Fault Tolerant Solutions for DSRC Safety Applications in VANET

A Dissertation
Presented in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
with a
Major in Computer Science
in the
College of Graduate Studies
University of Idaho
by
Sherif Ibrahim Morsy Hussein

Major Professor: Axel Krings, Ph.D.
Committee Members:
Robert Rinker, Ph.D.;
Ahmed Abdel-Rahim, Ph.D.;
Azad Azadmanesh, Ph.D.
Department Administrator: Fredrick Sheldon, Ph.D.

August 2018

Authorization to Submit Dissertation

This dissertation of Sherif Ibrahim Morsy Hussein, submitted for the degree of Doctor of Philosophy with a Major in Computer Science and titled "Fault Tolerant Solutions for DSRC Safety Applications in VANET", has been reviewed in final form. Permission, as indicated by the signatures and dates below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor:	Axel Krings, Ph.D.	Date
Committee Members:	Robert Rinker, Ph.D.	Date
	Ahmed Abdel-Rahim, Ph.D.	Date
	Azad Azadmanesh, Ph.D.	Date
Department Administrator:	Frederick Sheldon, Ph.D.	Date

Abstract

Vehicular Ad Hoc Networks (VANETs) are an emerging technology in Intelligent Transportation Systems (ITS) that aim to enhance traffic efficiency and safety. VANET deploy Dedicated Short Range Communications (DSRC) safety applications that are intended to alert drivers of dangerous road conditions and road hazards, e.g., during low visibility, to reduce accidents. DSRC offers the wireless support for Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communication to form the VANET. This requires that each vehicle be equipped with an On-Board Unit (OBU) and the infrastructure, such as an intersection, be equipped with a Road Side Unit (RSU). DSRC safety applications enable vehicles to exchange status information with their neighbors in a beacon message called Basic Safety Message (BSM). The reliability of safety applications heavily depends on the timely reception of these BSMs. However, DSRC safety applications might be exposed to the full range of malicious attacks associated with wireless technology.

In this dissertation, we concentrate on wireless jamming and GPS time spoofing attacks and their impact on the reliability of safety applications. First, we introduce a new hybrid jammer, that combines the properties of several types of jammers. The new jammer has the capabilities to 1) prevent legitimate nodes from accessing the medium to send their BSMs, and 2) make innocent nodes appear as misbehaving. As a mitigation strategy for this new jammer, we propose a detection algorithm that can differentiate between legitimate nodes subjected to the attack and misbehaving nodes. Then we present a series of algorithms to enhance safety application reliability in the presence of GPS time spoofing attacks. The mitigation strategies are as follows: 1) A decentralize clock synchronization algorithm is introduced that is capable of handling GPS time spoofing attacks; 2) An enhanced clock synchronization algorithm is proposed that considers a more realistic and stronger fault model. Finally, 3) a clock synchronization algorithm is presented that considers a mixed environment consisting of autonomous and connected vehicles in the presence of omission faults.

The proposed clock synchronization algorithms work as an augmentation to the current GPS synchronization protocol. Their effectiveness are demonstrated using laboratory and field experiments, as well as simulations using NS3 and SUMO. Finally, we show that the proposed mitigation strategies enhance the safety applications reliability with no extra hardware nor modifications of existing standards.

Acknowledgements

First and foremost, I would like to take this opportunity to thank God for being my strength and guide in this research. Without Him, I would not have had the wisdom or the physical ability to do so.

I express my sincere gratitude to my advisor Prof. Axel Krings for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. This work would not have been possible without his guidance in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my dissertation committee: Dr. Robert Rinker, Dr. Ahmed Abdel-Rahim, and Dr. Azad Azadmanesh, for their insightful comments and encouragement.

Moreover, I am thankful to the Egyptian Government for funding my Ph.D. study here at the University of Idaho.

I am also deeply grateful to the University of Idaho's Computer Science faculty, staff, and students for how they have affected my life and studies and for providing me with an environment that enabled me to work, especially with my best friend Mohamed. From outside the Computer Science faculty, I would like to thank Sameh Sorour and Ahmed Ibrahim for their support, help, and encouragement.

Finally, to my caring, loving, and supportive wife, Amira Mostafa: my deepest gratitude. Your encouragement when the times got rough are much appreciated and duly noted. It was a great comfort and relief to know that you were willing to provide management of our household activities while I completed my work. My heartfelt thanks.

Dedication

I would like to dedicate this dissertation to the memory of my father,

Ibrahim Morsy

who inspired scientific thinking in my life;

To my mother,

Amal Anawr

who provided me the endless support;

To my wife,

Amira Mostafa

for her patience, and encouragement toward the completion of this work;

Finally to my kids,

Ziad & Zaina

who make it all worthwhile.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AIFS	Arbitration Inter-Frame Space
AV	Autonomous Vehicles
BSM	Basic Safety Message
BSW	Blind Spot Warning
CCH	Control Channel
CLW	Control Loss Warning
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTS	Converging Time Synchronization
DCF	Distributed Coordination Function
DIFS	Distributed Inter-Frame Space
DNPW	Do Not Pass Warning
DoS	Denial of Service attack
DSRC	Dedicated Short Range Communication
EDCA	Enhanced Distributed Channel Access
EEBL	Emergency Electronic Brake Lights
FCC	Federal Communication Commission
FCW	Forward Collision Warning
FIFO	First-in First-out
FSPL	Free Space Path Loss
FTA	Fault Tolerant Average
FTM	Fault Tolerant MidPoint
GPS	Global Positioning System
HV	Host Vehicle
HCS	Hybrid Clock Synchronization
I2V	Infrastructure-to-Vehicle
ICA	Intersection Collision Avoidance
IEEE	Institute of Electrical and Electronics Engineer
IMA	Intersection Movement Assist

ITS	Intelligent Transportation Systems
LCW	Lane Change Warning
MAC	Media Access Control
MANET	Mobile Ad-Hoc Networks
MSR	Mean Subsequence Reduced
MSE	Mean Subsequence EgoCentric
MSEP	Mean Subsequence EgoPhobic
MSRH	Mean Subsequence Reduced History
MSFR	Mean Subsequence with Fixed Reduction
MSER	Mean Subsequence EgoCentric with Reduction
MSEPR	Mean Subsequence EgoPhobic with Reduction
NHTSA	National Highway Traffic Safety Administration
OBU	On-Board Unit
OMSR	Omission Mean Subsequence Reduced
PCAP	Packet Capture
PCF	Point Coordination Function
PDR	Packet Delivery Ratio
PER	Packet Error Rate
PHY	Physical Layer
PIFS	PCF Inter-frame Space
RSU	Road Side Unit
RV	Remote Vehicle
SAE	Society of Automotive Engineers
SCH	Service Channel
SIFS	Short Inter-frame Space
SNR	Signal to Noise Ratio
SOA	Strictly Omission Asymmetric
TTD	Time Table Diffusion
TTT	Time Table Transfer
USDOT	United States Department of Transportation
V2I	Vehicle-to-Infrastructure

V2V	Vehicle-to-Vehicle
VANET	Vehicular ad hoc Networks
WAVE	Wireless Access in Vehicular Environments
WDoS	Wireless Denial of Service attack (WDoS)
WSA	WAVE Service Advertisement
WSMP	WAVE Short Message Protocol
WSM	WAVE Short Messages

CHAPTER 1

Introduction

Motor vehicles are the most popular way of moving people from one place to another, especially for small distances. They are convenient, save people time, and are not confined to set departure and arrival times such as buses and trains. However, according to the National Highway Traffic Safety Administration (NHTSA) report from August 2016 [1], the United States had 35,092 fatalities from crashes on U.S. roadways during 2015, an increase from 32,744 in 2014. The number of people injured on the Nation's roads increased in 2015 from 2.34 to 2.44 million.

Crashes might occur due to reasons that are out of our control, such as low visibility, or due to driver behavior. The NHTSA estimated that the critical reason for crashes can be assigned to the driver 93% of the time [2]. It takes only one moment of inattention, e.g., checking phone messages or texting, to risk losing one's life. For example, Figure 1.1 shows the percentages of people using their phones while driving [1]. Some states prohibit all drivers from using hand-held cell phones while driving. However, there is still a large number of people injured or killed in the United States as the result of crashes linked to distracted driving. Vehicle improvements, including air bags and electronic stability control, have contributed to reducing such traffic fatalities. The NHTSA has predicted that the number of crashes could be decreased dramatically with the use of emerging Intelligent Transportation Systems (ITS) technologies [3].

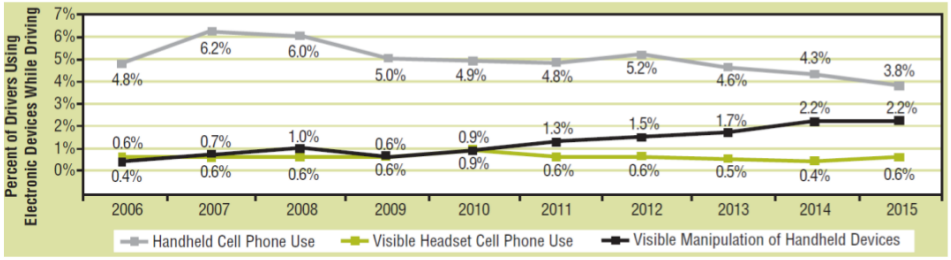

FIGURE 1.1: Driver use of electronic devices, 2006-2015 [1]

ITS integrates advanced communication technologies in transportation to improve traffic safety and efficiency. It deploys Dedicated Short Range Communications (DSRC) into vehicles and the infrastructure to enable both Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications to form the Vehicular Ad Hoc Network (VANET). The U.S. Federal Communications Commission (FCC), allocated 75MHz of DSRC spectrum at 5.9 GHz to be used exclusively for V2V and V2I communications. It requires that each vehicle in a VANET be equipped with an On-Board Unit (OBU) and each intersection with a Road Side Unit (RSU). ITS provide a variety of useful DSRC safety applications that aim to enhance traffic efficiency and increase driver awareness. For example, rear-end collision safety applications such as Emergency Electronic Brake Lights (EEBL) and Forward Collision Warning (FCW) can be used to alert drivers about possible collisions ahead. Such applications are very valuable, especially when drivers do not have direct visual contact to the hazard ahead.

