Improving Vehicular Networking Reliability and Efficiency in the Context of Platooning Applications

by

Song Gao

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 6, 2016

Keywords: Dedicated Short Range Communication, Vehicular Networking, Vehicle Platooning, Wireless Emulation, Compression, Congestion

Abstract

Vehicular networking is a technology that enables vehicles communication system. A joint effort from the automobile industry, transportation industry, and government offices is driving the adoption of this technology to build intelligent transportation systems that consist of smart vehicles. This study attempts to improve the reliability and efficiency of vehicular networking. The study assumes the context of platooning applications, but the contributions of this study can be applied to other vehicular applications as well.

There are two contributions in this study. First, a wireless emulator is designed and implemented to emulate IEEE 802.11 networks in real-time using the Ethernet infrastructure. The emulator replaces the MAC layer and physical layer of IEEE 802.11 networking stack with a real-time CSMA/CA model, thus reduces the cost of experiments. It provides upper layers the same interfaces as on a real device. As a result, the testing targets in the emulation are real-world software components as opposed to simulation scripts in a discrete event simulator. These software components can be routing protocols, transport protocols, or applications, and are the same code that can be deployed in the real world.

Second, an Interframe Compression Transmission Layer is designed and implemented, to provide efficient transmission of periodical messages in vehicular environments. The transmission layer compresses the difference between frames instead of frames themselves, and reduces bandwidth consumption significantly. To improve the behaviors of the transmission layer under different scenarios and configurations studied, an adaptive version of the algorithm is designed, which achieves more than 50% in reduction of bandwidth consumption using real-world platooning data trace. With lower bandwidth consumption, delivery ratio is vastly improved in congested networking environments.

Acknowledgments

First and foremost, I'd like to express my deepest gratitude to my advisor Dr. Alvin Lim. I'm grateful for his continuous support, encouragement, and guidance, without which this work would have been impossible. His patience, compassion, and positive altitude have greatly motivated me in the past five years. Dr. Lim has been a fantastic mentor to me in both research and life and I'm truly lucky to have met him.

My gratitude also goes to Dr. Saad Biaz, Dr. Hari Narayanan, and Dr. Xiao Qin for serving on my advisory committee, and Dr. Shiwen Mao for being my university reader. They have given helpful and insightful advices for my dissertation work.

I'd like to thank Dr. David Bevly for not only the help and guidance in research projects, but also providing opportunities for me to work with a wonderful multidisciplinary team. Thanks also go to Richard Bishop, Dr. Jim Shurbutt, and Austin Schuh for providing strong support in my research projects.

This research is supported in part by the *US Federal Highway Administration*, under Contract No. DTFH61-13-H-00019, entitled *Heavy Truck Cooperative Adaptive Cruise Control: Evaluation, Testing, and Stakeholder Engagement for Near Term Deployment*.

I'm grateful to have my lab-mates who have created a cheerful atmosphere and shared with me the joy of exploration, in particular, Jian Fang, Dr. Seungbae Lee, Dr. Dongjin Kim, Dr. Kanika Grover, James McCracken, Dongsuk Song, Abhishek Kulkarni, Amogh Kashyap, Ethan Mata, Yufei Yan, Kaushik Vedalaveni, Hannah Chappell, Matthew Rhodes, and Jason Perreault. My gratitude also goes to Dr. Shuang Li and Dr. Qing Yang for providing valuable advices and help in my graduate studies. I'd also like to thank the wonderful staff in CSSE, JoAnn Lauraitis, Barbara McCormack, Kecia Hill, Kelly Price, and Michelle Brown.

It has been a delightful experience working with my colleagues, from whom I have learnt tremendously and never failed to satisfy my curiosity, namely Lowell Brown, Luke Humphreys, Grant Apperson, Dan Pierce, Robert Cofield, Christopher Rose, Amos Smith, Scott Smith, Dr. Joshua Batterson, Andrew Watts, Nicholas Smith, Tianqi Gao Smith, Mikhail Gordon, Shraddha Praharaj, Jeffrey Wayne Nolen, and Willis Walker. In addition, I'd like to thank Jennifer Dukes and other staff at NCAT for making it possible to run real-world truck experiments. Thanks also go to Dennis Wilson and Richard Kent for teaching me how to move semi-trailer trucks.

My sincere gratitude also goes to Robert Chiniquy, who has been a great friend and an awesome mentor to me. I'm also grateful to have met Paul Querna, Russell Haering, Ryan Phillips, and Justin Gallardo. These people together have shaped my journey into the software industry.

I'd like to thank all the great people I've met in Auburn, especially, Jue Wang, Dr. Liang Tang, Yusheng Ding, Shuwei Zhang, Jiayi Xu, Yawei Chen, Dr. Qing Dai, Rong Zhang, Rui Guo, Selena Guo, Kang Sun, Dr. Chih-Jye Wang, Zhitao Gong, Haoran Zhang, Junchao Wei, Mary Olson, Orba Lee Humphrey, Ken Thomas, and Will Abercrombie, for allowing me to have a memorable and beautiful experience in Auburn.

Above all, none of these would have been possible without the support from my family. I'm grateful for my parents Yan Xiong, Lili Gao, and my grandparents Yingqing Xiong and Sufang Shu, for their dedication, love, and support throughout my life. My gratitude also goes to Yunfeng Bai, who has inspired and supported me like family. I'm deeply thankful that my grandfather Yingqing Xiong developed my initial interest in engineering and computer systems. He was an remarkable engineer and I'd be happy if I become half as good as him.

Last but not least, my sincere gratitude goes to my wife Xinyu Zhao, for her love, support, and always being there for me in the good and bad times. Life is a beautiful journey and I'm charmed to have her in my life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACC	Adaptive Cruise Control
ACK	Acknowledge Frame
ADR	Application Delivery Ratio
BO	Back-Off time counter
BSM	BasicSafetyMessage
BSS	Basic Servie Set
BSSID	Basic Service Set Identification
CACC	Cooperative Adaptive Cruise Control
CAN	Control Area Network
CCH	Control Channel
CL	Cycle Length
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CW	Contention Window
DIFS	Distributed coordination function Inter-Frame Space
DSRC	Dedicated Short Range Communication
FCC	Federal Communications Commission
GOP	Group of Pictures

HAL	Hardware Abstract Layer
ICTL	Interframe Compression Transmission Layer
ITS	Intelligent Transportation System
LLC	Logical Link Control
MAC	Media Access Control
MPR	Multi-Point Relay
NCAT	National Center for Asphalt Technology
NDR	Network Delivery Ratio
OCB	Outside Context of BSS
OFDM	Orthogonal Frequency Division Multiplexing
OLSR	Optimized Link State Routing
OSLV	Operating System Level Virtualization
PSID	Provider Service Identifier
RPC	Remote Procedure Call
SCH	Service Channel
SIFS	Short Inter-Frame Space
WAVE	Wireless Access in Vehicular Environment
WSM	Wave Short Message
WSMP	Wave Short Message Protocol
XOR	Exclusive Or

Chapter 1

Introduction and Motivation

Dedicated Short Range Communication (DSRC), is a communication technology designed for vehicular environments. By utilizing wireless radio, DSRC allows vehicles to communicate with nearby vehicles and road-side units efficiently. Wireless device vendors have been actively developing chipsets and integrated modules that provides DSRC support. Integrated devices that provides not only DSRC, but also supports for GPS and Controller Area Network (CAN) bus are also in market [1, 2, 3]. Automobile and transportation companies have also been actively integrating DSRC into vehicles and developing various DSRC-enabled applications.

Thanks to its low latency advantage, DSRC enables various applications that, among many other benefits, can enhance safety by augmenting human drivers' operations. Many of such applications have been designed or prototyped. An intersection collision warning system, for example, can emit warning messages through DSRC when a vehicle is going too fast towards an intersection with a red traffic light, so that other vehicles and pedestrians can be notified to avoid collision. As another example, an emergency braking warning system enables the vehicle to “see” another vehicle in front braking hard when the line-of-sight is blocked by a large vehicle, so that the vehicle can promptly decelerate before the driver realizes the situation.

Among many safety related DSRC applications, vehicle platooning, or cooperative adaptive cruise control [4], provides many incentives to commercial companies. It frees drivers from throttling and braking control by delegating it to the vehicle's computer and communication system. In addition to the safety enhancement, platooning makes vehicles run closer to each other. This improves aerodynamic efficiency and results in lower fuel cost. This study starts from this use case, and tries to improve efficiency and reliability of vehicular networking.

DSRC is a recent technology that was just standardized a few years ago. It has not been deployed to vehicles in the market yet. Most studies have focused on interactions with a few vehicles only. However, when the majority of vehicles on roads are equipped with DSRC, the wireless channels will be heavily utilized, and there can be scalability problems. This study looks at these potential problems, and tries to provide a way to improve efficiency and reliability of DSRC networking services.

There are two major contribution in this study:

- First, a wireless emulator is designed and implemented to emulate DSRC networks. It provides a way to test DSRC applications in a scalable way, using real application software. Unlike simulation approach where simulation scripts need to be written in order to test the idea, in the emulator the testing targets are application softwares that can be directly deployed in real-world devices.

- Second, a compression technique, Interframe Compression Transmission Layer (ICTL), is proposed. The compression layer compresses the difference between consecutive frames, rather than the frames themselves. It reduces bandwidth consumption in vehicular applications, particularly those that periodically sends messages that do not change dramatically. By reducing the bandwidth consumption, it makes channel congestions less likely to happen. Behaviors of ICTL under different configuration and scenarios are studied, and the findings lead to the design of an adaptive ICTL algorithm, which further lowers bandwidth consumption compared to the basic ICTL algorithms.

Chapter 2

Background

2.1 Dedicated Short Range Communication

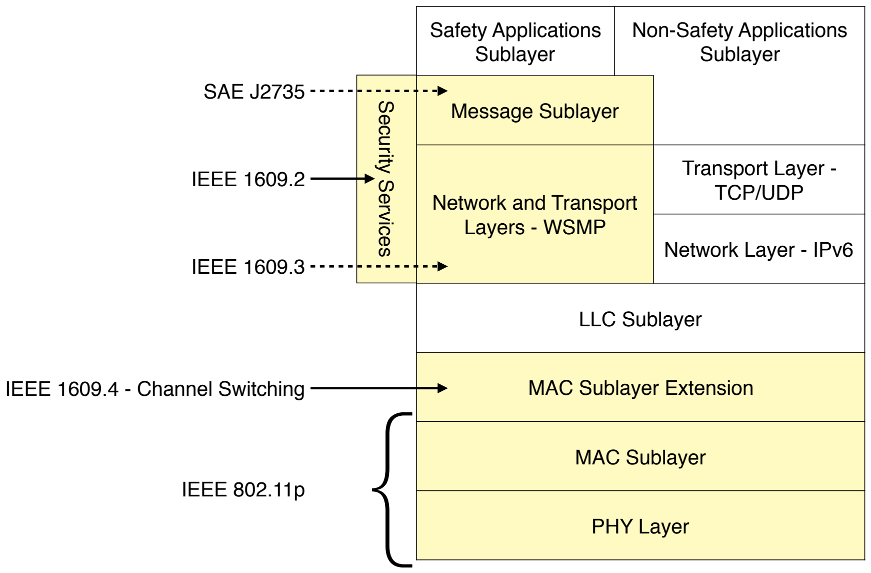

Safety Applications Sublayer
Non-Safety Applications Sublayer
SAE J2735
Message Sublayer
Security Services
IEEE 1609.2
Network and Transport Layers - WSMP
Transport Layer - TCP/UDP
Network Layer - IPv6
IEEE 1609.3
LLC Sublayer
IEEE 1609.4 - Channel Switching
MAC Sublayer Extension
MAC Sublayer
IEEE 802.11p
PHY Layer

Figure 2.1: DSRC protocol suite.

DSRC, also known as Wireless Access in Vehicular Environment (WAVE), is a protocol suite designed for low latency networking in vehicular environments. The protocol suite [5], as illustrated in Figure 2.1, is similar to TCP/IP over WiFi. In fact, it supports the IPv6 stack in parallel with a network and transport layer protocol called Wave Short Message Protocol (WSMP) that is dedicated to the DSRC suite. Since the WSMP stack is required to be used in safety applications such as platooning, this study focuses on the WSMP branch of the protocol suite.

2.1.1 IEEE 802.11p

IEEE 802.11p is derived from IEEE 802.11a, an early 5 GHz protocol used in WiFi. The Federal Communications Commission (FCC) has allocated the spectrum from 5.850 GHz to 5.925 GHz, i.e., the "5.9 GHz band", for DSRC operation in United States [6]. This spectrum is divided into seven 10 MHz channels (channel 172, 174, 176, 178, 180, 182, 184) with 5 MHz guard band at the low end [5]. Chanel $< 174, 176 >$ and $< 180, 182 >$ can be combined into 20 MHz channels. This spectrum is higher than the unlicensed 5.8 GHz spectrum used in the WiFi protocols, so DSRC applications do not suffer from interference generated by WiFi devices. As in the IEEE 802.11a protocol, the IEEE 802.11p uses Orthogonal Frequency Division Multiplexing (OFDM) for modulation and Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) for medium access control.

Table 2.1 shows data rates [5] for different modulation types and coding rates. With the same modulation type and same coding rate, 10 MHz channel gets only half the data rate compared to 20 MHz channels. As a result, the available bandwidth in upper layer is reduced by half as well.

Modulation Type	Coding Rate	Data rate 20 MHz	Data rate 10 MHz
BPSK	1/2	6	3
BPSK	3/4	9	4.5
QPSK	1/2	12	6
QPSK	3/4	18	9
16-QAM	1/2	24	12
16-QAM	3/4	36	18
64-QAM	2/3	48	24
64-QAM	3/4	54	27

Table 2.1: Data Rate in Mbps for 10 MHz and 20 MHz channels

Unlike other IEEE 802.11 protocols where stations have to join a Basic Service Set (BSS) before they can transmit or receive data, IEEE 802.11p defines an Outside Context of BSS (OCB) mode for the WSMP branch of the protocol suite. In OCB mode, Basic Service Set Identification (BSSID) field of the frame header is set to a wildcard value `FF:FF:FF:FF:FF:FF`. It allows stations to transmit and receive data without registering with an infrastructure device or an existing Ad-hoc network. As a result, the time required to activate a wireless device is significantly reduced.

In addition, the Media Access Control (MAC) sub-layer has an extension that supports channel switching, defined in IEEE 1609.4 [7]. One of the seven 10 MHz channels is dedicated as the control channel (CCH) while others work as service channels (SCHs). Channel switching allows concurrent access of CCH and SCHs. This is achieved by dividing each 100 milliseconds into a 46 milliseconds CCH interval and a 46 milliseconds SCH interval, each followed by a 4 milliseconds guard interval. Inherently, channel switching reduces usable bandwidth of the CCH to about half of that without channel switching.

2.1.2 Wave Short Message Protocol

WSMP, defined in IEEE 1609.3 [8], is the networking service in DSRC and serves the purposes of the network layer and transport layer from the TCP/IP stack. WSMP defines a message type that is efficient for 1-hop transmission. The message type is called Wave Short Message (WSM), whose minimum header size is 5 bytes, as shown in Table 2.2. Compared to UDP over IPv6, which is a similar configuration in the TCP/IP protocol stack that requires a minimum of 52 bytes of header, WSM's overhead is much smaller and causes less congestion. Since channel congestion is a significant concern in DSRC, the efficiency of WSMP is quite valuable [5]. On the other hand, being such a minimum protocol, WSMP does not provide many powerful transport layer functionalities other than multiplexing, which is achieved through the Provider Service Identifier (PSID) field in the WSM header.

Version	PSID	Extension	Element ID	Length	Payload
1 byte	1-4 bytes	variable	1 byte	2 bytes	variable

Table 2.2: WSM Header

2.1.3 Message Sub-layer

On top of the WSMP layer is the Message Sub-layer, which directly provides support to applications. An important standard in this layer is the SAE J2735 DSRC Message Set [9]. J2735 defines many message types that vehicular applications can utilize. The messages are encoded using ASN.1 format and always transmitted with WSMP. Each message is defined as a collection of constituent data structures called data elements and data frames. A data element is the most basic data structure in the J2735 standard. A data frame has a more complex data structure and is composed of one or more data elements or other data frames. [5] These data structures carry various information related to vehicular environment, from geometries of vehicle body to the dynamics of a running vehicle.

2.2 Vehicle Platooning

Vehicle platooning is one of safety applications that DSRC enables. In vehicle platooning, Cooperative Adaptive Cruise Control (CACC) plays an important role. CACC is based on Adaptive Cruise Control (ACC, also known as Autonomous/Active Cruise Control), which in addition to maintaining speed as in normal Cruise Control, can adjust vehicle speed based on distance between the vehicle and other vehicles in front of it, i.e., headway distances. The distance detection relies on radar sensors mounted at the front of the vehicle. Due to the latency from the time when a front vehicle brakes to the time when the ACC enabled vehicle reacts to the decreased headway, safe following distance is set to be quite high in ACC systems.

In CACC, however, headway distance can be further decreased without introducing extra safety issues. As shown in Figure 2.2, this is achieved by a signaling mechanism that results in shorter reaction time when front vehicle is braking. In this mechanism, as soon as the front vehicle's driver hit the brake pedal, even before the front vehicle starts to decelerate, the braking signal, along with other information, is broadcast through DSRC, making the vehicle that is following closely aware of the situation and brake in advance. Since this

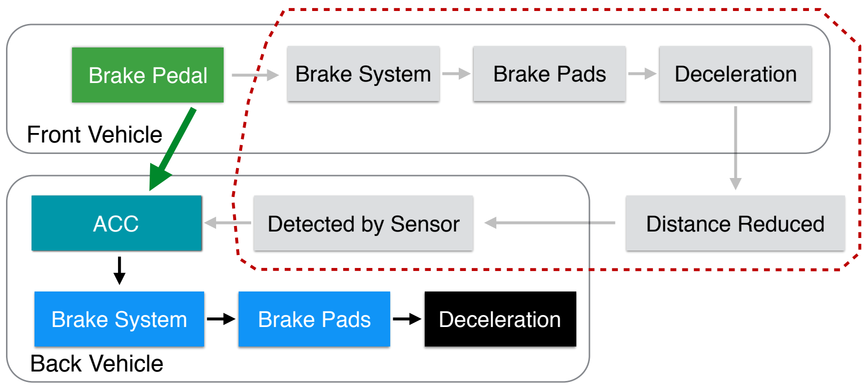


Figure 2.2: CACC shortcuts control flow of ACC thus reduces reaction time.

mechanism does not completely rely on actual headway distance change and sensor accuracy, the headway distance can be further reduced, to a point where the aerodynamic context can change significantly.

