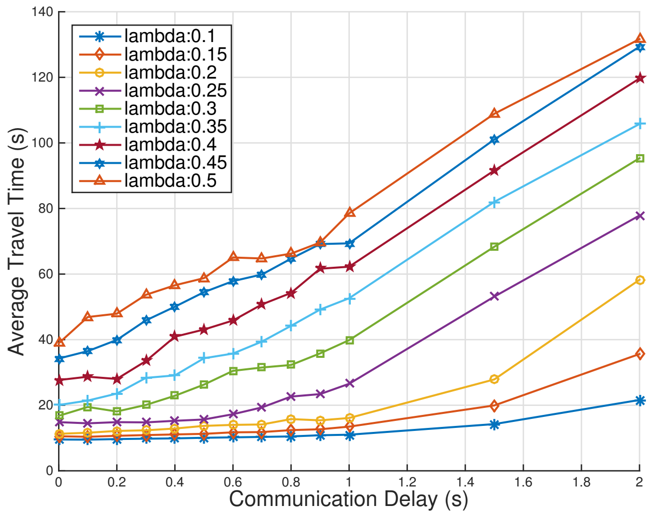
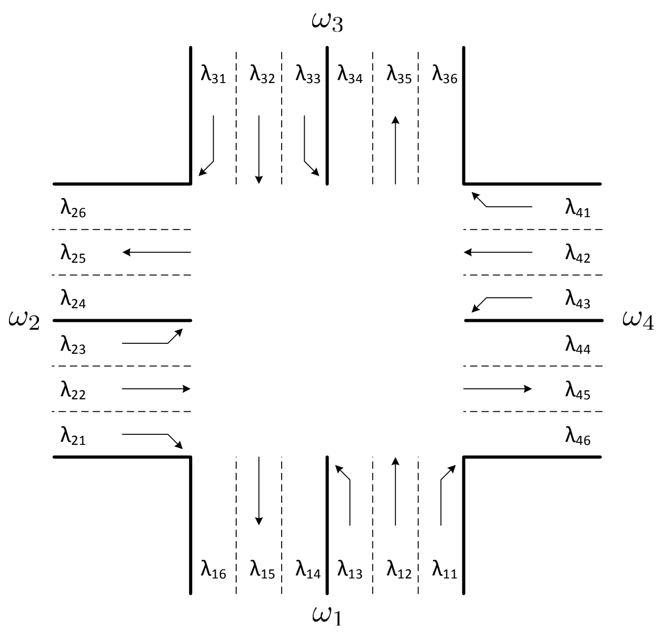


Figure 3.13: Performance of our protocol under different communication delays.

$(\lambda_{42}, \lambda_{25})$, $\pi_{12} = (\lambda_{43}, \lambda_{14})$. The length of each lane is 100 meters. In this simulation, vehicles are assumed to have a length of 5 meters, with maximum acceleration 0.8 m/s^2 and deceleration 4.5 m/s^2. The speed limit of the vehicle is 10 m/s. The routes of the vehicles are randomly generated, with the probability ratio of left turn, going straight and right turn set as 0.25:0.5:0.25. The arriving time of the vehicles follows Poisson distribution with an arriving rate denoting how many vehicles will arrive per second in average. In our experiment, the arriving rate ranges from 0.1 vehicle/s to 0.5 vehicle/s. The number of total vehicles entering the intersection is set as 300. The level of timing attack is represented by the delay added to the messages. The performance of the intersection is evaluated as the average traveling time of all the vehicles aiming to cross the intersection.

The simulation results are shown in Figure 3.15. The x-axis denotes the communication delay caused by the timing attack, and the y-axis denotes the average traveling



ω_3
λ_{31} λ_{32} λ_{33} λ_{34} λ_{35} λ_{36}
λ_{26} λ_{25} λ_{24}
ω_2
λ_{23} λ_{22} λ_{21}
λ_{41} λ_{42} λ_{43}
ω_4
λ_{44} λ_{45} λ_{46}
λ_{16} λ_{15} λ_{14} λ_{13} λ_{12} λ_{11}
ω_1

Figure 3.14: Setup for single-intersection simulations.

time of the vehicles as performance. We define traffic patterns in our simulation as the flow ratio of the vehicles arriving from north-south directions and the vehicles arriving from west-east directions. For example, traffic pattern "flow 0.5 : 0.1" denotes the average traffic flow from north-south directions is 0.5 vehicle/s and the average traffic flow from west-east directions is 0.1 vehicle/s. The figure shows that for each traffic pattern, the performance significantly decreases as the communication delay increases. For each specific delay, no matter symmetric or asymmetric traffic pattern, the trend of performance deterioration is similar.

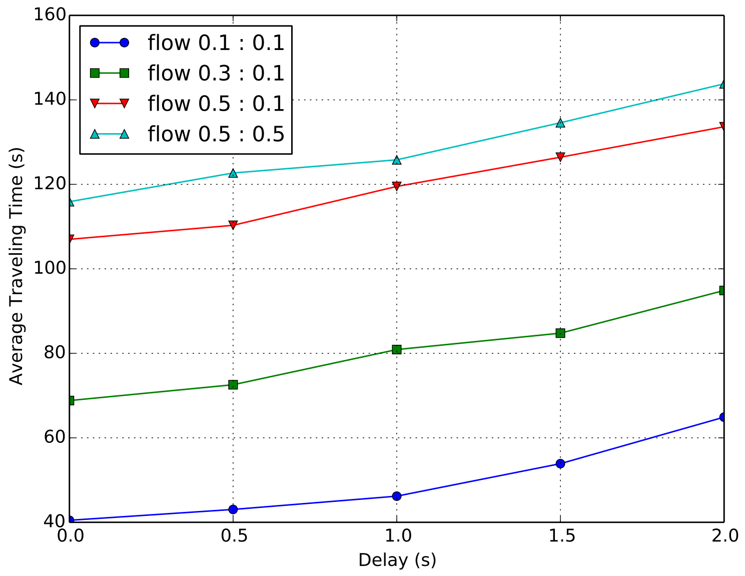


Figure 3.15: Performance of a single intersection under different communication delays.

Multiple Intersection with Multiple Lanes

In this experiment, the setup is a traffic network with nine intersections as in Figure 3.16 (b). Each intersection has the same setup as in Figure 3.14 with designated lanes for left turn, going straight and right turn. The connection of adjacent intersections is to connect the corresponding lanes together, i.e., connecting left lane with left lane, middle lane with middle lane, and right lane with right lane. In this simulation, vehicles can only arrive from the twelve entrances. At each intersection, the vehicle has a 0.25 probability to turn left, a 0.5 probability to go straight, and a 0.25 probability to turn right. The total number of vehicles entering the network is set as 1200. We first assume the attacker to launch timing attack to all the intersections, and then study the influence by attacking only one intersection.

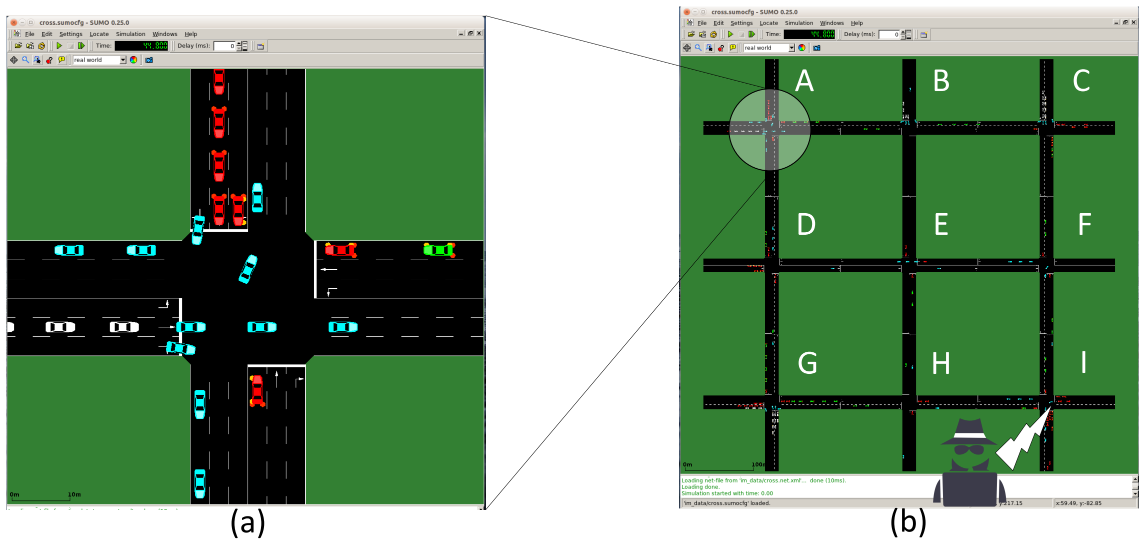


Figure 3.16: Simulations based on SUMO.

Figure 3.17 shows that average traveling time increases with communication delay applied to all nine intersections. In this case, the trend is similar to the single intersection with multiple lanes. For each traffic pattern, the average traveling time of vehicles increases as delay increases. Note that the performances of traffic patterns “0.5 : 0.1” and “0.5 : 0.5” are very similar and pattern “0.5 : 0.1” is even slightly better than “0.5 : 0.5” when the delay is 2.0 second.

Figure 3.18 shows the attack to only one of the intersections with a delay of 4.0 second. Letters ‘A’ to ‘I’ represent different intersections, and their positions are shown in Figure 3.16 (b). The y-axis denotes the average traveling time of vehicles.

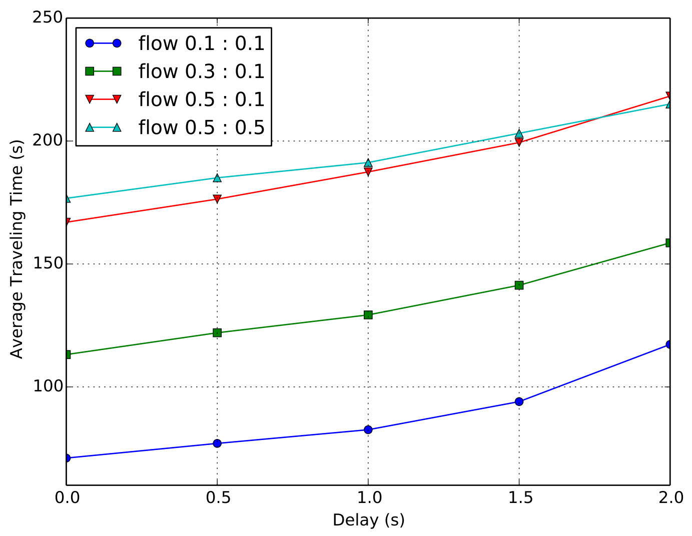


Figure 3.17: Performance of nine interconnected intersections under the same attack.

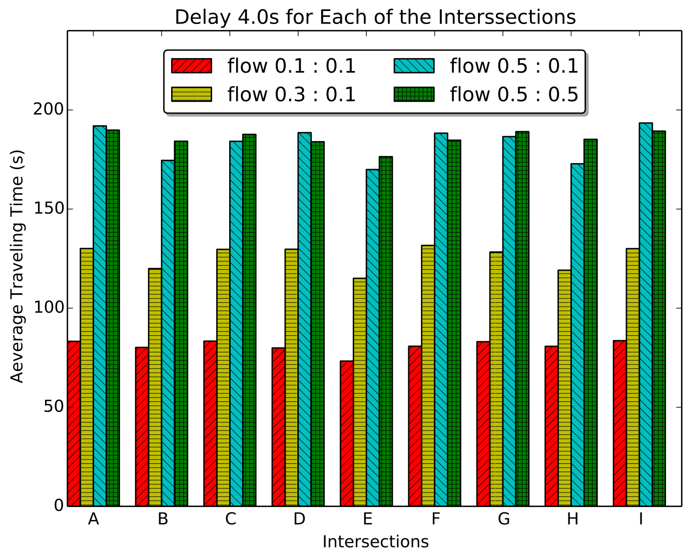


Figure 3.18: Performance of nine intersections (denoted by "A" to "I") if one intersection is under timing attack.

3.3 Security, Control, and Schedulability Codesign

In this section, we study software to hardware mapping with constraints from the application layer as well as the resource and communication constraints from the software and hardware layers. As described in Section 1.2 in Chapter 1, researchers have proposed various control-oriented approaches for attacks on cyber-physical systems [118, 54, 116, 139]. However, resource and real-time constraints are not considered in these approaches, and there is no guarantee of schedulability and control performance. To build correct, efficient and secure cyber-physical systems, it is crucial to quantitatively model the impacts of security techniques on other related metrics, and *address them together in a codesign environment.*

We utilize the ***cross-layer codesign framework*** to combine control-theoretic methods at the functional layer and cybersecurity techniques at the embedded platform layer. Furthermore, we address security together with other design metrics, in particular the control performance, under resource and real-time constraints.

3.3.1 General Codesign Formulations

Our work addresses a typical cyber-physical system, where multiple control loops share an embedded platform, with messages transmitted from sensors (vision sensors, GPS, ultrasound, etc.) to controllers and from controllers to actuators, as shown in Fig. 3.19 and similarly considered in [139]. Each controller (implemented as a control task) collects the sensed information, processes it on a shared computation unit (e.g., a single-core CPU)[5], and

[5]We assume all the control tasks are implemented on a single computation unit in this work. Our formulation can be extended to address multicore and multi-processor platforms with more complex models for schedulability and security level measurement, as planned in the future work.

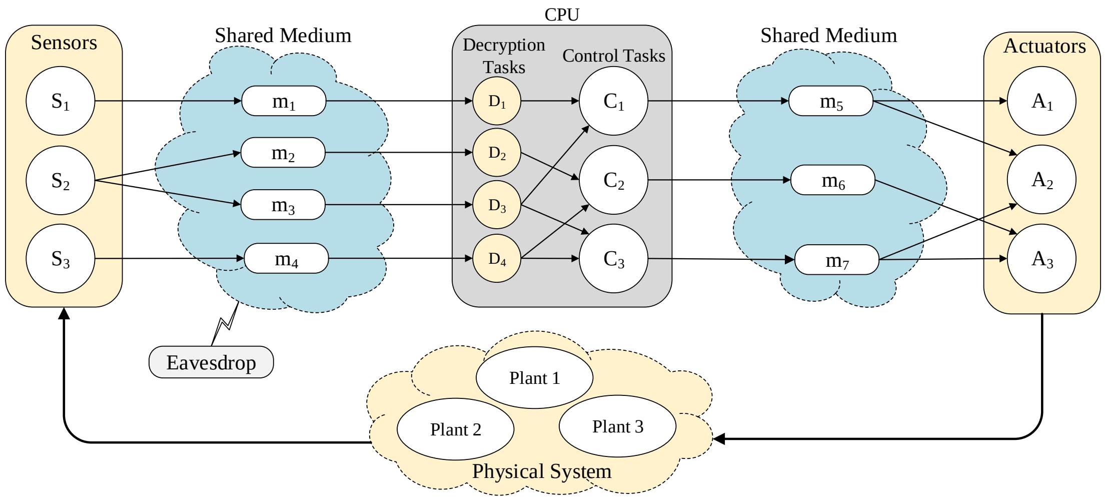


Figure 3.19: A cyber-physical systems with multiple control loops sharing an embedded platform. Attackers may eavesdrop on the communication medium and apply various attacks.

sends commands to various actuators. In our model, a message from a sensor may be sent to multiple control tasks for sharing information (which is common in many domains such as automotive systems). If a message is encrypted for security measurement, a dedicated *decryption task* is used for decrypting the message and send it to the receiving tasks (this approach reduces overhead, compared with carrying out the decryption of the same message within each receiving task[6]).

The attackers may be able to eavesdrop on the communication medium and further reconstruct the system state. This results not only in a loss of privacy, but can further be used as the basis for other malicious attacks. The system is resource-constrained, as control tasks compete for computation resources and messages compete for communica-

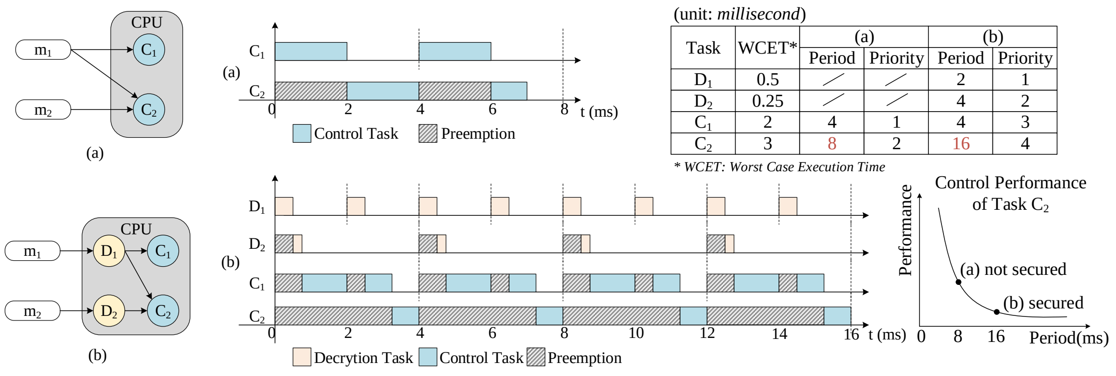
[6]In this work we assume the message is encrypted with the same key for all the receiving tasks. A more complex strategy with different keys may be used for higher level of security with significantly more overhead.

tion resources. Applying security techniques such as message encryption will introduce computation and communication overhead, through the elongation of message transmission time, the additions of decryption tasks, and consequently the elongation of control task execution time due to resource contention. This will in turn have a significant impact on system schedulability and control performance, as demonstrated in the following motivating example.

Motivating Example

A motivating example is shown in Fig. 3.20. In this example, there are two control tasks C_1 and C_2, two messages m_1 and m_2, and potentially two decryption tasks D_1 and D_2 if the corresponding messages are encrypted. We show only the control and decryption tasks here for simplicity, and will model sensing and actuation tasks later in problem formulation. All the tasks are implemented on a single-core CPU under the preemptive fixed-priority scheduling policy (commonly used in cyber-physical systems, such as automotive systems with OSEK standard [26]). In order to guarantee the correct execution order, the priorities of decryption tasks are set higher than the control tasks. We further assume C_1 has higher priority than C_2. The initial periods of C_1 and C_2 are set to $4ms$ and $8ms$, respectively.

We consider two scenarios: (a) no message is encrypted and the system is not protected, and (b) both messages are encrypted and the system is protected. In scenario (a), no decryption task is needed and there is no security overhead. Tasks C_1 and C_2 can be completed within their periods, i.e., before their next periodic activation. In scenario (b), the overhead of two decryption tasks elongates the time it takes to complete task C_2,



Task	WCET*	(a)		(b)	
		Period	Priority	Period	Priority
D_1	0.5	/	/	2	1
D_2	0.25	/	/	4	2
C_1	2	4	1	4	3
C_2	3	8	2	16	4

Figure 3.20: A motivating example: (a) no message is encrypted, (b) both messages are encrypted and the period of C_2 has to be increased, which leads to higher security level but lower control performance.

and consequently the period of C_2 cannot be smaller than $16ms$ (otherwise C_2 will not complete within its period and the system functionality may be incorrect). As discussed in the literature [135, 136], control performance typically decreases significantly when the control task period increases. Therefore, this motivating example has clearly shown the additions of security measurements may have a negative impact on system timing, and consequently control performance and system schedulability. It is essential to *quantitatively* analyze the trade off among these metrics. In the following, we will introduce our general codesign formulation for this purpose.

General Formulation

Our codesign framework addresses three design metrics: control performance, system security level, and platform schedulability. Control performance and system security level are measured at the functional layer, while schedulability is analyzed at the embedded platform layer. As shown in Fig. 3.21, to bridge these metrics, a set of *interface variables* are introduced, specifically the sampling period of every control task and the selection of messages to be encrypted. Intuitively, when the sampling period of a control task increases, its control performance decreases, and platform schedulability becomes easier with less frequent activation of the control task. On the other hand, when the number of messages being encrypted increases, the system security level increases, and platform schedulability becomes harder because of the increased overhead – the sampling periods may have to increase for schedulability concern thereby worsening the control performance. These relations are quantitatively modeled in our codesign formulation as introduced below.

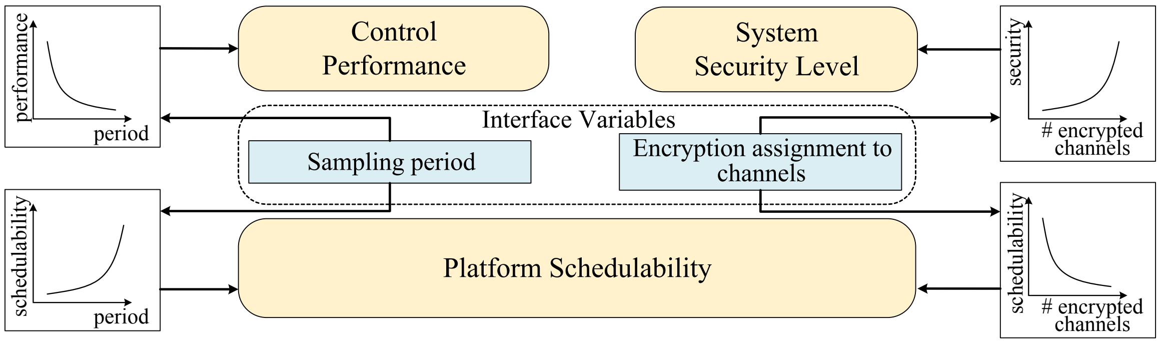


Figure 3.21: Illustration of the interface variables and their relations to design metrics.