1.1 THE BIG PICTURE

In recent decades, several applications were proposed for use in VANETs. DSRC applications can be classified into two main categories: non-safety and safety applications.

The first category includes non-safety applications aiming at enhancing the use of road networks in order to avoid highly congested road segments, thereby achieving efficient travel times and decreased fuel consumption. The applications aim to make the driving experience more comfortable, and provide access to different services such as weather forecast and paying tolls without the need to stop.

The second category, safety applications, aims to increase the safety of drivers, passengers, and pedestrians. Such applications save lives by avoiding or minimizing the effects of traffic accidents. These applications can predict and detect possible crashes and road hazards; they then broadcast warning messages over road network segments. However, as these applications are based on wireless communication,

they might be subjected to the full spectrum of security vulnerabilities associated with such technologies.

We will address some of the challenges that affect the reliability of safety applications and might cause them to fail. Such challenges are directly related to our research. In particular, we will focus on malicious attacks that affect data availability and consistency in VANET. Examples of attacks are as shown next.

Sybil attack: In this attack a vehicle pretends to be several vehicles at the same time but in different locations, thereby creating a huge security risks in the network. An attacker might use this attack for his own beneficial purposes. For example, the attacker transmits multiple messages with different identities to the other vehicles to make them feel that there is heavy traffic ahead. However, they may also serve to provoke false alerts to drivers, potentially resulting in unexpected reactions that may lead to accidents.

Message spoofing attack: Here a malicious vehicle sends incorrect or bogus information to other vehicles to deceive them about certain events. For instance, an attacker may transmit a message announcing "hazard ahead" to force others to change their direction or cause unexpected reactions.

DoS attack: This is one of the most serious attacks in any network, and it can be conducted using various techniques. A misbehaving vehicle may broadcast irrelevant or unimportant messages to use up large amounts of bandwidth, or jam the channel, thus preventing the legitimate nodes from sending warnings in the proper time. Such attack might cause safety applications to fail, as they are time critical applications and late warning renders them to be useless.

GPS spoofing attack: A GPS receiver attached to the OBU in each vehicle is the main sources of time and positioning information. An attacker might use a GPS satellite emulator to generate manipulated or spoofed GPS signals. GPS spoofing attacks may aim to cause a localization problem, or cause clock synchronization problems between vehicles. Synchronization faults might cause safety applications to fail as they will not be able to alert the drivers in the proper time to avoid a crash.

1.2 RESEARCH MOTIVATION AND OBJECTIVES

As mentioned above, DSRC safety applications are time sensitive applications, where delayed alerts might cause safety applications to fail. For example, imagine a person launching an object into traffic while disrupting the medium in the region around the induced hazard. This disruption of communication could cause failure of the safety application. A driver seeing the hazard would react, e.g., by braking hard. However, such attack will prevent safety applications from warning the drivers without visual contact, potentially leading to rear-end collisions. Such failure, whether due to benign or malicious reasons, has the potential to undermine public trust and acceptance of DSRC or VANET technologies. Since the safety applications operate in a critical infrastructure, where their failure could result in injury and loss of life, reliability is crucial. Specifically, reliability of the real time communication is necessary to respond to the warning messages before it becomes too late for a driver to react.

This research addresses the enhancement of safety application reliability in the presence of malicious attacks by using the principles of fault tolerance and survivability. Specifically, we will focus on the mitigation of wireless jamming and GPS time spoofing attacks. Our main objectives are to:

1. Introduce a new jammer attack and its potential impact on DSRC safety application reliability.

2. Design and implement mitigation strategies for the such jamming attacks.

3. Address the clock synchronization problem in VANET due to GPS time signal faults, and their impact on the reliability of different safety applications.

4. Propose distributed clock synchronization solutions capable of solving the clock synchronization problem when GPS-based approaches alone will fail. The solutions consider different safety applications and diverse attack models.

1.3 SUMMARY OF CONTRIBUTIONS

The major contributions of this research are based on real observations during field experiments using Arada Locomate Classic [4] devices, and simulations. The contributions are as follows:

- A new hybrid jammer is introduced that combines the properties of so-called constant and deceptive jammers in addition to characteristics resembling random jammers. The jammer is capable of causing safety application failure, and might cause legitimate nodes to appear misbehaving on the medium.

- A hybrid jammer detection algorithm is proposed as a mitigation strategy for hybrid jamming attacks. It can differentiate between misbehaving nodes and legitimate nodes subjected to this attack, thereby enhancing safety application reliability.

- A clock synchronization protocol capable of handling GPS time spoofing attacks is presented. The new protocol is based on approximate agreement and it does not require any extra hardware or message overhead. It works as an augmentation of the VANET's centralized clock synchronization approach, which is based on GPS.

- An Enhanced Clock Synchronization Algorithm (ECSA) for VANET is introduced. ECSA executes a new agreement algorithm addressing a more realistic and stronger fault model in the presence of GPS time spoofing attacks. The proposed algorithm aims to enhance the EEBL safety application reliability.

- A new clock synchronization algorithm for a mixed traffic model that includes autonomous vehicles is introduced. The proposed algorithm considers the Intersection Collision Avoidance safety application. It is able to deal with GPS time faults in the presence of omission faults. Field test results show that the augmentation of DSRC capability for autonomous vehicles is crucial.

1.4 DISSERTATION OUTLINE

The rest of the dissertation is structured as follows. In Chapter 2 we will give background related to the proposed work. A new hybrid jammer and a detection algorithm as a mitigation strategy will be introduced in details in Chapter 3. Chapter 4 presents a decentralized clock synchronization approach for VANET. An enhanced clock synchronization algorithm capable of handling omission faults and malicious nodes in VANET is introduced in Chapter 5. A clock synchronization protocol that considers a different safety application and fault model is presented in Chapter 6. Finally, Chapter 7 discusses conclusions and future work.

CHAPTER 2

Background

2.1 Intelligent Transportation System

ITS aim to improve road safety by integrating communication technologies into vehicles and the infrastructure. Connected vehicle technologies deploy DSRC safety applications that are intended to alert drivers of dangerous road conditions and road hazards, e.g., during low visibility, to reduce accidents. DSRC allows vehicles to exchange their status information, including GPS coordinates and time values, using V2V communication. This requires that each vehicle be equipped with an OBU. A GPS receiver attached to each OBU is considered as the main source of the exchanged information between vehicles. The OBU also enables V2I communication, which requires that the infrastructure, e.g., an intersection, is equipped with an RSU.

The FCC, in collaboration with the United States Department of Transportation (USDoT), considered 75 MHz of bandwidth at 5.9 GHz (5.850-5.925 GHz) to be utilized by DSRC communication [5, 6]. As shown in Figure 2.1, the DSRC bandwidth is divided into seven 10 MHz channels, one Control Channel (CCH) denoted by CH178, and six Service Channels (SCH), i.e., CH172, 174, 176, 180, 182, and 184. The remaining 5 MHz are reserved for future use. This research considers CH172, which is dedicated for V2V public safety communications and DSRC safety applications.

2.1.1 *DSRC Safety Applications*

In recent years, several DSRC safety applications were developed to operate in VANET. These applications focus on accident prevention and hazard avoidance. They enables vehicles to exchange their status information, including GPS coordinates and time values, using V2V communication. Each vehicle executes safety applications and contributes by sending or receiving information collaboratively. From a safety application point of view, we refer to the vehicle generating an alert

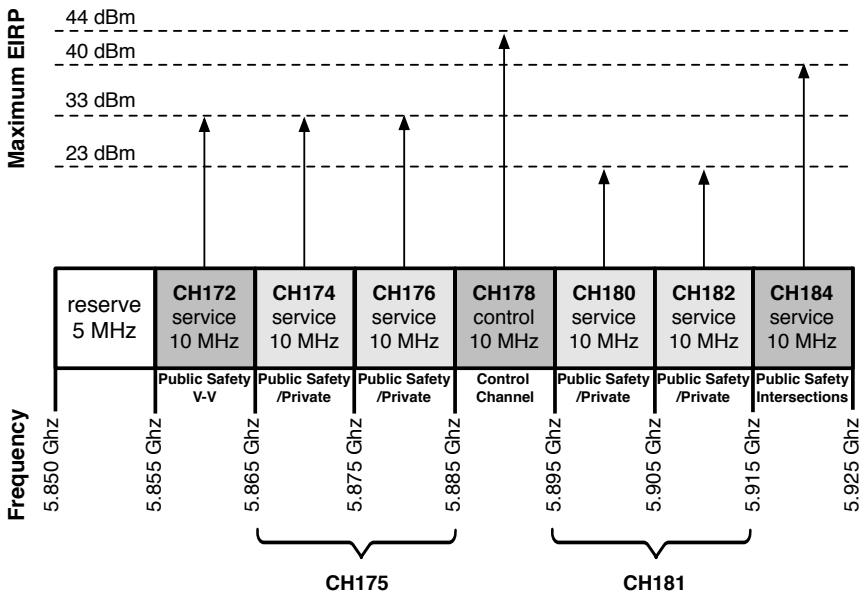

FIGURE 2.1: DSRC channels [7]

as Remote Vehicle (RV), and the vehicle making a decision in response to the alert as Host Vehicle (HV). A range of DSRC safety applications focusing on crash scenarios and their prevention have been described in [8] such as:

- *Forward Collision Warning* (FCW), shown in Figure 2.2, alerts the driver of HV in case of an imminent rear-end collision with the RV, driving ahead in the same lane and direction. FCW is useful in scenarios when approaching a vehicle that is decelerating or stopped.