2.2.1 Real-world Platooning Experiment

In a real-world deployment, antenna's mounting position matters because vehicle body can block line-of-sight and affects wireless propagation. The vehicle body can block the line-of-sight between the vehicle's antenna and vehicles next to it. The problem becomes greater for large size vehicles, such as semi-trailer trucks. To solve this problem, many researchers and engineers choose to mount two DSRC antennas on the vehicle [10, 11]. Many DSRC device vendors provide devices with two antenna interfaces [1, 2, 3].

A straightforward and easy-to-implement way to make the communication more reliably with dual-antenna design is to replicate each message on both antennas. In other words, the DSRC stack is configured to use both antennas alternately and each message is sent twice through DSRC. This way, as long as one side of antennas makes it through, the message can be delivered. It improves reliability but uses twice as much bandwidth, causing more channel congestion.

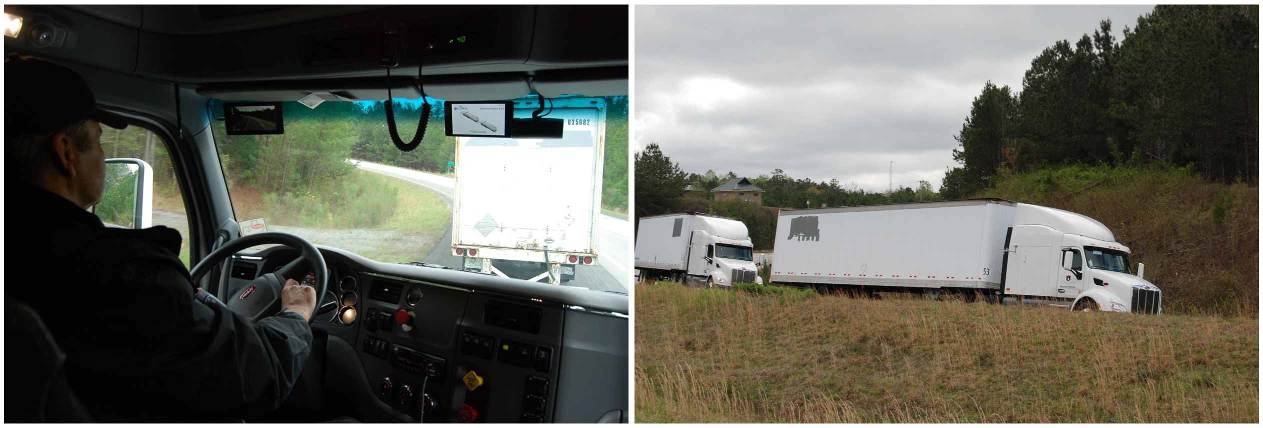


Figure 2.3: Platooning experiment conducted on the test track at National Center for Asphalt Technology

Figure 2.3[1] shows a platooning experiment conducted on a test track at National Center for Asphalt Technology (NCAT). The platoon consists of two semi-trailer trucks. On each truck, a pair of DSRC antennas are mounted on each side of the tractor, located near the back panel of the tractor. The upper ones are used by one of the two DSRC radios, for transmitting messages necessary for platooning control. Since these are safety messages, this DSRC radio runs the WSMP branch of the DSRC protocol suite. The lower two antennas are used by the other DSRC radio, for sending a video footage from front vehicle to the rear one, for driver comfortability. This DSRC radio runs an IP stack, essentially behave like a router between the electronic systems in the truck and the other truck. The antenna setting is shown in Figure 2.4. Since IEEE 1609.4 needs time synchronization, time is acquired from GPS satellites through a GPS antenna that is mounted on left side of the truck and is connected to the DSRC radio.

The DSRC radios and various sensors installed on the trucks allow the trucks to be virtually connected. The system can achieve less than 2 meters following distance, at speed of 72 kilometer per hour. This corresponds to a time interval of 0.1 second. Compared to commercial ACC solutions, for example, the 2016 Audi S7 [12], where the most aggressive

[1]Photos courtesy of Dr. Alvin Lim

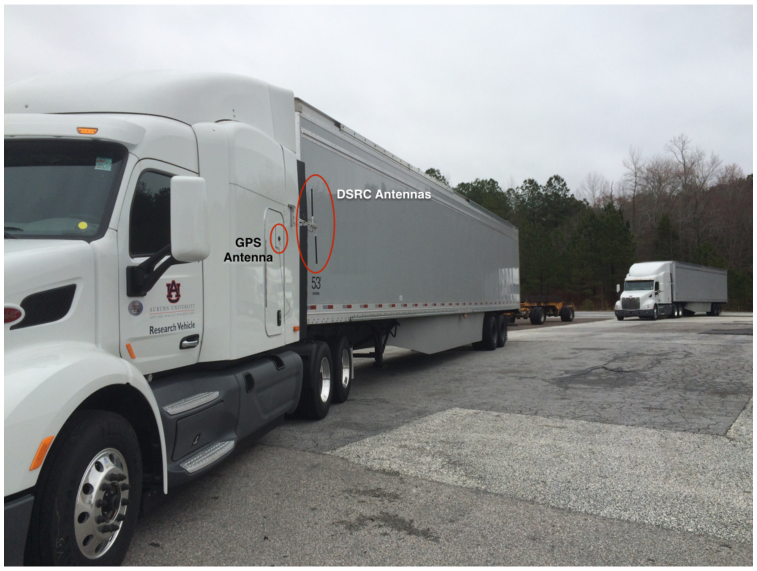


Figure 2.4: Left side of a truck with DSRC antennas and a GPS antenna

setting is a time interval of 1 second, CACC reduces the following distance by an order of magnitude.

Close following distance might not be very appealing to consumer cars, but it brings huge benefits to big commercial vehicles. To be specific, short following distance can reduce air resistance (drag) for the vehicles [13, 4] and the drag reduction results in fuel saving (up to 11% in some scenarios [14]). This is of great interest to transportation companies. For example, in the trucking industry, the single largest operating cost is diesel fuel, being 34% in year 2014 in the US [15]. 11% fuel saving yields 3.74% saving of the total operating cost.

2.3 Real-world Performance Attributes of DSRC in Platooning Scenarios

To obtain better understanding on how well DSRC can support platooning applications, we conducted an experimental study[16] primarily focusing on delivery ratio of broadcast messages under various circumstances and with different parameters. In the study, we conducted extensive real-world DSRC tests using the same devices and settings as the platooning experiment described in Section 2.2. The tests included a set of dynamic tests with two trucks running on the test track at NCAT as a general case and a few sets of static tests for studying performance in specific scenarios, such as when the road was not horizontal or when the front vehicle was turning.

In the study we used three antenna operating mode: alternate, left, and right. For alternate mode, we introduced pairwise delivery ratio. It is similar to the normal delivery ratio, except that it represents the ratio of the messages delivered from either of a pair of antennas to the total pairs of messages sent. In pairwise delivery ratio, delivery of any one of a pair of duplicate messages indicates successful transmission of the pair.

The study concluded that many factors can result in reduction in delivery ratio for broadcast messages. For example,

- if the front vehicle is turning, outside antenna is blocked by the vehicle body and results in poor delivery ratio;
- on straight road, some terrain can affect delivery ratio significantly;
- on hilly road, vehicle needs to rely on terrain's reflection because antennas are not parallel.

However, in all scenarios, using alternate mode improves delivery ratio significantly. To be more specific, the pairwise delivery ratio is above 90% for data rates up to 6 Mbps in all scenarios, even when using a side antenna only delivers less than 20% messages. In some scenarios, alternate mode even achieves above 90% delivery ratio for 27 Mbps. [16] For the dissertation, this is an important conclusion from the study. It suggests that it is most feasible to use alternate mode all the time. Since alternate mode requires more bandwidth, how to reduce bandwidth while maintaining reliability becomes a more important issue to solve.

Chapter 3

Emulation Infrastructure

3.1 Problem Statement

In networking, there are three experimental approaches to validate an algorithm or a protocol: real-world experiments, simulations, and emulations.

In real-world experiments, real wireless devices, such as DSRC radios, or laptops with WiFi cards, are used. An operating system with complete networking stack is installed on such devices, and applications can use various APIs to access the wireless hardware for real wireless communication. The algorithm being validated runs on top of existing protocols, or replaces an existing software component running on the device. Real-world experiments provide the most realistic results, since they are real experiments using hardware and software components that are used in the real world. The disadvantage of real-world experiments is that they are hard to conduct and require expensive devices and human resources to scale.

In simulation approach, everything is replaced by software components. A common way to implement network simulation is to use a discrete event simulator. The simulator converts the complex system into an ordered sequence of well defined events. The more specific the events are, the more accurate the simulation results will be and the more time it will take to run the simulation. It is normally easier to scale to large number of nodes with simulation, but since all hardware and software components are simulated, the accuracy of the simulation results largely relies on the quality of simulation models of each layer being simulated. In addition, once an idea is validated in simulation, it needs to be re-implemented as real software components in order to be used in the real world. Any bug in simulation scripts or the real implementation can result in different behaviors between the two. Thus, results from simulations become less trustworthy.

Emulations are a balance between real-world experiments and simulation approach. In emulations, some parts of the system are replaced by a piece of hardware, or a software model, and the rest still uses real components that are identical to what are used in the real world. More importantly, the testing target is a real-world software component that can be used in real-world applications. In other words, what is tested in emulation, is exactly what runs in the real world. This way, emulations provide relative high accuracy of test results, with relatively low cost of scalability.

For vehicular applications, particularly platooning applications, it is too risky and expensive to run real-world experiments with large number of vehicles. For example, placing several platoons of two-vehicle pairs on different lanes of a segment of a highway would affect the normal traffic and is costly by itself. On the other hand, the DSRC stack consists of many layers that are not formerly available through TCP/IP stack. Thus, there are limited simulations models that are well designed and tested. Without willing to sacrifice much accuracy, emulation is the best approach to take.

There are a few hardware channel emulators for IEEE 802.11 channels available. They replace part of physical layer of the networking stack and are primarily designed for wireless hardware (e.g. chipsets) testing. They require expensive hardware components and do not scale well. Hence, hardware channel emulators are not suitable for this study as well.

To emulate vehicular environment for testing platooning application, it is desirable to develop a software solution that can support large number of nodes, each of which has an independent networking stack, using reasonable amount of hardware resources.

3.2 Related Work

Most of non-experimental vehicular networking study has been focused on simulation approach. Popular simulators include but not limited to ns-2, ns-3[17], and OMNeT++[18]. Simulation models have been developed for ns-2 [19] and ns-3 [20, 21]. There are also integrated vehicular networking simulators such as Vehicular Networks Simulator [22] which is compatible with both ns-3 and OMNeT++. However, as discussed in Section 3.1, simulation approach requires simulation scripts to be written to describe the application or protocol being tested, where the accuracy largely relies on the quality of simulation models as well as the testing target scripts.

An emulated Dynamic Switch was proposed in [23]. It is a reconfigurable Ethernet switch. A master node with many Ethernet ports internally tracks positions of all emulated nodes. Emulated nodes connect to the master node through Ethernet. When packets go through the master node, the master determines whether the packet should be delivered based on emulated positions assigned to each node. The Dynamic Switch can emulate connectivity quite well, but it does not emulate interference. As a result, nodes will always achieve the same throughput regardless of whether the channel is congested or not.

The U.S. Naval Research Lab has developed Extendable Mobile Ad-hoc Network Emulator (EMANE) [24]. It is very similar to the emulator proposed in this study, and achieve roughly the same goal. Comparing the two, the emulator in this study has a simpler design and uses more straight-forward configuration format than EMANE. It is written in a safer language (Go vs C++) that supports static linking, which enables easier deployment in clusters. In addition, EMANE is a general emulation framework and does not have specific support for vehicular network as in the proposed emulator.

3.3 Design of Squirrel

3.3.1 Overview

In this study, an IEEE 802.11 emulation framework called Squirrel is developed, with a goal of trying to use as much real-world software components as possible. Conceptually, as shown in Figure 3.1, Squirrel replaces the wireless hardware, device driver's Hardware Abstract Layer (HAL) and the IEEE 802.11 kernel module. Layers above the MAC layer are not replaced by Squirrel, and run real software that is identical to what can be deployed in the real world. Figure 3.2 shows how Squirrel fits in the DSRC stack.

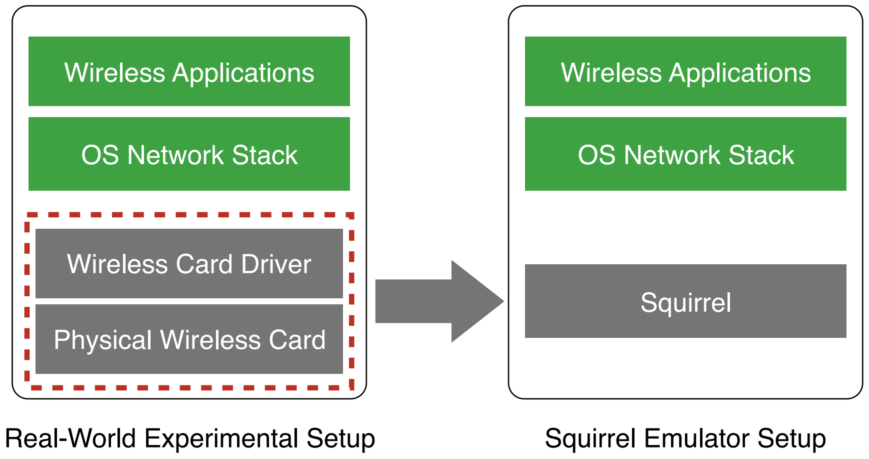


Figure 3.1: Layer concept of Squirrel

3.3.2 Architecture

Squirrel uses a multi-tier architecture, where a master component (`squirrel-master`) works as a server that coordinates communication among all participating nodes, and a worker component (`squirrel-worker`) that works as a client and handles communication details on actual nodes. There is only one `squirrel-master` in an emulation. It has an internal wireless model that determines whether data should be delivered on a frame-by-frame basis. The `squirrel-worker` component, on the other hand, has one instance running for each node being emulated. It connects the wireless components being tested with the `squirrel-master`, and handles the “first mile” and “last mile” of the network traffic.

Figure 3.3 illustrates the multi-tier architecture of Squirrel. Each host is a computer domain that has independent or isolated networking stack. It can represent a real computer system with independent hardware, a virtual machine, or an operating system level virtualization (OSLV) host such as a Linux process container [25] and a BSD-jail. Among the three options, OSLV has the lowest overhead and is thus the most scalable. It allows different hosts to share a same operating system kernel and relies on the kernel namespace for isolating the filesystem and networking stack.

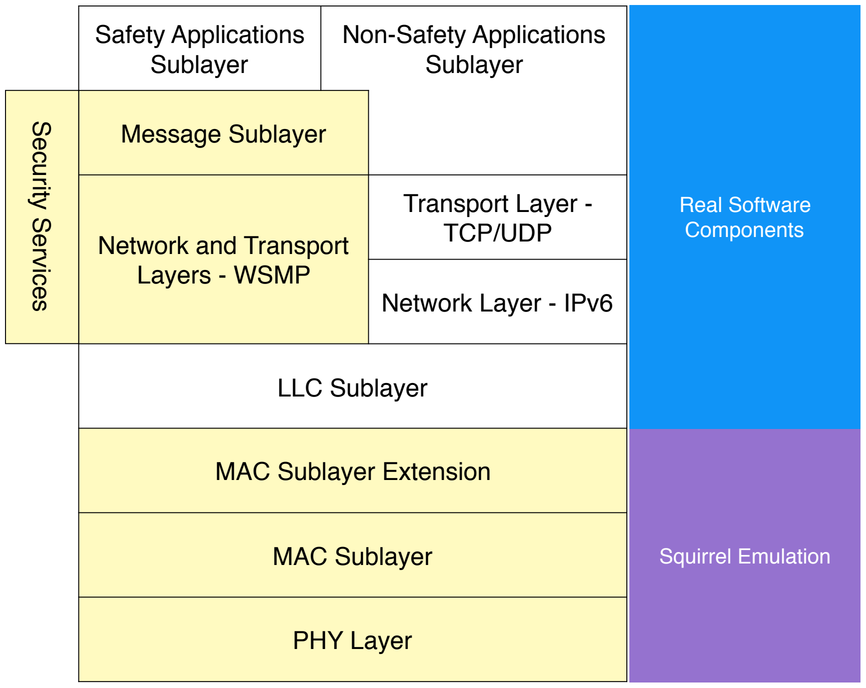


Figure 3.2: Squirrel in DSRC context

Squirrel uses a TAP[26] interface, `tap0` as in Figure 3.3, in each emulated wireless node as emulated wireless network interface. A TAP interface is a virtual network device that only exists in operating system kernels. It provides MAC layer frame reception and transmission for user space programs. From an application's perspective, the `tap0` interface can be seen as an Ethernet device, which, instead of receiving packets from a physical wireless network device, receives them from `squirrel-worker`, and instead of sending packets via a physical wireless network device, writes them to `squirrel-worker`. `squirrel-worker` has a file descriptor through which it can handle the `tap0` interface. Whenever an application writes any data into `tap0`, or any traffic is routed through the interface, `squirrel-worker` receives the data through the file descriptor. Whenever `squirrel-worker` writes into the file descriptor, a MAC frame is generated in kernel just like incoming traffic on a real device, and results in data received in the corresponding application.

`squirrel-worker` uses `eth0` interface to communicate with `squirrel-master`. This interface is not managed by Squirrel, but is an Ethernet interface managed by the operating

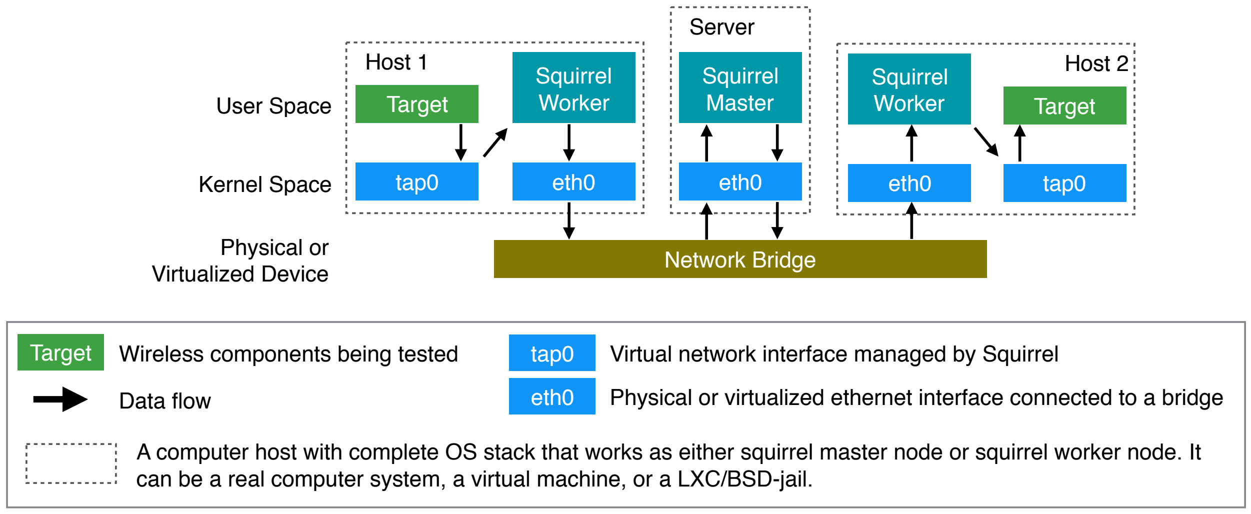

Server
Host 1
Host 2
User Space
Kernel Space
Physical or Virtualized Device
Target
Squirrel Worker
Squirrel Master
tap0
eth0
Network Bridge
Target Wireless components being tested
Data flow
tap0 Virtual network interface managed by Squirrel
eth0 Physical or virtualized ethernet interface connected to a bridge
A computer host with complete OS stack that works as either squirrel master node or squirrel worker node. It can be a real computer system, a virtual machine, or a LXC/BSD-jail.