First, we define the following notation for the cyber-physical system in Fig. 3.19: Tasks are represented by $\mathcal{T} = \mathcal{T}_\mathcal{S} \cup \mathcal{T}_\mathcal{C} \cup \mathcal{T}_\mathcal{D} \cup \mathcal{T}_\mathcal{A}$, where $\mathcal{T}_\mathcal{S} = \{ \tau_s^1, \tau_s^2, \ldots, \tau_s^l \}$ denotes the set of sensing tasks, $\mathcal{T}_\mathcal{C} = \{ \tau_c^1, \tau_c^2, \ldots, \tau_c^m \}$ denotes the control tasks, $\mathcal{T}_\mathcal{D} = \{ \tau_d^1, \tau_d^2, \ldots, \tau_d^p \}$ denotes the decryption tasks, and $\mathcal{T}_\mathcal{A} = \{ \tau_a^1, \tau_a^2, \ldots, \tau_a^q \}$ denotes the actuation tasks. Each task τ_i is associated with an activation period T_τ^i and worst case execution time C_τ^i. The worst-case execution time for sensing tasks include the time for processing the sensor data from crude input form. The relative deadline of each task is set equal to its period as in typical real-time systems. Messages are represented by $\mathcal{M} = \{m_1, m_2, \ldots, m_p\}$. $src(m_i)$ denotes the source task of message m_i, and the set $\{dst(m_i)\}$ includes all destination tasks of message m_i. Each message m_i is associated with a period T_{m_i} and an original transmission time C_{m_i} (without encryption). The relative deadline of each message is set equal to its period.

A control loop consists of a control task connected with a set of sensors, a set of actuators, and the corresponding plants. A control path p is a sequence of tasks (including sensing, control and actuation tasks) connected through messages.

In our codesign formulation, we explore the selection of messages for encryption and the assignment of periods to control tasks to address both control performance and system security while guaranteeing platform schedulability. In the following formulation from Equation (3.5) to (3.9), we set control performance as the optimization objective and assume constraints on the system security level. Then, by varying the requirements on the system security level, we can obtain the Pareto front between control performance and security level.

$$\text{maximize: } J(\overrightarrow{T_{\tau_c}}) \text{ (control performance)} \quad \text{s.t.} \tag{3.5}$$

$$S(\overrightarrow{o_m}) \geq S_0 \text{ (security constraint)} \tag{3.6}$$

$$U_e(\overrightarrow{o_m}, \overrightarrow{T_{\tau_c}}) \leq U_{e0} \text{ (computational resource)} \tag{3.7}$$

$$U_c(\overrightarrow{o_m}) \leq U_{c0} \text{ (communication resource)} \tag{3.8}$$

$$l_p(\overrightarrow{o_m}, \overrightarrow{T_{\tau_c}}) \leq D_p \text{ (end to end latency)} \tag{3.9}$$

As stated before, the design variables are the interface variables, which include the control task periods $\overrightarrow{T_{\tau_c}}$=\{ $T^1_{\tau_c}$, $T^2_{\tau_c}$, $\ldots$, $T^m_{\tau_c}$\}, and the selection of messages for encryption, denoted by $\overrightarrow{o_m} = \{o_{m_1}, o_{m_2}, \ldots, o_{m_k}\}$. The binary variable o_{m_i} is 1 if message m_i is encrypted, and 0 otherwise. The variables in Equation (3.5) to (3.9) are defined as follows and the details of these equations will be introduced in the rest of the section. J represents the control performance, which is a function of control task period $\overrightarrow{T_{\tau_c}}$ and works as the objective of this problem. S denotes the security level of the system, which is a function of the selection of messages for encryption $\overrightarrow{o_m}$. S_0 is the minimum security level set in the design requirements. U_e represents the utilization of the computation unit, and U_{e0} is the required maximum utilization. U_c denotes the utilization of the shared communication medium, and similarly, U_{c0} is the required upper bound on communication utilization. l_p represents the end to end latency of path p, and D_p is the required deadline for path p. In what follows, we introduce how each of the equations above are refined for modeling control performance, security level, and schedulability, with respect to the design variables.

Control Performance Modeling. We consider linear continuous-time dynamics for each physical plant:

$$\begin{aligned} \dot{x} &= Ax + Bu + w \\ y &= Cx + v \end{aligned} \tag{3.10}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $x : \mathbb{R} \to \mathbb{R}^n$ is the map describing the system state, $u : \mathbb{R} \to \mathbb{R}^m$ is the control input, and $y : \mathbb{R} \to \mathbb{R}^p$ is the measured output. Finally, $w : \mathbb{R} \to \mathbb{R}^n$ and $v : \mathbb{R} \to \mathbb{R}^p$ represent process and measurement noise, which we assume to be zero-mean, Gaussian, and white.

Each continuous-time physical plant is controlled by a digital controller τ_c^i with a sampling period $T_{\tau_c}^i$, which is also the activation period of the corresponding control task.

Thus, intuitively, the longer the sampling period, the worse the performance of the control system. For controller τ_c^i , we assume the control performance $J_{\tau_c}^i$ decreases exponentially as the sampling period $T_{\tau_c}^i$ increases, as in references [135, 136, 99]. [7] As in these works, we let the performance $J_{\tau_c}^i$ be an exponentially decaying function of $T_{\tau_c}^i$ defined as

$$J_{\tau_c}^i = \alpha^{-\beta T_{\tau_c}^i} \tag{3.11}$$

for appropriate constants $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}$ and $\alpha > 0$, $\beta > 0$. For each control task, α and β can be obtained by fitting the relation of control performance and sampling period with an exponential decay function, as described in Section 3.3.2.

For a system consisting of multiple control tasks, the overall control performance is calculated as the weighted average in Equation (3.12), where $\omega_{\tau_c}^i$ is the weight for each control task and $|\mathcal{T_C}|$ is the number of control tasks.

$$J = \frac{1}{|\mathcal{T_C}|} \sum_{\tau_c^i \in \mathcal{T_C}} \omega_{\tau_c}^i J_{\tau_c}^i \tag{3.12}$$

<u>*Security Level Modeling.*</u> We consider attackers with knowledge of the system dynamics (i.e. the matrices A, B, C in Equation (3.10)), and attempt to reconstruct the system state by eavesdropping messages containing sensor measurements. It should be noticed that reconstructing the system state results in not only a loss of privacy, but also in vulnerabilities to feedback attacks.

To protect against such attacks, a key-based encryption technique is adopted in our framework. As stated before, for simplicity and efficiency, we only consider the case

[7]In general, the relation between control performance and sampling period could be quite complex and may require simulations for accurate capturing. In those cases, we may approximate the control performance with a closed-form representation (if possible) and apply our codesign formulation, or we can combine our codesign formulation and simulated annealing algorithm directly with simulations for exploring control performance.

where a message is encrypted with the same key for all receiving tasks. Our formulation can be extended to address multiple-key distribution scenarios, using techniques similar to the ones in [95].

The security level of the control task τ_c^i is defined as the *complexity to eavesdrop the messages with sensor measurements and observe the system state.* It is modeled in Equation (3.13). Specifically, $P(n_{\tau_c}^i, \mu)$ represents the probability for an attacker to eavesdrop μ encrypted sensing messages for task τ_c^i, where $n_{\tau_c}^i$ is the number of encrypted sensing messages. $\xi(\Omega_\mu(\rho))$ represents the complexity for the attacker to estimate the system state from the ρ-th element of Ω_μ, which is the set containing all possible combinations of the eavesdropped messages, i.e., μ encrypted messages together with all unencrypted messages (its cardinality is $\binom{n_{\tau_c}^i}{\mu}$). More about $\xi(\Omega_\mu(\rho))$ is discussed later. $P(n_{\tau_c}^i, \mu)$ can be further defined as in Equation (3.14), where $\mathcal{D}(l_{key})$ denotes the probability for an attacker to decrypt one message with encryption key length l_{key}.

$$S_{\tau_c}^i = \sum_{\mu=0}^{n_{\tau_c}^i} \sum_{\rho=1}^{\binom{n_{\tau_c}^i}{\mu}} P(n_{\tau_c}^i, \mu)\xi(\Omega_\mu(\rho)) \tag{3.13}$$

$$P(n_{\tau_c}^i, \mu) = \mathcal{D}(l_{key})^\mu (1 - \mathcal{D}(l_{key}))^{n_{\tau_c}^i - \mu} \tag{3.14}$$

We use the example in Fig. 1 to illustrate the parameters shown above. Let us consider the solution in which only messages m_1 and m_3 are encrypted. In this case, the number of sensing messages of task C_1 is 2, i.e., m_1 and m_3, and the number of encrypted sensing messages of task C_1 is 2 as both m_1 and m_3 are encrypted. The number of sensing message of task C_2 is 2, i.e., m_2 and m_4, and the number of encrypted sensing messages of task C_2 is 0, as neither m_2 nor m_4 is encrypted. The number of sensing messages of task C_3

is 2, and the number of encrypted sensing messages of task C_3 is 1. Using C_3 as an example for Equation (3.14), the number of encrypted sensing messages $n^3_{\tau_c} = 1$. The attacker may eavesdrop 0 or 1 encrypted sensing message (i.e., μ is 0 or 1), with the probability defined as $P(1, 0)$ and $P(1, 1)$ in Equation (3.14). In addition, the attacker can always eavesdrop the unencrypted message m_4 to learn about C_3. Set Ω_μ contains m_4 and m_3 if $\mu = 1$ (i.e., encrypted message m_3 is eavesdropped), and only contains m_4 if $\mu = 0$. Equation (3.13) considers all possible situations for C_3 and computes its overall security level.

We simultaneously consider multiple control loops. A successful reconstruction of the system state of any control loop may lead to the whole system being attacked, therefore the system level security is defined as the minimum security level among all control tasks as in Equation (3.15).

$$S = \min_{\tau^i_c \in \mathcal{T}_C} S^i_{\tau_c} \tag{3.15}$$

The function $\xi(\Omega_\mu(\rho))$ can be defined in different ways. For deterministic systems (that is, without process and measurement noise), as shown in [117], $\xi(\Omega_\mu(\rho))$ can be defined based on the *Observability Gramian* [75]. This measure of observability quantifies the relative importance of different measurement channels based on the system dynamics only, and independently of any particular estimation scheme. For stochastic systems driven by process and measurements noise, $\xi(\Omega_\mu(\rho))$ can be defined based on the estimation error of an optimal *Kalman filter.* This measure of observability, which is inherently dependent on the Kalman estimation procedure, allows us to highlight the role of system noise with respect to the system security level. We now present these two metrics.

- *Option 1 – Observability Gramian.* Let $\mathcal{K} \subseteq \{1, \dots, p\}$ be the set of measurements decrypted by the attacker, and let $y_{\mathcal{K}}$ be the decrypted measurements. The Observability Gramian is defined as

$$\mathcal{O}_{\mathcal{K}} := \sum_{\tau=0}^{\infty} (A^{\mathsf{T}})^{\tau} C_{\mathcal{K}}^{\mathsf{T}} C_{\mathcal{K}} A^{\tau} \tag{3.16}$$

where $C_{\mathcal{K}}$ is the output matrix associated with the decrypted measurements ($y_{\mathcal{K}} = C_{\mathcal{K}} x$). The energy associated with the decrypted measurements $\mathcal{K}$ and due to the free evolution of the system from the x is

$$\begin{aligned} E(x) : &= \sum_{\tau=0}^{\infty} \|y_{\mathcal{K}}(\tau)\|^2 \sum_{\tau=0}^{\infty} y_{\mathcal{K}}^T y_{\mathcal{K}} = \sum_{\tau=0}^{\infty} x^T(\tau) C^T C x(\tau) \\ &= \sum_{\tau=0}^{\infty} x^T (A^T)^{\tau} C^T C A^{\tau} x = x^T \mathcal{O}_{\mathcal{K}} x \geq \lambda_{min}(\mathcal{O}_{\mathcal{K}}) \end{aligned} \tag{3.17}$$

where $\lambda_{min}(\mathcal{O}_{\mathcal{K}})$ denotes the smallest modulus of the eigenvalues of $\mathcal{O}_{\mathcal{K}}$. The following facts can be formally proven with standard methods [75]. First, the larger $\lambda_{min}(\mathcal{O}_{\mathcal{K}})$, the easier the reconstruction of the system state from measurements. Thus, the eigenvalue $\lambda_{min}(\mathcal{O}_{\mathcal{K}})$ measures the information of the system state contained in the measurements $y_{\mathcal{K}}$. Second, the eigenvalue $\lambda_{min}(\mathcal{O}_{\mathcal{K}})$ is a function of both the cardinality and the decrypted messages. Third, the inequality (3.17) holds with equality for certain system states and for an infinite observation horizon. Otherwise, $\lambda_{min}(\mathcal{O}_{\mathcal{K}})$ is a lower bound on the information retrieved by the attacker from the decrypted measurements $y_{\mathcal{K}}$. Thus, for deterministic systems we select $\xi(\Omega_{\mu}(\rho))$ as the equation shown below.

$$\xi(\Omega_{\mu}(\rho)) = \lambda_{min}^{-1}(\mathcal{O}_{\Omega_{\mu}(\rho)}) \tag{3.18}$$

- *Option 2 – Kalman filter.* Let $y_{\mathcal{K}}$ be the measurements decrypted by the attacker, and define the Kalman filter as

$$\hat{x}_{k+1} = A\hat{x}_k + K_k(y_k - C\hat{x}_k) + Bu_k$$

where the Kalman gain K_k and the error covariance matrix $P_{k+1} \triangleq \mathbb{E}\big[(\hat{x}_{k+1} - x_{k+1})(\hat{x}_{k+1} - x_{k+1})^T\big]$ can be calculated with the recursions

$$K_k = AP_kC^T(CP_KC^T + \Sigma_v)^{-1}$$
$$P_{k+1} = AP_kA^T - AP_kC^T(CP_kC^T + \Sigma_v)^{-1}CP_kA^T + \Sigma_w$$

with initial conditions $\hat{x}_1 = \mathbb{E}[x_1]$ and $P_1 = \mathbb{E}[x_1x_1^T]$. The matrices Σ_w and Σ_v are the process and measurements noise covariance matrices respectively.

If the system is detectable, the above recursion converges to the steady state $\lim_{k\to\infty} P_k = P$, where P can be obtained as the solution to an algebraic Riccati equation [75]. For the ease of presentation, we assume that the attacker uses a steady state Kalman filter, and we adopt $trace(P)$ to evaluate the complexity of the attacker's reconstruction of the state. Thus,

$$\xi(\Omega_\mu(\rho)) = trace(P) \tag{3.19}$$

Platform Schedulability. The encryption/decryption of messages puts overhead on computation and communication, and may have significant impact on platform schedulability as modeled below.

- *Decryption Tasks.* Every encrypted message m_i requires a decryption task ${\tau_d}^i$, with worst case execution time denoted by $C^i_{\tau_d}$. We use d_{m_i} to denote the decryption time of message m_i. For un-encrypted messages, we simply set $C^i_{\tau_d}$ to 0. The computation of $C^i_{\tau_d}$ thus can be modeled as in Equation (3.20), where o_{m_i} denotes whether the message is encrypted and d_{m_i} is a function of encryption key length l_{key} and message length l_{m_i}. The period of ${\tau_d}^i$, denoted as $T^i_{\tau_d}$, is equal to the message period T_{m_i} as shown in Equation (3.22).

$$C^i_{\tau_d} = d_{m_i} o_{m_i} \tag{3.20}$$

$$d_{m_i} \sim (l_{key}, l_{m_i}) \tag{3.21}$$

$$T^i_{\tau_d} = T_{m_i} \tag{3.22}$$

System scheduling has to ensure the functional dependencies among tasks, which include the dependencies between decryption tasks and corresponding control tasks. Later in our automotive domain formulation, where fixed-priority preemptive scheduling is assumed, we achieve this by setting the priorities of decryption tasks higher than the priorities of control tasks.

- *Computation Resource Utilization.* The original control tasks and the added decryption tasks are all allocated to a single computation unit. The constraint for computation resource utilization U_e is shown in Equation (3.23). As defined earlier, $\mathcal{T_C}$ represents the set of control tasks, where each control task τ^i_c has a worst case execution time $C^i_{\tau_c}$ and a period $T^i_{\tau_c}$. $\mathcal{T_D}$ represents the set of decryption tasks, with $C^j_{\tau_d}$ and $T^j_{\tau_d}$ similarly defined. U_{e0} (between 0 and 1) represents the utilization bound set by design requirement.

$$U_e = \sum_{\tau_c^i \in \mathcal{T}_\mathcal{C}} \frac{C_{\tau_c}^i}{T_{\tau_c}^i} + \sum_{\tau_d^j \in \mathcal{T}_\mathcal{D}} \frac{C_{\tau_d}^j}{T_{\tau_d}^j} \leq U_{e0} \tag{3.23}$$

- *Communication Resource Utilization.* The constraint on communication resource utilization U_c is shown in Equation (3.24), where the message transmission time C_{m_i} depends on the message length l_{m_i}, selection for encryption o_{m_i}, encryption key length l_{key}, and link data rate R.

$$U_c = \sum_{m_i \in \mathcal{M}} \frac{C_{m_i}}{T_{m_i}} \leq U_{c0} \tag{3.24}$$

$$C_{m_i} \sim (l_{m_i}, l_{key}, R, o_{m_i}) \tag{3.25}$$

- *End-to-end Path Latency* The end-to-end latency along a control path p (from sensor s_i to control task c_j to actuator a_k) is modeled in Equation (3.26). In the formulation, $t_{\tau_s}^i$, $t_{\tau_d}^m$, $t_{\tau_c}^j$, $t_{\tau_a}^k$ represent the maximum latency for sensing, decryption, control execution, and actuation, respectively. $t_{m_{s \to c}}$ represents the message transmission latency between sensor s_i and control task c_j, and $t_{m_{c \to a}}$ represents the message transmission latency between control task c_j and actuator a_k.

$$l_{p(\tau_s^i, \tau_c^j, \tau_a^k)} = t_{\tau_s}^i + t_{m_{s \to c}} + t_{\tau_d}^m + t_{\tau_c}^j + t_{m_{c \to a}} + t_{\tau_a}^k \tag{3.26}$$

3.3.2 Automotive Domain Codesign

To demonstrate the effectiveness of our approach, we customize and refine the general codesign formulation introduced in Section 3.3.1 to automotive systems.

Customization and Refinement of the General Formulation

Control Performance Refinement. In this work, we adopt exponential decay functions as control performance formulation in the automotive domain refinement. This is because for many automotive systems, the relation between control performance and sampling period can be captured by these functions with sufficient accuracy for our codesign.

As an example, we studied the model of an automotive electrohydraulic servomechanism controlled by a pulse-width modulated (PWM) solenoid in the Simulink library [142]. It is a feedback control loop with a high-level controller collecting data from sensors and sending commands to actuators. The control loop may conceivably be implemented in a distributed fashion, sharing sensors, actuators and computation node with other control loops. We can change the sampling period of the control loop, and measure its performance as the *reciprocal of the root mean square (RMS)* of the difference between the actuator position and the reference position (i.e. the error in the actuator position) of the electrohydraulic servo over the simulation process.

Fig. 3.22 shows this control performance measurement for different sampling periods. When the period increases, the performance decreases with larger error in the actuator position (i.e., larger RMS value). An exponential decay function can be used to approximate (fit) the functional dependency of the control performance on the sampling period. As shown in the figure, even when an exponential control cost function cannot be determined analytically, it is still possible to determine the parameters by fitting the cost values obtained through simulation runs for different sampling period values. In this case, the exponential fitting is very close to the simulation data, with an R-squared value of 0.972 (1

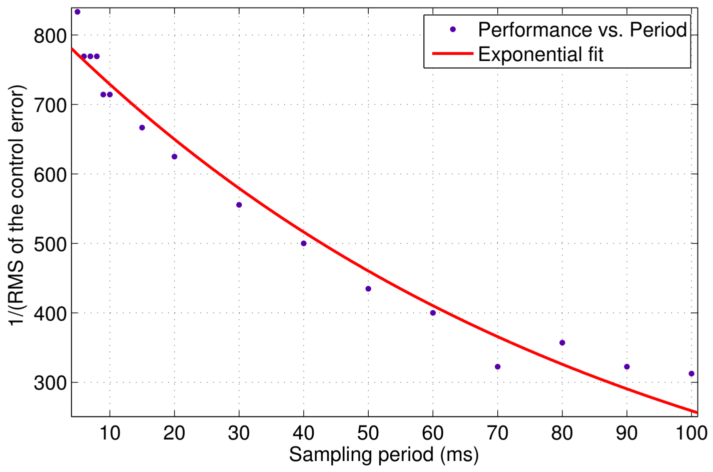


Figure 3.22: Fitting the control performance of the Simulink electrohydraulic servo example for different sampling periods with an exponential decay function.

is a perfect fit). We have also conducted experiments on a fuel control system example and an engine speed control example in the Simulink library, and observed similar exponential decay trend between control performance and sampling period (with R-squared values of 0.994 and 0.996 in the exponential fittings, respectively).