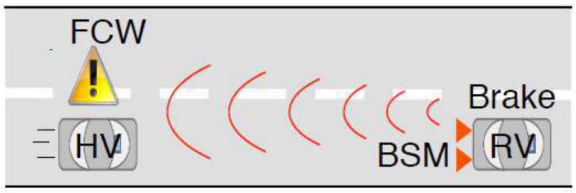

FIGURE 2.2: FCW safety application

- The *Emergency Electronic Brake Lights* (EEBL), depicted in Figure 2.3, alerts the driver of the HV to decelerate once receiving a hard brake event from an RV.
- The *Do Not Pass Warning* (DNPW) in Figure 2.4 warns the driver of the HV during a passing maneuver attempt that another vehicle is traveling in the opposite direction.

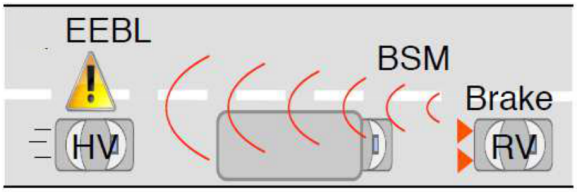

FIGURE 2.3: EEBL safety application

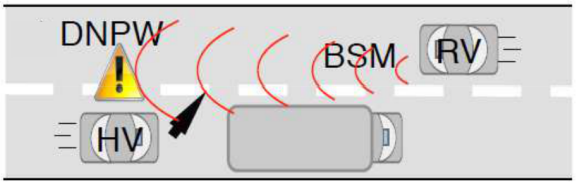

FIGURE 2.4: DNPW safety application

- *Blind Spot Warning + Lane Change Warning* (BSW+LCW), shown in Figure 2.5, warns the driver of the HV attempting to change into a lane, which happens to be occupied by another vehicle traveling in the same direction, but is in its blind-spot.

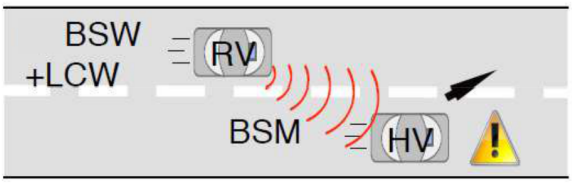

FIGURE 2.5: BSW + LCW safety applications

- *Intersection Movement Assist* (IMA), shown in Figure 2.6, warns the driver of an HV entering an intersection that there is a high probability of collision with an RV.

- *Intersection Collision Avoidance* (ICA) is similar to the IMA safety application, however, it adds autonomous braking as a system response in case that the driver ignores the warning.

In this research, we are concerned with rear-end and intersection collision scenarios as they represents the biggest percentages of all crashes occur. Specifically, in 2010, intersection collisions represented about 40% of all crashes, whereas rear

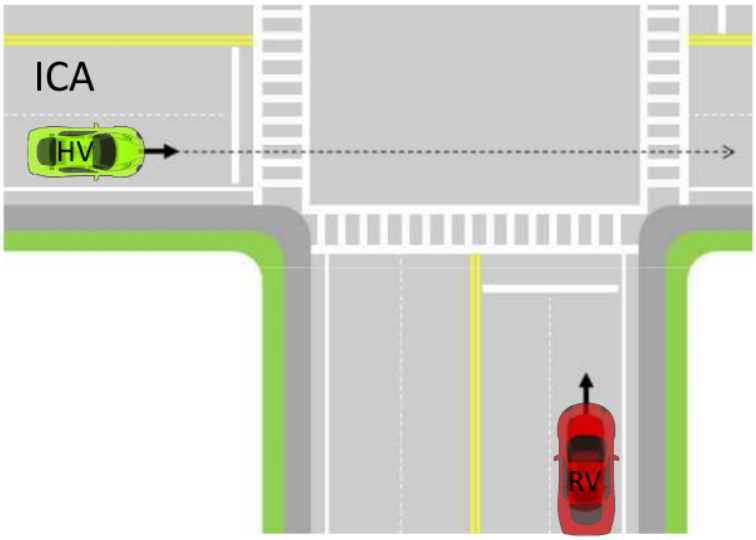

FIGURE 2.6: IMA safety application

end collisions represented 28% [9]. Thus, we consider both EEBL and Intersection Collision Avoidance (ICA) safety applications.

2.1.2 *Basic Safety Message*

DSRC safety applications rely on a beacon messages called Basic Safety Message (BSM). A BSM, also called heartbeat, is periodically exchanged between vehicles every 100ms [10]. As defined in standard SAE J2735 [11], a BSM consists of two parts. The first part is mandatory and contains specific BSMs such as speed, heading, location, brake status and time stamp. The second part is optional and includes additional information for certain applications. Figure 2.7 shows the BSM field names and their sizes.

BSM fields are described in standard SAE J2735 as follows:

- *DSRC_MessageID* is used to define the message type, and to inform the receiving application how to interpret the remaining bytes.
- *MsgCount* is used to sequence messages that were sent by the same sender with the same *DSRC_MessageID*.
- *TemporaryID* identifies the vehicle that is sending this BSM. This value changes periodically to ensure the overall anonymity of the vehicle.

```
BasicSafetyMessageVerbose ::= SEQUENCE {
   -- Part I, sent at all times
   msgID       DSRCmsgID,              -- App ID value, 1 byte

   msgCnt      MsgCount,               -- 1 byte
   id          TemporaryID,            -- 4 bytes
   secMark     DSecond,                -- 2 bytes
   -- pos      PositionLocal3D,
   lat         Latitude,               -- 4 bytes
   long        Longitude,              -- 4 bytes
   elev        Elevation,              -- 2 bytes
   accuracy    PositionalAccuracy,     -- 4 bytes

   -- motion   Motion,
   speed       TransmissionAndSpeed,   -- 2 bytes
   heading     Heading,                -- 2 bytes
   angle       SteeringWheelAngle,     -- 1 bytes
   accelSet    AccelerationSet4Way,    -- 7 bytes

   -- control  Control,
   brakes      BrakeSystemStatus,      -- 2 bytes

   -- basic    VehicleBasic,
   size        VehicleSize,            -- 3 bytes

   -- Part II, sent as required
   -- Part II,
   safetyExt   VehicleSafetyExtension OPTIONAL,
   status      VehicleStatus          OPTIONAL,
   ... -- # LOCAL_CONTENT
   }
```

FIGURE 2.7: Basic safety message structure

- *DSecond* provides the BSM's time of the sender. It consists of an integer value representing milliseconds within a minute.
- *Latitude* and *Longitude* provide the geographic latitude and longitude of the vehicle, expressed in $1/10^{th}$ integer micro degrees.
- *Elevation* represents the geographic position above or below sea level.
- *PositionalAccuracy* consists of multiple parameters to define the accuracy of the geographic position with respect to each axis.
- *TransmissionAndSpeed* expresses the current speed value in unsigned units of 0.02 meters per second, combined with a value to represent the vehicle's transmission state.
- *Heading* provides the current heading and the orientation of the vehicle.

- *SteeringwheelAngel* shows the rate of change of the angel of the steering wheel in either direction.
- *AccelerationSet4Way* defines the acceleration values in 3 orthogonal directions, in addition to yaw rotation rates.
- *BrakeSystemStatus* provides the current brake system status, (brake usage, anti-lock brake status, auxiliary brake status), in addition to system control activity of the vehicle.
- Lastly, *VehicleSize* indicates the vehicle length and width.

In this research, we consider the *DSecond* to be the most important field in the BSM as it can be used to calculate a BSM's lifetime by the difference in time when a BSM was timestamped at the transmitter side and the reception time at the receiver side. It will be shown that the BSM's lifetime is used to calculate the freshness of the received BSMs. DSRC safety applications rely on received BSMs to take the proper decision in the proper time. As this research uses the EEBL and ICA safety applications as a case study for clock synchronization, a closer look at how it works is beneficial.

2.1.3 *Electronic Emergency Light (EEBL)*

The main motivation of EEBL is to alert a driver of a hard-braking event from a vehicle ahead in the same lane. This application is especially valuable in situations with limited visibility, e.g., fog, or when visibility is blocked by other vehicles. The EEBL timing model is depicted in Figure 2.8. Assume that the front vehicle, denoted by Remote Vehicle (RV), is braking hard at time t_{brake} due to a road hazard. The BSMs of the RV indicate this braking event. Note that, the upper case T refers to time intervals, whereas lower case t denote instances of time.

For the EEBL application to be effective, the rear vehicle, denoted by Host Vehicle (HV), has to receive at least one BSM indicating the event before t_{react} to avoid a rear-end collision. In the figure, BSM_x is the last BSM that can be received before this cutoff time is reached. Any BSM received after this time is of no use, as an alert to

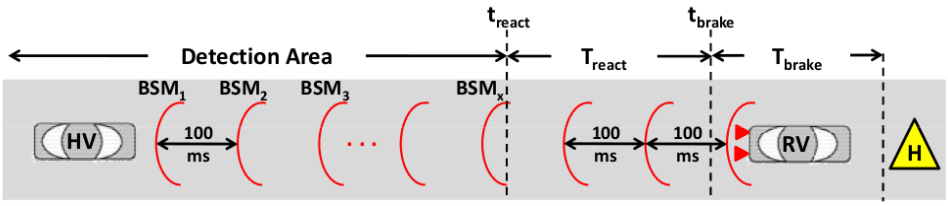

FIGURE 2.8: EEBL safety application timing model

the driver after t_{react} is too late, i.e., the EEBL reliability relies on the reception of the BSMs before time t_{react} in order to enable the driver to react and avoid a potential collision. Any delay, may it be due to benign reasons or due to tampering with the BSM time stamp may render the safety application useless.