Figure 3.3: Squirrel architecture and an example data flow

system. If the emulated node is a real computer device, `eth0` is mapped to a real Ethernet adapter. If the emulated node is a virtual machine or OSLV host, it is a virtualized interface managed by the virtualization layer. Conceptually, this interface should be considered as an Ethernet interface that connects to `squirrel-master` either directly or through a Ethernet bridge. `eth0` is normally not used by testing targets directly, but an interface used by Squirrel to coordinate traffic. An exception is when the testing target needs to use an out-of-band channel to notify the `squirrel-master` about its local situation. For example, Section 3.3.4 describes a way to update emulated positions using RPC services, which runs on `eth0`.

An example data flow is also shown in Figure 3.3, where emulated node 1 (host 1) is sending data to emulated node 2 (host 2). The data flows from a testing target into `tap0`, which is read by `squirrel-worker` as MAC frames, and encapsulated into Squirrel's packet, then sent to `squirrel-master` through `eth0`. The `squirrel-master` upon receiving a frame, applies the wireless model and determines whether it should be delivered to the destination node. If so, it sends the frame packet to host 2, which is parsed into a MAC frame by `squirrel-worker`, and written into the `tap0` interface on host 2. The testing target then reads the data from `tap0` interface.

`squirrel-master` has two important components: a mobility manager and a wireless model. Both of them are configurable through a plug-in system. The two components share a position manager, which is an internal component in Squirrel that keeps track of position of each emulated node. The mobility manager emulates the mobility of all emulated nodes. It interfaces with the position manager and updates positions of emulated nodes periodically. The position information is used by the wireless model to determine whether a frame should be delivered to its destination.

During the development and testing of Squirrel, several simplified mobility managers and wireless models are developed. For example, a static mobility manager allows pre-definition of a position for each node. A PassThrough wireless model instructs Squirrel to deliver each frame. These models are useful for testing the effectiveness and capability of Squirrel, but do not provide realistic throughput and packet loss for application in emulation.

The rest of this section is focused on realistic models that are used in this study. First, a CSMA/CA wireless model is designed to emulate the CSMA/CA algorithm used in real-world IEEE 802.11 networks. Second, a mobility manager that supports updating positions from different processes is designed.

3.3.3 CSMA/CA Model

In Ethernet, the collision domain is bounded to well controlled hardware components such as Ethernet wires and hubs. In wireless networking, however, since electro-magnetic wave propagates freely, the collision domain is much larger and the bounds are less pronounced. This introduces some challenges specific for wireless networking. A typical one is the hidden node problem, where a node transmitting to a second node may not see other nodes that are in interference range of the second node but not the first one, causing collision at second node. This is the motivation that lead to the design of collision avoidance in CSMA/CA algorithm that is widely used in IEEE 802.11 networks.

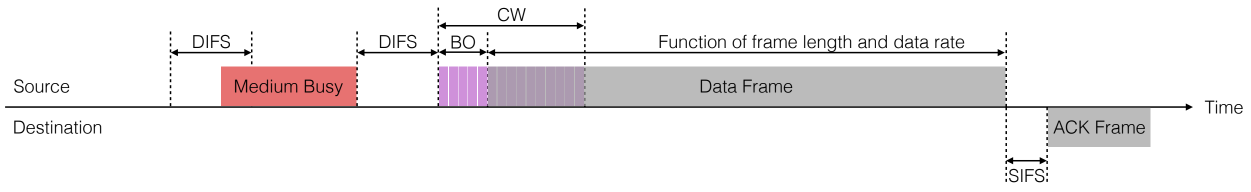


Figure 3.4: A CSMA/CA algorithm timeline of sending a data frame

Figure 3.4 shows a typical timeline of sending a data frame CSMA/CA algorithm. In CSMA/CA, when a node needs to transmit a data frame, it first senses the medium, i.e., the wireless channel. When the channel has been idle for a predefined period of time, Distributed coordination function Inter-Frame Space (DIFS), it starts a random Back-Off time counter (BO), which is equal to a number of $aSlotTime$ within Contention Window (CW). The counter decrements when the medium is idle. When BO hits zero, the data frame is transmitted. If this is a broadcast frame, that is the end of cycle for this frame. If this is a unicast frame, when the frame is successfully received at destination, the destination node waits for Short Inter-Frame Space (SIFS), which is shorter than DIFS, and then transmits an Acknowledgement Frame (ACK). The ACK allows the source node to be notified about the reception of the frame at destination node. If a frame is lost, the source retransmits the frame, and at the same time, increases the CW. When a frame is successfully transmitted, CW is reset to lowest value. Following this process, the time it takes to typically transmit a data frame, T_{data}, can be modeled with Equation 3.1:

$$T_{data} = t_{busy} + t_{DIFS} + t_{BO} \times aSlotTime + t_{data} \tag{3.1}$$

where t_{busy} is the time duration that the channel is busy, including partial DIFS's, as well as busy time during BO, and t_{data} is the time duration it takes to encode the actual data frame. The time it takes to transmit an ACK, T_{ACK}, can be modeled with Equation 3.2:

$$T_{ACK} = t_{SIFS} + t_{ACK} \tag{3.2}$$

where t_{ACK} is the time duration it takes to encode the actual ACK.

In IEEE 802.11 protocols, DIFS is defined as $SIFS + 2 \times aSlotTime$. In IEEE 802.11p with 10MHz channel, SIFS is defined to be 13 microseconds, and $aSlotTime$ is defined as 32 microseconds. CW's lowest value is 15, and the maximum that CW can grow into is 1023. [27]

Unlike in the real world, where the CSMA/CA algorithm can rely on network adapter to sense whether channel is available to use, in emulation, the model needs to approximately keep track of localized channel condition of each node. To be more specific, it needs to keep track of how much of the channel time is being used in the vicinity of each node being emulated. When a node needs to transmit, or when other nodes need to transmit to the node, the model needs to check the channel condition around this node, and determines whether the channel is available for the data to be delivered. This is achieved by using the leaky bucket algorithm.

Leaky bucket algorithm [28, 29] is commonly used in networking to shape and/or enforce throughput. The idea is to have a counter, i.e., the bucket, that is incremented whenever a packet goes through, and is decremented periodically. If the counter exceeds a threshold upon being incremented, the network discards the packet. The rate at which the counter is decremented determines the average bandwidth. Figure 3.5 shows the leaky bucket analogy.

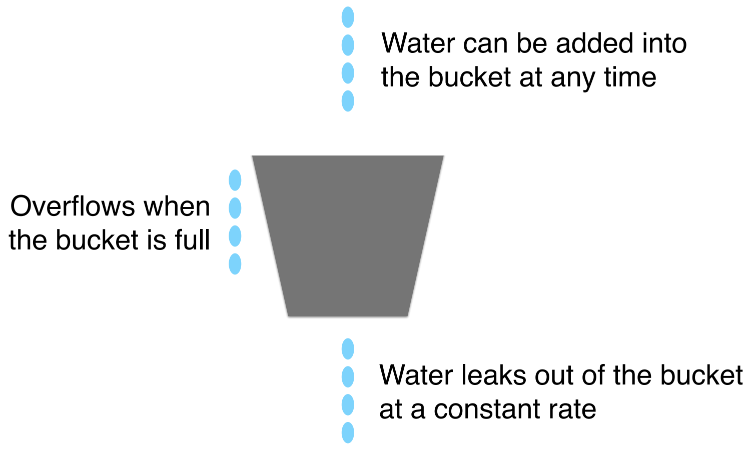

Water can be added into the bucket at any time
Overflows when the bucket is full
Water leaks out of the bucket at a constant rate

Figure 3.5: The leaky bucket analogy

In Squirrel's IEEE 802.11 model, one leaky bucket is allocated for each node to track the channel condition around the node. The unit used for the counter is nanosecond. The rate at which the bucket leaks, i.e., counter decrements, is as same as the rate at which time elapses, i.e. 1×10^9 per second. The counter is increased whenever an event happens, and the amount it increases is the time it takes for the event to happen. When the bucket is full, it indicates that the channel is congested around the corresponding node, and no more frames can be delivered to the node or sent from the node at the time.

Tracking channel condition with leaky buckets takes care of the process of "carrier sensing". As a result, in the emulation model, nodes do not actually sense the medium, but starts DIFS and BO directly for each transmitting frame. This eliminates the t_{busy} in Equation 3.1. So the equation for calculating time duration to transmit a data frame T_{data} in the emulation model is Equation 3.3:

$$T_{data} = t_{DIFS} + \frac{t_{cw}}{2} \times aSlotTime + t_{data} \tag{3.3}$$

where t_{cw} is the current effective CW. Since BO is uniformly distributed in $[0, CW]$, $\frac{t_{cw}}{2}$ is used as BO here to reduce time complexity of the model. Equation for calculating time duration to transmit an ACK remains the same as Equation 3.2.

In addition to the leaky buckets, Squirrel's CSMA/CA model also uses a probability function $P_{propagation}$ to model the process of propagation of electro-magnetic wave. The probability function, as defined in Equation 3.4, is based on the distance between source and destination nodes, as well as channel occupancy level, i.e., how full the bucket is.

$$P_{propagation} = ((1-u) * 0.1 + 0.9) * (1 - (\frac{d}{r})^3) \tag{3.4}$$

where $u(0 \leq u \leq 1)$is the usage ratio of the bucket, d is the distance between source and destination nodes, and r is the transmission range.

Putting it together, when a node transmits a unicast data frame, Squirrel's CSMA/CA model first tries to increment the bucket of the source node. If successful, it tries to increment all buckets within the interference range of the source node, based on Equation 3.3. If the destination bucket does not overflow, the probability function $P_{propagation}$ as defined in Equation 3.4 is applied. If the frame survives the probability function, the frame is delivered to the destination node. Any failure in this process results in retransmission, until the maximum number of transmission is reached. If the data frame is delivered to the destination node, the model starts the ACK process in which the destination node sends an ACK back to source node. This is a similar process, which includes incrementing buckets of nearby nodes, as well as the probability function. Although at this point, the data frame is for sure delivered to the destination node, the ACK can still be lost, in which case a retransmission still happens. Broadcast frames are handled in the same way, except that the model tries to deliver the data frame to all nodes within transmission range, and that there is not ACK process.

The CSMA/CA has configurable parameters, such as emulated MAC layer data rate, transmission range, and interference range. In addition, since different IEEE 802.11 protocol has different timing parameters, such as DIFS, SIFS, and aSlotTime, the CSMA/CA model has a configuration field for MAC layer protocol. The CSMA/CA model has a pre-defined parameter set for each supported protocol, which configures corresponding timing parameters for the model. An example of such parameter set is `802.11p10`, or `OFDM10`, which is configured to have 13 ms aSlotTime, 32 ms SIFS, and CW from 15 to 1023.

3.3.4 RPC Updated Mobility Manager

There are two approaches to design a mobility manager. The most straightforward way is to have a mobility model within the mobility manager, which emits new positions for emulated nodes based on the mobility model. This approach is useful for studying a specific mobility pattern, such as random waypoint, that can be easily well modeled with equations and software programs. Another approach is to follow an external trace which consists of a time series position data for each node, and update positions for emulated nodes accordingly. This approach provides a way to utilize external mobility simulators, or trace files from real-world experiments.

In this study, we acquired some real-world test traces. To produce most realistic results, the second approach is used. A Remote Procedure Call (RPC) updated mobility manager is designed. The mobility manager uses gRPC [30] protocol. It includes a server that exposes several RPC services that allows a different process to update positions for emulated nodes. As opposed to directly reading from a trace file, the RPC updated mobility manager delegates the work or interpreting trace files to a different process. This provides flexibility on trace file formats. In addition, in many cases, trace files include many types of data including GPS positions, DSRC send and receive history, etc.. The RPC updated mobility manager allows a dedicated process to replay all relevant events. This is used by `truck-playback`, which is described in Section 4.5.2.

To make sure the RPC services do not affect the CSMA/CA model's results, the RPC Updated Mobility Manager is only used through the `eth0` interface.

3.4 Implementation

Squirrel and its components are implemented in Go programming language [31]. The language is chosen because of its good balance between safety, development efficiency, and performance. Designed as a system language, Go has well designed support for system software, such as first-class concurrency, well-designed networking APIs and system call interface, and cross-compile tools. These features benefit the development of Squirrel, resulting in stability and high performance[1].

Despite the advantages, Go provides less control over memory management of the applications compared to other system languages such as C/C++. The garbage collection takes

[1]An earlier version of the emulator was developed using C++ with `Boost.Asio`, which achieved only about half throughput of that in the Go version.

care of reclaiming memory no longer used, so a straightforward way to code is to allocate new objects whenever needed, and let the garbage collection worry about freeing the memory. This approach works in most cases. However for Squirrel's use case, packets are generated and consumed at very high rate, which results in byte buffers of same size allocated and deallocated frequently. The allocation and deallocation increases processing burden and is inefficient. In Squirrel, a buffer pool is used for reusing existing byte buffers, using the free-list data structure. In addition, the buffer pool implements a thread-safe reference counting, similar to `std::shared_ptr` in C++, to allow multiple components share the same object, and reuse objects at precisely the time when they are not needed.

Squirrel has many configurable parameters. For example, the plug-in system allows different wireless models and mobility managers to be selected. The wireless models and mobility managers have their own parameters as well, such as data rate and interference range in CSMA/CA, and RPC service endpoint address of the RPC Updated Mobility Manager. Besides parameters, the `squirrel-master` needs to announce its endpoint address once it has started, so `squirrel-worker`s can connect to it. For these dynamic configurations, a fast and reliable distributed key-value store, Etcd [32], is used. It uses the Raft consensus algorithm [33] to manage a highly-available replicated log, which supports running multiple Etcd instances forming a cluster, to improve reliability and availability.

Since every frame goes through `squirrel-master`, it can easily become a bottleneck for the entire emulation infrastructure. Hence, it is important to optimize `squirrel-master` in a way that it is scalable by deploying it on a machine with more processing power and more simultaneous running threads. With this goal, `squirrel-master` uses a lock-free design whenever possible. For example, in the CSMA/CA model, the leaky bucket is implemented with `sync/atomic` package, using atomic operations that do not require a Mutex to guard resources. The position manager uses fine-grain locking mechanism, where accessing thread only blocks when read and write, or write and write happen for the same node's position at the same time. This design makes `squirrel-master` more parallelization friendly, thus scales better on multi-core systems.

The implementation of Squirrel is open-source and available at `https://github.com/squirrel-land/squirrel`.

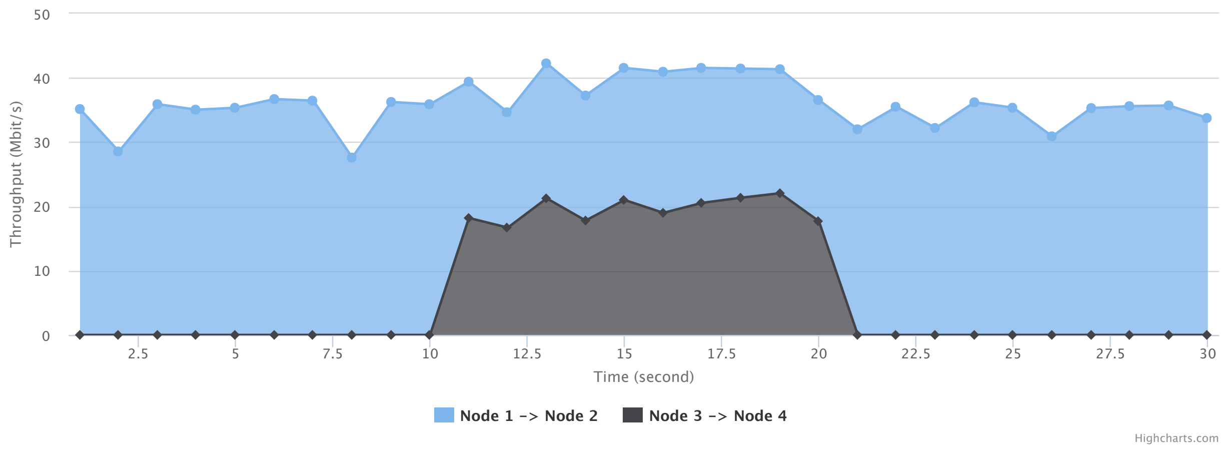
3.5 Tests

3.5.1 Effectiveness Test

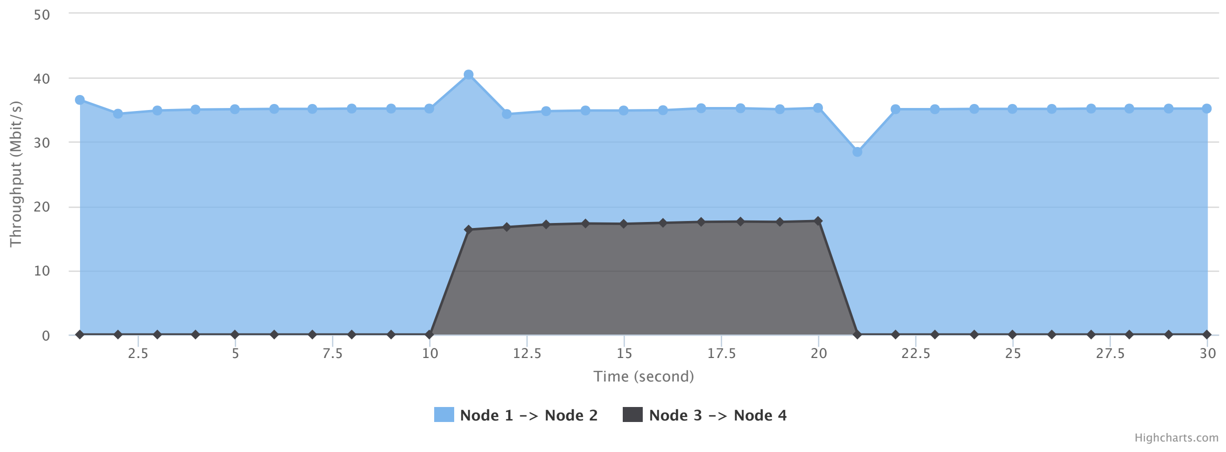
To validate that Squirrel provides networking service for applications comparable with real-world scenarios, some tests are conducted to run the same traffic on real-world WiFi devices and Squirrel.