We further normalize the performance of each control task with respect to its performance under an initial period $T^{i0}_{\tau_c}$ (which is obtained by solving the entire problem without encryption).

$$J^i_{\tau_c} = \alpha^{-\beta(T^i_{\tau_c} - T^{i0}_{\tau_c})} \tag{3.27}$$

The normalization is for fair consideration when performances of multiple control tasks are averaged to get overall system control performance, as shown in Equation (3.12) in the general formulations.

Security Level Refinement. For automotive systems, the presence of measurement noise motivates the use of the Kalman filter based approach to quantify the system security level (In the experiments, we also evaluate the security level using Observability Gramian. However, we think Kalman filter is a more suitable measurement for security level in automotive systems as it directly addresses measurement noises). For simplicity, we assume that all sensing messages for a control task reveal equal amount of information of the system state, that is, yield the same estimation error covariance matrix. Our approach can be extended to the case of inhomogeneous measurement channels at the cost of a more involved notation. We use $Kal(n)$ to represent the Kalman filter performance with any n measurement messages, i.e., how easy it is to reconstruct the system state from the information of n messages. The security level defined in Equation (3.13) can be refined to Equation (3.28) in below.

$$S^i_{\tau_c}(n^i_{\tau_c}) = \sum_{\mu=0}^{n^i_{\tau_c}} \binom{n^i_{\tau_c}}{\mu} P(n^i_{\tau_c}, \mu) Kal(N^i_{\tau_c} - n^i_{\tau_c} + \mu) \tag{3.28}$$

$$P(n^i_{\tau_c}, \mu) = 2^{-\mu l_{key}}(1 - 2^{-l_{key}})^{n^i_{\tau_c} - \mu} \tag{3.29}$$

$N^i_{\tau_c}$ denotes the total number of sensing messages for task τ^i_c, out of which $n^i_{\tau_c}$ messages are encrypted. $P(n^i_{\tau_c}, \mu)$ denotes the probability that μ encrypted messages (out of $n^i_{\tau_c}$) are hacked by an attacker. We assume the attacker uses brute-force attack by randomly guessing the key with probability $2^{-l_{key}}$ to decrypt one message in one try (Depending on the systems, more sophisticated attacks may be applied with different success probability. We plan to study some of those cases in future work, and our codesign formulation will still apply). $Kal(N^i_{\tau_c} - n^i_{\tau_c} + \mu)$ is the Kalman filter performance with the observation of μ hacked messages and $N^i_{\tau_c} - n^i_{\tau_c}$ un-encrypted messages (all un-encrypted messages are assumed as observable to the attacker). For fair comparison, the security level of each control task is normalized with respect to the maximum security level the control task may obtain:

$$S'^{i}_{\tau_c}(n^i_{\tau_c}) = S^i_{\tau_c}(n^i_{\tau_c}) / \max_{n^i_{\tau_c} \in [0, N^i_{\tau_c}]} \{S^i_{\tau_c}(n^i_{\tau_c})\} \tag{3.30}$$

The system security level is denoted by the task that has the lowest security level as show in Equation (3.15).

Platform Schedulability Refinement. In this work, we consider a single CAN (Controller Area Network) [146] or CAN-FD [62] bus as the communication medium. We use Bus_speed to denote the bus speed (bits/ms). Similar to [110, 68], we adopt the KASUMI encryption algorithm with encryption speed at En_speed bits/ms, decryption speed De_speed bits/ms and block size at B_size.

Based on the KASUMI algorithm, the size of an encrypted message should be elongated to $n{*}B\_size$ (n is an integer), thus the modified transmission time of an encrypted message m_i, denoted as c_{m_i}, should be calculated as Equation (3.31), where C_{m_i} denotes the original transmission time and o_{m_i} denotes whether m_i is encrypted.

The encryption time of message m_i, denoted as e_{m_i}, is computed according to Equation (3.32). Similarly, the decryption time of message m_i, denoted as d_{m_i}, is computed according to Equation (3.33). The worst case execution time of decryption task τ^i_d , denoted as $c^i_{\tau_d}$, is calculated as (3.34).

$$c_{m_i} = (1 - o_{m_i})C_{m_i} + o_{m_i} \frac{\left\lceil \frac{C_{m_i} \cdot Bus\_speed}{B\_size} \right\rceil B\_size}{Bus\_speed} \tag{3.31}$$

$$e_{m_i} = \frac{C_{m_i} \cdot Bus\_speed}{En\_speed} \tag{3.32}$$

$$d_{m_i} = \frac{C_{m_i} \cdot Bus\_speed}{De\_speed} \tag{3.33}$$

$$c^i_{\tau_d} = o_{m_i} d_{m_i} \tag{3.34}$$

The execution time of sensing task τ^i_s, denoted as $c^i_{\tau_s}$, is elongated as Equation (3.35), where every encrypted message sent by sensor τ^i_s introduces its encryption overhead.

$$c^i_{\tau_s} = C^i_{\tau_s} + \sum_{\tau^i_s \in src(m_i)} o_{m_i} e_{m_i} \leq T^i_{\tau_s} \tag{3.35}$$

We use task set $\mathcal{T}_{\mathcal{CD}} = \mathcal{T}_{\mathcal{C}} \cup \mathcal{T}_{\mathcal{D}}$ to denote all the tasks allocated on the shared computation node, including control tasks and decryption tasks. For automotive systems, we assume fixed-priority preemptive scheduling. We assign priorities following: 1) the priorities of decryption tasks are higher than control tasks to guarantee the correct execution order; 2) for tasks with the same type, the ones with shorter periods are assigned with higher priorities following the commonly-used Rate-Monotonic scheduling [98].

We conduct response time analysis to check platform schedulability, using techniques similarly as in [34, 163]. Task response time denotes the longest time it may take to complete the task, and should be less or equal to task period. For system with preemptive fixed-priority scheduling, task response time contains the computation time requirement from the task itself and the interference from higher priority tasks. Specifically, the task response time $r^i_{\tau_c}$ of control task τ^i_c is shown below in Equation (3.36). The set $hp(\tau^i_c)$ contains all the higher priority tasks compared to task τ^i_c. The first term $C^i_{\tau_c}$ is the worst case execution time of task τ^i_c. The second term is the summation of all the time τ^i_c being preempted by higher priority tasks on the same computation unit. The constraint between task response time and task period is shown in Equation (3.37).

$$r^i_{\tau_c} = C^i_{\tau_c} + \sum_{\tau^k \in hp(\tau^i_c) \cap \mathcal{T}_{\mathcal{CD}}} \lceil \frac{r^i_{\tau_c}}{T^k_\tau} \rceil C^k_\tau \tag{3.36}$$

$$r^i_{\tau_c} \leq T^i_{\tau_c} \tag{3.37}$$

The task response time of decryption task τ^i_d is shown below. If message m_i is not encrypted, the corresponding decryption task τ^i_d should not exist, thus the task response time of task τ^i_d should be 0.

$$r^i_{\tau_d} = o_{m_i} \cdot \left[C^i_{\tau_d} + \sum_{\tau^k \in hp(\tau^i_d) \cap \mathcal{T}_{\mathcal{CD}}} \lceil \frac{r^i_{\tau_d}}{T^k_\tau} \rceil C^k_\tau \right] \leq T^i_{\tau_d} \tag{3.38}$$

For messages transmitted through the CAN or CAN-FD bus, non-preemptive fixed-priority scheduling is applied. The response time r_{m_i} of message m_i is shown in Equation (3.39), where c_{m_i} denotes the worst-case transmission time of message m_i. Because message transmitted on CAN bus is not preemptable, a message may have to wait for a blocking time B_{m_i}, which is calculated as $\max_{j \in lp(i)} c_{m_j}$, where $lp(i)$ is the set of all lower priority messages that are allocated on the same bus with m_i. Similarly, message m_i itself is not subject to preemption from higher priority messages therefore the inferences from higher priority messages can only occur within $r_{m_i} - c_{m_i}$ time intervals.

$$r_{m_i} = c_{m_i} + B_{m_i} + \sum_{m_k \in hp(m_i)} \lceil \frac{r_{m_i} - c_{m_i}}{T_{m_k}} \rceil c_{m_k} \leq T_{m_i} \tag{3.39}$$

Finally, the end-to-end latency is shown in Equation (3.40), where path p is represented by the link $s_i \to c_j \to a_k$. Message m_{s_i,c_j} is transmitted between sensing task τ^i_s

and control task τ_c^j, and decryption task τ_d^i for decrypting m_{s_i,c_j} may be added to the path. Message m_{c_j,a_k} is transmitted between control task τ_c^j and actuation task τ_a^k. Because of the asynchronous nature of the automotive embedded systems, in the worst case, when any task/message on the path completes its execution/transmission, the receiving message/task might have just been activated and will need to wait for the next activation to continue processing, where the wait time can be arbitrarily close to its period. For the sensing task, the arrival of the external event has the similar effect, i.e. it may just have missed the activation of the sensing task. Thus, in the worst case scenario, the periods of all the tasks and messages on the path should be added into the latency. For more detailed discussion (and the cases where such worst case bound can be reduced), please refer to [34].

$$\begin{aligned} l_{p(\tau_s^i \rightarrow \tau_c^j \rightarrow \tau_a^k)} &= c_{\tau_s}^i + T_{\tau_s}^i + r_{m_{s_i,c_j}} + T_{m_{s_i,c_j}} + r_{\tau_d}^i \\ &+ o_{m_i} T_{\tau_d}^i + r_{\tau_c}^j + T_{\tau_c}^j + r_{m_{c_j,a_k}} + T_{m_{c_j,a_k}} \\ &+ c_{\tau_a}^k + T_{\tau_a}^k \end{aligned} \tag{3.40}$$

Optimization with Simulated Annealing

The final optimization formulation for this automotive system is shown in below, refined from the general formulation in Section 3.3.1. where the computation of variables such as security level $S_{\tau_c}^i$, response time $r_{\tau_c}^i$ and end-to-end latency l_p can be referred to the formulations in the previous Section 3.3.2.

$$\text{maximize: } J = \frac{1}{|\mathcal{T}_\mathcal{C}|} \sum_{\tau_c^i \in \mathcal{T}_\mathcal{C}} \omega_{\tau_c}^i \alpha^{-\beta(T_{\tau_c}^i - T_{\tau_c}^{i0})} \quad \text{s.t.} \tag{3.41}$$

$$\forall \tau_c^i \in \mathcal{T}_\mathcal{C},\ S_{\tau_c}^i \geq S_0 \text{ (security constraint)} \tag{3.42}$$

$$\forall \tau_c^i \in \mathcal{T}_\mathcal{C},\ r_{\tau_c}^i / T_{\tau_c}^i \leq 1 \text{ (computational constraint)} \tag{3.43}$$

$$\forall \tau_s^i \in \mathcal{T}_\mathcal{S},\ c_{\tau_s}^i / T_{\tau_s}^i \leq 1 \text{ (computational constraint)} \tag{3.44}$$

$$\forall m_i \in \mathcal{M},\ r_{m_i} \leq T_{m_i} \text{ (communicational constraint)} \tag{3.45}$$

$$\forall p \in \mathcal{P},\ l_p \leq D_p \text{ (end-to-end latency)} \tag{3.46}$$

The above formulation is complex, and direct use of a generic non-linear solver may be intractable for industrial size problems. Instead, we implement a simulated annealing (SA) algorithm to explore acceptable feasible solutions.

The algorithm shown in Algorithm 1 is based on the standard simulated annealing procedure. We first set every message not to be encrypted and obtain the initial period by solving the problem without encryption (line 1). Then we start from an initial temperature $heat_0$ to iteratively search the design space until the number of iterations $nIter$ exceeds a preset limit $maxIter$ or the temperature falls below a preset final temperature $heat_{final}$. During each iteration, we randomly explore changes to the current solution $curSol$ by considering either 1) selecting a message and changing its encryption status $curSol.o_{m_i}$ or 2) selecting a control task and changing its period $curSol.T_{\tau_c}^i$ (line 5 to 8). We evaluate the cost of such changes $tmpCost$, which is based on the objective value obj, the penalty proportional to the number of schedulability violations $timeVio$ and to the number of security violations $secuVio$ (line 12). If the new cost is smaller than the previous minimum cost $minCost$, the new solution $tmpSol$ will be accepted immediately; otherwise it will be accepted with a transition probability $P = \exp^{\gamma(curCost - tmpCost)/heat}$, where γ is a parameter and $heat$ is the current temperature. After each iteration, $heat$ is lowered with a cooling factor $coolFactor$.

We properly tune the values of parameters α_1, α_2, α_3, δ, γ, and $coolFactor$ to improve the SA performance.

3.3.3 Automotive Case Study Results

To evaluate the effectiveness of our codesign methodology and its refinement in automotive domain, we conducted experiments for an industrial automotive system example and a set of synthetic examples. All experiments are run on an Intel Core i7 CPU with 12GB memory. The results are discussed in below.

Industrial Example

We first conduct a case study that is derived from a subsystem of an experimental vehicle with active safety functions, similarly as the one used in [34, 163]. The vehicle supports distributed functions with end-to-end computations collecting data from 360° sensors and sending commands to the actuators, consisting of the throttle, brake, and steering subsystems and of advanced HMI (Human-Machine Interface) devices. Examples of active safety functions include Adaptive Cruise Control (ACC), Lane Departure Warning or Lane Keeping Systems. These functions are deployed together in a car electronics system, sharing the sensing and actuation layers and possibly also intermediate processing stages, such

Algorithm 1: Simulated Annealing()

```
Input: task graph, task execution time, message length, message period, priority,
       key length, encryption_speed, bus_speed
Output: sampling period $\overrightarrow{T_{\tau_c}}$, encryption assignment $\overrightarrow{o_m}$, best control performance
        found
1  $curSol \leftarrow initial\_solution()$; $heat \leftarrow heat_0$; $nIter \leftarrow 0$;
2  while $nIter < maxIter \wedge heat > heat_{final}$ do
3      $nTry \leftarrow 0$;
4      while $nTry < maxTry$ do
5          $nTry{+}{+}$; $i \leftarrow randIdx$;
6          if $move = randomMove1$ then
7              $tmpSol.o_{m_i} \leftarrow !curSol.o_{m_i}$;
8          end
9          else if $move = randomMove2$ then
10             $tmpSol.T^i_{\tau_c} \leftarrow curSol.(T^i_{\tau_c} + \delta(r^i_{\tau_c} - T^i_{\tau_c}))$;
11         end
12         $tmpCost \leftarrow \alpha_1 \cdot obj^{-1} + \alpha_2 \cdot timeVio + \alpha_3 \cdot secuVio$;
13         if $tmpCost < minCost$ then
14             Accept $tmpSol$;
15         end
16         else if $randNum < e^{\gamma(curCost - tmpCost)/heat}$ then
17             Conditionally accept $tmpSol$;
18         end
19     end
20     $heat \leftarrow heat * coolFactor$; $nIter{+}{+}$;
21 end
```

as the sensor fusion and object detection functions or the actuator arbitration layers. The result is a complex graph of functions (programmed as tasks) with a high degree of communication dependency and deadlines on selected pairs of endpoints. In this case study, we select a subsystem of those functions, including their tasks and communication signals [8]. The example consists of 14 tasks (including 6 sensing tasks, 5 control tasks, and 4 actuation tasks), 17 messages from sensing tasks to control tasks, and 13 messages from control tasks to actuation tasks. As we explore the encryption of sensing messages, up to 17 additional encryption tasks may be added. The structure of the example is shown in Fig. 3.23. The task execution times are in the range of 0.2ms to 20ms, and the initial task periods are in the range of 10ms to 100ms. The message lengths are in the range of 1 to 64 bits, and the message periods are in the range of 10ms to 100ms. In our study, we derived system dynamics from two automotive examples in the Simulink library (these systems and their derivations are used for the control loops in the industrial example and the synthetic examples). The first system dynamics is linearized from Simulink Vehicle Suspension Model, and the equation is shown in (3.47). The second dynamics is linearized from Simulink Engine Speed model. The equation is shown in (3.48).

There are 5 control loops in the system (corresponding to 5 control tasks). The system control performance, calculated as in Section 3.3.2, is in the range of $[0, 1]$ (where 1 represents the best possible performance obtained without encryption). The α value in Equation (3.27) is set as the Euler's number e and β is set as 1.

[8] Addressing the entire system available requires models for more complex functional graph and multiple computation units (planned in the future work).

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -50.4 & -4.671 & 7.105 * 10^{-15} & -0.1429 \\ 0 & 0 & 0 & 1 \\ 7.105 * 10^{-15} & -0.25 & -81.67 & -7.5 \end{bmatrix} x(t)$$
$$+ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -21.6 & -1.929 & 24 & 2.143 & -28.8 & -2.743 & -24 & -2.286 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 42 & 3.75 & -46.67 & -4.167 & -42 & -4 & -35 & -3.333 \end{bmatrix} u(t)$$
$$+ w(t) \tag{3.47}$$

$$\dot{x}(t) = \begin{bmatrix} -7.146 & -0.02545 & 0 & 0 \\ 592.9 & 0.4642 & -2.323 * 10^{6} & 0 \\ 0.2043 & 0.0001861 & -400.1 & -5.335 * 10^{4} \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t)$$
$$+ \begin{bmatrix} 0 & 0.6771 \\ -7.143 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u(t) + w(t) \tag{3.48}$$

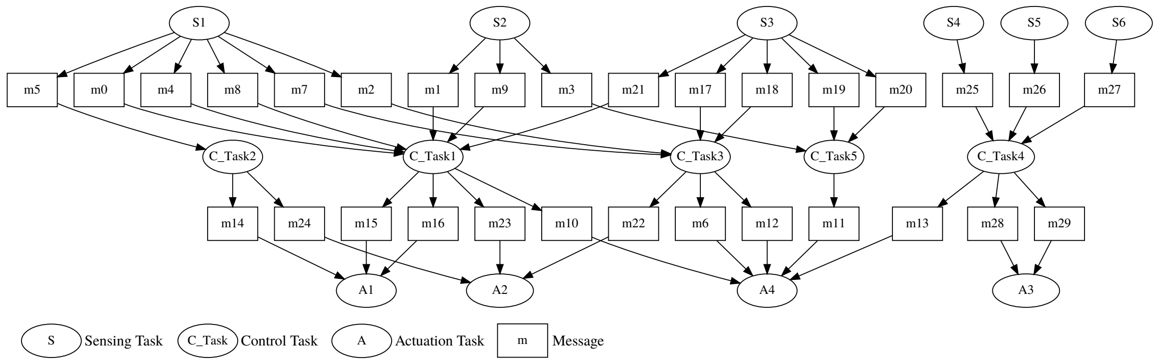


Figure 3.23: The modified automotive subsystem used in the case study.

The attacker conducts eavesdropping on 17 sensing messages. The security level of every control loop is measured by the complexity for the Kalman filter to retrieve information, and normalized to be within $[0, 1]$ as shown in Section 3.3.2, where 1 represents the best possible security level, i.e., every sensing message of the control loop is encrypted. The system security level is the minimum of all control loops' security level, and is within the range of $[0, 1]$. System security level reaches 1 when all 17 messages are encrypted, and is 0 when none is encrypted.

In our experiments, we explore the selection of messages for encryption and the assignment of control sampling periods, to address security level together with control performance while guaranteeing platform schedulability.

Trade off between control performance and system security level. As we explore the selection of sensing messages for encryption, the security levels of control loops are significantly affected. For instance, Table 3.1 shows the security level of control loop 3 (the control loop associated with control task C_Task 3 in Fig. 3.23) when different number of sensing messages is encrypted, as directly computed from Equation (3.30). The system dynamics with regard to control loop 3 is shown in Equation (3.47). The measurement noise is set to $0.01 * eye(4, 4)$.

We further conduct simulations to report the actual mean and variance of the Kalman filter performance, i.e., the attacker performance, as a function of the number of encrypted messages and key length (varied from 4 to 64). For each configuration for control loop 3, we run 1000 simulations to compute the mean and the variance of the Kalman filter performance, measured as the inverse of the trace of the Kalman covariance error.

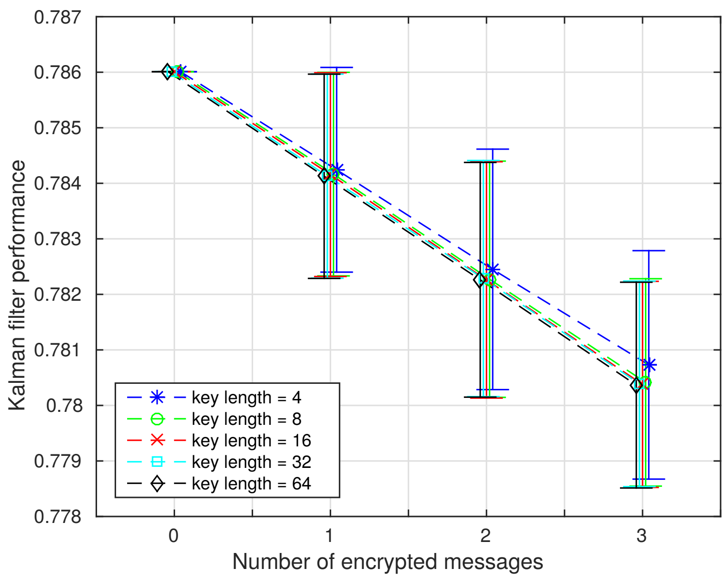


Figure 3.24: Mean and standard deviation of the Kalman filter performance under different number of encrypted messages and different key lengths (simulations of control loop 3).

As shown in Fig. 3.24, the means are noted with markers while the standard deviations (square roots of the variances) are shown as the vertical bars. We can see from the figure that as the number of encrypted messages increases or the key length increases, the attacker performance decreases, i.e., the control loop has a higher security level.