2.1.4 *Intersection Collision Avoidance (ICA)*

The main motivation of the ICA application is to alert the driver of an imminent collision at an intersection. It relies on V2V communications as vehicles can calculate their relative position with respect to others. This application is valuable for both human driven vehicles and autonomous vehicles as the field of view might be blocked by nearby buildings or other vehicles.

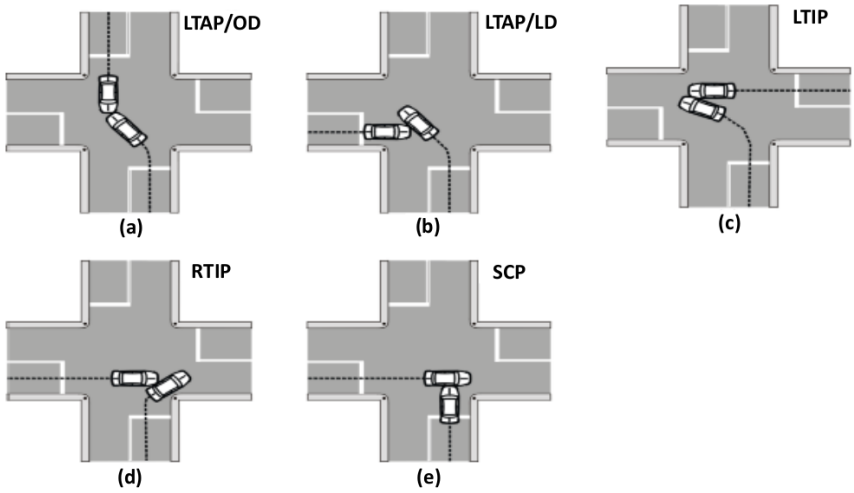

FIGURE 2.9: Intersection collision scenarios [12]

Figure 2.9 depicts several intersection crash scenarios that can be addressed by the ICA [13]: (a) Left Turn Across Path - Opposite Direction (LTAP/OD), (b) Left Turn Across Path - Lateral Direction (LTAP/LD), (c) Left Turn Into Path (LTIP), (d)

Right Turn Into Path (RTIP), and (e) Straight Crossing Paths (SCP). According to [12], the SCP scenario is the most prevalent scenario as it causes 70% of the crashes for stop signs and 43% in intersections controlled by traffic signals.

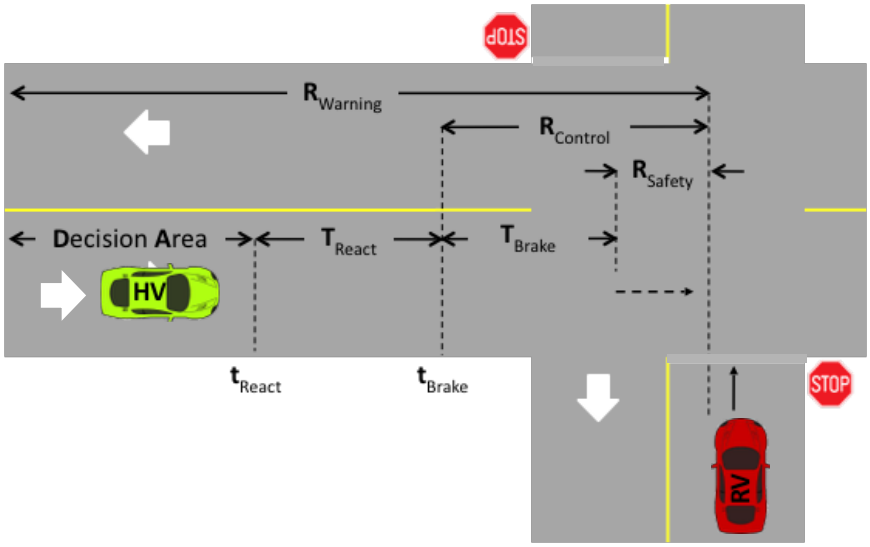

FIGURE 2.10: ICA safety application timing model

Figure 2.10 illustrates the timing for the ICA safety application. As the HV approaches an intersection, the ICA warns the HV driver in the range $R_{Warning}$ about a possible crash with an RV moving in the crossing traffic. Two possible cases are addressed in the Figure. In the first case it is assumed that the ICA system applies an automatic control action in the range labeled $R_{Control}$ if the driver of the HV ignores the ICA warning. Thus automatic braking would be initiated at t_{Brake}. The ICA automatic control should bring the HV to a complete stop in range R_{Safety}, thereby avoiding a collision with the RV. In the second case, it is assumed that the HV driver was alerted and reacts before t_{react}. The HV driver should be able to bring the HV to a complete stop before a collision with the RV as long as the driver brakes before time t_{Brake}.

In the cases above, the brake distance depends on the speed and road conditions. To get a feeling for brake distances, Table 2.1 shows the braking distances, derived from [14], for light vehicles moving with different speeds over wet asphalt. It should be noted that trucks need more time to stop.

TABLE 2.1: Braking distances on level roadways for wet asphalt [14]

Speed (mph)	Speed (Km/h)	Braking Distance (m)
25	40.2	18.3
30	48.3	26.3
35	56.3	35.8

The highly dynamic nature of VANET requires that each vehicle will have a fair and efficient access to the medium to send its BSMs. In the following section, we briefly describe the channel access rules.

2.2 802.11P

DSRC communication uses the IEEE 802.11p Medium Access Control (MAC) protocol, which uses Enhanced Distributed Channel Access (EDCA), an improvement of the Distributed Coordination Function (DCF) used in IEEE 802.11 [15].

2.2.1 *EDCA Channel Access Rules*

The EDCA allows different devices to access the channel based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). It uses four different categories of Access Classes (AC) associated with each level of priorities:

- AC0 for Background traffic (BK),
- AC1 for Best Effort traffic (BE),
- AC2 for Video traffic (VI) and
- AC3 for Voice traffic (VO).

Assume an OBU has a BSM to transmit, it senses the media first, and if it is idle for an Arbitration Interframe Space (AIFS), the OBU delays transmission by a random back-off time. Specifically, it selects a random back-off time from a Contention Window (CW) defined as $[0, CW + 1]$, which is initialized to CW_{min}. If the transmission attempt fails, the interval size is doubled, until it reaches CW_{max}. The

backoff value will only be decreased when the channel is sensed to be idle. The OBU will send its BSM immediately when the back-off value reaches zero. Figure 2.11 depicts the timing related to channel access for different inter-frame spacings.

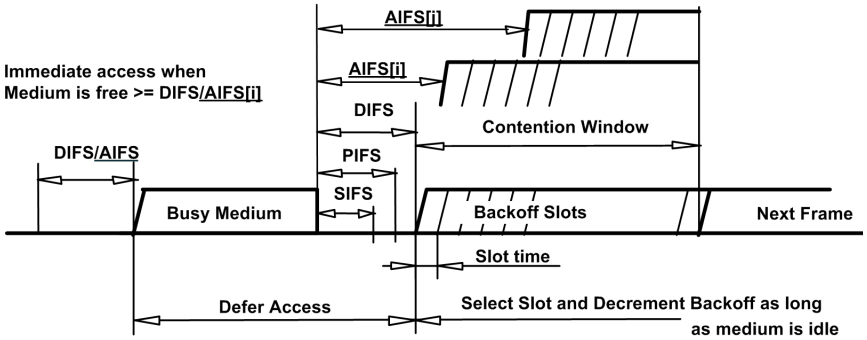

FIGURE 2.11: EDCA channel access prioritization [16]

It should be noted that OBU transmission performance is affected by the EDCA along with the buffering and scheduling mechanisms used at the transmitter side.

2.2.2 *Transmission Queue Behavior*

Buffering and scheduling mechanisms have a great impact on the IEEE 802.11p beaconing performance. The buffering mechanism defines the procedure of how BSMs are handled when they arrive at the queue, whereas the scheduling mechanism shows how BSMs leave the queue. The best known buffering mechanism is called Tail-Drop and is also known as Newest Packet Drop (NPD) [17]. Depending on the queue status, if the queue has an empty slot, the new BSM is buffered in the queue for transmission. Otherwise, the new BSM will be dropped as the queue is full.

In [18] another buffering mechanism called Oldest Packet Drop (OPD) was proposed. It is also known as head-drop queuing. Rather than dropping the newest BSMs, the OPD keeps the freshest BSMs in the queue. It drops the packet that contains the oldest information from the queue, creating an empty space for the newest arriving BSM.

Once the OBU has access to the media, e.g., using CSMA/CA, the BSM is taken from the queue for transmission using First-in First-out (FIFO) scheduling, where BSMs are sent in the order in which they arrived to the queue. Another scheduling

mechanism called Last-in First-out (LIFO) was discussed in [18], where the scheduler sends out the most recent arrived BSM. However, in reality, we have observed that the Arada locomate OBU used in during our field and lab experiments, is deploying the NPD buffering mechanism and FIFO scheduling.

Based on these observations and with respect to timeliness, it is concluded that, the longer a BSM is queued, the more outdated its information becomes. In the absence of misbehavior or jamming, given the relatively slow rate of 10 BSM/s, BSMs are unlikely to queue if traffic density is not overloading the medium. Otherwise, BSM might be considered to be outdated and discarded by the safety application.