There are four nodes involved in the tests. In Android configuration, the four nodes are connected to an Ad-Hoc WiFi network at 2.4 GHz, placed right next to each other, with distance less than 100 millimeters. In the Squirrel configuration, CSMA/CA wireless model is used, with the four emulated nodes placed 150 units from each other. Communication range is set to 150,000 units and interference range is set to 300,000. In other words, in both configurations the nodes are very close to each other so that interference rather than distance is the major affecting factor to wireless performance.

A simple networking testing tool, `iperf`, is used as testing target. UDP traffic is used since it does not have congestion control and gives a good perception of above-MAC-layer throughput. Two pairs of `iperf` tests are run, between two different pairs of nodes. The first pair (node1 to node2) starts first. It is set to run for 30 seconds. The second pair (node3 to node4) starts when the first has started for 10 seconds, and runs for 10 seconds. This way, during the first and last 10 seconds, only one pair of nodes are transmitting, while from 10th second and 20th second, both pairs are transmitting.



(a) Real Android UDP throughput



(b) Squirrel UDP throughput

Figure 3.6: Squirrel CSMA/CA model testing: UDP throughput in stacked area plot for Android v.s. Squirrel

Figure 3.6[2] shows a stacked area plot of the UDP throughput result from Android tablets and Squirrel. They both show that, in the first 10 seconds the first pair (node1 to node2) gets about 33 Mbps throughput. After the second pair (node3 to node4) starts at 10th second, the first pair's throughput drops to below 20 Mbps, while the second pair gets roughly the same level of throughput.

The general trends match very well between Android and Squirrel configuration and show the expected results. Squirrel is able to emulate interference based on positions of

[2]Plots in this dissertation are generated with Highchart JS (`https://http://www.highcharts.com/`) provided by Highsoft AS. Highchart JS is not free for commercial and Governmental use. The use of Highchart JS in this dissertation is under Creative Commons Attribution-NonCommercial 3.0 License.

emulated nodes even though the traffic is between two different pairs that are completely independent. The difference is that, there seems to be more randomness in the Android test. This is likely due to networking channel condition variations due to external factors such as other devices beyond our control using 2.4 GHz, or even a microwave oven next door heating macaroni and cheese.

3.5.2 Throughput Test

Squirrel is a real-time emulator. Unlike in discrete event simulation where the model can take as long as it needs to calculate results, in real-time emulation, models need to make decisions as fast as events happen in the real world. In other words, raw throughput of Squirrel effectively bounds the scalability of emulation.

To find out this boundary, `iperf` is used to measure the throughput on Squirrel with PassThrough wireless model, which delivers all frames. The test was conducted on a single workstation with 6-core processor. All `squirrel-worker` nodes, as well as the `squirrel-master` run on the same workstation.

With a single pair of sender/receiver, UDP throughput achieves around 480 Mbps. This is mainly bounded by processor. As the number pairs increases to 2, more processor cores are able to work in parallel. As a result both pair get the same throughput, doubling the total throughput. However, when the number of pairs keep increasing, more parallel routines than the number of cores are requested, in which case the total throughput cannot increase linearly as number of pairs increases. Figure 3.7 is a stacked area plot showing UDP throughput with different pairs simultaneously transmitting. The total throughput tops at around 1200 Mbps, with each transmitting pair gets about 200 Mbps throughput.

With more workstations handling `squirrel-worker` nodes and more number of cores on the workstations, the throughput can further increase. In other words, the throughput capability of Squirrel is mainly bounded by hardware resources. For the purposes in this study, a single workstation is able to provide enough throughput for the emulation.

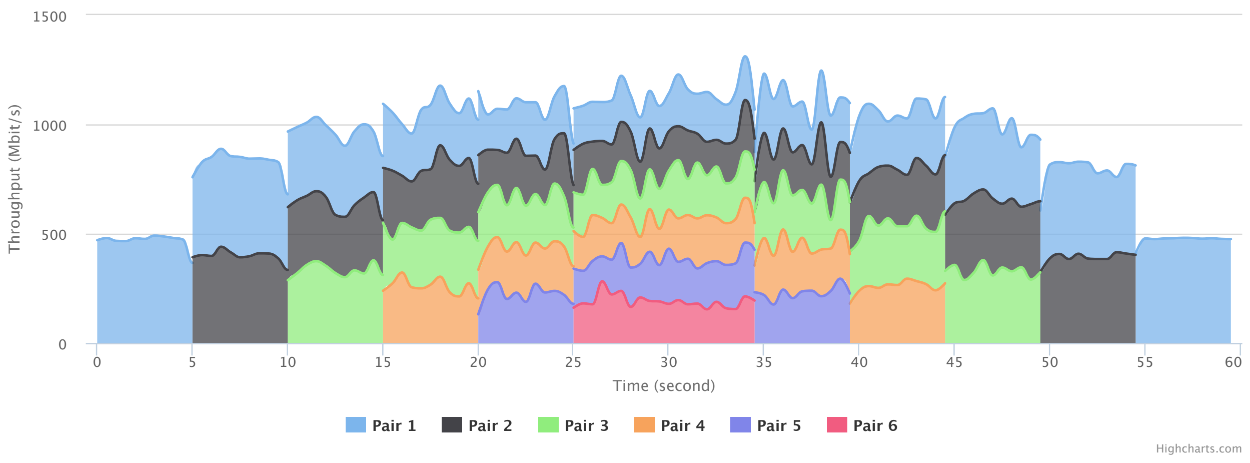


Figure 3.7: Stacked throughput using PassThrough wireless model

Chapter 4

Interframe Compression Transmission Layer

4.1 Problem Statement

In many cases for IEEE 802.11 communication, channel congestion is a key issue that affects performance and needs to be relieved through various techniques. For DSRC channel congestion becomes a more important issue to consider, especially when the technology deployment scales to all vehicles on the road. There are several key factors that makes congestion problems in DSRC applications, especially safety application, more challenging:

1. DSRC uses 10 MHz, for the purpose of supporting many parallel types of applications. However, the narrower channel width means the available bandwidth for the same channel is significantly reduced. Although applications can use different channels through channel switching, only the control channel is guaranteed to be accessed in every 100 millisecond window. Crucial safety applications such as vehicle platooning is less tolerant of channel unavailability, thus is likely to use control channel or a dedicated safety channel that has performance guarantees. As a result, the 10 MHz channel width brings extra challenge to safety applications.

2. As described in Section 2.3, a previous study [16] has shown that using two antennas on each vehicle and sending each message once on each antenna can significantly improve message delivery ratio. It is likely that the industry will adopt this approach more widely. This implies that twice as much bandwidth would be required for the same application layer communication strategy, causing more demand on the channel.

3. The term "Short Range" in DSRC is meant to convey the fact that the communication range is shorter than cellular and WiMax services, but is still expected to reach a few

hundred meters [5], which is much longer than the range of a typical WiFi deployment. Longer communication range means larger interference range. As a result DSRC causes congestion more easily than other IEEE 802.11 based protocols suites.

4. Safety applications require real-time low latency data, thus is less tolerant of channel congestion. For example, in vehicle platooning, the back vehicle relies on periodical dynamics data from the front vehicle to maintain a close following distance. When a channel is congested, several messages can be lost consecutively. In this case, the back vehicle has to make the worst assumption about the front vehicle and apply strong braking in order to avoid collision.

To show channel congestion can happen in busy road segments, some analysis is done to estimate bandwidth requirement in a typical platooning scenario on a busy segment of a highway environment. For analysis purposes, several assumptions are made and justified to assemble a reasonable heavy-use case:

1. Average following distance: 20 meters. Study [13] has shown that fuel saving is significant when following distance is below 40 feet (about 13.3 meters). Volve [34] has done real-world experiments with following distance of 7 meters. When there is traffic jam, following distance can be further reduced. Thus, assuming nearly full market penetration of Intelligent Transportation System (ITS), 10 meters (or 0.33 seconds headway at 110 km/h) is a reasonable assumption for a moderately heavy-use case.

2. Platooning message size: 150 bytes. SAE J2735 defines BasicSafetyMessage (BSM), which includes various vehicle information helpful to vehicular safety applications. It has about 50 bytes of data that is a required portion and appears in every transmission. After the required portion, vendors can attach proprietary data structures. Since the required portion of BSM is not sufficient for platooning applications, the industry may use the custom portion extensively. 100 bytes is assumed for this part, which adds up to roughly 150 bytes for each platooning message.

3. Total lanes: 10. In urban area, it is common for highways to have 5 lanes per direction at busy segments, especially near an interchange. In some cases, there are multiple highways in the same area, making congestion easier to happen. An example is an area[1] near Atlanta airport in US, where the Interstate 285 has 5 lanes per direction, with other roads or highways nearby. Thus, it is reasonable to use 10 lanes (for both directions) in analysis for heavy-use cases.

4. Interference range: 400 meters. Interference range is larger than effective communication range. With DSRC ratios designed to reach several hundred meters, 400 meters interference range is a reasonable assumption.

5. MAC layer data rate: 6 Mbps. Generally speaking, higher data rate results in lower the signal-to-noise ratio under the same environment. Lower signal-to-noise causes higher packet error rate. This affects the reliability of transmission. Study [16] shows that even using alternating mode, the delivery ratio may drop significantly when the transmission rate is above 6 Mbps. Safety applications like platooning requires high reliability, so it is assumed that the 6 Mbps is selected all the time. This is not the most conservative choice. In some scenarios, 3 Mbps may be used to achieve higher reliability.

6. Platooning message rate: 20 Hz. This assumes dual-antenna configuration and alternate mode. In other words, at 20 Hz, the effective message rate is 10 Hz, or one message every 100 milliseconds. 100 milliseconds results in about 3 meters in distance at a speed of 110 km/h. Depending on the braking system performance of the vehicle, this normally means tolerance of no more than 2 consecutive message losses before the control system has to conservatively apply a hard braking.

Consider a target vehicle on a busy segment of a highway. With 400 meters' interference range, interference can be caused by vehicles within 400 meters radius, which means

[1]https://www.google.com/maps/@33.6205067,-84.4576489,1936m/data=!3m1!1e3

an 800 meters long segment on the highway. Assuming full market penetration and full road utilization, linear vehicle density is $\frac{1vehicle}{20(meters*lane)} = 0.05vehicle/meter/lane$. With 10 lanes in total for both directions, there are $800meters * 10lanes * 0.05vehicle/meter/lane = 400vehicles$ within the interference range. At 20 Hz per vehicle, messages are being sent at $400vehicles * 20Hz/vehicle = 8,000Hz$ within the interference range. Multiplying this by the message size 150 bytes, the total required bandwidth can be calculated as $8,000Hz * 150bytes = 1.2MB/second$, or 9.6 Mbps.

With variables defined as in Table 4.1, the calculation above can be represented in a streamlined equation as shown in Equation 4.1.

Variable	Definition
B	Demanded bandwidth within interference range
r	Interference range
l	Total number of lanes in both directions
d	Average following distance
s	Platooning message size
f	Frequency at which platooning messages are sent

Table 4.1: Variable definition for Equation 4.1

$$\begin{aligned} B = \frac{2r}{d} * l * s * f = \frac{2 * 400m/lane}{20m/vehicle} * 10lanes * 150bytes * 20Hz/vehicle \\ = 1.2MB/second = 9.6Mbps \end{aligned} \tag{4.1}$$

9.6 Mbps is already greater than the MAC layer data rate 6 Mbps. Furthermore, due to the channel switching strategy defined in IEEE 1609.4 (see Section 2.1.1), only half of the channel time can be used. This reduces the available data rate to 3 Mbps. Other than this, there are MAC layer overheads, such as the contention window and various spacing in CSMA, management frames and control frames. These overheads further reduce the bandwidth available for upper layers.

Even worse, the calculation above only considers the platooning message, which is an extended version of BSM. In addition to BSM, SAE J2735 also defines other message types usable in vehicular environments, including but not limited to: Common Safety Request, Emergency Vehicle Alert Message, Roadside Alert, Traveler Information, NMEA Corrections [9, 5]. These messages all increase bandwidth consumption, making the congestion worse.

In conclusion, the DSRC technology will have congestion problems when it is fully deployed. It is necessary to apply various techniques to reduce bandwidth consumption in DSRC applications in order to relieve channel congestion.

To reduce bandwidth consumption for platooning application and other potential vehicular applications, a transmission mechanism that uses interframe compression is designed in this study.

4.2 Related Work

In [35], the authors found that non-safety use of DSRC may have to be severely restricted during peak hours of traffic to insure that automotive safety is not compromised. This is consistent with the analysis in Section 4.1.

A study in [36] proposed an adaptive congestion control method that adapts communication rate and power based on the dynamics of a vehicular network and safety-driven tracking process. In [37], authors studied the relationship between channel occupancy, as a readily available feedback measure, controllable network parameters, and network performance. It then proposed a closed loop congestion monitoring and control based on limited feedback from the network. A study in [38] proposed a linear message rate control algorithm, which rather than using a binary control, takes advantage of full precision control inputs that are available on the wireless channel.

SAE J2945.1 [39], a standard that specifies the system requirements for vehicle-to-vehicle safety communications, includes a congestion control algorithm for BSMs. The algorithm inspects channel busy percentage and packet error rate, to make transmission decisions, including transmission schedule and power.

These techniques in vehicular networks reduces bandwidth consumption by various congestion control methods. They reduce bandwidth consumption of vehicular applications significantly in congestion, but takes a different approach than this study:

- First, the congestion control algorithms make decisions on when to limit message rate or transmission powers. Consequently, message frequency or the amount of information transmitted are sacrificed, i.e., the networking service level degrades. In other words, to transmit the same amount of data, the required bandwidth is not reduced.
- Second, such congestion control algorithms are most effective when congestion already happens. They serve as a remedy for avoiding the consequences from congestion. However, the congestion control algorithms generally have little effect in prevention of congestion.

In this study, however, rather than trying to deal with the congestion when it happens, this study introduces a compression algorithm that helps postpone the point where congestion happens, by reducing bandwidth consumption without sacrificing the amount of information transmitted.

4.3 Inspiration and Discussion

Interframe compression is a fundamental idea of many video compression algorithms such as MPEG-2 [40] and H.264 [41]. In video data, although entropy of each individual frame is relatively high, consecutive frames in a video are normally very similar, i.e., entropy of difference between consecutive frames is relatively low. Thus, encoding frames individually results is a waste of bandwidth. With interframe compression, video codecs encode complete frames (Intra-frames, or I-frames) only every once a while. Between these complete frames, only changing parts are encoded (Predictive-frames or P-frames, and Bidirectional-frames, or B-frame).

These frames are packed into Group of Pictures (GOP) structures. A typical GOP structure can be represented as `IBBPBBP...PBBI`. Since B-frames and P-frames are much smaller than I-frames, the complete video stream uses much less bandwidth.

Messages in vehicular network can benefit from interframe compression as well. An important attribute of video streams that allows interframe compression is that consecutive frames are very similar, with only slight changes in parts. This is true for messages in vehicular environments as well. Many messages passed by DSRC applications only gradually change. Take platooning message as an example. Within a small portion of the road, e.g. 60 meters, or roughly 2 seconds at 110 km/h, many dynamic attributes of the vehicle, yaw rate, acceleration, brake system, steering, etc., have relatively stable values, as long as the vehicle is not doing a significant maneuver. Even when they change, they change gradually, which results in only a few flipped bits in binary form.

However, video compression algorithms cannot be directly applied to vehicular networks.

First, platooning messages, among many types of messages in vehicular environment, are temporal messages that are only meaningful within a period of time. As much as high delivery ratio matters, the message is of little use to the actual real-time applications if it is not delivered in time. As a result, the communication system should not attempt to delivery an old message by sacrificing latency of newer messages. In other words, retransmission should not happen in case of message losses. As a result, data stream from one vehicle to another should not be considered reliable if the key frames can be missing. Hence, a long dependency chain like `IBBPBBP...PBBI` is not suitable for vehicular network.

Second, messages like platooning require lossless compression. While recent video compression techniques have improved prediction techniques and result in high quality video stream, they are not lossless and data through compression may still differ from original data.

Hence, to use interframe compression, a new algorithm needs to be designed to suit the needs of vehicular networks.

4.4 Basic Design

To make the transmission mechanism suitable for general purposes in vehicular environment rather than specifically for platooning, it is designed as a layer between Message Sublayer and WSMP layer in the DSRC stack as shown in Figure 2.1. We call it Interframe Compression Transmission Layer (ICTL). In context of the DSRC protocol suite in Figure 2.1, ICTL can be placed either between Safety Applications Sublayer and the Message Sublayer or between the Message Sublayer and WSMP.

In traditional DSRC, applications constructs a SAE J2735 message, and directly encapsulates it in a WSM and send through IEEE 802.11p using WSMP. With ICTL, the message is either sent as it is or converted to a smaller byte stream containing changes from a previous complete message before being sent out. ICTL takes care of compression and decompression, and applications see the same message identical to the original.

To support this, two types of frames are introduced:

- Key Frame (KF). A KF is like an I-frame in video compression. It contains the complete message being sent. It does not require any extra information to decode a KF.
- Differential Frame (DF). A DF only carries difference from a previously sent KF. Decoding original message from a DF requires the KF that it refers to as well.

4.4.1 Header and Layout

To separate DF from KF, identify each message, and carry extra information regarding the codings, ICTL uses a 4-byte header. Table 4.2 shows the layout of an ICTL packet. Zooming out to a bigger picture, a typical message with all headers has a layout shown in Table 4.3.

Reserved	Frame Type	Compression Options	Frame ID	Payload
4 bits	4 bits	8 bits	16 bits	variable length

Table 4.2: ICTL Packet Layout

MAC Header (IEEE 802.11)	WSM Header	ICTL Header	Payload
34 bytes	5-20 bytes	4 bytes	variable length

Table 4.3: ICTL Packet Layout With WSM Header

The Reserved field is for use in newer versions of ICTL. For now it should always be set to `0b0000`. In the future, this field can be used as a version field. Frame Type indicates whether the packet contains a KF (`0b0001`) or a DF (`0b0010`). Compression options carries information about compression algorithms applied to the payload (see Section 4.4.3). Frame ID is an increasing unsigned integer that wraps at boundaries. It is used to identify frames sent from the same source. For a KF, it is the ID of the actual message. For a DF, it is the ID of the KF that it refers to. Payload is the actual complete message (KF) or differential message (DF). Since WSM header already has a field for payload length and ICTL header has a fixed width of 4 bytes, the length of ICTL payload can be easily inferred, thus does not require an extra length field.