Table 3.1: Security level of control loop 3 with different number of encrypted messages.

# encr. msgs	0	1	2	3	4
security level	0	0.2902	0.4349	0.4371	1

On the other hand, encrypting and decrypting messages introduces significant timing overhead, which may cause the increase of control sampling periods due to schedulability constraints and thus reduce the control performance. Based on our formulation introduced

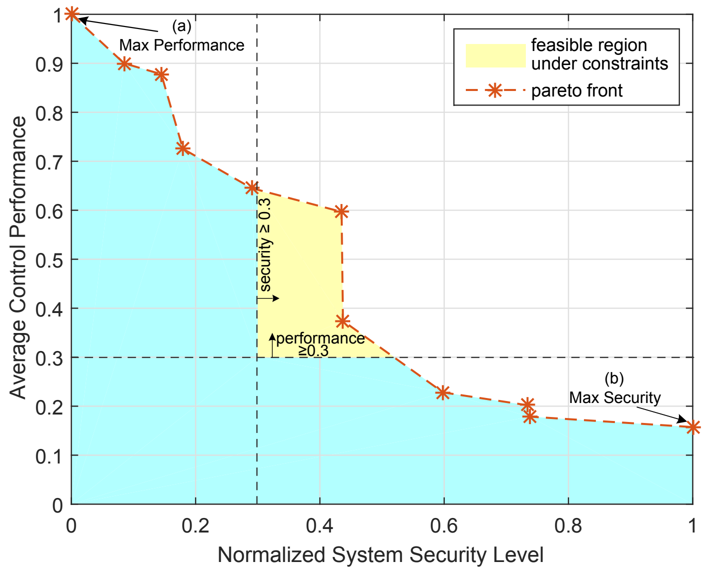


Figure 3.25: Pareto front between normalized control performance and security level for the industrial example. An example feasible region denotes all feasible solutions under requirement that control performance ≥ 0.3 and security level ≥ 0.3.

in Section 3.3.2 and using the simulated annealing algorithm shown in Section 3.3.2, we are able to explore the design space while quantitatively analyzing the trade-offs between control performance and security level. Fig. 3.25 shows the Pareto front between the two normalized metrics for the automotive case study. The relative noise of each sensing message (i.e., how much the measurement deviates from the true value) is set as 10% of the operation point – the noise impact on security will be discussed later in this section.

From Fig. 3.25, we can clearly see the trade-offs between control performance and security level. During design, constraints on these two metrics may be set according to system requirements. The Pareto front generated by our approach will provide a ***feasible region that is important for making decision choices***. For instance, an example feasible region is shown in the figure, under the requirements that the system control performance should be no less than 0.3 and the system security level should be no less than 0.3. Without our codesign approach, it is impossible to identify the feasible designs under such requirements. Instead, the designers might get a solution that violates security requirement if they only optimize for control performance (point (a) in the figure), or a solution that violates performance requirement if they simply choose to encrypt all messages (point (b)). This shows the importance of our codesign framework.

Impact of sensing noise on system security level and the number of encrypted messages. In our experiments, we observe that the noise on sensor measurements has a significant impact on system security level. Intuitively, as the attacker tries to reconstruct the system state from hacked sensing messages and un-encrypted messages, the lower the sensing noise, the easier it is for the attacker (thus the system is less secure). To quantitatively analyze this relation, we conduct a series of experiments: we set the sensing noise to different levels (measured by the relative error of the sensing measurements with respect to the control operating point (op)), and evaluate how many messages need to be encrypted for certain system security level requirement, while maximizing control performance.

A heat map demonstrating the relation among noise level, system security level, and the number of encrypted messages is shown in Fig. 3.26(a). We can clearly see the trend that when the noise level increases, we need fewer messages to be encrypted to reach certain security level. Fig 3.26(b) and Fig 3.26(c) further extract one horizontal line and one vertical line from the heat map, respectively. For (b), we can see that as the security level requirement increases, we need to encrypt more messages. More interestingly, for (c), we can see that as the noise level increases, we may encrypt fewer messages.

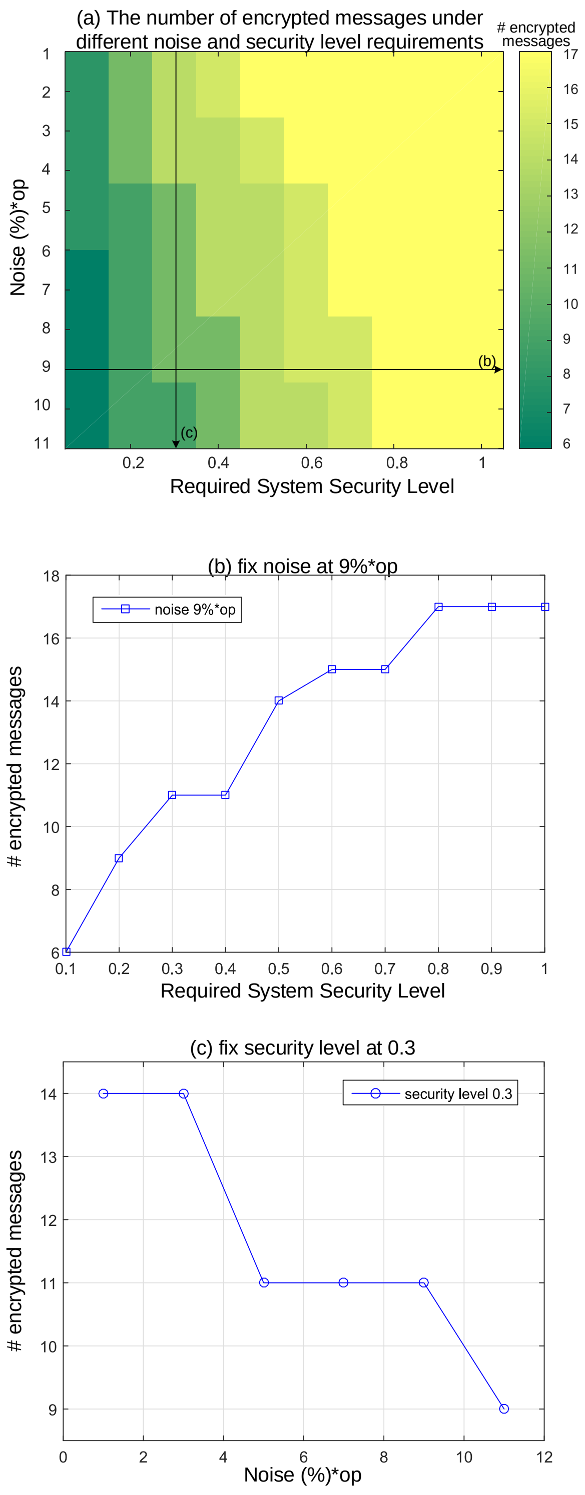


Figure 3.26: Impact of sensing noise on security level and encrypted messages for the industrial example: (a) The number of messages encrypted under different noise levels and different security requirements. (b) The noise level is fixed at 9%*op. (c) The security level set as 0.3.

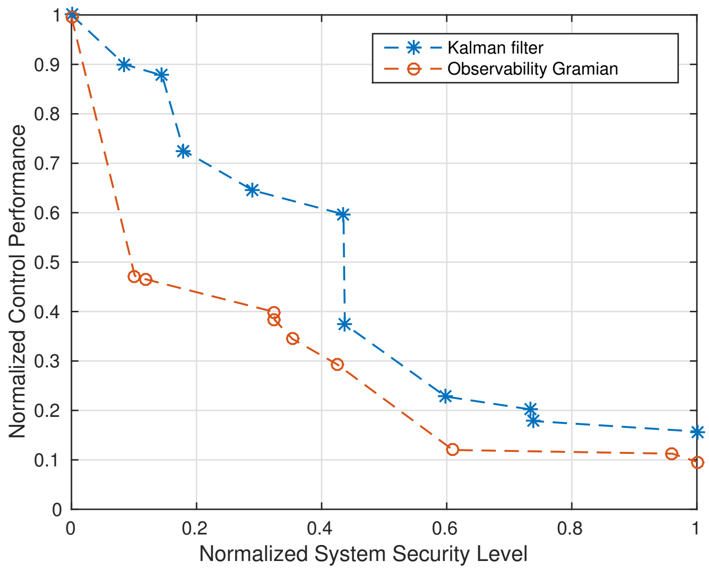


Figure 3.27: Pareto front between normalized control performance and security level using Observability Gramian and Kalman filter for the industrial example.

Security evaluation using Observability Gramian. As stated in Section 3.3.1, we may also use the *Observability Gramian* to measure the complexity for an attacker to estimate the system states, and further compute the system security level. In this experiment, we conduct experiments with Observability Gramian as the security level measurement, and evaluate its trade off with the control performance for the industrial example. Fig. 3.27 shows such trade off, along with the trade off using Kalman filter (which is the same as the one from Fig. 3.25). The two security level measurements, Observability Gramian and Kalman filter, are based on different perspectives and consequently provide different values. Despite this, the two Pareto fronts show similar trade off trend between control performance and security level. Furthermore, for automotive systems with measurement noise, we think Kalman filter based approach is a more suitable measurement with its consideration of measurement noises.

Synthetic Examples

For more comprehensive study of our codesign approach, we conduct a set of experiments with synthetic examples that have varying number of tasks, messages, execution times and periods.

We randomly generate task graphs that have the same structure as the system model in Fig. 3.19. Specifically, we first vary the number of control tasks from 5 to 25, and then randomly generate a number of sensing tasks and actuation tasks as well as their connections with the control tasks (the numbers of sensing tasks and actuation tasks are proportional to the number of control tasks in average). The period of each task is randomly generated between 10ms to 100ms, and the execution time is randomly generated within the period. When randomly generating the task periods and execution times, we keep the total utilization of the computation unit around 60% (before adding the decryption tasks). Each message may have multiple successive control tasks, and each message length is randomly generated between 1 to 32 bits. We set the message transmission speed at 1000 bit/ms, and message decryption speed at 250 bit/ms.

Similarly as for the industrial example, we evaluate the trade off between control performance and security level in codesign for various synthetic examples. Fig. 3.28 shows the Pareto fronts for synthetic examples with different number of control tasks, using Kalman filter as the security level measurement. Every point in this figure is the average of 10 randomly generated examples. We can see that the trade off between control performance and security level is similar to the industrial example. Furthermore, as the number of control tasks increases, the control performance decreases faster with respect to

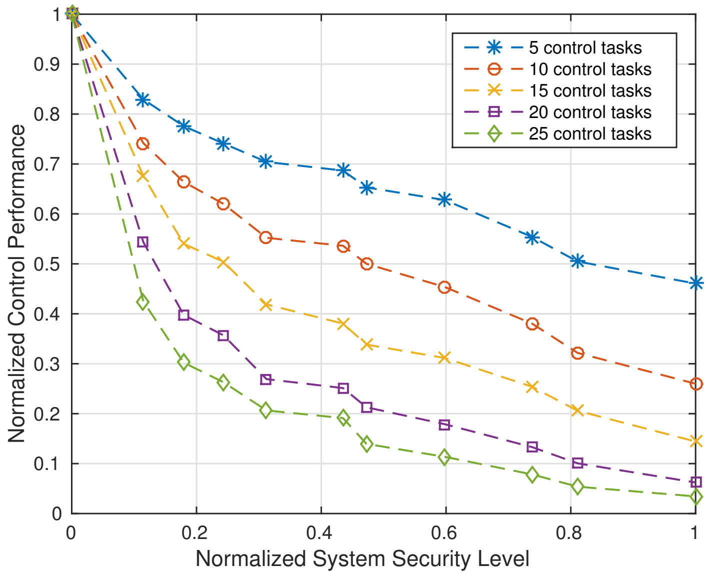


Figure 3.28: Pareto front between normalized control performance and security level for synthetic examples with different number of control tasks (using Kalman filter based security level measurement).

increasing security level (i.e., the Pareto front curve is lower as shown in the figure). This is because to achieve the same system security level, more messages need to be encrypted for a larger task set. This leads to more decryption tasks added to the computation unit, and consequently harder scheduling (even for similar level of utilization), and eventually longer periods for control tasks and worse control performance.

We also conduct experiments using Observability Gramian as the security level measurement for the synthetic examples. The results in Fig. 3.29 demonstrate the similar trend as using Kalman filter.

The runtime of our algorithm depends on the problem size (in particular the number of control tasks) and the tuning parameters in simulated annealing. In Table 3.2, we

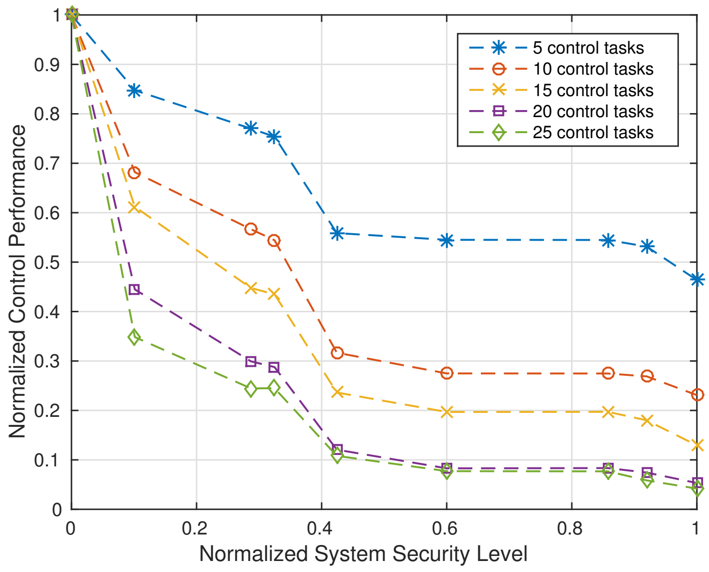


Figure 3.29: Pareto front between normalized control performance and security level for synthetic examples with different number of control tasks (using Observability Gramian based security level measurement).

record the runtime of our algorithm for the synthetic examples under different sizes of control task set and the same tuning parameters. The time we record is the average time for running the algorithm once, i.e., for obtaining one point in Fig. 3.28 and Fig. 3.29. As the number of control tasks increases, we let the numbers of messages, sensing tasks and actuation tasks increase proportionally. Note that the Observability Gramian and Kalman filter performance for each control loop under different number of encrypted sensing messages are calculated and stored in arrays before running the simulated annealing algorithm. The calculation time of both metrics is small compared to simulated annealing. Therefore, we only record the average runtime of the simulated annealing algorithm, which is almost the same for the two metrics.

Table 3.2: Algorithm runtime of simulated annealing under different number of control tasks

control tasks #	5	10	15	20	25
runtime (s)	43	126	181	482	745

3.4 Summary

We use three applications as case studies to demonstrate the effectiveness of CONVINCE to address the codesign of application and software layers.

The case study of CACC demonstrates the effectiveness of CONVINCE in analyzing the impact of security attacks in vehicular communication and ultimately the application performance.

The case study of the intelligent intersection management shows the quantitative analysis of communication delays in the framework. It presents a delay-tolerant intersection management protocol, and the modeling, simulating and verifying the safety, liveness and performance of the proposed protocol. Experiments demonstrate the effectiveness of both the proposed protocol and the framework.

By utilizing the framework, we identify the key interface variables in cyber-physical systems among security, performance and schedulability, in particular the sampling periods and the selection of sensing messages for encryption. We quantitatively model the relation between message encryption and system security level, and model the impact of message encryption and sampling periods on control performance and platform schedulability. The general framework is refined to automotive systems, and a simulated annealing algorithm is developed for exploring the design space based on the refined codesign formulation. An automotive case study and synthetic examples demonstrate the effectiveness of our approach.

Chapter 4

Cross-Layer Design of Software and Hardware Layers

4.1 Fault-Tolerance-Aware Mapping

As described in Section 1.2 in Chapter 1, we use the same categorization of online error detection techniques as the one in [51], namely embedded error detection (EED) and explicit output comparison (EOC). EED refers to the broad collection of error detection techniques that do not rely on redundant execution. EED typically has a performance overhead which is reflected as elongated execution time. EOC can take the form of the classic triple modular redundancy (TMR) architecture, or a scaled down version that executes the same program/task twice, and rely on re-executions in the event of an output mismatch [51].

Applying EOC or EED techniques and corresponding task re-executions improves the system's capability to tolerate soft errors, but also introduces significant timing overhead and may be detrimental to meeting real-time constraints. It is therefore crucial to ***quantitatively model the impact of various soft error tolerance mechanisms on timing constraints to ensure system safety***. In a real-time system, some critical tasks (e.g., feedback control tasks directly related to safety) may require EOC techniques for their higher error coverage, while other less critical tasks (e.g., multimedia tasks in a vehicle) may choose to employ EOC or EED or neither to trade off between error coverage and timing overhead. Quantitative analysis is needed to ensure that after applying error detection and recovery mechanisms, the tasks and their copies and re-executions can be completed within the original task deadlines, especially for critical tasks (deadlines may be relaxed for less critical tasks [83, 82]). Based on such analysis, task scheduling and allocation and in some cases the choice of architecture platform may be explored to find the feasible solutions that meet both timing and fault tolerance requirements (e.g., some tasks may require EOC). Furthermore, optimization may be conducted to select the proper error tolerance mechanism for each of those tasks that have multiple options (and are not enforced to select one of them), to maximize error coverage at the system level while meeting all requirements.

In comparison to previous works, our approach formulates the impact on system timing for different error tolerance mechanisms including both EOC and EED based techniques, and optimizes the task-level selections of tolerance mechanisms, for various fault models and task execution models on representative single-core, multicore and distributed platforms.

4.1.1 System Model

Function, Architecture and Timing

A real-time embedded application is captured by a functional model similarly as in [34, 163]. The functional model is represented by a task graph $\mathcal{G} = (\mathcal{T}, \mathcal{S})$, where $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is a set of tasks and $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ is a set of signals exchanged among tasks. Each task is activated repeatedly with a period T_{τ_i}, and each signal is activated with a period T_{s_i}. A path p is an ordered interleaving sequence of tasks and signals that contribute to a function (e.g., from the collection of sensor data at the first task to the actuation at the last task), denoted as $p = [\tau_{k_1}, s_{k_1}, \tau_{k_2}, s_{k_2}, \ldots, s_{k_{p-1}}, \tau_{k_p}]$.

During implementation, the embedded application may be mapped onto single-core, multicore, or distributed platforms. In this work, we first address platforms with one single-core CPU. The tasks running on the single core are scheduled with preemptions according to static priorities, which is common in real-time systems supported by standards such as OSEK [113]. The worst case execution time (WCET) of task τ_i is denoted by C_{τ_i}. The analysis for single-core platform sets the foundation of our work by modeling the timing impact of error tolerance mechanisms and exploring task-level mechanism selection and priority assignment.

We then address the platforms with one multicore CPU, and consider two task execution models. In *task duplication model*, each task is duplicated on two homogeneous cores and the two copies are executed at the same pace. In *task concentration model*, each task and its re-executions are allocated to the same core. The task scheduling on each core is independent in this case (i.e., partitioned scheduling model). We explore the allocation of tasks to cores, together with error tolerance mechanism selection and priority assignment for each task.

Finally, we consider the problem for the distributed platforms in which a set of single-core CPUs are connected through a single communication bus. In particular, we address the widely-used CAN (Controller Area Network) bus protocol, which features non-preemptive static priority based message scheduling. We explore the allocation of tasks to CPUs, error tolerance mechanisms for tasks, priority assignment for tasks on the same CPU and for messages on the CAN bus.

The distributed platform case shares some similarities with the multicore case but also has two major differences: First, while the communication within a CPU is conducted through shared memory, the communication on the distributed platform is carried out through bus messages (by mapping signals to messages) and requires explicit modeling of message delays that are affected by message priority assignment. Second, the task duplication model for the multicore case is not considered for the distributed platform case because the CPUs are assumed as asynchronous. Our formulations and algorithms may also be extended to address the distributed platforms in which a set of multicore CPUs are connected through one or multiple buses, with a more complex communication model.