2.3 FAULT MODEL

As DSRC is based on wireless communication, they inherit the full spectrum of challenges and attacks associated with such technology. In general terms we speak of faults. In the context of the following fault models, nodes are classified as either faulty, i.e., they generate erroneous values, or non-faulty, generating correct values. Figure 2.12 shows different fault models. The top level of the figure shows only one fault mode, which is the classic Byzantine fault (BYZ-1) [19], where no restrictions are made on the values received by different nodes.

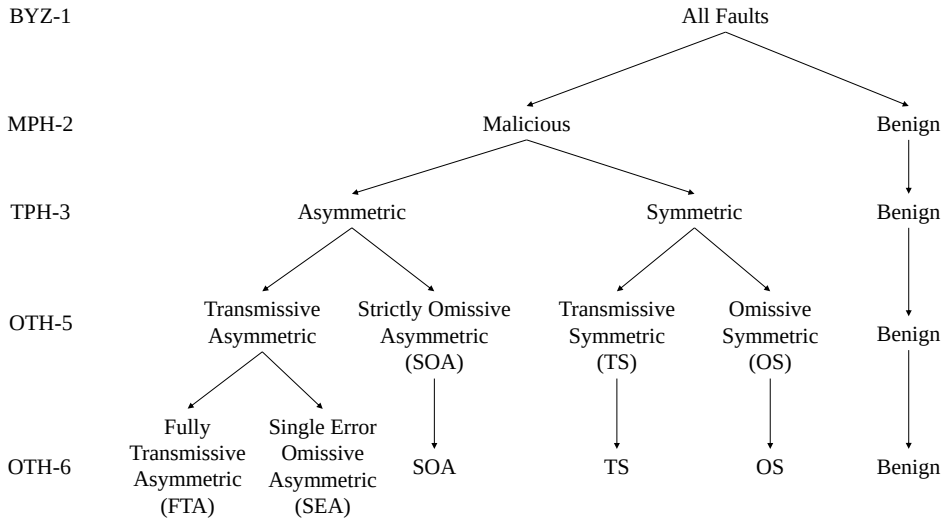

FIGURE 2.12: Omissive/Transmissive Six-Mode (OTH-6) Fault Model [20]

The second level of the figure represents the Hybrid Two-Mode fault model (MPH-2) [21], where faults are partitioned into Malicious and Benign faults.

The third level shows the Hybrid Three-Mode Fault Model TPH-3 of [22]. In this model, the malicious faults were partitioned into two faults based on their behavior. The first is the symmetric fault, where all non-faulty nodes are delivered the same value from the same faulty node. The second is the asymmetric fault, where the non-faulty nodes do not receive the same value.

The fourth level of the figure shows the Omissive/Transmissive Hybrid Five-Mode Fault Model (OTH-5) [23]. In this model both the symmetric and asymmetric faults were divided into two groups each. The symmetric faults are divided into Omissive Symmetric faults that are caused by the inability of the sender to deliver any value to any nodes, and Transmissive Symmetric faults, which occur when a faulty node transmits the same erroneous value to all other nodes. The asymmetric faults are divided into 1) Strictly Omissive Asymmetric faults that take place when the sender delivers correct values to some nodes and no value to one or more other nodes, and 2) Transmissive Asymmetric faults, which are the classic Byzantine fault.

The last fault model in Figure 2.12, is the Omissive/Transmissive Six-Mode Fault Model (OTH-6) [20]. This model divides the Transmissive Asymmetric faults of the OTH-5 fault model into Fully Transmissive Asymmetric and Single Error Omissive Asymmetric faults. The latter fault occurs when some nodes receive a single value while others do not receive any. Whereas the Fully Transmissive Asymmetric is the same as Transmissive Asymmetric fault defined in OTH-5. In this research we consider the OTH-5 and OTH-6 fault models.

2.4 VANET MALICIOUS ATTACKS

As indicated before, due to the wireless nature of DSRC the safety applications may be subjected to all security vulnerabilities of wireless communications. Several VANET security challenges and their causes were presented in [24]:

- Message spoofing attack: an eavesdropping vehicle modifies an existing message and sends it to its surrounding vehicles to deceive them. For example, an

attacker may transmit false information about traffic conditions so that other vehicles change their route and clear the way.

- Message replay attack: an attacker saves a copy of an old message and replays it later. In VANET systems, when a vehicle broadcasts a warning message, e.g., an accident warning, the attacker keeps a copy of this message and uses it later, causing unnecessary vehicle stops or traffic jams.

- Denial of service (DoS) attack: the attacker tries to prevent the legitimate nodes from accessing the medium and thus reduces the efficiency of the network, potentially preventing important messages from being sent in time.

- Timing attack: sending messages from one vehicle to the other vehicles in the proper time is of significant importance in VANET, especially in case of a hazard. In timing attacks, the malicious vehicle does not send the emergency message to its neighboring vehicles at the right time or deliberately introduces some delay, causing them to react lately.

- GPS time spoofing attack: the primary aim of the attacker is to spoof the timestamp contained in the GPS signal received by the vehicles under attack [25]. This act might cause the received messages from such vehicles to be perceived as outdated and hence cause serious problems in the VANET.

- Sybil Attack: a malicious vehicle in a VANET pretends to be multiple vehicles by sending multiple messages with different source identity, thereby creating an illusion to the other vehicles in the network for the benefits of this attacker. Such act is an active attack which causes performance degradation in the VANET system.

In this research, we consider two kind of attacks. The first is based on DoS attacks, which can be conducted using wireless jamming or MAC layer selfish/malicious misbehavior. The second is based on GPS time spoofing attacks in the presence of omissions and malicious nodes.

2.4.1 *DoS attacks*

Denial of Service (DoS) is the most common destructive attack found in VANET regarding communication networks. Such attack denies all services provided by theVANET and causes safety applications to fail. The goal of traditional DoS attacks is to prevent good nodes from accessing the medium. DoS attacks could range from simple attacks that do not require any protocol knowledge to intelligent attacks, where the attacker is aware of its surroundings and the communication protocol. There are various types of attacks that can cause DoS, as will be described in the following subsections.

Jamming Models — Jamming is the act of emitting radio signals to interfere with communication between nodes in a wireless network [28]. It also can be defined as the "disruption of existing wireless communications by decreasing the signal-to-noise ratio at receiver sides through the transmission of interfering wireless signals" [29]. One goal of the jammer may be to decrease the Signal to Noise Ratio (SNR), thus making reception unreliable or impossible, or perhaps to destroy network packets. However, jamming can also be viewed from a data point of view as a DoS, where the injection of what appears to look like valid data denies other nodes access to the media. Jamming models can be categorized into 1) simple jamming, and 2) intelligent jamming [28, 30].

A *Constant Jammer* is the first jamming model in the simple jamming category. It emits a constant stream of random data that does not follow the MAC layer protocol. As a result, the medium appear to be constantly busy, thus blocking legitimate nodes from communicating with each other. However, it may also result in the corruption of ongoing packets. This type of jammer is very easy to launch and can damage network communications to the point that no one can communicate at any time. However, it is energy inefficient and can be easily detected.

A *Deceptive Jammer* is the second jamming model. It does not follow the channel access protocol by continually injecting a stream of what appears to be valid packets without any gaps between them. This jammer is more difficult to detect than a

constant because it transmits legitimate packets instead of random bits. However, it is also energy inefficient.

A *Random Jammer* is different from the above two jammers, as it aims at saving energy and being harder to detect. It switches randomly between periods of jamming and sleeping. During the jamming period its behavior resembles that of a constant jammer. Sleeping and jamming time periods are either fixed or random.

The *Reactive Jammer* is the last jamming model in this category. It senses the medium for ongoing communication, and when it senses a packet transmission, it emits a radio signal that collides with the packet, thus corrupting it. This type of jammer is much more difficult to detect.

Finally, an *Intelligent Jammer* is a protocol-aware jammer, which can target specific packets or packet types.

Beside the obvious difference in jamming behavior, the key issues for different jammers is detectability and power consumption, as described in [28, 30].

Misbehavior attack Models — The MAC protocol described in Subsection 2.2.1 assumes that every node that wants to access the medium plays by the rules. A node is considered to be misbehaving if it does not follow the protocol rules, e.g., to gain an unfair advantage in transmitting its packets, or to deny other legitimate nodes from transmitting packets. The misbehaving nodes initial intent may be to achieve higher channel access frequency (selfish) by manipulating backoff value timers. However, as the greediness of such nodes increases, the network starts to suffer with low/moderate/high level DoS attacks (malicious) [31]. This DoS attack is difficult to be detected since the misbehaving nodes are considered to be protocol aware nodes, and they are not sending a stream of traffic such as conventional jamming attacks. MAC layer misbehavior can be classified into two general categories [32], 1) selfish misbehavior [33, 34] and 2) malicious misbehavior [28, 35].

A selfish node deliberately violates its backoff timer to obtain a larger portion of the shared channel. In particular, it can reduce its back-off time (or consistently use the minimum value for the back-off congestion window), and/or use another

distributions for the congestion window [30]. The selfish node thereby increases its data transmission rate at the cost of other nodes.

A malicious node could prevent other nodes from communicating by either constantly generating strong signals to disrupt a normal node's signal, or by transmitting fake packets to occupy the medium. The Sybil attack also has malicious misbehavior [36, 37], as the malicious node impersonates several other nodes in order to disrupt the network. Compared to selfish behavior, malicious misbehavior is more difficult to detect and can result in more serious problems, greatly degrading the performance for normal users [32].