4.4.2 Differential Encoding

Compressing vehicular messages needs to be lossless. As a result, video compression techniques such as prediction cannot be used. Instead, a simpler approach using Exclusive Or (XOR) is proposed to encode differential messages in a DF.

Given a previously transmitted KF, whose payload is

$$p^0 = b_0^0, b_1^0, b_2^0, b_3^0, b_4^0, b_5^0, \ldots, b_{n-2}^0, b_{n-1}^0 \tag{4.2}$$

where b_i^0 represents one bit (zero based index) of the payload p^0, and a new message

$$p^1 = b_0^1, b_1^1, b_2^1, b_3^1, b_4^1, b_5^1, \ldots, b_{n-2}^1, b_{n-1}^1 \tag{4.3}$$

where b_i^1 represents one bit (zero based index) of the payload p^1, then the payload of DF, or in other words, the differential message $d^{0,1}$ between p^0 and p^1, can be produced as

$$d^{0,1} = b_0^0 \oplus b_0^1, b_1^0 \oplus b_1^1, b_2^0 \oplus b_2^1, b_3^0 \oplus b_3^1, b_4^0 \oplus b_4^1, b_5^0 \oplus b_5^1, \ldots, b_{n-2}^0 \oplus b_{n-2}^1, b_{n-1}^0 \oplus b_{n-1}^1 \tag{4.4}$$

where $\oplus$ is the XOR operator[2].

Since the messages that are sent within a short time window have very similar content, the majority bits in the differential message should be zero. Such message has low entropy, and can be efficiently compressed using a general purpose compression algorithm.

4.4.3 Encoding and Decoding Process

ICTL is applied to each application rather than to each link. Since WSMP already handles multiplexing through PSID, ICTL does not further multiplex messages into multiple streams. In other words, one ICTL encoder/decoder should be allocated for each PSID.

When ICTL receives a request to transmit message from upper layer for a given PSID, it first determines whether the message should be transmitted using KF or DF. This depends on either a predefined parameter that describes how many DFs can be sent between two adjacent KFs, or an internal model to make the decision.

For KF, the message is compressed, and prepended with an ICTL header. The header indicates that it is a KF and provides the increasing unsigned integer to identify the message. For DF, the differential message is first calculated using process described in Section 4.4.2, then compressed and prepended with ICTL header. The header indicates that it is a DF,

[2]When implementing in a programming language, this can be done efficiently by using bitwise XOR operator (normally written as ˆ), and computed one byte, or even one word (32 bits or 64 bits) at a time.

and references the ID of the KF that the DF is produced from. Normally, the referenced KF is the most recently sent KF, but can be older ones as well depending the networking condition when the KF was sent. If the network condition is bad when the most recently sent KF was transmitted, it might not have been successfully received by many nearby vehicles. In this case, ICTL can choose a previously sent KF to base on. In this study, however, for simplicity, ICTL always choose the most recently sent KF as reference frame.

The following is a pseudo-code listing for encoding KFs and DFs using Go-like syntax:

```
var KFs map[uint16][]byte
var currentID uint16

/*
* Omitted functions' signatures:
* func shouldDF() bool
* func chooseKFID() uint16
* func XOR(src1 []byte, src2 []byte) []byte {}
* func compress(original []byte) (compressionAlgorithm string, compressed []byte) {}
* func prependHeader(isDF bool, compressionAlgorithm string,
                                 id uint16, message []byte) []byte {}
*/

func encode(message []byte) []byte {
  if shouldDF() {
    return encodeDF(message)
  } else {
    return encodeKF(message)
  }
}

func encodeKF(message []byte) []byte {
  algo, data := compress(message)
  currentID++
  map[currentID] = make([]byte, len(message)
  copy(map[currentID], message)
  return prependHeader(false, algo, currentID, data)
}

func encodeDF(message []byte) []byte {
  baseID := chooseKFID()
  diff := XOR(baseID, message)
  algo, data := compress(diff)
  currentID++
  return prependHeader(true, algo, baseID, data)
}
```

Decoding is basically the reverse operation of encoding. Since the sender may choose to send a differential message based on an older message, a list of recently received KF needs to be maintained. The following is a pseudo-code listing for decoding KFs and DFs using Go-like syntax:

```
var KFs map[uint16][]byte

/*
* Omitted functions' signatures:
* func XOR(src1 []byte, src2 []byte) []byte
* func decompress(original []byte, compressionAlgorithm string) []byte
* func decodeHeader(packet []byte)
        (isDF bool, compressionAlgorithm string, id uint16, payload []byte)
*/

func decode(packet []byte) []byte {
  isDF, algo, id, payload := decodeHeader(packet)
  data := decompress(payload, algo)
  if isDF {
    return decodeDF(data, id)
  } else {
    return decodeKF(data, id)
  }
}

func decodeKF(message []byte, id uint16) []byte {
  KFs[id] = make([]byte, len(message))
  copy(KFs[id], message)
  return message
}

func decodeDF(diff []byte, referredID uint16) []byte {
  base, ok := KFs[referredID}
  if !ok { // not found
    return nil
  } else {
    return XOR(base, diff)
  }
}
```

The implementations of `shouldDF()` are particularly tricky and require guidance from tests in order to achieve good performance. The following sections of this chapter explains this process, in which an adaptive ICTL algorithm (Section 4.7) is designed.

For analysis and experimental purposes, the concept of ICTL Cycle, and Cycle Length (CL) is introduced here:

- ICTL Cycle is defined as a sequence of ICTL frames, starting with a KF, followed by a number of DFs, given that the last DF, i.e., the end of cycle, is either the last frame of transmission, or followed immediately by a KF, which marks the start of next cycle. In the case of this study where ICTL always pick the most recently transmitted KF as reference frame for DFs, an ICTL Cycle can also be defined as a sequence of messages including exactly one KF and all the DFs that use this KF as reference frame;
- CL is defined as the number of messages included in an ICTL Cycle. For example, for an ICTL strategy that always sends 3 DFs after each KF before sending a new KF, the CL is always 4. CL can be a predefined fixed value, in which case it is a configurable parameter, or dynamically changing according to an internal model in ICTL implementation, in which case CL is merely an observed value driven by the ICTL implementation.

4.5 Implementation and Experimental Setup

4.5.1 Implementation

For the same reasons explained in Section 3.4, Go programming language is chosen to implement ICTL. Similarly, it uses a buffer pool for memory management. ICTL is implemented as a library package that encapsulates details of encoding and decoding behind a simple interface `ictl.Endpoint`, which represents an ICTL endpoint for encodings and decodings. An object of `ictl.Endpoint` can be created using a configuration object that specifies the behaviors of ICTL algorithm. The endpoint has methods for encoding and decoding messages. Each of such methods accepts a `context` parameter, which is used to differentiate different streams of data. In case of DSRC, a combination of MAC address and PSID is a good choice of `context`. The endpoint allocates one encoder/decoder for each context, and maintains the separation between different data streams.

The implementation of ICTL is open-source and available at `https://github.com/songgao/ictl`.

4.5.2 Trace File Playback

Squirrel was designed to facilitate running real-world codes. Everything above MAC layer assumes a real wireless device, and process real network frames. The idea here is to develop research projects that can be not only reproducible, but also more directly beneficial to real-world applications. With the same spirit, we acquired real-world platooning data traces and use them to study the ICTL algorithm to improve performance. To utilize such trace files, a software component is written, called `truck-playback`, to replay events from one vehicle.

The trace files were logged by a prototype platooning application running on two heavy vehicles, while the trucks were being tested in platooning mode on a closed test track. The trace files include many events, such as GPS location, DSRC messages transmission and reception, various events from vehicle's CAN bus, radar measurements, etc.. For the purpose of studying ICTL, two events are relevant:

- GPS location events, which are needed to update emulated locations of the truck in Squirrel;
- DSRC transmission events, which indicate when to send a DSRC message and what particular data should each DSRC message carry.

`truck-playback` reads the two types of events from a logged trace file and replays these events in emulation. The GPS location events are converted to RPC requests, and sent through the RPC Updated Mobility Manager's service (see Section 3.3.4) to update the current emulated position of the node. The DSRC events drives the `truck-playback` to send WSMs over DSRC at predefined time points, using traditional method, or encoded with ICTL. The content of a WSM is determined by the event content, which includes values of each data field. Sending exactly the same information as trace files is important because the content of messages and how fast the content changes affect how well ICTL compresses the messages.

Each event from the trace file is tagged with a timestamp. `truck-playback` synchronizes the timeline so that a time point from wall-clock during emulation is aligned with a time point from trace file's time when the tests were conducted in the real world. In other words, a fixed offset is set between the playback's timeline and the trace file's timeline. When `truck-playback` replays an event, it first calculates a deadline based on the timeline offset. In most cases, the deadline is after current wall-clock time. A difference is calculated between the two, and `truck-playback` waits for such time duration by calling `sleep()` before actually replaying the event. This ensures that events happen at exactly same relative times. For example, DSRC messages are sent at same frequencies as the trace file dictates. When many nodes are emulated on the same computer, occasionally, the deadline may have already passed at the time it is calculated. In this case, this particularly event is sent immediately (without calling `sleep()`, but not able to meet the deadline when replayed. If too many events cannot meet their deadline, the results would not be authentic. As a result, `truck-playback` logs the such incidents to be examined afterwards.

The `truck-playback` uses a WSMP implementation written in Go, which uses system call interfaces to pass messages into or read messages from the Logical Link Control (LLC) layer in the kernel. In Squirrel, LLC is handled by `squirrel-worker` through `tap0` interface, as demonstrated in Figure 3.3. Conceptually, `truck-playback` works as a DSRC application as shown in Figure 4.1.

When `truck-playback` replays a DSRC transmission event, it logs the replay, including a timestamp, the GPS position of the node when the message is sent, a message counter, and frame size. Similarly, when it receives a message from WSMP, it logs the reception as well, including a timestamp, a message counter, source MAC address, and frame size. With

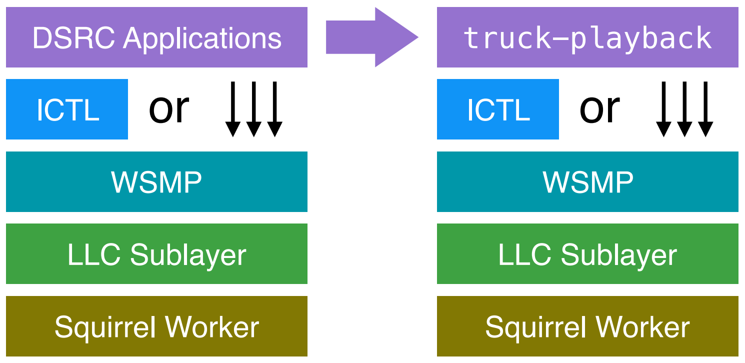


Figure 4.1: `truck-playback` as DSRC application

these information, we can study the bandwidth and delivery ratio of ICTL, under different parameter scenarios.

4.5.3 Deployment

Squirrel can be deployed on a cluster, where one node runs `squirrel-master`, and `squirrel-worker` nodes run on other nodes. In this study, however, the cluster only consists of one physical workstation. The workstation has one Intel(R) Xeon(R) CPU W3680 with 6 cores. The CPU does not have hyper-threading technology, so at most six threads (as opposed to 12 threads had it support hyper-threading technology) can run simultaneously on the CPU at any time.

Due to its lightweight property and scalability, OSLV is chosen to provide proper isolation for different emulated nodes. Specifically, each `squirrel-worker` node runs in an application container [25]. With different kernel namespaces, one isolated complete networking stack is allocated for each `squirrel-worker` node. There are several open source softwares providing Linux application container runtime, namely Docker, LXC, and Rkt. Each of them provides a set of tools for building container images, running and configuring application containers. In this study, Rkt [42] is used. The operating system on the workstation is CoreOS [43], a modern Linux distribution designed specifically for running application containers.

Since all components are written in Go, they are statically linked into independent executable binaries, which makes packaging Rkt images very straightforward. The `squirrel-master` image only contains the `squirrel-master` binary. The `squirrel-worker` node image contains the `squirrel-worker` binary, `truck-playback` binary. For convenience in debugging, some common tools such as `tcpdump` and a command-line shell are also included. Since application containers provide filesystem isolation as well, proper filesystem mount points are configured for `squirrel-worker` nodes to provide input data trace files as well as to collect logs from `truck-playback`.

CoreOS uses `systemd` as its init system. To manage large number of `squirrel-worker` nodes for experiments, `systemd` unit files are written to use `systemd` to automatically spawn Rkt containers, attach directories to mount points, and provide initial parameter settings. A wrapper unit file is also written to automatically schedule series of tests consisting of different number of nodes and configurations.

4.5.4 Triggering Congestion

A major goal that ICTL tries to achieve by reducing bandwidth consumption is to improve delivery ratio when network utilization is high. As a result, to study and improve ICTL, it is necessary to create congestion in the emulation environment. Section 4.1 uses 400 vehicles in the analysis. However, on a single workstation, it is hard to emulate 400 nodes in real time. The data trace files we acquired use 50 Hz DSRC message rate, which is higher than 20 Hz in the analysis. This makes it easier to trigger congestion. In addition, several methods are used to help trigger congestion with less than 15 emulated nodes.

1. 3 Mbps (instead of 6 Mbps) is used as the MAC layer data rate setting in Squirrel's CSMA/CA model.

2. Each message is replicated 12 times, in 12 separate WSMs, before being encapsulated and sent as a LLC layer frame. This increases frame size, based on the original data trace, from 117 bytes to 1250 bytes.

Part of replicated messages in the second method can also be seen as adding other DSRC messages into the frame. An important example is re-broadcast messages that are required to deliver message to vehicles outside the transmission range.

4.5.5 First Glance

A simple test is conducted in the emulation platform to demonstrated the effectiveness of ICTL. In Figure 4.2, three strategies are compared. "Raw" series shows the required bandwidth of the original data trace; "Raw Compressed" shows the required bandwidth of the data trace compressed with a traditional compression algorithm [3]; and "ICTL CL=3" shows the demanded bandwidth of the data trace compressed using ICTL, with fixed CL 3.

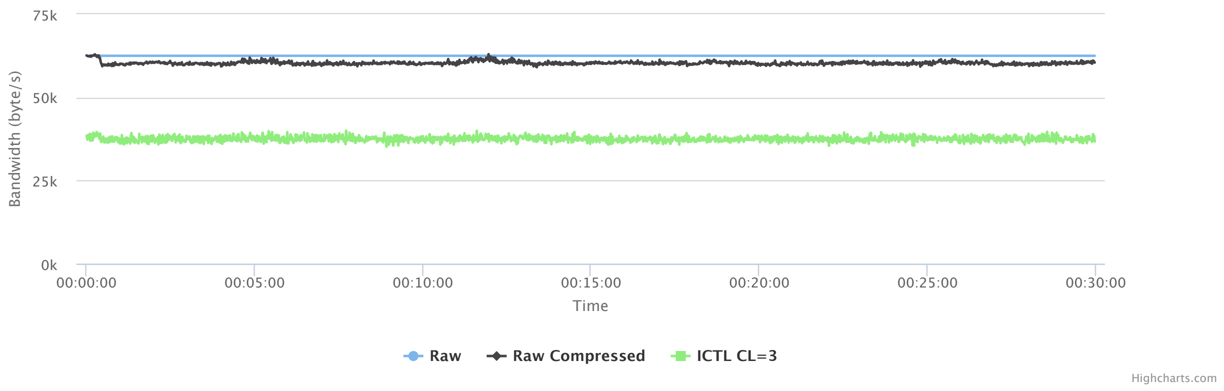


Figure 4.2: Bandwidth test: original, traditional compression, and ICTL

In this test, the original data trace uses a constant bandwidth of 62.5 KB/s. Using traditional compression method directly over the original data trace uses $\geq$ 59 KB/s. With ICTL, the required bandwidth is always below 40 KB/s, yielding more than 30% bandwidth savings.

The test demonstrates the effectiveness of the basic idea of ICTL. The following sections in this chapter study the behavior and performance attributes of ICTL and describe ways to optimize the algorithm for bandwidth and delivery ratio.

[3] `flate` is used here. Please refer to Section 4.6.4 for more information on choice among different compression algorithms.

4.6 Performance Studies

4.6.1 Cycle Length and Application Delivery Ratio

Delivery ratio is one of the ultimate metrics that researchers are constantly trying to optimize. In this case, two kinds of delivery ratio are involved:

- Application Delivery Ratio (ADR) is the delivery ratio observed in the application layer. Lost frames or errors in any of the lower layers can affect this metric. This is the metric that we are trying to optimize.
- Network Delivery Ratio (NDR) is the delivery ratio observed in the layer immediately below ICTL. Events such as MAC layer frame loss directly affect this metric.

In ICTL, each DF carries differential information from a previously transmitted KF. In other words, decoding a DF depends on the success of a KF transmission. As a result, losing a KF can be critical, since it implies the loss of all frames within the cycle starting with the lost KF. This makes the relationship between ADR, CL and NDR more complicated. On one hand, longer CL means more application frame loss whenever a key frame is lost; on the other hand, longer CL means that there are less KFs in the same transmission, thus losing KFs is less likely to happen given the same NDR.

Some mathematical analysis is done to further study this problem. Table 4.4 shows variables used in the analysis, along with their meanings.

Assume that N is sufficiently large, then p_n is equal to NDR. ADR can be calculated as $\frac{n_a}{N}$. Starting from each ICTL Cycle, the expected number of messages delivered at application layer within an ICTL Cycle, $E[n_c]$, can be calculated by dividing into two cases:

1. When the KF is lost, the entire ICTL Cycle is lost, i.e., 0 is delivered. The probability of this happening is $1 - p_n$;

Variable	Definition
N	The number of messages needed to be transmitted
n_a	The number messages delivered at application layer during entire transmission
n_c	The number of messages delivered at application layer within one ICTL Cycle
m	Cycle Length
p_n	The probability that an ICTL frame is delivered, i.e., not lost in lower layers
NDR	Network Delivery Ratio
ADR	Application Delivery Ratio

Table 4.4: Variable Definition for Analysis in Section 4.6.1

2. When the KF is delivered, each message in the rest of the ICTL Cycle is delivered at probability of p_n, so the expected number of delivery of the cycle in this case is $1 + (m - 1) * p_n$ The probability of this happening is p_n.