Task timing model. During task execution, timing constraints enforce that the tasks should be completed before their deadlines (assumed as task periods in our experiments). The worst-case response time r_{τ_i} represents the longest time it may take to complete task τ_i, and can be formulated in Equation (4.1), similarly as in [60, 163]. It includes the WCET C_{τ_i} and the time it takes to wait for the preemptions from higher priority tasks on the same core. $hp(\tau_i)$ denotes the set of higher priority tasks which can preempt the execution of lower priority tasks.

$$r_{\tau_i} = C_{\tau_i} + \sum_{\tau_j \in hp(\tau_i)} \lceil \frac{r_{\tau_i}}{T_{\tau_j}} \rceil C_{\tau_j} \tag{4.1}$$

<u>*Message timing model.*</u> For the distributed platform case only, we need to consider message delays. As stated, we assume non-preemptive scheduling based on static priorities for messages transmitted over a CAN bus. In this work, we assume each signal is mapped to is own message, so in the following sections we use s_i to denote both the signal s_i and its corresponding message. The worst-case response time r_{s_i} of message s_i is shown in Equation (4.2), where C_{s_i} denotes the worst-case transmission time of message s_i and B_{max} denotes the maximum blocking time (approximated as the longest transmission time of any message in the system). The summation term denotes the time waiting for higher priority messages before transmission (not during transmission as the bus is non-preemptive).

$$r_{s_i} = C_{s_i} + B_{max} + \sum_{s_j \in hp(s_i)} \lceil \frac{r_{s_i} - C_{s_i}}{T_{s_j}} \rceil C_{s_j} \tag{4.2}$$

<u>*End-to-end path latency model.*</u> Deadlines may be imposed on selected paths as application requirements. The *worst-case* end-to-end timing latency incurred when traveling a path p is denoted as l_p, which represents the largest possible time interval for an input value change at the first task to be propagated and cause an output value change at the last task. A deadline D_p may be imposed as the upper bound of l_p. As shown below in Equation (4.3), the computation of l_p includes the worst-case response times and the periods of all the tasks and signals on the path. The periods need to be included because of the asynchronous sampling nature of the communication data (more details can be found in [34]). Note that

for communicating tasks with harmonic periods on the same CPU, the analysis may be less pessimistic if we assume the designers can select the relative activation phases of tasks, as explained in [39, 34][1].

$$l_p = \sum_{\tau_i \in p} (r_{\tau_i} + T_{\tau_i}) + \sum_{s_i \in p} (r_{s_i} + T_{s_i}) \tag{4.3}$$

For single-core and multicore platforms, signals are exchanged through shared memory and the worst-case response times r_{s_i} correspond to memory access latencies (modeled as a small constant in this work). For distributed platforms, the signals between tasks on the same CPU are also exchanged through shared memory and associated with memory access latencies. The signals between tasks on different CPUs are transmitted on the bus through messages, and their worst case response times r_{s_i} are modeled as in Equation (4.2).

Error Model and Detection

We use K to denote the number of errors that occur within the hyperperiod T_{hyper} of the task set, which is the least common multiple of the task periods in the task set. The goal of our formulations is to 1) check whether certain error tolerance mechanisms can be applied to a set of tasks without violating timing constraints in the existence of K errors during hyperperiod, and 2) explore task scheduling and allocation as well as error tolerance mechanisms for some tasks (for which multiple options exist) to find feasible solutions and in some cases optimize the coverage of these K errors (the error coverage is defined later).

We consider two error detection mechanisms, EED and EOC. For task τ_i, we use α_{τ_i} to denote the error detection rate (probability) when EED is used, and use β_{τ_i} to

[1] In those cases, depending on whether it is oversampling or undersampling, some periods may not need to be included. For simplicity, we did not consider those in this work.

denote the error detection rate when EOC is used. For EED, even when the state-of-the-art CFC techniques can detect almost all crashes and control flow violations, these errors are only around 70% of the total errors that could occur during execution [119, 149]. In other words, EED cannot guarantee the detection of all possible errors. EOC can largely eliminate this concern and achieve close to 100% detection rate in many cases, but it will incur at least 100% performance overhead (when using temporal redundancy) or resource overhead (when using spatial redundancy), plus the additional time to compare the outputs as shown in [51]. In general purpose computing, EOC may be realized through redundant multithreading [150, 52] with less performance penalty, however such techniques are hard to adopt for most embedded architectures and hence are not considered in our model.

When exploring the usage of EED and EOC for real-time system in this work, we assume that: 1) The necessary support for EED and EOC are available, including source code modifications, compiler improvements and in some cases additional hardware (e.g., lightweight checkers/comparators). 2) The inputs of each task instance are stored for redundant execution and potential re-executions within the task period. Two executions of the same task instance will produce the same outputs if no error occurs in either of them.

Error Coverage and its Approximation

To measure the overall error resiliency of a designed real-time system that employs EED or EOC or no detection mechanism for each of its tasks, we define a system-level *error coverage* metric as: the *probability* that all errors are either 1) detected and recovered within the hyperperiod by EED or EOC while all timing constraints are met, or 2) regarded as covered when they occur during the idle time. When the system is idle (i.e., no task is running on the core), it is still susceptible to cosmic ray strikes and memory errors etc., but we assume the probability that the program outputs are affected is negligible[2]. In other words, the errors occurred during idle time are assumed as "covered".

The precise evaluation of the system error coverage metric requires detailed information of the error occurrence profile (i.e., types of errors, when the errors occur and on which cores the errors occur) and detailed analysis of the timing schedulability, and in general cannot be captured in closed-form formulation. Therefore, to explore the design space in our work, we derive an *approximated error coverage formulation* as the optimization objective. Initial approximation is introduced in below, and further approximation to linear formulations is introduced later for corresponding problems.

First, within the hyperperiod, we let t_{eoc}, t_{eed} and t_{none} denote the accumulative time spent (including re-executions) by tasks using EOC, EED and no error detection, respectively. t_{idle} denotes the idle time. $T_{hyper} = t_{eoc} + t_{eed} + t_{none} + t_{idle}$. Assuming K soft errors of arbitrary types occur within the hyperperiod following uniform distribution on a single-core platform *and* assuming the timing constraints can be met, the error coverage P can be approximated as:

$$P \approx \sum_{i=0}^{K} \sum_{j=0}^{i} \binom{K}{i} \binom{i}{j} (\frac{\alpha \cdot t_{eed}}{T_{hyper}})^{j} (\frac{\beta \cdot t_{eoc}}{T_{hyper}})^{i-j} (\frac{t_{idle}}{T_{hyper}})^{K-i} \tag{4.4}$$

where α and β are the average probabilities that an error can be detected by EED and by EOC, respectively. The formulation essentially calculates the probability that all K errors are either detected and recovered by EED or EOC, or occurred during idle time. Note that the errors occurred during idle time are assumed as "covered".

[2]Memory is typically well protected. Our formulation can also be extended to address idle-time errors.

Equation (4.4) needs to be further approximated and adjusted (for multicore cases) in the following sections to linear formulations for our MILP-based exploration and optimization. Then, after the solutions are identified in solving the MILP formulations, we use our Monte Carlo based simulation engine introduced in Section 4.1.5 to more accurately evaluate their system error coverage. The simulator addresses the factors that are not considered in our approximated objective, including the timing constraints, the task re-execution time, and the existence of multiple errors. In Section 4.1.6, our experiments demonstrate that our work provides an *effective approximation* as optimization objective.

4.1.2 Single-core CPU Platform

We use $Cdec_{\tau_i}$ to denote the *execution time with detection* for task τ_i, which includes WCET C_{τ_i} and the overhead for EED or EOC error detection but not the error recovery time. On a single-core CPU, EOC needs to run a task twice to determine whether any error has occurred – if the outputs of the two runs are different, error(s) are assumed to have occurred. Therefore when EOC is used for τ_i, $Cdec_{\tau_i} = 2C_{\tau_i} + \Lambda_{\tau_i}$, where Λ_{τ_i} is the time for comparing the outputs of the two runs and typically much smaller than C_{τ_i}. EED only needs to run a task once with built-in error detection techniques. Therefore when EED is used for τ_i, $Cdec_{\tau_i} = C_{\tau_i} + \Delta C_{\tau_i}$, where ΔC_{τ_i} is the timing overhead for EED detection.

$Crec_{\tau_i}$ denotes the *error recovery time* for τ_i. On a single-core CPU, we assume that a re-execution of τ_i is scheduled immediately after an error(s) is detected. When EED is used for τ_i, $Crec_{\tau_i} = C_{\tau_i} + \Delta C_{\tau_i}$. When EOC is used for τ_i, $Crec_{\tau_i} = C_{\tau_i}$ if there is only one error in the system, and $Crec_{\tau_i} = C_{\tau_i} + \Lambda_{\tau_i}$ if there are multiple errors.

Illustrating Example

	C	T
Task1	0.2	1
Task2	0.45	2

EOC	Λ	Cdec	Crec
Task1	0.02	0.42	0.2
Task2	0.05	0.95	0.45

EED	ΔC	Cdec	Crec
Task1	0.1	0.3	0.3
Task2	0.15	0.6	0.6

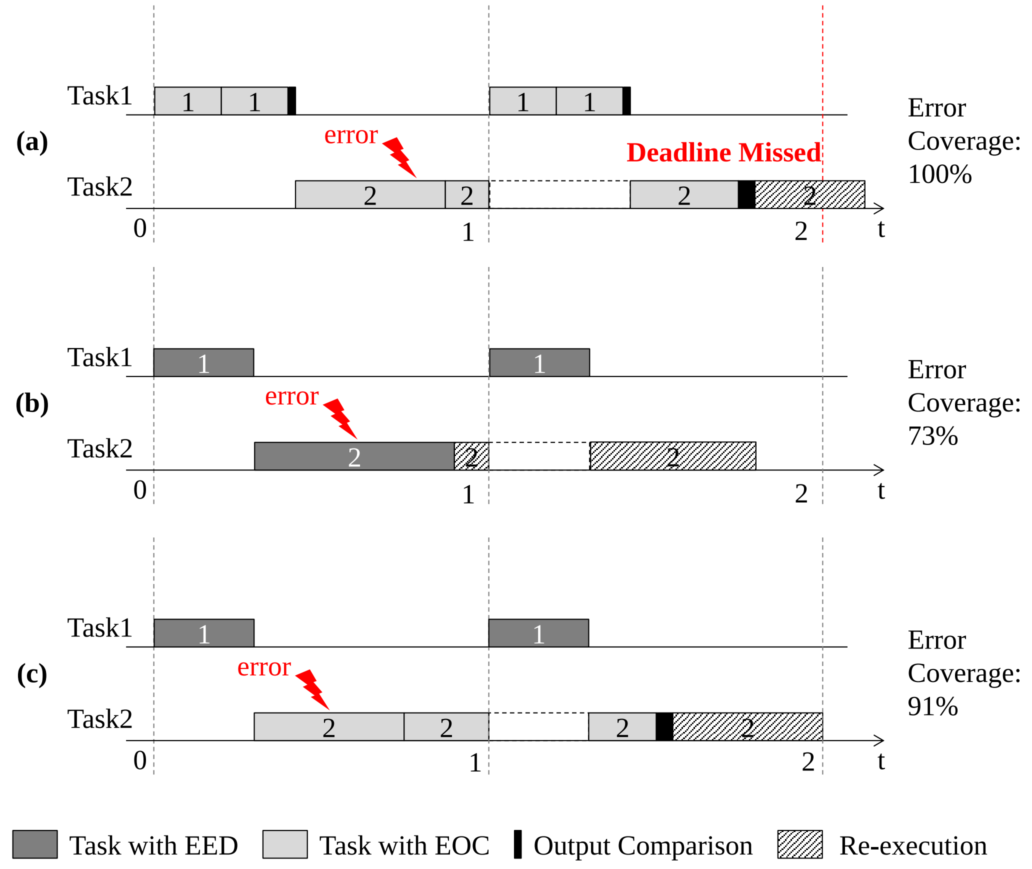


Figure 4.1: A fault-tolerant design example for single-core CPU platform: (a) only EOC is used, (b) only EED is used, (c) EOC/EED selection.

Fig. 4.1 shows two tasks running on a single-core CPU. The task characteristics are shown at the top, including the original WCET C, the task period T, the execution time with detection $Cdec$, and the error recovery time $Crec$. We assume EED error detection rate $\alpha_{\tau_i} = 70\%$ and EOC detection rate $\beta\tau_i = 100\%$ (The EOC error detection rate depends on its implementation. If all output bits are compared, EOC has near perfect (100%) detection rate as assumed in [51]. We use 100% for experiments but our formulations are general). Task1 has higher priority than Task2. We assume one soft error of arbitrary type occurs at an arbitrary time within the hyperperiod. For simplicity, we consider a solution as infeasible if timing constraints are violated. For feasible solutions, we compare them using the approximated error coverage in Equation (4.4) (as stated before, our final error coverage evaluation of the solutions incorporate timing violations through simulations).

In (a), we deploy EOC for both tasks for higher error coverage, however this solution is infeasible since it causes deadline miss when an error occurs in Task2. To satisfy timing constraints, solution (b) weakens the error detection strength and deploys EED for both tasks. The approximated error coverage of this solution is 73% based on Equation (4.4) and the schedule can tolerate any single detected error. Solution (c) uses a hybrid solution: EED for Task1 and EOC for Task2. It can still tolerate any single detected error, while achieving a higher error coverage of 91%.

One-Error Formulation

The illustrating example above demonstrates the importance to quantitatively model and explore the selection between EOC and EED. In below, we present an MILP formulation to explore the EOC/EED selection and task scheduling (through priority assignment) on a single-core CPU platform, with respect to an approximated error coverage objective and timing constraints. We first consider the case in which one error occurs within the hyperperiod (i.e., $K = 1$), and explore designs that will guarantee to *satisfy timing constraints when the number of errors within hyperperiod is not more than 1.*

We use Boolean variables ρ_{τ_i} and o_{τ_i} to represent the selection of error detection mechanism for task τ_i. ρ_{τ_i} is 1 if either EOC or EED is used for τ_i, and 0 if neither is used. o_{τ_i} is 1 if EOC is chosen, and 0 if EED is chosen. We use p_{τ_i,τ_j} to denote whether τ_j has higher priority than τ_i (1 if τ_j has higher priority and 0 otherwise). The MILP formulation includes the following aspects.

Execution time with detection and Error recovery time. As derived before, the execution time with detection $Cdec_{\tau_i}$ is $2C_{\tau_i} + \Lambda_{\tau_i}$ for EOC, $C_{\tau_i} + \Delta C_{\tau_i}$ for EED, and C_{τ_i} if neither is chosen. Incorporating the Boolean variables, we have the following equation for $Cdec_{\tau_i}$.

$$Cdec_{\tau_i} = C_{\tau_i} + \left[o_{\tau_i}(C_{\tau_i} + \Lambda_{\tau_i}) + (1 - o_{\tau_i})\Delta C_{\tau_i}\right]\rho_{\tau_i} \tag{4.5}$$

Similarly, the equation below models error recovery time.

$$Crec_{\tau_i} = (C_{\tau_i} + (1 - o_{\tau_i})\Delta C_{\tau_i})\rho_{\tau_i} \tag{4.6}$$

The nonlinear binary multiplication $o_{\tau_i}\rho_{\tau_i}$ in Equation (4.5) and (4.6) can be linearized by introducing a new variable b_{τ_i} with constraints $\rho_{\tau_i} + o_{\tau_i} - 1 \leq b_{\tau_i}$, $b_{\tau_i} \leq \rho_{\tau_i}$ and $b_{\tau_i} \leq o_{\tau_i}$.

Worst-case fault-tolerant task response time. A key aspect in our formulation is to extend the worst-case response time formulation in Equation (4.1) to include the consideration of fault-tolerant techniques. We use r_{τ_i,τ_j} to denote the worst-case response time for task τ_i if an error occurs during task τ_j's execution and leads to its re-execution. If τ_j has higher priority than τ_i, r_{τ_i,τ_j} will be affected by τ_j's recovery time $Crec_{\tau_j}$, as shown below.

$$r_{\tau_i,\tau_j} = Cdec_{\tau_i} + Crec_{\tau_j}p_{\tau_i,\tau_j} + \sum_{\substack{\tau_k \in \mathcal{T} \\ \wedge \tau_i \neq \tau_k}} \lceil \frac{r_{\tau_i,\tau_j}}{T_{\tau_k}} \rceil Cdec_{\tau_k}p_{\tau_i,\tau_k}$$

Then at the system level, the worst-case fault-tolerant response time r_{τ_i} is determined by the higher priority task that has the largest $Crec_{\tau_j}$, as shown below in Equation (4.7).

$$\begin{aligned} r_{\tau_i} = &Cdec_{\tau_i} + \underset{\tau_j \in \mathcal{T}}{max} \left\{ Crec_{\tau_j} p_{\tau_i,\tau_j} \right\} \\ &+ \sum_{\substack{\tau_k \in \mathcal{T} \\ \wedge \tau_i \neq \tau_k}} \lceil \frac{r_{\tau_i}}{T_{\tau_k}} \rceil Cdec_{\tau_k} p_{\tau_i,\tau_k} \end{aligned} \tag{4.7}$$

The *max* function can be linearized with a new variable m and a set of constraints $\forall \tau_j, m \geq Crec_{\tau_j} p_{\tau_i,\tau_j}$. The ceiling function $\lceil r_{\tau_i}/T_{\tau_k} \rceil$ can be linearized with an integer variable x_{τ_i,τ_k} and constraint $0 \leq x_{\tau_i,\tau_k} - r_{\tau_i}/T_{\tau_k} < 1$. Then integer-binary multiplication such as $x_{\tau_i,\tau_k} p_{\tau_i,\tau_k}$ can be linearized with a new variable y_{τ_i,τ_k} using the "big M" formulation as in [163]: $x_{\tau_i,\tau_k} - M(1 - p_{\tau_i,\tau_k}) \leq y_{\tau_i,\tau_k}$, $y_{\tau_i,\tau_k} \leq x_{\tau_i,\tau_k}$, and $y_{\tau_i,\tau_k} \leq M \times p_{\tau_i,\tau_k}$, where M is a very large constant.

The timing constraints enforce that the worst-case response time of each task should be within its period, and the end-to-end latency of each selected path (whose computation follows Equation (4.3)) should be within its deadline.

$$\forall \tau_i \qquad r_{\tau_i} \leq T_{\tau_i} \tag{4.8}$$

$$\forall p \qquad l_p \leq D_p \tag{4.9}$$

Optimization objective. To optimize the system error coverage, we further approximate the Equation (4.4) defined in Section 4.1.1 with a linear formulation in below by 1) setting $K = 1$, and 2) assuming total task re-execution time is negligible compared to total time for regular executions (this is typically true when K is small).

$$
\begin{aligned}
P &\approx \sum_{i=0}^{K}\sum_{j=0}^{i}\binom{K}{i}\binom{i}{j}(\frac{\alpha\cdot t_{eed}}{T_{hyper}})^{j}(\frac{\beta\cdot t_{eoc}}{T_{hyper}})^{i-j}(\frac{t_{idle}}{T_{hyper}})^{K-i}\\
&= \frac{\alpha\cdot t_{eed}+\beta\cdot t_{eoc}+t_{idle}}{T_{hyper}}\\
&= 1-\frac{(1-\alpha)t_{eed}+(1-\beta)t_{eoc}+t_{none}}{T_{hyper}}\\
&\approx 1-\frac{\sum_{\tau_i\in\mathcal{T}}(1-\varepsilon_{\tau_i})Cdec_{\tau_i}(T_{hyper}/T_{\tau_i})}{T_{hyper}}\\
&= 1-\sum_{\tau_i\in\mathcal{T}}(1-\varepsilon_{\tau_i})\frac{Cdec_{\tau_i}}{T_{\tau_i}} \qquad (4.10)
\end{aligned}
$$

where ε_{τ_i} is the error detection rate for task τ_i that depends on the choice of error detection mechanisms:

$$
\varepsilon_{\tau_i}=\begin{cases}0 & \text{no detection is used}\\ \alpha_{\tau_i} & \text{EED is used}\\ \beta_{\tau_i} & \text{EOC is used}\end{cases} \qquad (4.11)
$$

Note that from line 1 to line 2 in the derivation, we set $K = 1$. From line 2 to line 3, we use $T_{hyper} = t_{eoc} + t_{eed} + t_{none} + t_{idle}$. From line 3 to line 4, we refine the α and β at the system level to ε_{τ_i} at the task level, and use the assumption that task re-execution time is negligible.

We use Equation (4.10) as the objective function to optimize system error coverage, by exploring task scheduling and error detection mechanism selection among all feasible solutions that satisfy timing and fault tolerance requirements.

K-Error Formulation ($K > 1$)

For single-core CPU platforms, K-error formulation ($K > 1$) is a simple extension of the one-error case. We define the following formulation to explore designs that guarantee to *meet the timing constraints when the number of errors within hyperperiod is not more than K*.

Since the K errors can occur repeatedly on one task, the error recovery time $Crec_{\tau_i}$ should be modified as below in Equation (4.12) (for those tasks that do not use any fault-tolerant technique, the recovery time is still 0).

$$Crec_{\tau_i} = (C_{\tau_i} + o_{\tau_i}\Lambda_{\tau_i} + (1 - o_{\tau_i})\Delta C_{\tau_i})\rho_{\tau_i} \tag{4.12}$$

The worst-case fault-tolerant response time for task τ_i happens when all the K errors are located at the higher priority task that has the longest recovery time. So the worst-case fault-tolerant task response time for K errors is formulated in Equation (4.13).

$$\begin{aligned} r_{\tau_i} = {} & Cdec_{\tau_i} + K \underset{\tau_j \in \mathcal{T}}{max} \left\{ Crec_{\tau_j} p_{\tau_i,\tau_j} \right\} \\ & + \sum_{\substack{\tau_k \in \mathcal{T} \\ \wedge \tau_i \neq \tau_k}} \lceil \frac{r_{\tau_i}}{T_{\tau_k}} \rceil Cdec_{\tau_k} p_{\tau_i,\tau_k} \end{aligned} \tag{4.13}$$

We still use the same optimization objective as in the one-error case for approximation in the multi-error case, based on the observation that in practice K is usually very small during the hyperperiod and the amount of time spent on re-execution is also small compared to regular executions. Other constraints and their linearization are similar to the one-error case.

4.1.3 Multicore CPU Platform

We consider two task execution models for a single homogeneous multicore CPU platform. In task duplication model, two copies of the same task are allocated onto two homogeneous cores and executed at the same pace, as a form of spatial redundancy. If re-execution is needed for error recovery, both cores need to rerun the same task to ensure their timing are exactly the same. For this model, we only address dual-core CPU and single error within the hyperperiod in this work. In task concentration model, each task and all its re-executions can only happen at the same core. For this model, we address any number of cores and any number of errors within the hyperperiod.