2.4.2 *GPS time spoofing attack*

In order to ensure safety application data consistency and reliability, vehicles participating in VANET should be synchronized with a clock value within a specified tolerance. Recall that a GPS receiver attached to the OBU in each vehicle is considered to be the main source of BSM data to determine their own system time. It is used as a centralized clock synchronization approach between vehicles. However, GPS receiver may be subjected to GPS signal outage or GPS time spoofing attacks. The attacker is assumed to be able to manipulate the received GPS signal in a limited geographical area in an arbitrary way. The primary aim of the attacker is to manipulate the time used by the OBUs of vehicles under attack in such a way that the time difference between time stamps of vehicles is large.

According to [38], the BSM time-to-live, which is the difference between the *timestamps* of the HV and RV, should be no more than 500ms. Any BSM that violate this time restriction should be considered outdated. Based on the time-to-live rule and GPS time spoofing attack, safety applications might face a serious denial of service capabilities if the attacker succeeds in making the time difference between the sending time of the BSM by RV and the reception time of the BSM at the HV more than the time-to-live limit. This act will cause the safety application to fail by forcing the HV to discard all such BSMs received from its neighbors.

2.5 SAFETY APPLICATION RESILIENCE AND FAULT TOLERANCE

Safety application reliability is related to the reception of BSMs in the proper time to alert the driver before it is too late to react. Benign faults or malicious acts, which cannot be fully predicted, could render safety applications useless. For example, a jamming attack might delay vehicles from sending BSMs to the point that the BSMs become outdated. As a result receivers discard the BSMs even if they contain an alert. Imagine that an intelligent attacker launches this attack combined with creating a physical hazard. Such attack might cause safety application failure and could result in catastrophic consequences like injury and loss of life. Furthermore, it could result in the public's loss of confidence in the underlying technologies. Thus, reliability and fault-tolerance are crucial.

To fulfill the fault-tolerance needs of such critical infrastructure system, this research is concerned with introducing mitigation for different attack models. This can be achieved by adding fault tolerant mechanisms, that enables the safety applications to continue operating properly in the presence of either benign or malicious faults. Not all faults are avoidable, but minimizing their impact can be an achievable goal.

CHAPTER 3

A New Hybrid Jammer and its Impact on the EEBL Safety Application

Wireless jamming has become a major research problem in VANET due to the ease in blocking communication between communicated vehicles. It can be conducted using one of the different jammer types discussed in Section 2.4.1. The objective of a jammer is to interfere with legitimate wireless communications, thus making reception unreliable or impossible, or perhaps to destroy network packets.

This chapter introduce a new hybrid jammer that combines the properties of constant and deceptive jammers, in addition to characteristics resembling random jammers. It is shown that the hybrid jammer is capable of 1) delaying the transmission of BSMs to the point of causing safety applications to fail, 2) making innocent nodes appear to be misbehaving, and 3) rendering voting based protocols useless as it affects the freshness of the transmitted BSMs. We consider the EEBL Safety Application and demonstrate the impact of different jamming scenarios on the safety application reliability. Furthermore, we introduce a detection algorithm as a mitigation strategy for the new jammer. The proposed algorithm is capable of differentiating between vehicles subjected to the hybrid jamming attack and misbehaving vehicles. Finally, a GUI system will be presented, that is used to measure the effectiveness of the hybrid jamming detection algorithm.

3.1 RELATED WORK

Recall that the mediums capacity is limited, the BSMs transmission rate is 10 BSMs/second, and the vehicular density is not fixed and can become very high during traffic jams. No vehicle should be allowed to violate the IEEE 802.11p MAC protocol to obtain a larger portion of the shared medium. Otherwise the transmission medium may become congested and vehicles starts to queue their BSMs. A node is considered to be misbehaving if it does not obey the IEEE 802.11p MAC layer

protocol rules, e.g., to gain an unfair advantage in transmitting its packets, or to deny other legitimate nodes transmitting packets. Different MAC layer misbehaving attacks, classified as selfish or malicious misbehaving, were described in Section 2.4.1. In the following subsections, we will summarize the previous research on the detection of misbehaving nodes.

3.1.1 *Selfish Misbehavior Detection Techniques*

In [33] the authors proposed a modification to the IEEE 802.11 protocol to detect selfish misbehaving nodes. The protocol enables the receiver to select a back-off value to be used by the sender. At the end of each transmission the receiver checks if the sender deviates from the selected back-off value. The receiver is capable of penalizing the sender, if a deviation in a transmission is detected. If the sender continues on deviating from the selected back-off value over multiple transmissions, the receiver identifies the sender as misbehaving.

A detection of greedy behavior in the MAC layer of IEEE 802.11 public networks (DOMINO) was proposed in [39], that relies on a large amount of historical data to perform its detection. DOMINO can be implemented on any access point (AP) in the network. It periodically collects traffic traces of active nodes during short intervals of time called monitoring periods. The collected traffic traces are then analyzed to detect any violations of the MAC layer protocol.

In [32] a schemes was presented to detect and defend against MAC-layer selfish misbehavior in IEEE 802.11 multi-hop ad hoc networks. The proposed algorithm calculates the channel occupation durations or channel occupation ratio r for each active node in the network. A node is said to be selfish if it occupy the channel more than normal nodes.

3.1.2 *Malicious Misbehavior Detection Techniques*

Recall that a selfish node deliberately violates its back-off timer to obtain a larger portion of the shared channel, and as the greediness of such nodes increase, the network starts to experience DoS. In such case, the node was said to be a malicious

misbehaving node. In [35], the authors proposed a monitoring technique that detects malicious misbehaving attacks. The authors considered the malicious misbehaving attacks behaving as intelligent jamming attacks in an IEEE 802.11 network that could increases the number of packet collisions. The proposed technique monitors the network and calculates the probability of collisions. The network is said to be subjected to malicious misbehaving attacks as the probability of collisions increases.

In [40] VANET DoS attack detection was based on a so-called "Packets entropy", by monitoring traffic traces during short monitoring windows. Based on the fact that malicious nodes emit more data packets, and given that when the probability of emission of packets changes, the entropy will also change. The distinction can be made between a normal network and a network under attack by calculating packets entropy in each case.

The previous detection approaches were mainly based on monitoring the medium and counting of transmitted packets or channel occupation duration. However, it will be shown that the new hybrid jammer has the potential to make innocent nodes appear selfish misbehaving for such detection algorithms.

3.2 A NEW HYBRID JAMMER FOR VANET

We now describe a new hybrid jammer that combines properties of constant, deceptive, and random jammers. The jammer emits continuous random bits like a constant jammer but the bits appear as regular packets as in deceptive jammers, without following the CSMA protocol. In addition the jammer will be dormant for most of the time and only jam the medium for specific time durations, e.g., half a second to a few seconds, which make it appear like a random jammer.

During jamming all other nodes believe that legitimate transmissions are taking place. As a result nodes refrain from transmitting and queue their BSMs until the medium becomes available again when jamming stops. No BSM will be lost as long as the queues of the nodes do not overflow. Due to this, jamming cannot be detected as a malicious attack by mechanisms that use packet error rates or delivery ratios. A formal definition of the new hybrid jammer will be given in conjunction with the

EEBL safety application's BSM timing and queuing model, and its impact on the application reliability will be analyzed.

3.2.1 *EEBL Safety Application Reliability*

The reliability of the EEBL safety application is conditioned on the reception of BSMs and making the correct decision in the proper time. Assume that the distance d between the HV and RV is equivalent to t_d seconds. Given that BSMs are spaced 100ms in time, this distance accounts for $b = t_d/0.1$ safety messages. However, one can only consider those BSM that are received at or before t_{react}. Different values for reaction time have been used, e.g., in [41] a driver's minimum reaction time used was 0.7 seconds, whereas [42, 43] assumed it to be 1s.

Let t_r denote the reaction time. Then reaction time accounts for $r = t_r/0.1$ BSMs. In line with the standard definition of reliability, i.e., $R(t)$ is the probability that the system is working to specifications during the entire time interval $[0, t]$ [44], we can define the EEBL application reliability as the probability of receiving at least one BSM message at or before t_{react}, i.e., one of BSM_i, for $i = 1, .., x$, where $x = b - r$. The safety application fails only if no BSM message is received at or before t_{react}. If one assumes that the reliability of one BSM is independent of that of another BSM, and using unreliability $Q(t) = 1 - R(t)$, the probability of all x messages being lost is

$$Q(t) = \prod_{i=1}^{x} Q_i(t_i) \tag{3.1}$$

where $Q_i(t_i)$ is the probability that BSM_i was not received and t_i is the time it should have been received. In [43] Q_i was computed based on packet error rates and packet delivery ratio. Equation 3.1 assumes that packet failure is independent.

3.2.2 *Transmission Queue Behavior and Field Test Observations*

After an OBU generates a BSM it is placed in the transmission queue, which is a FIFO queue [45, 46]. Once the node has access to the media, e.g., using CSMA/CA, the BSM is taken from the queue for transmission. Should the node not be able to

send the BSM before the subsequent BSM arrives, i.e., within 100ms, the new BSM is also queued. This could go on until the capacity of the transmission queue overflows, in which case packets are dropped.

In Section 2.2.2, two different queuing mechanisms were discussed. The first, NPD, also known as tail-drop queuing, implies that when a packet arrives at a full queue the newest packet is dropped. The second, OPD, also known as head-drop queuing, drops the oldest packet when a new one arrives at a full queue.