So $E[n_c]$ can be calculated using Equation 4.5:

$$\begin{aligned} E[n_c] &= (1 - p_n) * 0 + p_n * [1 + (m - 1) * p_n] \\ &= p_n^2 * m - p_n^2 + p_n \end{aligned} \tag{4.5}$$

With N frames in total and CL being fixed at m, there are $\frac{N}{m}$ ICTL cycles in total. So ADR can be calculated using Equation 4.6:

$$ADR = \frac{E[n_a]}{N} = \frac{\frac{N}{m}E[n_c]}{N} = \frac{E[n_c]}{m} \tag{4.6}$$

Combining Equation 4.5 and 4.6,

$$\begin{aligned} ADR = \frac{E[n_c]}{m} &= \frac{p_n^2 * m - p_n^2 + p_n}{m} \\ &= p_n^2 + \frac{p_n - p_n^2}{m} \end{aligned} \tag{4.7}$$

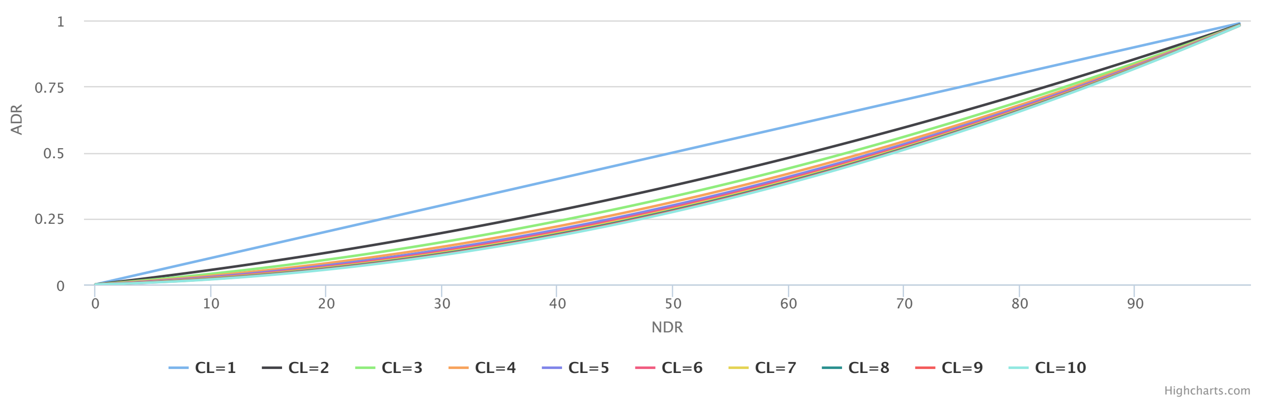


Figure 4.3: ADR as function of NDR for different CL

Equation 4.7 shows the relationship between ADR, NDR, and CL. This is illustrated in Figure 4.3, which shows ADR as function of NDR for different CL values. Although higher CL values reduce ADR overall, ADR is not as sensitive to CL changes as to NDR changes. In other words, ADR is more affected by NDR.

Bandwidth consumption affects NDR in a major way, in the sense that lower bandwidth consumption means less congestion, which results in higher NDR. As a result, for the purpose of increasing ADR, it is reasonable for ICTL to simply optimize for low bandwidth.

4.6.2 Location and Bandwidth

ICTL compresses difference between frames, so it uses less bandwidth when frames are similar, but uses more bandwidth when frames change more drastically. When a vehicle goes through a curve road, the vehicle's status, such as yaw rate, may change more than when it is on a straight line. Intuitively, one may think ICTL should result in higher bandwidth consumption at curved roads compared to on straight roads. To find out whether this is true, Figure 4.4, a 3D version of Figure 4.2, is generated, to show bandwidth consumption at different locations.

As shown in the figure, ICTL does not require more bandwidth at curved roads. We believe this is because the changes of vehicle status at curved roads are gradual and slow

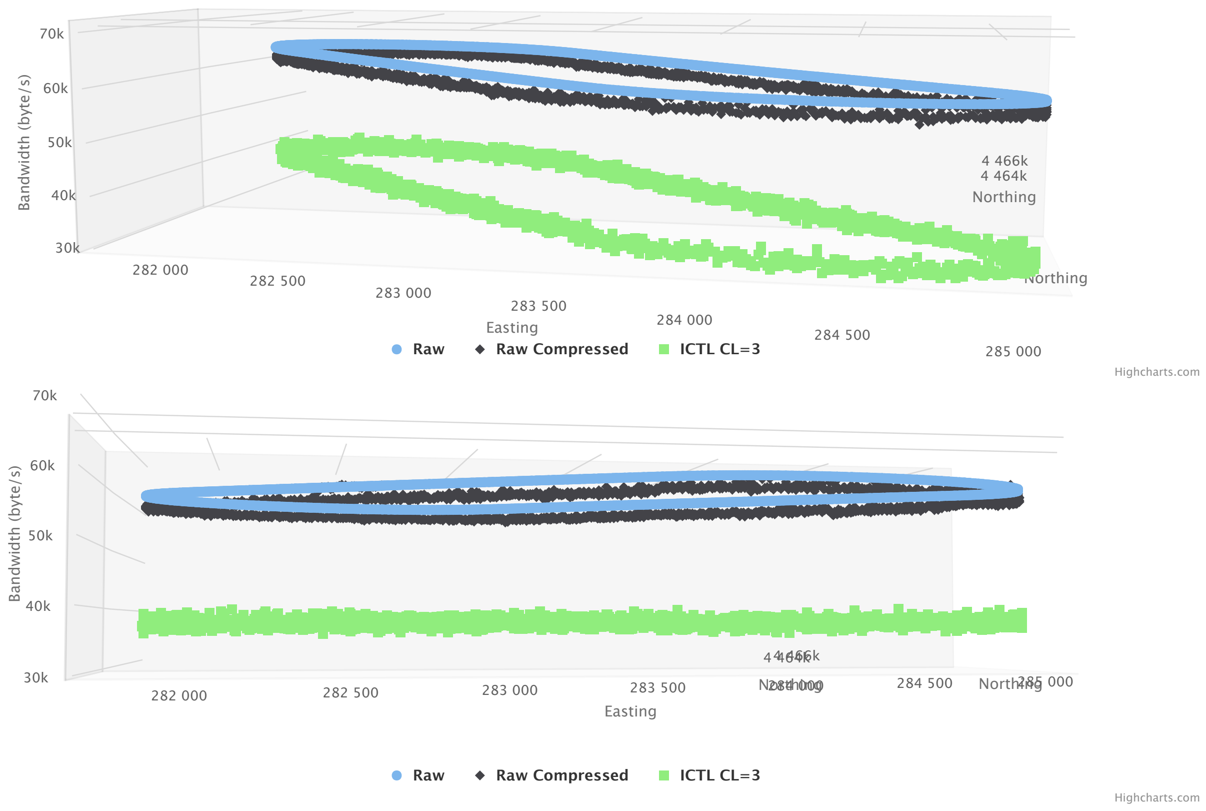


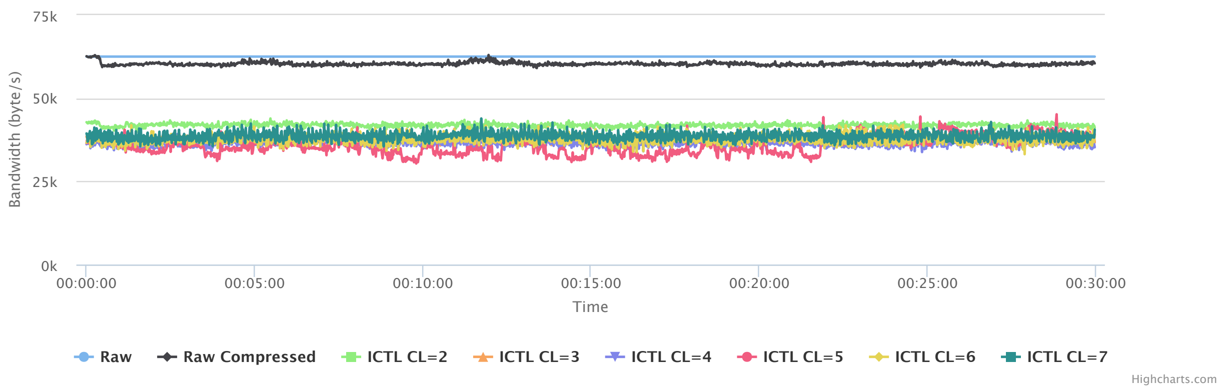
Figure 4.4: 3D bandwidth graph from two viewing angles

enough to not cause drastic change in DSRC messages. Thus, the relatively slow change in the message is reflected in the ICTL bandwidth consumption.

4.6.3 Cycle Length and Bandwidth

CL affects the required bandwidth in a major but complex way. DFs are generally much smaller than KFs. But the actual size of a DF depend on how different it is from the KF it uses as the reference frame. On the one hand, since DFs are smaller, more DFs should be sent, thus longer CL is better. On the other hand, longer CL means the DFs towards the end of an ICTL Cycle is much more different from the KF, increasing the size of the DFs. As a result, longer CL does not always result in lower required bandwidth.

This theory is verified by results from out emulation tests. Figure 4.5 shows the required bandwidth of ICTL with different fixed CLs, using the same data trace. Apparently when CL is set to 7, the required bandwidth is, for most of the time, higher than when CL is set to



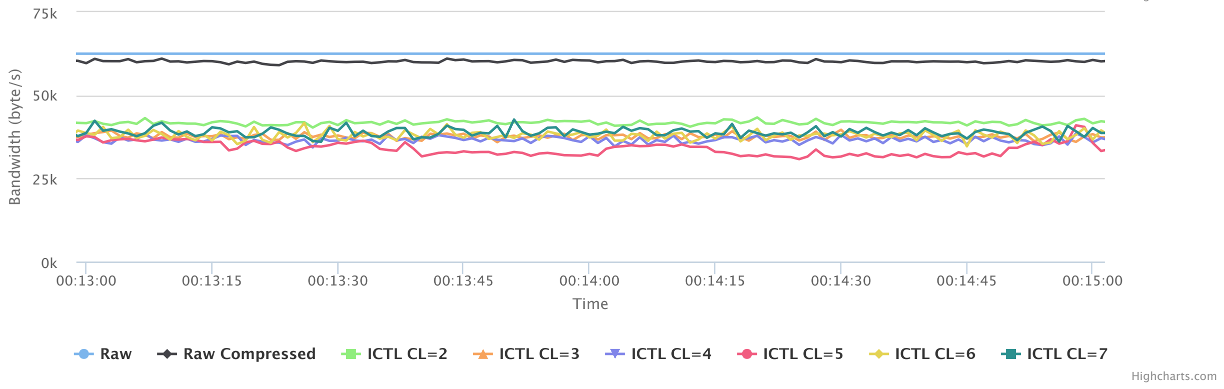


Figure 4.5: Bandwidth with different CL: 30-minute view and 2-minute zoom-in view

5, but they are both generally lower than when CL is set to 2. Furthermore, different data series cross each other, meaning that even for the same data trace, there is no CL setting that works the best all the time.

This shows that, it is challenging to set a proper fixed CL, and that with a fixed CL, ICTL is unable to always yield low required bandwidth. With this in mind, an adaptive algorithm is designed, as described in Section 4.7.

4.6.4 Payload Compression

ICTL has a header field of Compression Options, which carries information on how the payload is compressed before transmitted. This enables the receiver to dynamically configure the decoder based on the header and accept different compression algorithms and configurations. The compression algorithm used to compress DF payload affects overall size of DFs, so it is important to use one that performs well for such data.

At the beginning of this study, it was believed that, the best performing compression algorithm should be different for different DFs. In other words, there should be no algorithm that worked for all DFs. As a result, a mechanism was designed in the ICTL implementation that, if the compression algorithm was not specified, it would iterate through a predefined selection of algorithms, and pick the one that yields smallest compressed payload.

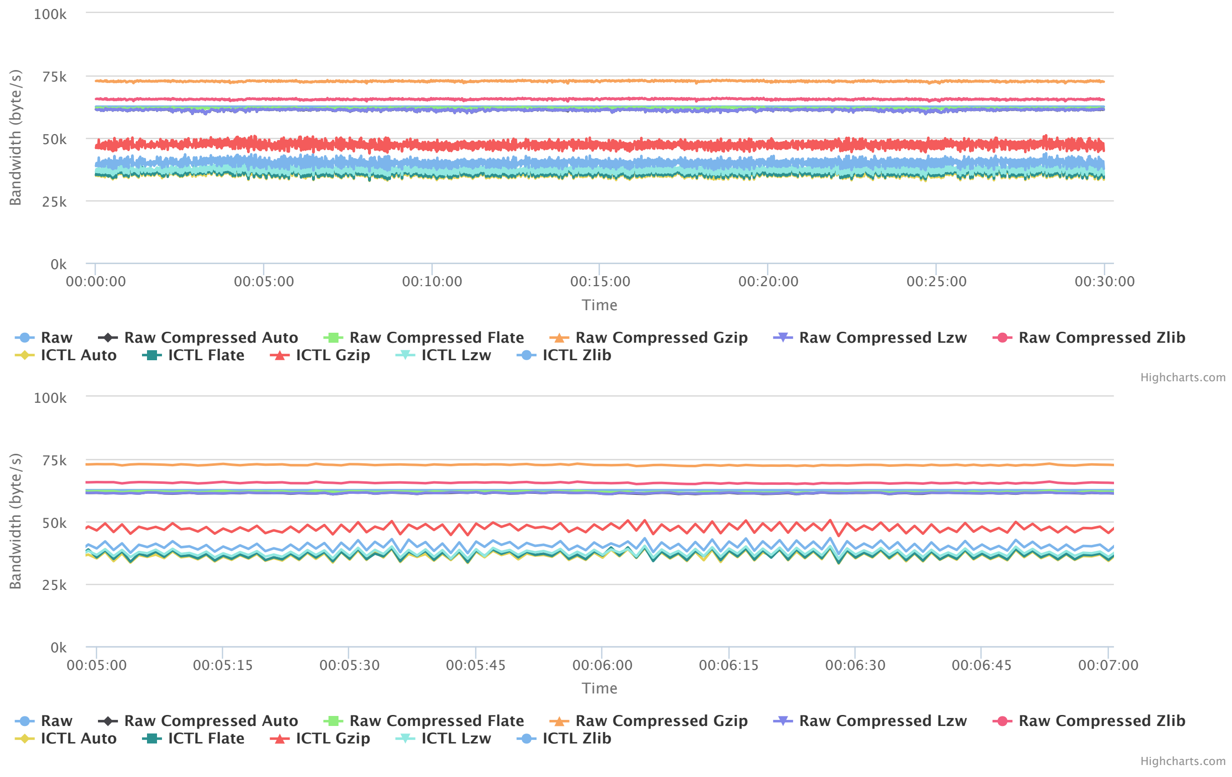


Figure 4.6: Bandwidth with different payload compression algorithms: 30-minute view and 2-minute zoom-in view

To find out whether it makes sense to do so, tests are done to compare different compression algorithms for the same payload of frames. Specifically, `flate`, `zlib`, `gzip`, `lzw` are tested. Figure 4.6 presents the results of the tests, which is contradictory to what was expected. Different data series generally do not cross each other, meaning their performance is fairly stable across different DFs. Among the four algorithms, `flate` and `lzw`, always yield smallest compressed payloads, with `flate` being slight better most of time.

Interestingly, `zlib` and `gzip` produces larger data than uncompressed. This suggests that `zlib` and `gzip` have compressing header overheads that are large enough to make them not suitable for compressing small size data, thus should not be directly used to compress payload of KFs and DFs.

Based on the findings here, the rest of our tests in this study use `flate` in ICTL.

4.7 Adaptive ICTL

Section 4.6.3 shows that a fixed CL cannot constantly perform well in terms of bandwidth consumption. In other words, CL should be chosen dynamically as ICTL is running. From a different perspective, CL is dependent on the observed behavior of ICTL. What should be determined dynamically is actually, at any given time, whether ICTL should send a DF, i.e., continue within the same ICTL Cycle, or a KF, i.e., start a new ICTL Cycle. The `shouldDF()` function in Section 4.4.3 is where this decision is made.

This section describes an adaptive ICTL algorithm, in which a `shouldDF()` is designed to optimize for bandwidth consumption.

4.7.1 Design

In an imaginary world where a node knows exactly what it will be transmitting for sufficiently large amount of time and there are usable quantum computers that make it viable to exhaust large search space, this would be an optimization problem where the goal is to find globally optimal sequence of CLs for transmitting all the frames that produces the lowest bandwidth consumption throughout the transmission. However, in the real world, ICTL is in no way able to predict what will be transmitted, and the node has limited hardware resources for searching for the optimal solution. What ICTL can do is to look at locally available information and make the best decision for bandwidth consumption based on a well selected metric.

In this study, ICTL uses a metric of average frame size of the ICTL Cycle, calculated up to the most recent frame. More formally, for each frame being transmitted, the metric μ_{cycle} for nth frame within the ICTL Cycle is defined as Equation 4.8:

$$\mu_{cycle}^{n} = \frac{S_{KF} + \sum_{i=2}^{n} S_{DF}^{i}}{n} \tag{4.8}$$

where S is the size of corresponding frame, and i is the frame number within the ICTL Cycle. i is greater than or equal to 2, because the first frame in an ICTL Cycle is always a KF.

μ_{cycle}^{x} is the key metric used by ICTL to determine whether a DF should be used for xth frame in the ICTL Cycle. First of all, ICTL keeps track of the immediately previous calculated metric, μ_{cycle}^{x-1}. When ICTL transmits a frame x, it first encodes a DF, and inspects its size S_{DF}^{x}, with which $\hat{\mu}_{cycle}^{x}$ is calculated according to Equation 4.8. If $\hat{\mu}_{cycle}^{x} < \mu_{cycle}^{x-1}$, it implies that, by sending another DF in this ICTL Cycle, the average frame size within the cycle can be further reduced, so ICTL should send a DF for this frame. In this case, $\hat{\mu}_{cycle}^{x}$ becomes μ_{cycle}^{x}, and the DF is sent. Otherwise, a KF is sent, and a new ICTL Cycle is started.

The following is a pseudo-code listing for logic to determine whether a DF should be sent, using Go-like syntax:

```
type adaptiveCycleLength struct {
  cycleSentTotalSize int
  cycleSentCount     int
}

func (a *adaptiveCycleLength) sentKF(size int) {
  a.cycleSentTotalSize = size
  a.cycleSentCount = 1
}

func (a *adaptiveCycleLength) sentDF(size int) {
  a.cycleSentTotalSize += size
  a.cycleSentCount++
}

func (a *adaptiveCycleLength) shouldSendThisDF(size int) bool {
  return a.cycleSentTotalSize/a.cycleSentCount > size
}
```

ICTL calls `sentDF()` and `sentKF()` methods whenever it sends a DF or a KF. It calls `shouldSendThisDF()` method to determine whether the next frame should be sent as DF or KF.

This algorithm minimizes average frame size within an ICTL Cycle. Although it only utilizes local information, the optimization is a step towards optimized CLs and achieves much lower bandwidth consumption.

4.7.2 Experimental Results

Our emulation tests are conducted to verify the effectiveness of the adaptive ICTL algorithm. Figure 4.7 shows the bandwidth consumption with adaptive ICTL and basic ICTL with different fixed CL settings.