Note that memory contention may occur when multiple cores access the shared memory at the same time, and the worst case execution time of the interfered tasks may be elongated (compared with single-core case) with memory access delays [157, 81]. In the following, we still use C_{τ_i} to denote the worst case execution time of task τ_i and assume the consideration of potential memory access delays is included (e.g., by using the techniques from [157, 81]).

Task Duplication Model under One Error

In the task duplication model, two copies of the same task are allocated onto two cores and executed at the same pace, as a form of spatial redundancy. If re-execution is needed for error recovery, both cores need to rerun the same task to ensure their timing are exactly the same. For this model, we only address dual-core CPU and single error within the hyperperiod in this work. Fig. 4.2 shows an example of task duplication model on a dual-core CPU platform where the two cores always execute the copies of the same task at the same pace. For EOC, the execution time with detection is reduced because of the spatial redundancy. For EED, no recovery time is needed since at least one of the two copies will produce correct results under a single error model[3].

	C	T
Task1	0.5	1
Task2	0.5	3

	Tech	Cdec	Crec
Task1	EED	0.53	0
Task2	EOC	0.55	0.5

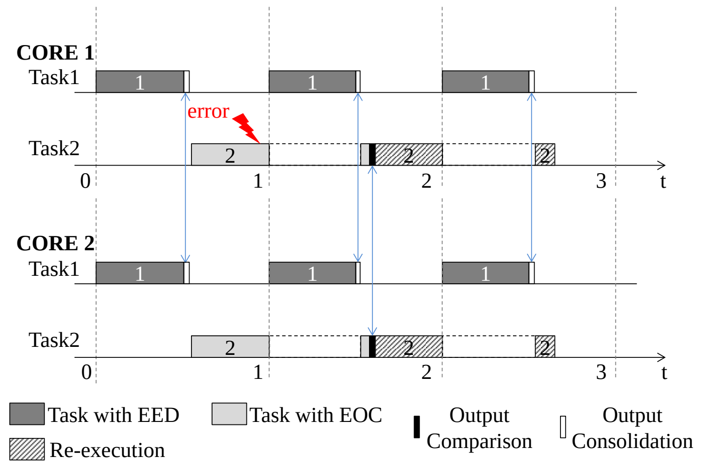


Figure 4.2: Task duplication model on a dual-core CPU

<u>*Execution time with detection and Error recovery time.*</u> Because of the spatial redundancy, the execution time with detection for EOC is $C_{\tau_i} + \Lambda_{\tau_i}$. The execution time with detection for EED is $C_{\tau_i} + \Delta C_{\tau_i} + \Gamma_{\tau_i}$, where Γ_{τ_i} is the consolidation time similarly as in [51]. The MILP formulation with the selection Boolean variable is as follows.

$$Cdec_{\tau_i} = C_{\tau_i} + (o_{\tau_i}\Lambda_{\tau_i} + (1 - o_{\tau_i})(\Delta C_{\tau_i} + \Gamma_{\tau_i}))\rho_{\tau_i} \tag{4.14}$$

[3]We may also compare the outputs of two EED copies as in EOC to increase the error detection coverage, but this may lead to re-execution if the correct copy cannot be identified. We plan to model this hybrid technique in future work.

The error recovery time for EOC is the original WCET, while the error recovery time for EED is 0. The MILP formulation is as follows.

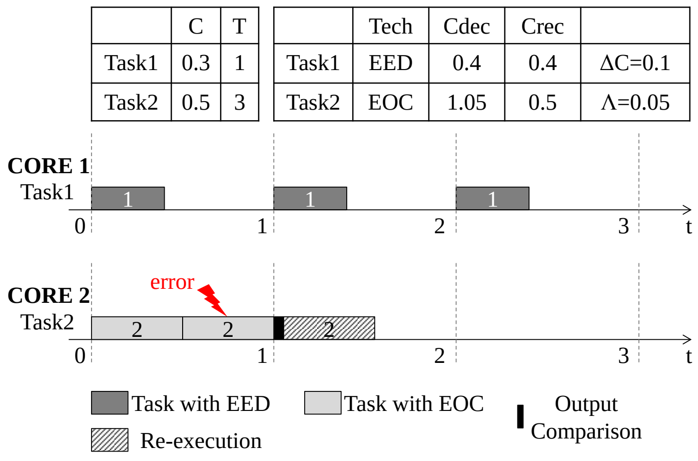
$$Crec_{\tau_i} = C_{\tau_i} o_{\tau_i} \rho_{\tau_i} \tag{4.15}$$

The linearization and other parts of the MILP formulation including the optimization objective are similar to the single-core CPU case in Section 4.1.2. The formulation explores designs that guarantee timing feasibility when the number of errors across two cores within hyperperiod is not more than 1.

Task Concentration Model

In the task concentration model, each task and its re-executions can only happen on the same core. Fig. 4.3 shows an example of task concentration model on a dual-core CPU platform. For this model, we address any number of cores and any number of K errors within the hyperperiod. We explore static allocation of tasks to cores together with error detection mechanism selection and priority assignment, to optimize error coverage while guaranteeing timing feasibility when the number of errors across all cores within hyperperiod is within K.

Task allocation. We use $\Phi = \{\phi_1, \phi_2, \ldots, \phi_n\}$ to represent a set of homogeneous cores (our formulation can be easily extended to heterogeneous cores for which we use C_{τ_i,ϕ_j} to denote the core specific WCET). Boolean variable a_{τ_i,ϕ_k} is 1 if task τ_i is allocated on core ϕ_k, and 0 otherwise. Boolean variable h_{τ_i,τ_j,ϕ_k} is 1 if task τ_i and τ_j are allocated on the same core ϕ_k, and 0 otherwise. Constraint (4.16) ensures that each task is only allocated onto one



	C	T
Task1	0.3	1
Task2	0.5	3

	Tech	Cdec	Crec	
Task1	EED	0.4	0.4	ΔC=0.1
Task2	EOC	1.05	0.5	Λ=0.05

Figure 4.3: Task concentration model on a dual-core CPU

core. Constraints (4.17) to (4.19) enforce the value for h_{τ_i,τ_j,ϕ_k}.

$$\sum_{\phi_k \in \Phi} a_{\tau_i,\phi_k} = 1 \tag{4.16}$$

$$a_{\tau_i,\phi_k} + a_{\tau_j,\phi_k} - 1 \leq h_{\tau_i,\tau_j,\phi_k} \tag{4.17}$$

$$h_{\tau_i,\tau_j,\phi_k} \leq a_{\tau_i,\phi_k} \tag{4.18}$$

$$h_{\tau_i,\tau_j,\phi_k} \leq a_{\tau_j,\phi_k} \tag{4.19}$$

Worst-case fault-tolerant task response time. The modeling of $Cdec_{\tau_i}$ and $Crec_{\tau_i}$ are the same as the single-core case. The worst-case response time for task τ_i is affected by the higher priority tasks on the same core, which can be modeled with added allocation variables as below.

$$
\begin{aligned}
r_{\tau_i} = & \sum_{\phi_m \in \Phi} a_{\tau_i,\phi_m} Cdec_{\tau_i} + \\
& + K \max_{\tau_j \in \mathcal{T}} \{ \sum_{\phi_m \in \Phi} Crec_{\tau_j} p_{\tau_i,\tau_j} h_{\tau_i,\tau_j,\phi_m} \} \\
& + \sum_{\substack{\tau_k \in \mathcal{T} \\ \wedge \tau_i \neq \tau_k}} \sum_{\phi_m \in \Phi} \lceil \frac{r_{\tau_i}}{T_{\tau_k}} \rceil Cdec_{\tau_k} p_{\tau_i,\tau_k} h_{\tau_i,\tau_k,\phi_m}
\end{aligned} \tag{4.20}
$$

Optimization objective. In Section 4.1.2, Equation (4.10) is derived for single-core platforms as optimization objective. For multicore platforms, assuming $K = 1$ (one error for the entire system), similar derivation process results in the following equation where N is the number of cores.

$$P \approx 1 - \frac{1}{N} \sum_{\tau_i \in \mathcal{T}} (1 - \varepsilon_{\tau_i}) \frac{Cdec_{\tau_i}}{T_{\tau_i}} \tag{4.21}$$

We use Equation (4.21) as the optimization objective. Other constraints are similar to the single-core case.

4.1.4 Distributed Platform

The formulation of a distributed platform with single-core CPUs communicating through a single CAN bus can leverage the formulation from the task concentration model in Section 4.1.3. The main difference is the added consideration of message delays on the bus.

Task allocation. We use $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ to represent the set of CPUs. Boolean variable a_{τ_i,e_k} is 1 if task τ_i is allocated on CPU e_k, and 0 otherwise. Boolean variable h_{τ_i,τ_j,e_k} is 1 if task τ_i and τ_j are both allocated on CPU e_k, and 0 otherwise. The following formulation is similar to the case in Section 4.1.3.

$$\sum_{e_k \in E} a_{\tau_i,e_k} = 1 \tag{4.22}$$

$$a_{\tau_i,e_k} + a_{\tau_j,e_k} - 1 \leq h_{\tau_i,\tau_j,e_k} \tag{4.23}$$

$$h_{\tau_i,\tau_j,e_k} \leq a_{\tau_i,e_k} \tag{4.24}$$

$$h_{\tau_i,\tau_j,e_k} \leq a_{\tau_j,e_k} \tag{4.25}$$

Worst-case fault-tolerant task response time. The worst-case fault-tolerant response time for task τ_i is modeled as follows.

$$\begin{aligned} r_{\tau_i} = & \sum_{e_m \in E} a_{\tau_i,e_m} Cdec_{\tau_i} \\ & + K \max_{\tau_j \in \mathcal{T}} \{ \sum_{e_m \in \mathcal{E}} Crec_{\tau_j} p_{\tau_i,\tau_j} h_{\tau_i,\tau_j,e_m} \} \\ & + \sum_{\substack{\tau_k \in \mathcal{T} \\ \wedge \tau_i \neq \tau_k}} \sum_{e_m \in \mathcal{E}} \lceil \frac{r_{\tau_i}}{T_{\tau_k}} \rceil Cdec_{\tau_k} p_{\tau_i,\tau_k} h_{\tau_i,\tau_k,e_m} \end{aligned} \tag{4.26}$$

Worst-case signal response time. First, we use Boolean variable g_{s_i} as below to denote whether signal s_i is a global signal, i.e., a signal transmitted on the bus.

$$g_{s_i} = 1 - \sum_{e_m \in \mathcal{E}} h_{src(s_i),dst(s_i),e_m} \tag{4.27}$$

The worst-case response time of a signal r_{s_i} is modeled in below: If s_i is a global signal, it is transmitted as a CAN bus message and its response time is modeled following Equation (4.2) in Section 4.1.1; otherwise, it is transmitted in local memory and its response time is approximated with a small constant latency C_{local}.

$$\begin{aligned} r_{s_i} = & g_{s_i}(C_{s_i} + Bmax + \sum_{s_k \in S} C_{s_k} p_{s_i,s_k} \lceil \frac{r_{s_i - C_{s_i}}}{T_{s_k}} \rceil) \\ & + (1 - g_{s_i}) C_{local} \end{aligned} \tag{4.28}$$

In Equation (4.28), C_{s_i} denotes the worst-case transmission time of a CAN bus message and can be computed based on the length of its payload and the bus speed as follows, where β_{s_i} is the number of bits in the data payload and *speed* is the bus speed (CAN message has an overhead of 47 bits).

$$C_{s_i} = \frac{\beta_{s_i} + 47}{speed} \tag{4.29}$$

The end-to-end latency formulation follows Equation (4.3), with the consideration of messages. The objective function is similar to the task concentration model in Section 4.1.3.

4.1.5 Simulator for Evaluating Error Coverage

In order to accurately evaluate the error coverage of system solutions (Equation (4.10) and (4.21) used in our MILP formulations are only approximations), we build a Monte Carlo based error injection and runtime simulation engine. The simulation engine randomly injects soft errors, simulates the task executions and re-executions based on error occurrences and types, and checks whether timing constraints are violated or if any error escaped detection, and records the error detection and recovery statistics. The error coverage of a system solution is measured by *the percentage of simulation runs* during which all generated errors are either detected and recovered by EOC or EED or occurred during idle time, *and* all timing constraints are met.

The pseudo code of the simulator implementation is shown in Algorithm 2. We discretize the system and assume the tasks are activated at the integer multiplies of the

```
Algorithm 2: Simulation()

initialize_system();
for $t \leftarrow 0$; $t < T_{hyper}$; $t \leftarrow t+step$ do
    update_error() based on preset distribution;
    update_running_tasks();
    update_ready_tasks();
    check_timing_violations();
end
```

```
Algorithm 3: update_running_tasks()

if $runtask \neq \emptyset$ then
    if $Cleft_{runtask} = 0$ then
        if $error\_mark_{runtask} \neq 0$ then
            check error types; record failure if error has escaped;
        end
    end
    else
        $Cleft_{runtask} \leftarrow Cleft_{runtask} - step$;
    end
end
```

Algorithm 4: update_ready_tasks()

```
sort all $\tau_i$ activated at time $t$ by priority;
enqueue these $\tau_i$ to ready_queue[];
if $P_{readytask} > P_{runtask}$ then
    enqueue $runtask$ to ready_queue[] by priority;
    $runtask \leftarrow readytask$ ;
end
else if $Cleft_{runtask} = 0$ and $error\_mark_{runtask} = 0$ then
    if $ready\_queue[] \neq \emptyset$ then
        $runtask \leftarrow readytask$ ;
    end
    else
        $runtask \leftarrow \emptyset$ (CPU is free) ;
    end
end
```

smallest time step. The sub-routine *udpate_error*() injects random errors and updates the error information for affected tasks. Sub-routine *update_running_task*() checks whether the currently running task (if any) is completed and whether a re-execution is needed, based on the error tolerance mechanism for the task and the error types – EOC may cover all error types and EED can only cover some of them (the percentage of injected errors that can be covered by EED simulates the EED detection rate). In our experiments we only need to use two error types, which can be easily extended to simulate other cases such as those with less than 100% EOC coverage. $Cleft_{runtask}$ denotes how much time is left for the currently running task instance (either a regular execution or a re-execution). Sub-routine *update_ready_tasks*() updates the list of tasks that are activated and checks whether there is an preemption. Sub-routine *check_timing_violations*() checks whether timing constraints are violated, and stops the simulation and records failure if they are.

4.1.6 Experiments

We apply our approach to an industrial case study and a set of synthetic examples. The industrial case study is derived from a subsystem of an experimental vehicle that incorporates advanced active safety functions (similar to the system in [163]). The vehicle supports distributed functions with end-to-end computations collecting data from 360° sensors to the actuators, consisting of the throttle, brake and steering subsystems and of advanced HMI (Human-Machine Interface) devices. The functional model of the application includes 41 tasks with given periods and WCETs. The tasks communicate through 81 signals, and there are 171 paths with deadlines ranging from 100ms to 300ms. The synthetic examples are generated by the TGFF tool [40], with 30 to 50 tasks, and are assigned with random periods and WCETs. Based on the task graph, we sort end-to-end paths and assign each path a reasonable deadline.

Feasibility analysis and design exploration for industrial case study

In the industrial example, we demonstrate the *effectiveness of our formulations in evaluating the feasibility and error coverage* of the system under various architecture options and timing constraints. Among the 41 tasks in the industry example, 16 critical ones are assumed to require EOC for higher soft error coverage, and others may use either EOC or EED.

Multicore platforms. First, we use the multicore task concentration formulation introduced in Section 4.1.3 to analyze whether feasible solutions exist to satisfy the above fault tolerance requirements while also meeting the timing constraints, for platform configurations with various number of cores, and optimize the system error coverage if feasible solutions exist. The EED overhead is set as 30% and the EED error detection rate is 70%, and the EOC error detection rate is assumed as 100% (these numbers are also used for synthetic examples).

First, we find that a minimum of 5 cores are required to meet the fault tolerance requirements without violating the timing constraints, by using our formulation to explore task allocation and scheduling while enforcing EOC detection for the 16 critical tasks. We then evaluate how much the error coverage can be improved by increasing the number of cores (which enables applying EOC or EED to more tasks), with results based on 1000 simulation runs shown in Table 4.1. During each simulation run, we assume exactly one error occurs within the system hyperperiod and generate the error following uniform distribution

Core #	≤ 4	5	6	7	8	9
Coverage	infeasible	0.614	0.774	0.876	0.959	0.961

Table 4.1: System error coverage under different number of cores for the industrial example

within the hyperperiod. We record the system error coverage as the percentage of simulation runs during which all errors are covered and all timing requirements are met.

We can see that the error coverage improvement slows down when the number of cores reaches 8. Information in Table 4.1, which can only be obtained through quantitative modeling and exploration as in our formulations, will facilitate designers analyze system feasibility and assess potential fault tolerance level for various solutions.

We then explore the quantitative impact of using a faster CPU on system feasibility and error coverage. Table 4.2 shows the results of feasibility and error coverage analysis when all cores of the CPU speed up by a factor between 1.2 and 2 (modeled by reducing the task WCETs by the same factor). All other assumptions are the same as before.

We can see that for each CPU speed, having more cores leads to better error coverage and it saturates at certain point. For the same number of cores, having faster CPUs leads to better fault coverage in general and the improvement varies. These trends are not surprising but the quantitative differences may facilitate decisions at early design stages.

Distributed platforms. We also map the industrial example onto a distributed automotive platform with several single-core ECUs (Electronic Control Units) connected with a CAN

Core #	≤ 2	3	4	5	6	7	8
1.2×speed	infeasible		0.550	0.769	0.899	0.964	0.984
1.4×speed	infeasible		0.609	0.896	1.000	1.000	1.000
1.6×speed	infeasible	0.597	0.804	0.963	1.000	1.000	1.000
1.8×speed	infeasible	0.784	0.909	0.992	1.000	1.000	1.000
2.0×speed	infeasible	0.816	1.000	1.000	1.000	1.000	1.000

Table 4.2: System error coverage under different CPU speeds and number of cores for the industrial example (multicore platform)

bus, while analyzing the feasibility and error coverage[4]. Comparing with the above single-CPU multicore case, the consideration of message latencies leads to different results and design focuses. In particular, to analyze the impact of the end-to-end latency constraints on error coverage, we vary the end-to-end latency requirements D_p to 0.7×, 1.3×, 1.6×, and 2.0× of the original requirements (100 ms to 300 ms on different paths). Table 4.3 shows the error coverage results under different number of ECUs and different end-to-end latency requirements. We can clearly see the quantitative impact of end-to-end latencies on the system feasibility and error coverage. We can also see that for the original requirements, the distributed platform needs 7 ECUs to have a feasible solution while in the multicore case we only need 5 cores (we assume same WCETs). This is because the bus message transmission delays make it more difficult to meet the end-to-end latency requirements.

System error coverage optimization in synthetic examples

We then apply our approach to a set of synthetic examples to further demonstrate its *effectiveness in exploring error detection mechanisms for enhancing system error cover-*

[4]The original design we received was implemented on a similar CAN-based platform, which is the currently prevalent automotive architecture (while mulitcore ECUs/CPUs are projected to be adopted in the future).

e2e requirement	0.7×	1×	1.3×	1.6×	2×
5 ECU	infeasible		0.569	0.647	0.663
6 ECU	infeasible		0.752	0.756	0.757
7 ECU	infeasible	0.763	0.827	0.867	0.878
8 ECU	infeasible	0.824	0.932	0.933	0.951

Table 4.3: System error coverage under different end-to-end latency requirements and number of ECUs for the industrial example (distributed platform)

age. We compare five sets of solutions through simulations: the solutions from our MILP formulations exploring EOC/EED selections with task allocation and scheduling (denoted as "Optimized EOC/EED"), the solutions from our MILP formulations but assuming that due to system constraints or capabilities only EOC or only EED is available (denoted as "Optimized EOC only" and "Optimized EED only", respectively), and the solutions from two naive implementations in which either EOC or EED is used for all tasks without optimization (denoted as "Naive EOC only" and "Naive EED only", respectively).

Single-core CPU platform. Fig. 4.4 shows the comparison among the five methods under different original core utilizations ($\sum_{\tau_i} C_{\tau_i}/T_{\tau_i}$) for a single-core platform. For each utilization, 1000 simulation runs are conducted and the error coverage is reported. We assume exactly one error occurs within the hyperperiod (i.e., $K = 1$).