In the absence of misbehavior jamming, given the relatively slow rate of 10 BSM/s, BSMs are unlikely to queue if traffic density is not overloading the media. However, during field tests related to the study of the impact of jamming on V2V communications, we observed excessive queuing. In this experiment, three vehicles, V1, V2, and V3, were equipped with OBUs, specifically, Arada LocoMate Classic OBUs [4]. An additional LocoMate Classic OBU was configured to be a jammer capable of operating at different data rates. Vehicles were moving at speed of 15 m/sec in a 2 lane road, whereas the jammer was at fixed position in the middle of the test area. The exact parameters for the field test below are shown in Table 3.1. Experiment configurations, OBU commands, and a complete set of parameter options is given in Appendix A.

TABLE 3.1: Field test parameters

OBU	Arada Systems LocoMate Classic
Vehicle speed	15 *m/s*
Test range	straight 2-lane road
Test range length	1.35 km
Jammer position	600m from starting point
BSM rate	10 BSM/s (a BSM every 100ms)
Channel	Safety Channel 172
Transmitter power	18 dBm
Data rate	3 and 6 Mbps
Jammer power	18 dBm
Jammer data rates	3, 6, and 12 Mbps

Figure 3.1 shows logging of BSMs at V3 received from V1 and V2 just before the media was completely jammed. The test area can be divided into sequences of three regions: *No jamming region*: here BSMs were transmitted and received with

the expected spacing. *Jamming region*: in is the region the jammer starts to affect the transmission medium and gaps in reception were observed, as expected. *After jamming region*: here bursts of BSM were logged that follow the gaps of reception. After careful analysis of the timing and content of packets we could confirm that the bursts were due to OBU message queuing as the media was jammed. This queuing and subsequent burst behavior will be exploited by the hybrid jammer. After investigation of the log files of this experiment, we discovered that the queue of the Arada locomate OBU can hold up to 40 BSM. The knowledge of the queue size can be important information for an attacker, as will be described in Section 3.3.2. Also it was found that the Arada OBUs are using Drop Tail buffering or NPD, which drops the newest BSMs if the queue is full.

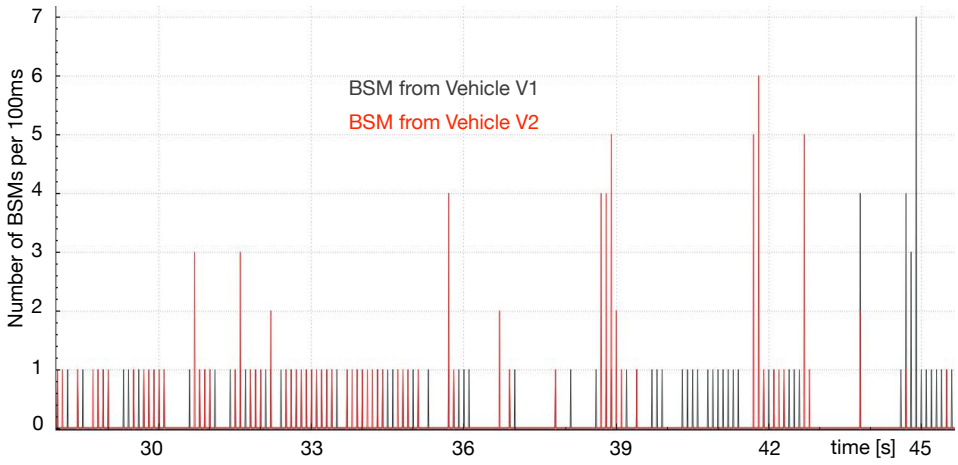

FIGURE 3.1: BSMs from vehicles 1 and 2 received by vehicle 3

3.3 HYBRID JAMMER SYSTEM MODEL

This section formally describes the behavior of the new hybrid jammer. Let T_{jam} denote the duration of jamming and let t^s_{jam} and t^e_{jam} denote the time jamming starts and ends respectively. A jamming period T_{jam} is thus $T_{jam} = t^e_{jam} - t^s_{jam}$. Now assume that the queue size of the OBUs is q. Jamming for T_{jam} will theoretically result in each OBU in the jamming area queuing $m = T_{jam}/0.1$ BSMs, where 0.1s is the BSM spacing, i.e., 100ms. Several issues arise:

- With respect to timeliness of BSMs, jamming for a duration of T_{jam} will result in a reception delay d_i for each BSM$_i$ queued, i.e., after jamming stops at t^e_{jam} the minimum message delay of a queued BSM$_i$ is

$$d_i \geq T_{jam} + \sum_{1}^{i} t_{min}, \quad 1 \leq i \leq m \tag{3.2}$$

  where t_{min} is the lower bound on the BSM transmission time from a specific OBU.

- A jamming period of T_{jam} will not result in the lost of BSMs if i) the period is short enough to not overflow an OBU's transmission queue, i.e., if $m \leq q$, and if ii) subsequent queue flushing of affected OBUs does not cause congestion.

An attacker can take advantage of both issues by selectively choosing T_{jam} to 1) intentionally causing BSM delays suiting its attack objectives, e.g., causing the EEBL to fail, and 2) ensuring the jamming duration does not cause queues to overflow. The consequences are multi-fold:

- The attacker can minimize being detected by carefully selecting the smallest BSM delay that renders a DSRC safety application useless. For example, if one blocks reception of x BSMs for the HV in the scenario depicted in Figure 2.8, the driver of the HV will not have ample time to react to the hazard. A jamming period of $T_{jam} = x0.1s$ would theoretically achieve this.

- The jammer makes other vehicles appear to be misbehaving. Since jamming causes each affected OBU to queue BSMs that are subsequently send in bursts after t^e_{jam}, the OBUs appear to be selfishly misbehaving by the algorithms described in Subsection 3.1.1. Specifically, each OBU's burst will be interpreted as getting disproportional access to the media, since the BSM rate of each node is expected to be 10 BSMs per second.

- Jamming detection mechanisms relying on Packet Delivery Ratio (PDR), e.g.,[47], will be ineffective as the jammer does not cause packets to be lost.

From the EEBL safety application point of view, the safety application fails if no BSMs indicating an event are received in time to alert the driver before it is too late to react. Even assuming BSM omissions were independent, which they are not, this would be an instance in which the unreliability in Equation 3.1 would evaluate to one, $Q(t) = 1$ or $R(t) = 1 - Q(t) = 0$. This constitutes failure of the EEBL application.

After discussing the hybrid jammer impact on EEBL safety application, the next subsection will introduce two possible attack models using the hybrid jammer. Then we will investigate the impact of both attacks and show that this jammer can cause the safety application to fail.

3.3.1 *Attack Model*

Stationary Attack Model — The stationary version of the jammer is demonstrated in the scenario depicted in Figure 3.2a). Assume a hazard is introduced and

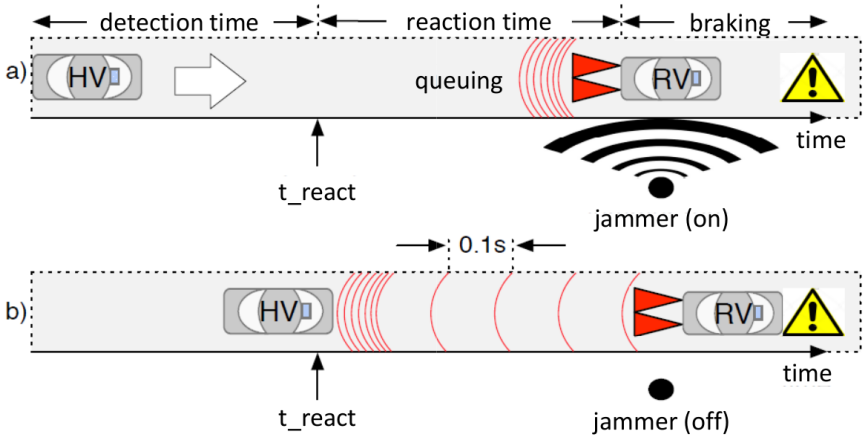

Figure 3.2: Stationary jammer

the jammer, positioned on the roadside next to the RV, jams for T_{jam} in coordination with the creation of the hazard. This causes the OBU of the RV to queue BSMs, as it cannot access the media during the jamming time. Once jamming stops, as indicated in Figure 3.2b), the RV sends all queued BSMs in a burst, followed by regularly spaced BSMs. The EEBL in the HV, which did not receive BSMs during T_{jam}, will receive its first BSM from the burst when it is already too late to react, i.e.,

after time t_{react}. Recall that, t_{react} represents the cutoff instance of time, where any BSM received after this time is of no use.

Mobile Attack Model — The scenario in Figure 3.3 shows how two collaborating attack vehicles H and J can cause the safety application to fail. Assume, as

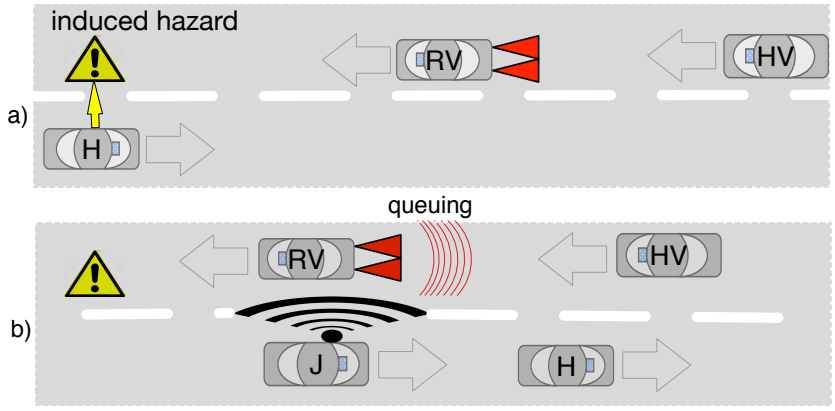

Figure 3.3: Mobile jammer

shown in part a) of the figure, that vehicle H causes a hazard, e.g., by launching an obstacle into oncoming traffic. The driver of the RV, who sees the hazard, will take some time to react. Now consider the scenario just before RV's driver reacts, as shown in Figure 3.3b). The collaborating vehicle J, which has a mobile jammer, follows vehicle H at a distance that positions it close to the RV just before the driver of the RV is expected to react to the hazard. Specifically, vehicle J keeps a distance from H short of 0.7s, the minimum reaction time. After vehicle H induced the hazard, vehicle J jams for T_{jam}, which can be determined based on the speed v_{HV} of the HV and the distance $d_{RV,HV}$ between the HV and RV. For example, if we consider 0.7s as the reaction time, $T_{jam} = d_{RV,HV}/v_{HV} - 0.7s$, i.e. T_{jam} is equal to the time separation between the RV and HV minus the reaction time. As in the case of the stationary jammer, the mobile jamming model is capable of delaying the reception of RV's BSMs to the point that it is too late for the driver of HV to react.