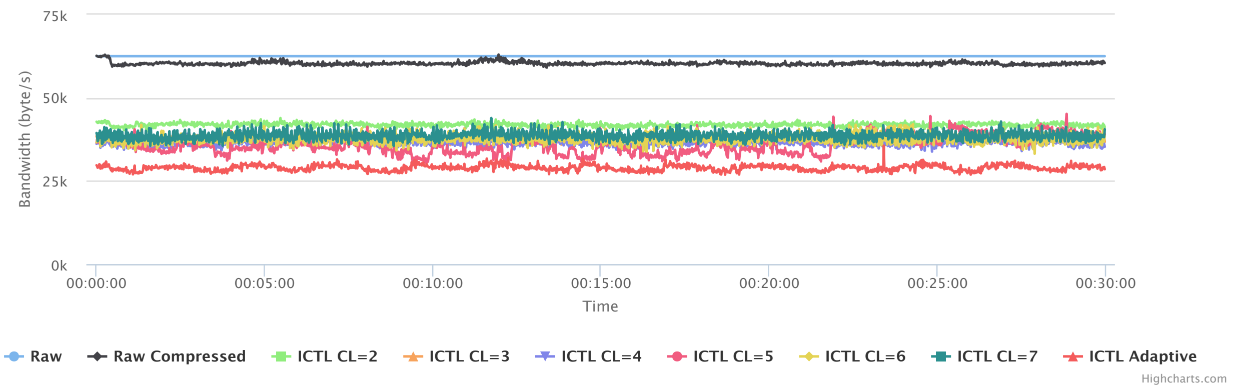


Figure 4.7: Bandwidth consumption with adaptive ICTL and fixed CLs: 30-minute view

The figure shows that adaptive ICTL's bandwidth consumption is constantly the lowest among all the data series. Furthermore, even the lowest one among fixed CLs is still 5 KB/s, or 15% higher than adaptive ICTL. This verifies that a dynamic decision making process like adaptive ICTL works better than using a fixed CL, yielding much lower bandwidth consumption. Based on the absolute numbers, adaptive ICTL's bandwidth consumption stays below 30 KB/s most of time. Compared to the original data trace which constantly consumes 62.5 KB/s bandwidth, adaptive ICTL yields more than 50% bandwidth saving.

As discussed in Section 4.6.1, since NDR is the main factor affecting ADR and that bandwidth affects NDR in a major way, ICTL should optimize for bandwidth in order to achieve good ADR. To verify this, delivery ratio, specifically ADRs, of different congestion levels are studied as well. As explained in Section 4.5.4, messages are already replicated 12 times to make it easier to trigger congestion. Here different number of nodes are used to emulate different congestion level.

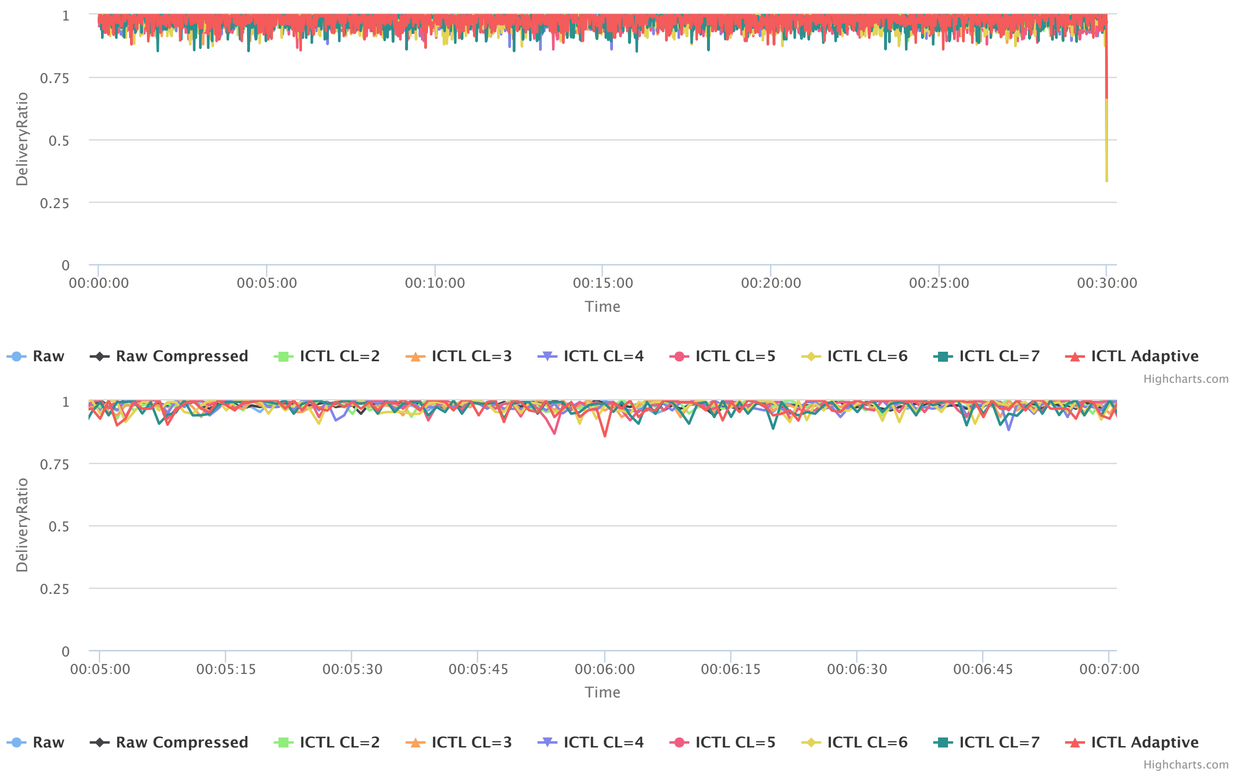
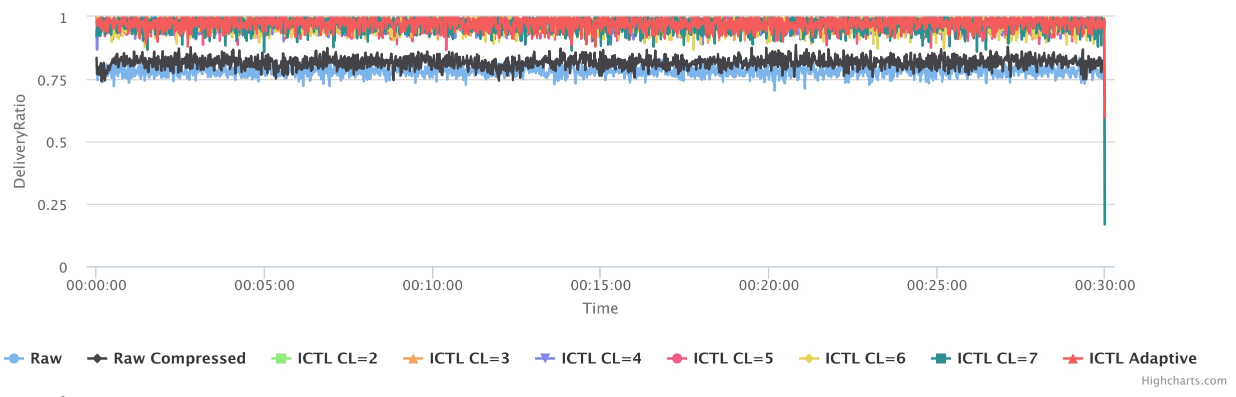


Figure 4.8: ADR in adaptive ICTL compared other transmission strategies with 4 nodes: 30-minute view and 2-minute zoom-in view

Figure 4.8 shows ADR for adaptive ICTL compared with other transmission strategies when there are 4 nodes in emulation. In this case, all data series show a good delivery ratio, close or equal to 100%. Note that even congestion is not triggered for any transmission strategy in this case, there are still packet losses. This is because Squirrel's CSMA/CA model looks at leaky bucket usage and uses a probability function based on it to determine whether a frame should be delivered. Arguably, this is emulating a realistic real-world experimental environment.



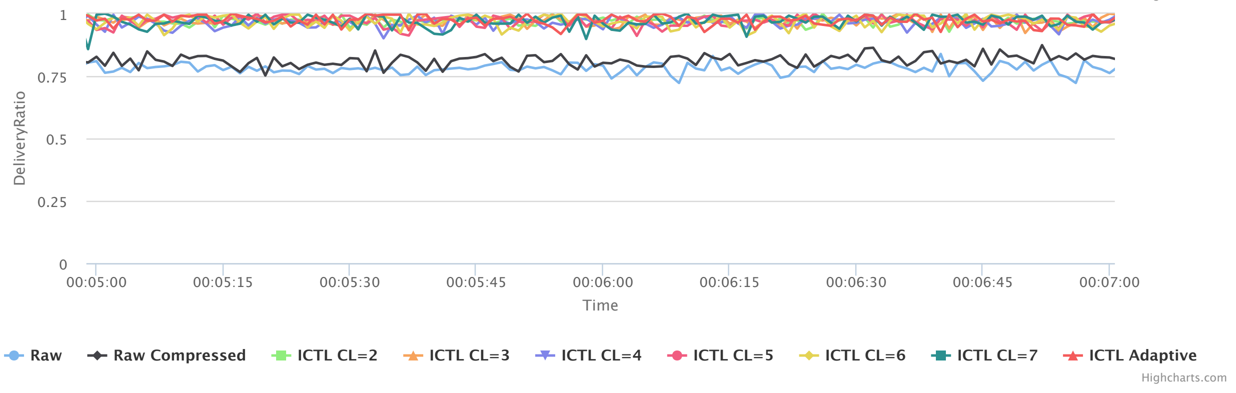
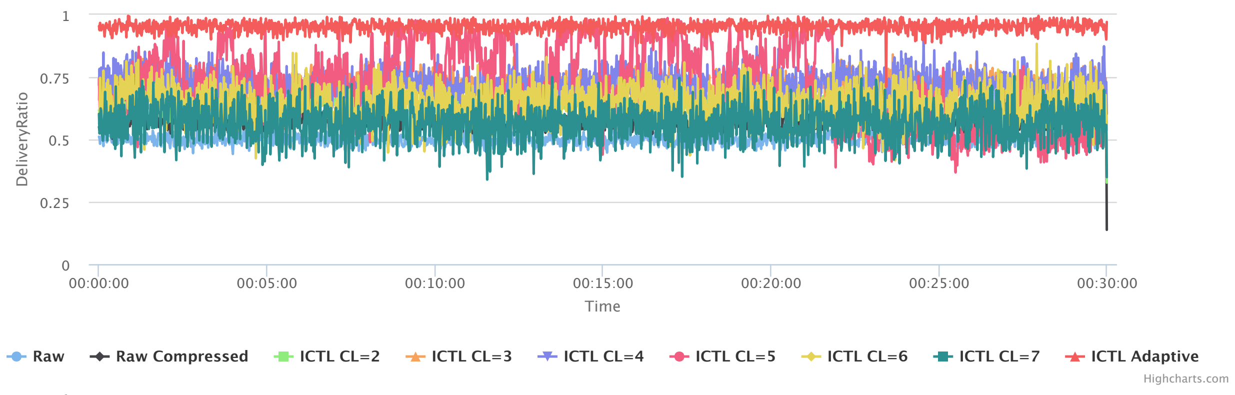


Figure 4.9: ADR in adaptive ICTL compared other transmission strategies with 6 nodes: 30-minute view and 2-minute zoom-in view

Figure 4.9 shows ADR results when there are 6 nodes in emulation. In this case, congestion starts to happen for the original data trace as well as using traditional compression method. All ICTL compressed data series still have close or equal to 100% ADR.



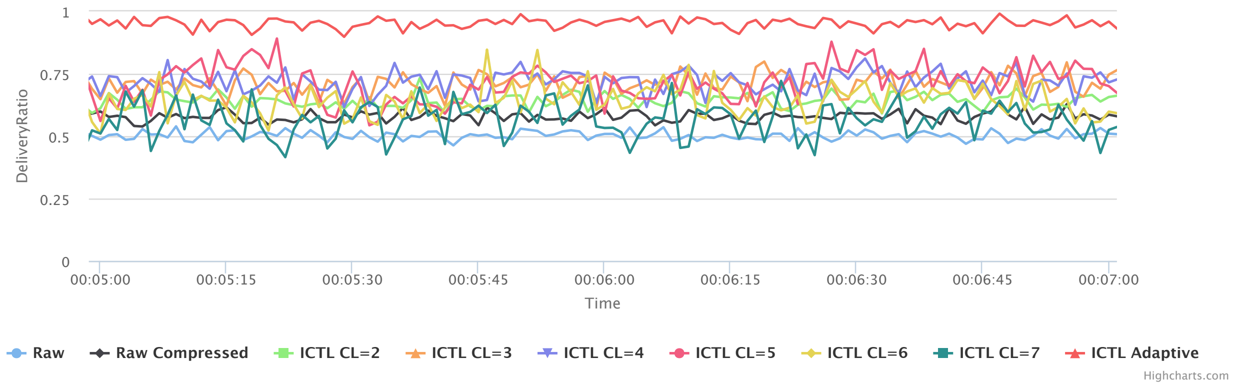
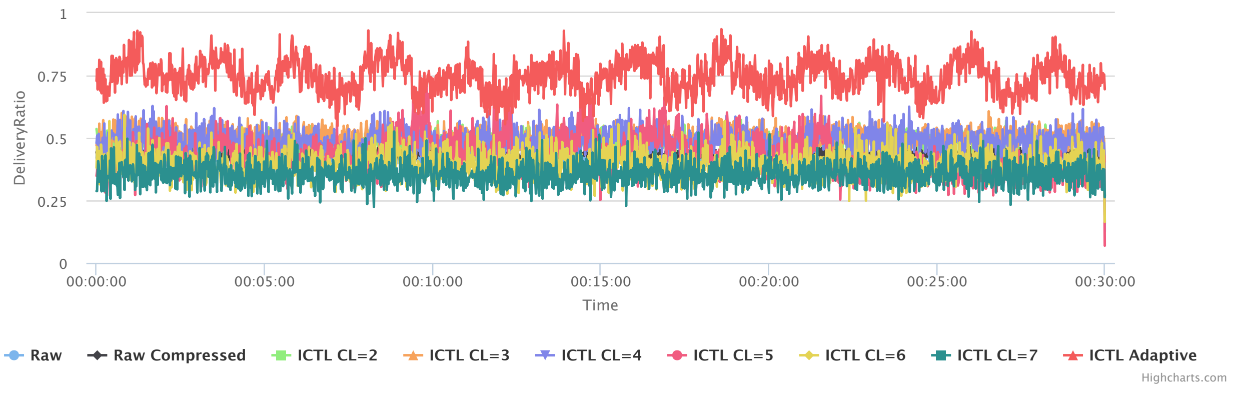


Figure 4.10: ADR in adaptive ICTL compared other transmission strategies with 10 nodes: 30-minute view and 2-minute zoom-in view

Figure 4.10 shows ADR results when there are 10 nodes in emulation. In this case, congestion starts to happen for ICTL with fixed CL settings. However, adaptive ICTL still delivers close to 100% of messages.



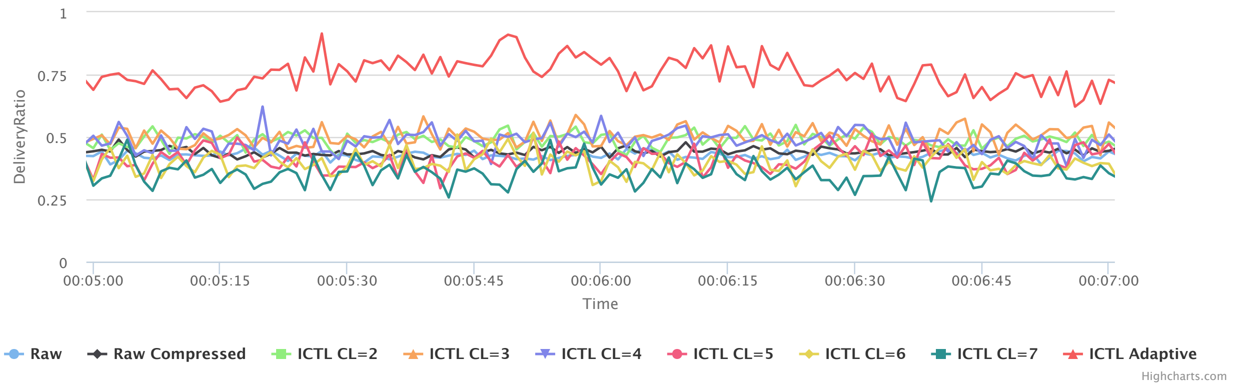


Figure 4.11: ADR in adaptive ICTL compared other transmission strategies with 12 nodes: 30-minute view and 2-minute zoom-in view

Figure 4.11 shows ADR results when there are 12 nodes in emulation. In this case, even adaptive ICTL starts to suffer from congestion. Although, even with congestion, adaptive ICTL still has the best ADR among all transmission strategies.

Putting these results together, Figure 4.12 aggregates results from emulations with different number of nodes into a single plot, showing ADR in different transmission strategies, with different number of nodes in emulation.

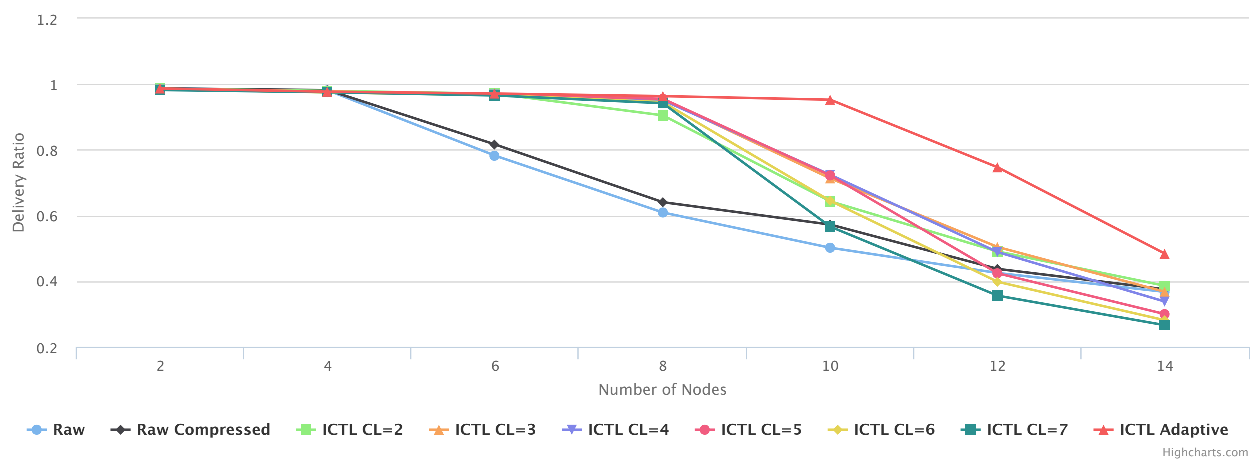


Figure 4.12: ADR in adaptive ICTL compared other transmission strategies with different node configurations aggregated in one plot

As shown in the figures, adaptive ICTL has the best ADR among all fixed CL settings, and has significantly high ADR compared to the original data trace or using a traditional compression algorithm, when wireless channel is congested.