Fig. 4.4 demonstrates that 1) the optimized EOC only and EED only solutions provide better error coverage than naive EOC only and EED only solutions when the utilization increases, and 2) the optimized EOC/EED (when both options are available for the platform) provides further significant improvements over optimized EOC only or EED

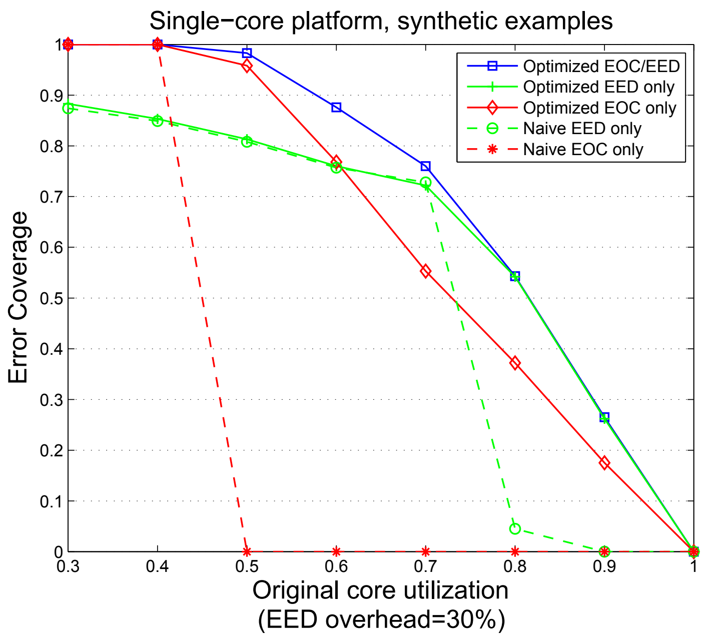


Figure 4.4: Comparison of various methods for synthetic examples on a single-core platform

only. This clearly shows the benefits of our formulations in quantifying the impact of EOC and EED and exploring the selections of them at task level. Furthermore, our formulations guarantee that the solutions always satisfy timing constraints when the number of errors is within K, which provides much better *timing predictability* than naive solutions.

Multicore platform. Fig. 4.5 shows the error coverage comparison among the five methods under different original core utilizations (x-axis is per core utilization), for the synthetic examples with task concentration model on a dual-core platform. We assume $K = 2$, with one error occurs on each core within the hyperperiod. The error coverage is based on 1000 simulations. We see similar trend as in the single-core case. Our approach provides significantly better results overall. We also conduct experiments for a dual-core platform with the task duplication model. The duplication of task execution on two cores leverages the spatial redundancy in a straightforward fashion. Since in this case almost no timing overhead is introduced for EOC error detection (except for small output comparison time), our MILP formulation will try to select EOC for tasks as long as the timing constraints are satisfied. Overall, the duplication model provides similar results to the concentration model when the utilization is under 80%, and slightly worse when it is higher than 80%.

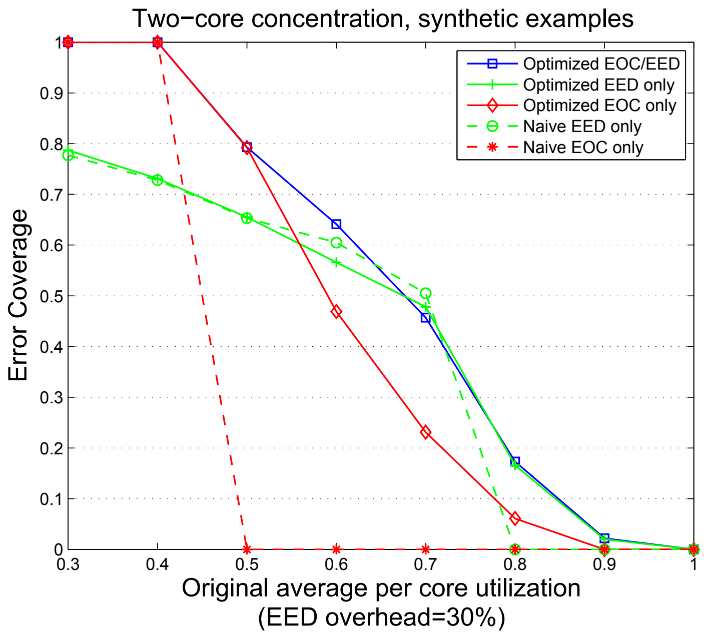


Figure 4.5: Comparison of various methods for synthetic examples on a dual-core platform (task concentration model)

Impact of EED overhead and detection rate

We also study the impact of EED overhead and detection rate on the selection of EED versus EOC. We conduct experiments on a single-core platform with a modification of the industrial example (the task WCETs are scaled to make the core utilization at 60% since the original WCETs will not yield feasible solution on a single-core platform). Fig. 4.6 shows the percentage of EOC used and EED used out of the entire task set (some tasks may use neither mechanism because of timing constraints). The sub-figures demonstrate results with EED overhead ranging from 0% to 100%, and EED detection rate for a task ranging from 50% to 80%. Clearly, we can see the significant impact of EED overhead and detection rate on the selections of EOC versus EED. Intuitively, higher overhead and lower detection rate will lead to more EOC selection, and vice versa. Such trade-off demonstrates the importance of having quantified formulations and algorithms in selecting the appropriate error detection mechanisms for each task during task scheduling. Also, it reveals important performance overhead and error detection rate goals when designing the detailed EED implementations. Finally, together with the previous figures, it can be seen that EOC still maintains its value in fault-tolerant scheduling, despite its simplicity and high performance overhead.

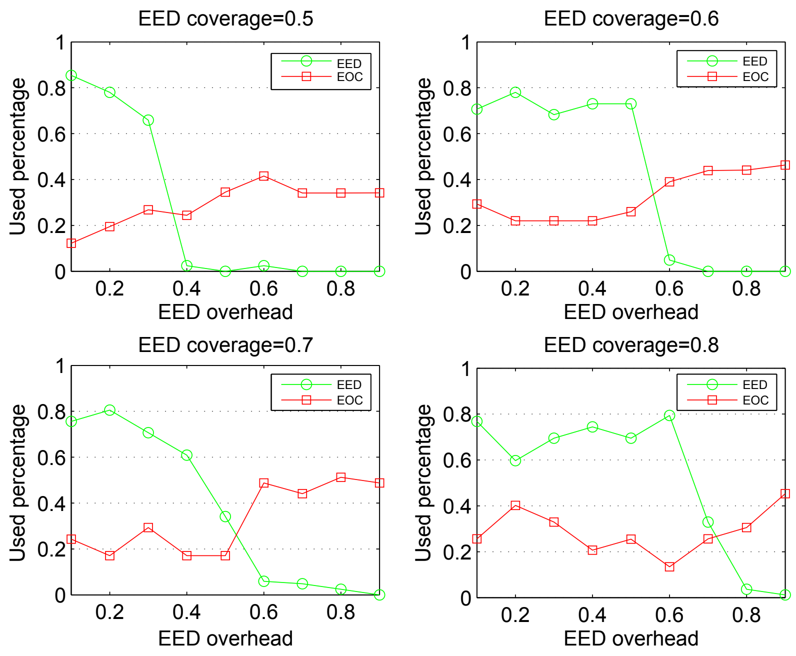


Figure 4.6: The percentage of EOC used vs. EED used under different EED overhead and error detection rate for modified industrial example

4.2 Security-Aware Mapping

As described in Chapter 1, the in-vehicle design encounters various security challenges. Figure. 4.7 shows an example of the masquerade attack on the in-vehicle bus system. Without authentication, the compromised node can masquerade a trusted node and send the "Lock brake" message to the brake. One countermeasure is to use authentication techniques, like authentication codes (MACs), for the brake and the trusted node to check identity. In this example, symmetric keys are used.

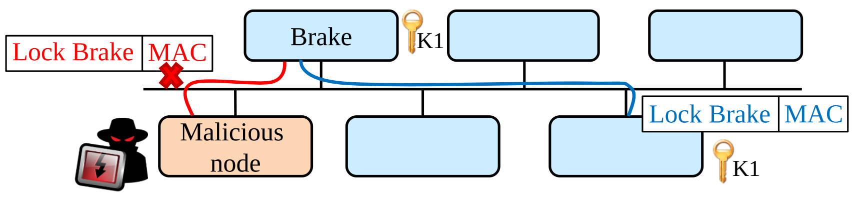


Figure 4.7: The security challenges in automotive systems

Adding security mechanisms may lead to computation overhead caused by authentication and transmission overhead with added authentication fields. The timing behavior may be significantly affected by these overheads. Therefore, the security mechanisms should be considered at the early design stage.

In our work [94], we consider the security-aware mapping for CAN-based and TDMA based systems. The functional model is captured by a task graph with signals between tasks. The goal is to decide security mechanisms together with task mapping. The security mechanism is to divide the nodes into groups and in each group, the same symmetric authentication key is used. The purpose of key-sharing groups is to reduce the overhead of

MACs if the destinations of the message are in one group. If we use pair-wise key sharing, each destination will result in one more MAC appended to the message. However, grouping more nodes in one group increase the security risks. Therefore, we quantitatively model the risks as the objective with real-time constraints on computation and communication. The decision variables include group allocation, MAC length, task allocation, task scheduling, signal packing, and message scheduling as in [94, 96, 95].

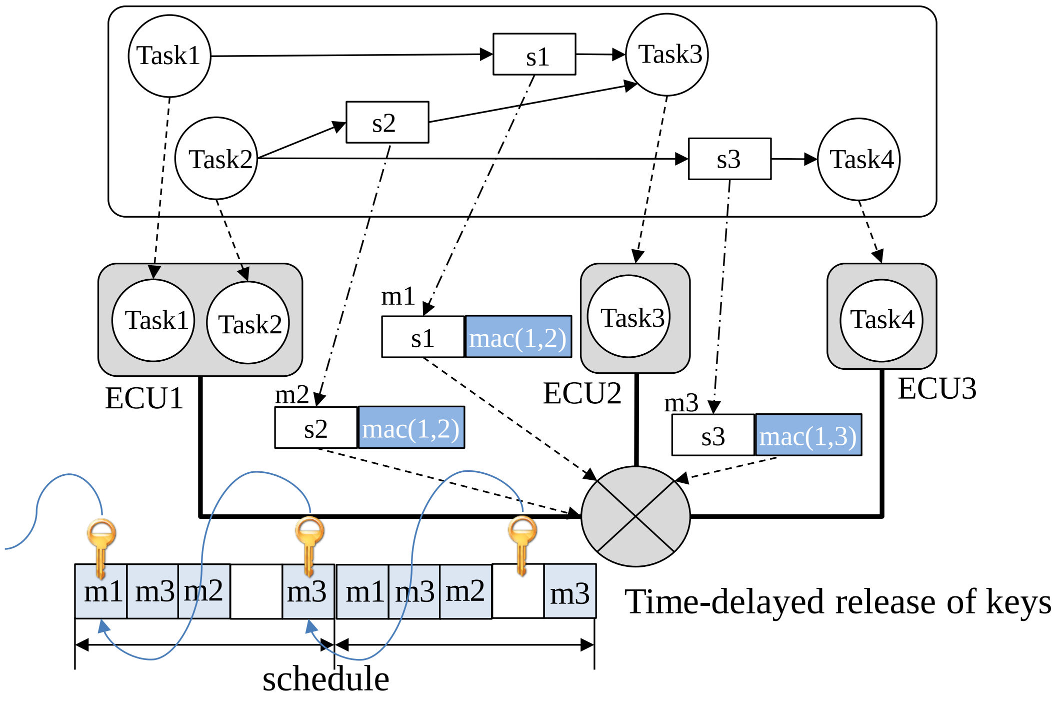


Figure 4.8: Security-aware mapping for TDMA-based systems

Figure. 4.8 shows the security-aware mapping for the TDMA-based system. TDMA-based buses assign pre-defined time slots to messages, therefore, making denial-of-service attacks more difficult. However, masquerade attacks and replays are still possible without authentication. Similar to the CAN-based systems, we can quantitatively trade-off between security and computational overhead by utilizing message authentication with delayed-release of keys [94, 97].

4.3 Summary

When addressing the safety of real-time systems, it is important to consider the timing, the fault tolerance requirements, and the security requirements. In this chapter, we quantitatively model the impact of fault-tolerance and security techniques on system timing. For fault-tolerance, two different types of soft error detection mechanisms (EOC and EED) are considered under various fault models, task execution models, and architecture platforms. We present a set of MILP formulations to explore the selections of EOC versus EED, together with task allocation and scheduling, to enhance the system soft error resiliency while meeting timing constraints. These formulations may facilitate designers to analyze system feasibility under various fault tolerance requirements, trade off various design options, and optimize system error coverage. Similarly, the security level can be quantitatively modeled and optimized for both CAN-based and TDMA-based systems.

Chapter 5

Conclusions and Future Directions

In this thesis, we discuss the design challenges for building a secure and reliable cyber-physical systems. We present a cross-layer modeling, exploration and validation framework CONVINCE considering various design objectives and metrics, among which security and fault-tolerance are our focuses.

At the application and software layers, we consider cooperative adaptive cruise control and intelligent intersection management to address the challenges from communication delays and possible security attacks. We present a delay-tolerant protocol for intelligent intersection management and conduct modeling, simulation, and verification for analyzing the safety, liveness, and performance of the protocol. This protocol guarantees safety and assures that as long as the communication delays are bounded, deadlock-free and liveness can be guaranteed. We also develop a codesign approach for addressing the trade-off between security and control performance with the consideration of implementation feasibility. This approach quantitatively models the impact of security techniques on control performance and platform schedulability, and explores trade-offs between security level and control performance while guaranteeing real-time constraints for cyber-physical systems.

At the software and hardware layers, we address the software to hardware mapping considering fault-tolerance and security. We conduct fault-tolerance design to improve system-level error recovery rate by applying soft error detection and recovery mechanisms with real-time constraints. This approach formulates the impact on system timing for different error tolerance mechanisms including both EOC and EED based techniques, and optimizes the task-level selections of tolerance mechanisms, for various fault models and task execution models on representative single-core, multi-core, and distributed platforms. We also present the security-aware mapping that can quantitatively study the impact of applying security mechanisms for both CAN-based and TDMA-based systems with limited resources and strict timing constraints.

There are several future directions.

- As the safety and liveness properties are related to the consensus and latency trade-offs in asynchronous distributed systems, we can study the impact of consensus levels on connected vehicle applications through the cross-layer framework. In particular, we can define appropriate consensus levels, study what levels can be reached under communication delays and losses, and further analyze how the consensus levels may affect objectives, such as safety and transportation efficiency.

- We can address other connected vehicle applications through our framework, for example, roundabout management, lane merging, do not pass warning, dynamic ridesharing, etc.

- We can study other types of cyber-physical attacks that threaten cyber-physical systems, for example, the attacks on the time synchronization protocol, the spoofing attacks, the physical attacks, etc.

Bibliography

[1] As U.S. investigates fatal Tesla crash, company defends autopilot system. http://www.nytimes.com/2016/07/13/business/tesla-autopilot-fatal-crash-investigation.html.

[2] Google car: Data hog at speeds topping 2700gb per hour. http://www.roboticsbusinessreview.com/article/google_car_data_hog_at_speeds_topping_2700gb_per_hour.

[3] Google self-driving car caught on video colliding with bus. https://www.theguardian.com/technology/2016/mar/09/google-self-driving-car-crash-video-accident-bus.

[4] NS-3. https://www.nsnam.org/.

[5] OMNeT++. https://www.nsnam.org/.

[6] Simulink. http://www.mathworks.com/products/simulink/.

[7] SUMO. http://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/.

[8] Ubers fatal self-driving crash: all the news and updates. https://www.theverge.com/2018/3/28/17174636/uber-self-driving-crash-fatal-arizona-update.

[9] Veins. http://veins.car2x.org/.

[10] UPPAAL. https://www.uppaal.org/, 2017.

[11] U. Abelein, H. Lochner, D. Hahn, and S. Straube. Complexity, quality and robustness - the challenges of tomorrow's automotive electronics. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 870–871, March 2012.

[12] Ulrich Abelein, Alan Cook, Piet Engelke, Michael Glas, Felix Reimann, Laura Rodriguez Gomez, Thomas Russ, Jurgen Teich, Dominik Ull, and H-J Wunderlich. Non-intrusive integration of advanced diagnosis features in automotive E/E-architectures. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[13] M. Ahmane, A. Abbas-Turki, F. Perronnet, J. Wu, A. El Moudni, J. Buisson, and R. Zeo. Modeling and controlling an isolated urban intersection based on cooperative vehicles. *Transportation Research Part C: Emerging Technologies*, 28:44–62, 2013.

[14] Mohammed Saeed Al-kahtani. Survey on security attacks in vehicular ad hoc networks (VANETs). In *Signal Processing and Communication Systems (ICSPCS), 2012 6th International Conference on*, pages 1–9. IEEE, 2012.

[15] Mani Amoozadeh, Hui Deng, Chen-Nee Chuah, H Michael Zhang, and Dipak Ghosal. Platoon management with cooperative adaptive cruise control enabled by vanet. *Vehicular Communications*, 2(2):110–123, 2015.

[16] R. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige. Intersection management using vehicular networks. Technical report, SAE Technical Paper, 2012.

[17] R. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige. Stip: Spatio-temporal intersection protocols for autonomous vehicles. In *ICCPS'14: ACM/IEEE 5th International Conference on Cyber-Physical Systems (with CPS Week 2014)*, pages 1–12. IEEE Computer Society, 2014.

[18] S.R. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige. Reliable intersection protocols using vehicular networks. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 1–10. ACM, 2013.

[19] Peter Bailis, Shivaram Venkataraman, Michael J Franklin, Joseph M Hellerstein, and Ion Stoica. Probabilistically bounded staleness for practical partial quorums. *Proceedings of the VLDB Endowment*, 5(8):776–787, 2012.

[20] Y. O. Basciftci, F. Chen, J. Weston, R. Burton, and C. E. Koksal. How vulnerable is vehicular communication to physical layer jamming attacks? In *2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall)*, pages 1–5, Sept 2015.

[21] S. Bastani, B. Landfeldt, and L. Libman. On the reliability of safety message broadcast in urban vehicular ad hoc networks. In *Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '11, pages 307–316, New York, NY, USA, 2011. ACM.

[22] Robert C Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *Device and Materials Reliability, IEEE Transactions on*, 5(3):305–316, 2005.

[23] Steve Bono, Matthew Green, Adam Stubblefield, Ari Juels, Aviel D Rubin, and Michael Szydlo. Security analysis of a cryptographically-enabled rfid device. In *USENIX Security*, volume 5, pages 1–16, 2005.

[24] Martin Buechel, Jelena Frtunikj, Klaus Becker, Stephan Sommer, Christian Buckl, Michael Armbruster, Andre Marek, Andreas Zirkler, Cornel Klein, and Alois Knoll. An automated electric vehicle prototype showing new trends in automotive architectures. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 1274–1279. IEEE, 2015.

[25] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[26] Giorgio C Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[27] Arquimedes Canedo, Jiang Wan, Al Faruque, and Mohammad Abdullah. Functional modeling compiler for system-level design of automotive cyber-physical systems. In *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pages 39–46. IEEE, 2014.

[28] Robert N. Charette. This Car Runs on Code. *IEEE Spectrum*, February 2009.

[29] Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.

[30] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, Tadayoshi Kohno, et al. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco, 2011.

[31] Lei Chen and Cristofer Englund. Cooperative intersection management: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 17(2):570–586, 2016.

[32] Lewis M Clements and Kara M Kockelman. Economic effects of automated vehicles. *Transportation Research Record: Journal of the Transportation Research Board*, (2606):106–114, 2017.

[33] Anup Das, Akash Kumar, Bharadwaj Veeravalli, Rishad Shafik, Geoff Merrett, and Bashir Al-Hashimi. Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 43–48. EDA Consortium, 2015.

[34] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 278–283, 2007.

[35] G de A Lima and Alan Burns. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. *Computers, IEEE Transactions on*, 52(10):1332–1346, 2003.

[36] P. Deng, F. Cremona, Q. Zhu, M. Di Natale, and H. Zeng. A Model-Based Synthesis Flow for Automotive CPS. In *Cyber-Physical Systems (ICCPS), 2015 ACM/IEEE International Conference on*, pages 198–207, April 2015.

[37] P. Deng, Q. Zhu, A. Davare, A. Mourikis, X. Liu, and M. Di Natale. An efficient control-driven period optimization algorithm for distributed real-time systems. *IEEE Transactions on Computers*, PP(99):1–1, 2016.

[38] M. Di Natale and A.L. Sangiovanni-Vincentelli. Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. *Proceedings of the IEEE*, 98(4):603 –620, april 2010.

[39] Marco Di Natale, Paolo Giusto, Sri Kanajan, Claudio Pinello, and Patrick Popp. Architecture exploration for time-critical and cost-sensitive distributed systems. In *SAE Technical Paper*, 2007.

[40] Robert P. Dick, David L. Rhodes, and Wayne Wolf. Tgff: task graphs for free. In *Proceedings of the 6th international workshop on Hardware/software codesign*, CODES/CASHE '98, pages 97–101, Washington, DC, USA, 1998. IEEE Computer Society.

[41] Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal worst-case timing analysis of ethernet topologies with strict-priority and avb switching. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, pages 1–10. IEEE, 2012.

[42] Université Laval. Département d'informatique et de génie logiciel and Sylvain Hallé. *Architectures for collaborative driving vehicles: From a review to a proposal*. Sainte-Foy, Quebec: Dép. d'informatique et de génie logiciel, Université Laval, 2003.

[43] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*, 31:591–656, 2008.

[44] Michael Eberl, Michael Glaß, Jürgen Teich, and Ulrich Abelein. Considering diagnosis functionality during automatic system-level design of automotive networks. In *Proceedings of the 49th Annual Design Automation Conference*, pages 205–213. ACM, 2012.

[45] Richard Gilles Engoulou, Martine Bellaïche, Samuel Pierre, and Alejandro Quintero. VANET security surveys. *Computer Communications*, 44(0):1–13, 2014.

[46] Y. P. Fallah and M. K. Khandani. Analysis of the coupling of communication network and safety application in cooperative collision warning systems. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ICCPS '15, pages 228–237, New York, NY, USA, 2015. ACM.

[47] Yaser P. Fallah and Masoumeh K. Khandani. Analysis of the coupling of communication network and safety application in cooperative collision warning systems. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, ICCPS '15, pages 228–237, New York, NY, USA, 2015. ACM.

[48] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (AADL): An introduction. Technical report, DTIC Document, 2006.

[49] Pedro Fernandes and Urbano Nunes. Platooning with ivc-enabled autonomous vehicles: Strategies to mitigate communication delays, improve safety and traffic flow. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):91–106, 2012.

[50] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[51] Yue Gao, Sandeep K. Gupta, and Melvin A. Breuer. Using Explicit Output Comparisons for Fault Tolerant Scheduling (FTS) on Modern High-performance Processors. In *DATE 2013*.

[52] M. Gomaa, C. Scarbrough, T.N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 98–109, June 2003.

[53] L. Guo, Q. Zhu, P. Nuzzo, R. Passerone, A. Sangiovanni-Vincentelli, and E.A. Lee. Metronomy: A Function-Architecture Co-Simulation Framework For Timing Verification Of Cyber-Physical Systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on*, pages 1–10, Oct 2014.

[54] Abhishek Gupta, Cédric Langbort, and Tamer Basar. Optimal control in the presence of an intelligent jammer with limited actions. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 1096–1101. IEEE, 2010.

[55] Simon Hallé and Brahim Chaib-draa. A collaborative driving system based on multiagent modelling and simulations. *Transportation Research Part C: Emerging Technologies*, 13(4):320–345, 2005.

[56] Simon Hallé, Julien Laumonier, and Brahim Chaib-Draa. A decentralized approach to collaborative driving coordination. In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pages 453–458. IEEE, 2004.

[57] Y. Han and E. Oruklu. Real-time traffic sign recognition based on zynq fpga and arm socs. In *IEEE International Conference on Electro/Information Technology*, pages 373–376, June 2014.

[58] Peter Hank, Thomas Suermann, and Steffen Müller. Automotive Ethernet, a holistic approach for a next generation in-vehicle networking standard. In *Advanced Microsystems for Automotive Applications 2012*, pages 79–89. Springer, 2012.

[59] Mohammad A Haque, Hakan Aydin, and et al. Real-time scheduling under fault bursts with multiple recovery strategy. In *RTAS 2014*, pages 259–268. IEEE, 2013.

[60] M. Gonzalez Harbour, M. Klein, and J. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1), January 1994.

[61] John Harding, Gregory Powell, Rebecca Yoon, Joshua Fikentscher, Charlene Doyle, Dana Sade, Mike Lukuc, Jim Simons, and Jing Wang. Vehicle-to-vehicle communications: Readiness of V2V technology for application. Technical report, 2014. National Highway Traffic Safety Administration, DOT HS 812 014.

[62] Florian Hartwich. CAN with flexible data-rate. In *13th International CAN Conference (iCC2012), Hambach, Germany*, 2012.

[63] Matthew Hausknecht, Tsz-Chiu Au, and Peter Stone. Autonomous intersection management: Multi-intersection optimization. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4581–4586. IEEE, 2011.

[64] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security requirements for automotive on-board networks. In *Proceedings of the 9th International Conference on Intelligent Transport System Telecommunications (ITST 2009), Lille, France*, 2009.

[65] F. Homm, N. Kaempchen, J. Ota, and D. Burschka. Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1006–1013, June 2010.

[66] Jia Huang, Andreas Raabe, Kai Huang, Christian Buckl, and Alois Knoll. A framework for reliability-aware design exploration on MPSoC based systems. *Design Automation for Embedded Systems*, 16(4):189–220, 2012.

[67] PB Hunt, DI Robertson, RD Bretherton, and M Cr Royle. The scoot on-line traffic signal optimisation technique. *Traffic Engineering & Control*, 23(4), 1982.

[68] Ondrej Hyncica, Pavel Kucera, Petr Honzik, and Petr Fiedler. Performance evaluation of symmetric cryptography in embedded systems. In *Intelligent data acquisition and advanced computing systems (IDAACS), 2011 IEEE 6th international conference on*, volume 1, pages 277–282. IEEE, 2011.

[69] IEEE. IEEE standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks. *IEEE Std 802.1AS-2011*, pages 1–292, March 2011.

[70] IEEE. IEEE approved draft standard for a transport protocol for time sensitive applications in a bridged local area network. *IEEE P1722/D16, November 2015*, pages 1–247, Jan 2015.

[71] Viacheslav Izosimov, Ilia Polian, Paul Pop, Petru Eles, and Zebo Peng. Analysis and optimization of fault-tolerant embedded systems with hardened processors. In *Design, Automation and Test in Europe Conference and Exhibition, 2009. DATE'09.*, pages 682–687. IEEE, 2009.

[72] Viacheslav Izosimov, Paul Pop, Petru Eles, and Zebo Peng. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, DATE '05, pages 864–869, Washington, DC, USA, 2005. IEEE Computer Society.

[73] Q. Jin, G. Wu, K. Boriboonsomsin, and M. Barth. Advanced intersection management for connected vehicles using a multi-agent systems approach. In *Intelligent Vehicles Symposium (IV), 2012 IEEE*, pages 932–937. IEEE, 2012.

[74] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car - part I: distributed system architecture and development process. *Industrial Electronics, IEEE Transactions on*, 61(12):7131–7140, 2014.

[75] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[76] Algimantas Kajackas, Antanas Vindašius, and Šarūnas Stanaitis. Inter-vehicle communication: Emergency message delay distributions. *Journal of Electronics and Electrical Engineering*, 8(96):33–38, 2009.

[77] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil. Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions. *Communications Surveys Tutorials, IEEE*, 13(4):584–616, Fourth 2011.

[78] Shin Kato, Sadayuki Tsugawa, Kiyohito Tokuda, Takeshi Matsui, and Haruki Fujii. Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications. *IEEE Transactions on Intelligent Transportation Systems*, 3(3):155–161, 2002.

[79] Matthias Kauer, Damoon Soudbakhsh, Dip Goswami, Samarjit Chakraborty, and Anuradha M Annaswamy. Fault-tolerant control synthesis and verification of distributed embedded systems. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 56. European Design and Automation Association, 2014.

[80] John B Kenney. Dedicated short-range communications (DSRC) standards in the United States. *Proceedings of the IEEE*, 99(7):1162–1182, 2011.

[81] Hyoseung Kim, Dionisio de Niz, Bjorn Andersson, Mark Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in COTS-based multicore systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*, pages 145–154. IEEE, 2014.

[82] Junsung Kim, Gaurav Bhatia, Ragunathan Rajkumar, and Markus Jochim. SAFER: System-level architecture for failure evasion in real-time applications. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 227–236. IEEE, 2012.

[83] Junsung Kim, Karthik Lakshmanan, and Ragunathan Rajkumar. R-BATCH: Task partitioning for fault-tolerant multiprocessor real-time systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1872–1879. IEEE, 2010.

[84] Alexander Kordes, Ben Vermeulen, Abhishek Deb, and Michael G Wahl. Startup error detection and containment to improve the robustness of hybrid FlexRay networks. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[85] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. Experimental security analysis of a modern automobile. In *2010 IEEE Symposium on Security and Privacy*, pages 447–462. IEEE, 2010.

[86] H. Kowshik, D. Caveney, and PR. Kumar. Provable systemwide safety in intelligent intersections. *IEEE transactions on vehicular technology*, 60(3):804–818, 2011.

[87] Shankara Narayanan Krishna, Ganesh Narwane, S Ramesh, and Ashutosh Trivedi. Compositional modeling and analysis of automotive feature product lines. In *Proceedings of the 52nd Annual Design Automation Conference*, page 57. ACM, 2015.

[88] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[89] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[90] C. Lee, S. W. Kim, and C. Yoo. VADI: GPU virtualization for an automotive platform. *IEEE Transactions on Industrial Informatics*, 12(1):277–290, Feb 2016.

[91] Edward A Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[92] GM Lima and Alan Burns. An effective schedulability analysis for fault-tolerant hard real-time systems. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 209–216. IEEE, 2001.

[93] Chung-Wei Lin and Alberto Sangiovanni-Vincentelli. Cyber-security for the Controller Area Network (CAN) communication protocol. In *Cyber Security (CyberSecurity), 2012 International Conference on*, pages 1–7. IEEE, 2012.

[94] Chung-Wei Lin, Bowen Zheng, Qi Zhu, and Alberto Sangiovanni-Vincentelli. Security-aware design methodology and optimization for automotive systems. *ACM Trans. Des. Autom. Electron. Syst.*, 21(1):18:1–18:26, December 2015.

[95] Chung-Wei Lin, Qi Zhu, C. Phung, and A. Sangiovanni-Vincentelli. Security-aware mapping for CAN-based real-time distributed automotive systems. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 115–121, Nov 2013.

[96] Chung-Wei Lin, Qi Zhu, and A. Sangiovanni-Vincentelli. Security-Aware Modeling and Efficient Mapping for CAN-Based Real-Time Distributed Automotive Systems. *Embedded Systems Letters, IEEE*, 7(1):11–14, March 2015.

[97] Chung-Wei Lin, Qi Zhu, and Alberto Sangiovanni-Vincentelli. Security-aware mapping for TDMA-based real-time distributed systems. In *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '14, pages 24–31, Piscataway, NJ, USA, 2014. IEEE Press.

[98] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[99] Xue Liu, Qixin Wang, S. Gopalakrishnan, Wenbo He, Lui Sha, Hui Ding, and Kihwal Lee. Ortega: An efficient and flexible online fault tolerance architecture for real-time control systems. *Industrial Informatics, IEEE Transactions on*, 4(4):213–224, Nov 2008.

[100] Xiaomin Ma and Xianbo Chen. Delay and broadcast reception rates of highway safety applications in vehicular ad hoc networks. In *2007 Mobile networking for vehicular environments*, pages 85–90. IEEE, 2007.

[101] John Paul MacDuffie and Takahiro Fujimoto. Why Dinosaurs Will Keep Ruling the Auto Industry. *Harvard Business Review*, 88(6):23–25, 2010.

[102] Florian Many and David Doose. Scheduling analysis under fault bursts. In *RTAS 2011*, pages 113–122. IEEE, 2011.

[103] Franois Michaud, Pierre Lepage, Patrick Frenette, Dominic Letourneau, and Nicolas Gaubert. Coordinated maneuvering of automated vehicles in platoons. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):437–447, 2006.

[104] Pitu Mirchandani and Larry Head. A real-time traffic signal control system: architecture, algorithms, and analysis. *Transportation Research Part C: Emerging Technologies*, 9(6):415–432, 2001.

[105] G. Miremadi, J. Harlsson, U. Gunneflo, and J. Torin. Two software techniques for on-line error detection. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 328–335, July 1992.

[106] Philipp Mundhenk, Sebastian Steinhorst, Martin Lukasiewycz, Suhaib A Fahmy, and Samarjit Chakraborty. Security analysis of automotive architectures using probabilistic model checking. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[107] NHTSA National Highway Traffic Safety Administration. Fatality analysis reporting system (FARS). `https://www.nhtsa.gov/research-data/fatality-analysis-reporting-system-fars`, 2017.

[108] R. Naumann, R. Rasche, J. Tacken, and C. Tahedi. Validation and simulation of a decentralized intersection collision avoidance algorithm. In *Intelligent Transportation System, 1997. ITSC'97., IEEE Conference on*, pages 818–823. IEEE, 1997.

[109] Dennis K Nilsson, Ulf E Larson, Francesco Picasso, and Erland Jonsson. A first simulation of attacks in the automotive network communications protocol flexray. In *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*, pages 84–91. Springer, 2009.

[110] D.K. Nilsson, U.E. Larson, and E. Jonsson. Efficient in-vehicle delayed data authentication based on compound message authentication codes. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, Sept 2008.

[111] J-H Oetjens, Nico Bannow, Matthias Becker, Oliver Bringmann, Andreas Burger, M Chaari, Shiladri Chakraborty, Rolf Drechsler, Wolfgang Ecker, Kim Gruttner, et al. Safety evaluation of automotive electronics using virtual prototypes: State of the art and research challenges. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[112] N. Oh, P.P. Shirvani, and E.J. McCluskey. Control-flow checking by software signatures. *Reliability, IEEE Transactions on*, 51(1):111–122, Mar 2002.

[113] OSEK. OSEK OS Version 2.2.3 specification. available at `http://www.osek-vdx.org`, 2006.

[114] Umit Ozguner, Tankut Acarman, and Keith Redmill. *Autonomous ground vehicles.* Artech House, 2011.

[115] Sujan Pandey and Bart Vermeulen. Transient errors resiliency analysis technique for automotive safety critical applications. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 9. European Design and Automation Association, 2014.

[116] Zhong-Hua Pang, GP Liu, and Zhe Dong. Secure networked control systems under denial of service attacks. In *18th IFAC World Congress*, 2011.

[117] F. Pasqualetti and Q. Zhu. Design and operation of secure cyber-physical systems. *Embedded Systems Letters, IEEE*, PP(99):1–1, 2014.

[118] Fabio Pasqualetti, F Dorfler, and Francesco Bullo. Attack detection and identification in cyber-physical systems. *Automatic Control, IEEE Transactions on*, 58(11):2715–2729, 2013.

[119] András Pataricza, István Majzik, Wolfgang Hohl, and Joachim Hönig. Watchdog processors in parallel systems. *Microprocessing and Microprogramming*, 39(2):69–74, 1993.

[120] Alexandre Petrenko, Omer Nguena Timo, and S Ramesh. Model-based testing of automotive software: Some challenges and solutions. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[121] Claudio Pinello, Luca P Carloni, and Alberto L Sangiovanni-Vincentelli. Fault-tolerant distributed deployment of embedded control software. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(5):906–919, 2008.

[122] Radha Poovendran, Krishna Sampigethaya, Sandeep KS Gupta, Insup Lee, K Venkatesh Prasad, David Corman, and J Paunicka. Special issue on cyber-physical systems [scanning the issue]. *Proceedings of the IEEE*, 100(1):6–12, 2012.

[123] Paul Pop, Viacheslav Izosimov, Petru Eles, and Zebo Peng. Design optimization of time-and cost-constrained fault-tolerant embedded systems with checkpointing and replication. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(3):389–402, 2009.

[124] Rene Queck. Analysis of Ethernet AVB for automotive networks using network calculus. In *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*, pages 61–67. IEEE, 2012.

[125] Maxim Raya and Jean-Pierre Hubaux. Securing vehicular ad hoc networks. *J. Comput. Secur.*, 15(1):39–68, January 2007.

[126] Maxim Raya, Panos Papadimitratos, and Jean-Pierre Hubaux. Securing vehicular communications. *IEEE Wireless Communications Magazine, Special Issue on Inter-Vehicular Communications*, 13(LCA-ARTICLE-2006-015):8–15, 2006.

[127] Felix Reimann, Michael Glaß, Alan Cook, Laura Rodriguez Gomez, Jurgen Teich, Dominik Ull, H-J Wunderlich, Ulrich Abelein, and Piet Engelke. Advanced diagnosis: SBST and BIST integration in automotive E/E architectures. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[128] SAE. Time-Triggered Ethernet. *SAE Standard AS6802*, November 2011.

[129] Florian Sagstetter, Sidharta Andalam, Peter Waszecki, Martin Lukasiewycz, Hauke Stähle, Samarjit Chakraborty, and Alois Knoll. Schedule integration framework for time-triggered automotive architectures. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 20:1–20:6, New York, NY, USA, 2014. ACM.

[130] Florian Sagstetter, Martin Lukasiewycz, Sebastian Steinhorst, Marko Wolf, Alexandre Bouard, William R Harris, Somesh Jha, Thomas Peyrin, Axel Poschmann, and Samarjit Chakraborty. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 458–463. EDA Consortium, 2013.

[131] Alberto Sangiovanni-Vincentelli. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[132] Bernhard Schatz, Sebastian Voss, and Sergey Zverlov. Automating design-space exploration: optimal deployment of automotive SW-components in an ISO26262 context. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[133] F. Schwiegelshohn, L. Gierke, and M. Hbner. Fpga based traffic sign detection for automotive camera systems. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on*, pages 1–6, June 2015.

[134] Michele Segata, Falko Dressler, Renato Lo Cigno, and Mario Gerla. A simulation tool for automated platooning in mixed highway scenarios. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 389–392. ACM, 2012.

[135] Danbing Seto, John P Lehoczky, Lui Sha, and Kang G Shin. On task schedulability in real-time control systems. In *Real-Time Systems Symposium, 1996., 17th IEEE*, pages 13–21. IEEE, 1996.

[136] Lui Sha, Xue Liu, Marco Caccamo, and Giorgio Buttazzo. Online control optimization using load driven scheduling. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4877–4882. IEEE, 2000.

[137] Mahdi Shahbakhti, Mohammad Reza Amini, Jimmy Li, Satoshi Asami, and J Karl Hedrick. Early model-based design and verification of automotive control system software implementations. *Journal of Dynamic Systems, Measurement, and Control*, 137(2):021006, 2015.

[138] Michael Short and Julián Proenza. Towards efficient probabilistic scheduling guarantees for real-time systems subject to random errors and random bursts of errors. In *ECRTS 2013*, pages 259–268. IEEE, 2013.

[139] Yasser Shoukry, Jose Araujo, Paulo Tabuada, Mani Srivastava, and Karl H Johansson. Minimax control for cyber-physical systems under network packet scheduling attacks. In *Proceedings of the 2nd ACM international conference on High confidence networked systems*, pages 93–100. ACM, 2013.

[140] Shanker Shreejith and Suhaib A Fahmy. Security aware network controllers for next generation automotive embedded systems. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[141] Arthur G Sims and Kenneth W Dobinson. The sydney coordinated adaptive traffic (scat) system philosophy and benefits. *IEEE Transactions on vehicular technology*, 29(2):130–137, 1980.

[142] Simulink. http://www.mathworks.com/products/simulink/.

[143] Jill Slay and Michael Miller. *Lessons learned from the maroochy water breach.* Springer, 2008.

[144] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. Tomorrow's in-car interconnect? a competitive evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE, 2012.

[145] Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, and Youssef Laarouchi. Survey on security threats and protection mechanisms in embedded automotive networks. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–12. IEEE, 2013.

[146] K.W. Tindell, H. Hansson, and A.J. Wellings. Analysing real-time communications: controller area network (CAN). In *Real-Time Systems Symposium, 1994., Proceedings.*, pages 259–263, Dec 1994.

[147] Sasayuki Tsugawa, Shin Kato, Takeshi Matsui, Hiroshi Naganawa, and H Fujii. An architecture for cooperative driving of automated vehicles. In *Intelligent Transportation Systems, 2000. Proceedings. 2000 IEEE*, pages 422–427. IEEE, 2000.

[148] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[149] R. Venkatasubramanian, J.P. Hayes, and B.T. Murray. Low-cost on-line fault detection using control flow assertions. In *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*, pages 137–143, July 2003.

[150] T. N. Vijaykumar, Irith Pomeranz, and Karl Cheng. Transient-fault recovery using simultaneous multithreading. *SIGARCH Comput. Archit. News*, 30(2):87–98, May 2002.

[151] Armin Wasicek, Patricia Derler, and Edward A Lee. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.

[152] Christopher Weaver, Joel Emer, Shubhendu S Mukherjee, and Steven K Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *ACM SIGARCH Computer Architecture News*, volume 32, page 264. IEEE Computer Society, 2004.

[153] Marko Wolf, André Weimerskirch, and Christof Paar. Security in automotive bus systems. In *Workshop on Embedded Security in Cars*, 2004.

[154] Y. Yao, L. Rao, X. Liu, and X. Zhou. Delay analysis and study of ieee 802.11p based dsrc safety communication in a highway environment. In *2013 Proceedings IEEE INFOCOM*, pages 1591–1599, April 2013.

[155] Yuan Yao, Lei Rao, Xue Liu, and Xingshe Zhou. Delay analysis and study of IEEE 802.11 p based DSRC safety communication in a highway environment. In *INFOCOM, 2013 Proceedings IEEE*, pages 1591–1599. IEEE, 2013.

[156] Huafeng Yu, Prachi Joshi, Jean-Pierre Talpin, Sandeep Shukla, and Shinichi Shiraishi. The challenge of interoperability: Model-based integration for automotive control software. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 58:1–58:6, 2015.

[157] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 299–308. IEEE, 2012.

[158] B. Zheng, P. Deng, R. Anguluri, Q. Zhu, and F. Pasqualetti. Cross-layer codesign for secure cyber-physical systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):699–711, May 2016.

[159] B. Zheng, C. W. Lin, H. Yu, H. Liang, and Q. Zhu. Convince: A cross-layer modeling, exploration and validation framework for next-generation connected vehicles. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, Nov 2016.

[160] Bowen Zheng, Yue Gao, Qi Zhu, and Sandeep Gupta. Analysis and optimization of soft error tolerance strategies for real-time systems. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 55–64. IEEE Press, 2015.

[161] Bowen Zheng, Hengyi Liang, Qi Zhu, Huafeng Yu, and Chung-Wei Lin. Next generation automotive architecture modeling and exploration for autonomous driving. *IEEE Computer Society Annual Symposium on VLSI*, 2016.

[162] F. Zhu and S. V. Ukkusuri. A linear programming formulation for autonomous intersection control within a dynamic traffic assignment and connected vehicle environment. *Transportation Research Part C: Emerging Technologies*, 55:363–378, 2015.

[163] Qi Zhu, Haibo Zeng, Wei Zheng, Marco Di Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4):85, 2012.