In addition, adaptive ICTL also postpone the point when congestion happens, thus pushes back the point when a congestion control algorithm needs to be activated. This helps maintain a high service level of DSRC networking services.

4.8 ICTL Running Time Benchmarks

Adaptive ICTL is a fairly complicated process compared to simple networking operations such as sending a packet. To study the processing efficiency of adaptive ICTL, some benchmarks are done against adaptive ICTL to study the running time of the encoding process, using Go's built-in benchmark framework, on a laptop computer with Intel(R) Core(TM) i5-5287U processor.

In the benchmarks, adaptive ICTL is configured to either automatically select the compression algorithm by iterating through four algorithms or always use `flate`. The data trace is composed of messages that are either completely random, or always identical to each other. With random data, the differential data is large, thus it takes more time for adaptive

ICTL to compress the payload. With identical data, differential data is full of zeros, and it takes significantly less time for adaptive ICTL to compress the payload. Thus, identical data determines the lower bound of running time of adaptive ICTL, while random data trace is a good estimate of upper bound.

Unit: ns/op	Random	Identical
Iterating	2,434,206	1,232,683
Flate	701,801	305,154

Table 4.5: Adaptive ICTL Running Time Benchmarks: 256-byte messages

Unit: ns/op	Random	Identical
Iterating	2,942,048	1,318,164
Flate	882,314	321,307

Table 4.6: Adaptive ICTL Running Time Benchmarks: 1024-byte messages

Table 4.5 shows running time of encoding operations of four configurations for data traces composed of 256-byte messages. With fixed `flate` compression, the running time per operation is about 0.3 ms for identical data and around 0.7 ms for random data.

Table 4.6 shows the results for 1024-byte messages. Identical data and random data takes about 0.3 ms and 0.8 ms to encode respectively. With larger message size, running time of each operation is slightly larger, but not significantly larger. This suggests that larger message size does not affect adaptive ICTL encoding time significantly.

This suggests that, users of adaptive ICTL should take into account the fact that adaptive ICTL can cause up to 1 ms delay. 1 ms is comparable to the time it takes for networking stack on a device to send a network packet. Depending on how critical 1 ms is to the application and how efficient the rest of networking stack is, this can be significant. Implementing other operations asynchronously can help improve overall system response time, whereas the running time of ICTL is a hard limit. Thus, whether or not to use ICTL is a matter of trade-offs between low latency and low bandwidth consumption.

4.9 Conclusion

This chapter introduces an ICTL algorithm as a way to reduce bandwidth consumption in DSRC applications. The ICTL algorithm is implemented and tested on Squirrel. Behaviors of ICTL under different scenarios and configurations have been studied. Based on the findings, adaptive ICTL is designed to reduce the bandwidth consumption even lower.

By using adaptive ICTL, bandwidth consumption of the platooning data trace used in this study is reduced by over 50%. Consequently, in congested networking environment, adaptive ICTL delivers the highest ADR in all tested transmitted strategies. With 12 nodes, the ADR is 74.7% for adaptive ICTL compared to 50.4% for the best fixed CL ICTL. Thus, adaptive ICTL helps postpone the point when congestion control algorithms must be activated, which indirectly improves the service reliability of DSRC networking in vehicular environments.

The running time of adaptive ICTL's encoding operation is less than 1 ms, but greater than 0.3 ms. Depending on how sensitive the application is to latencies, this may or may not be significant.

Chapter 5
Conclusions

In context of a real-world application, vehicle platooning, this dissertation strives to improve the efficiency and reliability of vehicular networking, using an emulated experimental approach.

To start with, an emulation infrastructure is designed and implemented to provide a scalable yet realistic experimental platform. As a general framework for wireless networking emulation, Squirrel is designed to use as much real-world software components as possible. Conceptually, it replaces the wireless hardware, device driver's Hardware Abstract Layer (HAL) and the IEEE 802.11 kernel module, and provides upper layers with interfaces that are same as on real wireless devices. Consequently, testing targets on Squirrel are real software components that are deployable to real-world devices. Squirrel is configurable through a plug-in system. This allows different wireless models and mobility managers to be developed for Squirrel.

Studies in this dissertation use a CSMA/CA model that emulates the behaviors of IEEE 802.11 networks. It uses a leaky bucket to track channel condition in vicinity of each node. As an event happens near a node, the corresponding leaky bucket increases by the number of nanoseconds the event takes to finish. All leaky buckets leak at the same rate as time, i.e. 1×10^9 per second. With this technique, channel condition is independently tracked for each node, which enables the model to not only shape throughput of traffic, but also reflect interference generated by other transmitting nodes. As a result, channel congestion is properly emulated.

Effectiveness tests are conducted for the CSMA/CA model in Squirrel. It demonstrates that Squirrel effectively reflect network congestion in the test and show a similar trend of iperf UDP throughput compared to the same test conducted on Android devices. In addition, Squirrel's throughput is also tested using the PassThrough network model. On a workstation with a 6-core CPU, Squirrel achieves 1200 Mbps throughput in total.

Second, an interframe compression method, ICTL, is designed and implemented. Unlike traditional compression methods where payload of each frame is compressed directly, ICTL compresses the difference between consecutive frames. The interframe compression design is inspired by video compression algorithms, but is adapted to be more suitable for vehicular networking. Two message types, KF and DF, are introduced by ICTL. A KF encodes application data directly, while DF encodes the difference between the application data and a previously transmitted KF. In a general transmission sequence, a KF is followed by a number of DFs. Since a DF is much smaller than a KF, this reduces average bandwidth consumption.

Behaviors of ICTL are studied under different configurations and scenarios, using analytical method as well as experimental method on Squirrel. Based on mathematical analysis, it is speculated that reducing bandwidth consumption should help increase ADR as well. With this in mind, an adaptive ICTL is designed and implemented. Unlike in basic ICTL where a fixed CL is used, adaptive ICTL goes through a process for each frame to determine whether it should be a KF or a DF should be send, based on average frame size within current ICTL Cycle.

The adaptive ICTL was tested on Squirrel, under different network utilization level. It has been shown that adaptive ICTL achieves over 50% in reduction of bandwidth consumption and maintains relative high ADR when network is congested. The adaptive ICTL algorithm achieves the lowest bandwidth consumption and the highest ADR among all the tested ICTL and tradition compression algorithms, in all test cases.

In an effort to promote reproducibility and applicability of the research, the implementation of Squirrel is open-source and available at `https://github.com/squirrel-land/`

`squirrel`, and the implementations of ICTL as well as adaptive ICTL are open-source and available at `https://github.com/songgao/ictl`.

Chapter 6

Future Work

As an extension of this work, multi-hop scenarios can be studied. This chapter describe a general idea of using Multi-Point Relays (MPRs) in multi-hop vehicular networks.

In this study, the effort has been focused on improving efficiency and reliability in single hop transmissions. However, for large vehicle platoons, this is not sufficient. Normally one of vehicles in the platoon serves as controller, and helps with coordinating different vehicles. As a result, it is important that platooning messages from the control vehicle can reach all vehicles in the platoon. For large platoons that are composed of many vehicles, it is normally impossible to transmit message from one end of platoon all the way to the other end within one hop. Hence, an efficient yet reliable rebroadcasting scheme needs to be developed for platooning applications.

One of the ways to design such a rebroadcasting scheme is to use MPRs. MPR is a concept proposed by Optimized Link State Routing (OLSR) Protocol [44]. As its name suggests, a MPR node relays messages for other nodes. To be more specific, each node has a minimum set of 1-hop neighbors as MPRs, so that there exists a path to each of its 2-hop neighbors via a node selected as an MPR. This way, a naive broadcast reaches all 1-hop neighbors, while MPRs relays the messages to 2-hop neighbors. Since only a subset of 1-hop neighbors are selected as MPRs, this reduces relaying overhead significantly. OLSR uses MPRs to relay routing information, but the concept of MPRs can be used for general purpose flooding as well. In vehicular network, platooning applications can construct MPRs to propagate platooning messages within the entire platoon.

To calculate MPR set, each node needs to know its 1-hop neighbors and 2-hop neighbors. OLSR uses Hello messages to discover local topology. A hello message is like a beacon, which contains the node's basic information as well as its 1-hop neighbors. By exchanging Hello messages, nodes are able to sense their 2-hop neighbors.

For a dynamic network like vehicular network, it is important to keep MPR set up to date in order to achieve good delivery ratio. The frequency that these messages are broadcast is a key parameter to performance. If the messages are broadcast too frequently, they generate a lot of overhead. If the messages are too infrequent, the MPR set may often be out of date, especially for a dynamic network where topology changes very often.

In vehicular networks, vehicles are already broadcasting beacon-like messages through WSMP, such as BSM. These messages allows all vehicles sense 1-hop neighbors. If the vehicles construct Hello messages as well, they can select MPRs in the same way as OLSR. This is a viable and straight-forward solution, but may cause large overhead for vehicular network, since the topology is more dynamic due to traffic condition, road condition, and terrains nearby.

Fortunately, applications running on vehicles have access to vehicular dynamics data as well as GPS positions. They enable some interesting ways to maintain a healthy MPR set for each node.

- GPS information can be used to assist selecting MPRs. The 1-hop neighbor set constructed with beacon messages provide information on communication range. Assuming that this range does not change dramatically within short period of time, it can be used to estimate 2-hop neighbors, allowing selecting MPR set without Hello messages.

- Vehicular dynamics data provides a history as well as current status of the vehicle's speed, acceleration, and steering information. Along with GPS positions, it is possible to predict each vehicle's trajectory. This is particularly useful when a vehicle is about to leave the platoon, or when a new vehicle is joining the platoon. This is the time when network topology of the platoon changes dramatically. Prediction on such an event allows each vehicle in the platoon to adjust MPR set accordingly to avoid high message loss rate by temporarily rebroadcasting more aggressively.

In short, vehicular environment and DSRC provide a way to construct MPR set without introducing Hello messages to the system, and a way for vehicles to predict topology changes, making them more responsive to dynamic environments.

A properly designed MPR algorithm for DSRC applications can reduce bandwidth consumption, thus improving ADR for multi-hop transmissions. Thus, this is a promising direction to improve vehicular networking efficiency and reliability for DSRC applications.

Bibliography

[1] Arada Systems. Locomate products. [Online]. Available: http://www.aradasystems.com/products/

[2] DENSO INTERNATIONAL AMERICA, INC. (2012, 1) Denso connected vehicle technology. [Online]. Available: http://www.densomedia-na.com/press_kits/naias2012/technology/Jan_9_V2X_Brochure_final%20.pdf

[3] Unex Technology Corp. Unex: V2x and v2x fusion technology. [Online]. Available: http://www.unex.com.tw/v2x

[4] D. Bevly, C. Murray, A. Lim, R. Turochy, R. Sesek, S. Smith, G. Apperson, J. Woodruff, S. Gao, M. Gordon *et al.*, "Heavy truck cooperative adaptive cruise control: Evaluation, testing, and stakeholder engagement for near term deployment: Phase one final report," 2015.

[5] J. Kenney, "Dedicated short-range communications (dsrc) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, July 2011.

[6] F. C. Commission *et al.*, "Fcc 99-305," *FCC Report and Order*, 1999.

[7] "Ieee standard for wireless access in vehicular environments (wave)–multi-channel operation," *IEEE Std 1609.4-2010 (Revision of IEEE Std 1609.4-2006)*, pp. 1–89, Feb 2011.

[8] "Ieee standard for wireless access in vehicular environments (wave) - networking services," *IEEE Std 1609.3-2010 (Revision of IEEE Std 1609.3-2007)*, pp. 1–144, Dec 2010.

[9] S. International, "SAE J2735 Dedicated Short Range Communications (DSRC) Message Set Dictionary," *Society of Automotive Engineers, DSRC Tech Committee*, 2016.

[10] C. Bergenhem, E. Hedin, and D. Skarin, "Vehicle-to-vehicle communication for a platooning system," *Procedia - Social and Behavioral Sciences*, vol. 48, pp. 1222 – 1233, 2012, transport Research Arena 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877042812028340

[11] Peloton Technology. Peloton technology - about. [Online]. Available: http://www.peloton-tech.com/about/

[12] Audi AG, *Owner's Manual 2016 Audi A7 Sportback / S7 Sportback*, 2016.

[13] A. Watts, "Computational Characterization of Drag Reduction for Platooning Heavy Vehicles," Ph.D. dissertation, Auburn University, 2015.

[14] M. P. Lammert, A. Duran, J. Diez, K. Burton, and A. Nicholson, "Effect of Platooning on Fuel Consumption of Class 8 Vehicles Over a Range of Speeds, Following Distances, and Mass," *SAE International Journal of Commercial Vehicles*, vol. 7, no. 2014-01-2438, pp. 626–639, 2014.

[15] W. F. T. IV and D. Murray, "An analysis of the operational costs of trucking: 2015 update," 2015. [Online]. Available: http://atri-online.org/wp-content/uploads/2015/09/ATRI-Operational-Costs-of-Trucking-2015-FINAL-09-2015.pdf

[16] S. Gao, A. Lim, and D. Bevly, "An Empirical Study of DSRC V2V Performance in Truck Platooning Scenarios," submitted to Digital Communications and Networks.

[17] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, 2008.

[18] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops.* ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[19] Q. Chen, D. Jiang, V. Taliwal, and L. Delgrossi, "Ieee 802.11 based vehicular communication simulation design for ns-2," in *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, ser. VANET '06. New York, NY, USA: ACM, 2006, pp. 50–56. [Online]. Available: http://doi.acm.org/10.1145/1161064.1161073

[20] H. Arbabi and M. C. Weigle, "Highway mobility and vehicular ad-hoc networks in ns-3," in *Proceedings of the Winter Simulation Conference*, ser. WSC '10. Winter Simulation Conference, 2010, pp. 2991–3003. [Online]. Available: http://dl.acm.org/citation.cfm?id=2433508.2433880

[21] J. Benin, M. Nowatkowski, and H. Owen, "Vehicular network simulation propagation loss model parameter standardization in ns-3 and beyond," in *Southeastcon, 2012 Proceedings of IEEE*, March 2012, pp. 1–5.

[22] R. Fernandes, F. Vieira, and M. Ferreira, "Vns: An integrated framework for vehicular networks simulation," in *Vehicular Networking Conference (VNC), 2012 IEEE*, Nov 2012, pp. 195–202.

[23] T. Lin, "Mobile ad-hoc network routing protocols: methodologies and applications," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.

[24] U.S. Naval Research Lab. Extendable mobile ad-hoc network emulator (emane). [Online]. Available: http://www.nrl.navy.mil/itd/ncs/products/emane

[25] P. B. Menage, "Adding generic process containers to the linux kernel," in *Proceedings of the Linux Symposium*, vol. 2. Citeseer, 2007, pp. 45–57.

[26] M. Krasnyansky, M. Krasnyansky, and F. Thiel, "Universal tun/tap device driver," 2002. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/tuntap.txt

[27] I. S. Association *et al.*, "802.11-2012-ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *Retrived from http://standards.ieee.org/about/get/802/802.11.html*, 2012.

[28] J. Turner, "New directions in communications (or which way to the information age?)," *Communications Magazine, IEEE*, vol. 24, no. 10, pp. 8–15, October 1986.

[29] N. Yin and M. Hluchyj, "Analysis of the leaky bucket algorithm for on-off data sources," in *Global Telecommunications Conference, 1991. GLOBECOM '91. 'Countdown to the New Millennium. Featuring a Mini-Theme on: Personal Communications Services*, Dec 1991, pp. 254–260 vol.1.

[30] grpc: A high performance, open source, general rpc framework that puts mobile and http/2 first. [Online]. Available: http://www.grpc.io/

[31] (2016) The go programming language. [Online]. Available: https://golang.org

[32] (2016) Using etcd with coreos. [Online]. Available: https://coreos.com/etcd/

[33] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 305–319.

[34] R. Janssen, H. Zwijnenberg, I. Blankers, and J. de Kruijff, "TRUCK PLATOONING DRIVING THE FUTURE OF TRANSPORTATION," TNO Innovation for Life, Tech. Rep., Feburary 2015. [Online]. Available: http://orfe.princeton.edu/~alaink/SmartDrivingCars/PDFs/TruckPlatooning-TNO.pdf

[35] Z. Wang and M. Hassan, "How much of dsrc is available for non-safety use?" in *Proceedings of the Fifth ACM International Workshop on VehiculAr Inter-NETworking*, ser. VANET '08. New York, NY, USA: ACM, 2008, pp. 23–29. [Online]. Available: http://doi.acm.org/10.1145/1410043.1410049

[36] C.-L. Huang, Y. Fallah, R. Sengupta, and H. Krishnan, "Adaptive intervehicle communication control for cooperative safety systems," *Network, IEEE*, vol. 24, no. 1, pp. 6–13, Jan 2010.

[37] Y. Fallah, C. Huang, R. Sengupta, and H. Krishnan, "Congestion control based on channel occupancy in vehicular broadcast networks," in *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, Sept 2010, pp. 1–5.

[38] J. B. Kenney, G. Bansal, and C. E. Rohrs, "Limeric: A linear message rate control algorithm for vehicular dsrc systems," in *Proceedings of the Eighth ACM International Workshop on Vehicular Inter-networking*, ser. VANET '11. New York, NY, USA: ACM, 2011, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/2030698.2030702

[39] S. International, “SAE J2945.1 On-Board System Requirements for V2V Safety Communications,” *Society of Automotive Engineers, DSRC Tech Committee*, 2016.

[40] D. Le Gall, “Mpeg: A video compression standard for multimedia applications,” *Communications of the ACM*, vol. 34, no. 4, pp. 46–58, 1991.

[41] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the h. 264/avc video coding standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[42] (2016) rkt, a security-minded, standards-based container engine. [Online]. Available: https://coreos.com/rkt

[43] (2016) Coreos is linux for massive server deployments. [Online]. Available: https://coreos.com/

[44] E. T. Clausen and E. P. Jacquet, “Optimized link state routing protocol (olsr),” October 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3626.txt

Postface

Research projects ultimately serve humanity. In exploring the unknowns, seeking challenges, and enjoying the joy of solving problems, we should always strive to discover possibilities to make the world a better place to live in.

The author truly believes that research should be driven by real-world problems, and that, research should be reproducible and applicable in the real world. This belief has led to the efforts towards the real-world data driven, realistic targeted emulation approach used in this dissertation.