3.3.2 *Jamming Impact on Transmission Queues*

As indicated in the discussion of Figure 3.1 in Subsection 2.2.2, queuing of BSMs due to jamming was observed in real tests using commercially available OBUs. To validate the hybrid jammer's effect on queuing and to investigate if queuing is deterministic, we conducted several lab experiments using three Arada locomate Classic OBUs from Arada Systems [4]. One OBU was programmed to act as the hybrid jammer, the second served as the RV, and the third as the HV. The experiments were conducted in a controlled environment with no objects interfering with communications. The test parameters used are shown in Table 3.2.

TABLE 3.2: Hybrid jammer parameters

OBU Model	Arada Systems LocoMate Classic
Number of OBUs	3 (2 OBUs for two vehicles and 1 for the stationary jammer)
BSM generation	10 packets/s
Channel	Safety Channel 172
Transmitter power	21 dBm
Data rate	6Mbps
Jammer power and data rate	18 dBm, 6Mbps

The impact of jamming with $T_{jam} = 1, 2, 3$ and 4s of a typical experiment can be seen in Figure 3.4, where the number of BSMs that the HV received from the RV per 100ms is shown. After each jamming period a burst of BSMs, consistent with the number of BSMs expected to have been queued based on T_{jam}, can be observed.

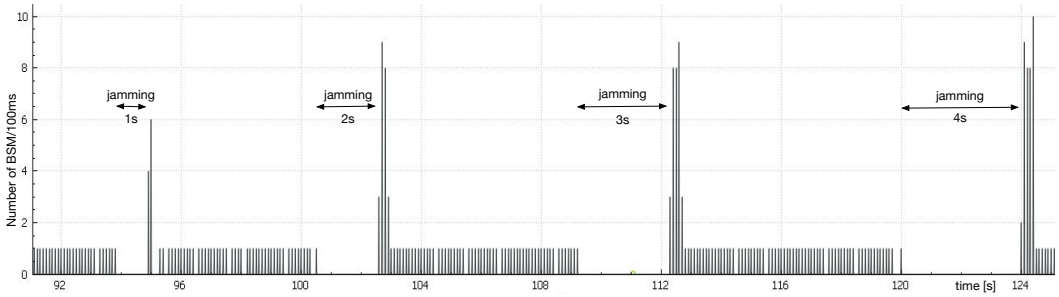

FIGURE 3.4: Queuing effect for jamming periods of 1, 2, 3, and 4s, showing the BSMs received by the HV from RV

After careful examination of the experiment, it was found that jamming in practice is not as precise as in theory, and we observed several factors that introduced

variability. First, the time from starting the jammer until it effectively jammed the media was nondeterministic, as it was not possible to start jamming precisely at the time intended. We attribute the observed differences to process initialization delays and runtime overhead of the Arada LocoMate Classic's operating system, which is Linux based, and the overhead associated with the jammer program. This delay can be observed in the scenario with $T_{jam} = 1$, which actually resulted in an effective jamming period slightly longer than 1s.

The second factor that created nondeterminism was attributed to the Arada Locomate Classic OBUs way of flushing their buffer. Specifically, experiments revealed that BSM spacing during flushing was on average 12.5ms, with minimum and maximum observed spacings of 10ms and 16ms respectively, and a standard deviation of 1.6. In Figure 3.4 this behavior was responsible for several spikes after the jamming period rather than one large spike.

Thirdly it should be noted that the figure only shows BSMs sent by the RV and received by the HV. The HV also queued messages during jamming, which also accessed the media using CSMA/CA. However, due to the low utilization of the media this should have had minimal impact on the data in the figure.

The experiments further suggested that the Arada Locomate Classic OBUs queued up to around 40 BSMs before messages were dropped. In general, such approach could be used by an attacker to experimentally determine the queue size of any device (OBU/RSU). The attacker can then use this information to conduct an attack to force the legitimate nodes to queue BSMs to a point that they became useless, and at the same time no BSM is lost. Thus PDR is not affected, and no detection methods based on PDR are effective. Specifically, detection based on PDR can be fooled, if the detection window is large enough. Also, vehicles under attack will be accused as misbehaving based on counting-based approaches.

3.4 HYBRID JAMMER DETECTION

This section introduces a hybrid jamming detection algorithm capable of detecting hybrid jamming and differentiating between misbehaving nodes and legitimate nodes subjected to such attack.

3.4.1 *Detection Algorithm*

Algorithm 1 outlines the hybrid jamming detection algorithm executing in the receiver thread of each OBU. Each vehicle maintains a Vehicles Neighborhood Table (VNT) to keep the latest information received from its neighbors. The i^{th} record in the *VNT* stores 1) the last BSM received from vehicle i, 2) receiving time of this BSM, and 3) the number of missing BSMs, *MissingBSM*, for vehicle i. The *VNT* is used to detect omissions as the receiving time of the latest BSM is saved.

Whereas the algorithm considers any BSM, it should be noted that for all practical purposes only BSMs from vehicles ahead of the HV could be selected, as they are relevant to the EEBL application. Since a BSM is expected from each vehicle approximately every 100ms, the omission of a BSM from vehicle i can be detected, e.g., using a watchdog mechanism, as indicated in the first "if" statement of the algorithm. Moreover, based on the information saved in *VNT*, vehicles can calculate the time passed since the last BSM was received from their neighbors. A vehicle j might consider vehicle i to be out of its range if no BSM was received for a maximum time period denoted by T_{max}. In this case, vehicle j deletes the i^{th} record from its *VNT*. Different research addressed detecting out of range vehicles, such as [42], which is based on distance prediction between vehicles. However, in this research, we simply used predefine value for T_{max} to indicate the out of range vehicles.

If a BSM is received from a vehicle that is not in the *VNT*, then a new record is created. A missed BSM from a vehicle does not necessarily imply the presence of a jammer, but would most likely be due to environmental conditions or collisions. We therefore declare a threshold α to account for such benign message losses. If the number of missed BSMs surpasses α ongoing jamming is assumed.

```
START: Receiving Thread
if (BSM not received) then
    if (time since last BSM reception > T_max) then
        Delete vehicle record from VNT;
    else
        Increment VNT[i].MissingBSM;
    end
else
    if (this is the first BSM) then
        Add new vehicle record to VNT;
    else
        Diff_1 ← |Dsecond_new - Dsecond_saved| ;
        Diff_2 ← |Dsecond_HV - Dsecond_new| ;
        if (Diff_1 «100ms) then
            Call misbehaving detection technique ;
        else
            if (MissingBSM ≤ α) then
                VNT[i].BSM ← NewBSM;
                VNT[i].MissingBSM ← 0;
            else
                VNT[i].BSM ← NewBSM;
                Decrement VNT[i].MissingBSM;
                if (Diff_2/100ms ≈ VNT[i].MissingBSM) then
                    Jamming detected. Mark vehicle as victim of jamming;
                end
            end
        end
    end
end
Goto: START;
```

Algorithm 1: Hybrid jamming detection algorithm

Two values $Diff_1$ and $Diff_2$ are used to identify if misbehavior is occurring or if a node is falsely accused of such behavior. Specifically, $Diff_1$ is the difference in time between the creation of the last received and the currently received BSM. If this time is much less than 100ms a misbehavior detection algorithm should be executed. The difference in time between the *Dsecond* field of the received BSM and the time at the HV, denoted by $Diff_2$, is used to identify if a vehicles is innocently framed as behaving selfishly. The value in $Diff_2/100ms$ should be the number of missing BSMs

if a burst occurred. This is used to determine that the vehicle is not misbehaving, but sending a burst queued due to jamming.

3.4.2 *Detection Algorithm Implementation and Testing*

The hybrid jamming detection algorithm was implemented inside the receiving thread of an Arada locomate Classic OBU using the C programming language. Sample functions of the detection algorithm code can be found in Appendix B. Several lab experiments were conducted to test the detection rates of the algorithm, using three Arada locomate Classic OBUs. The first one was configured to be a transmitter that transmited a BSM every 100ms, the second OBU was configured to be the receiver that ran the hybrid jamming detection code in the receiving thread, and the third OBU was programmed to be the hybrid jammer. Both the transmitter and the receiver were subjected to the hybrid jamming attack for different jamming periods and the results were logged inside the receiver. Table 3.3 shows the test parameters for the hybrid jamming detection setup.

TABLE 3.3: Hybrid jammer detection algorithm test parameters

OBU Model	Arada Systems LocoMate Classic
Number of OBUs	3 (2 OBUs for two transmitter and receiver and 1 for the stationary jammer)
Test environment	Lab test
BSM generation	10 packets/s
Channel	Safety Channel 172
Transmitter power	21 dBm
Data rate	6Mbps
Jammer power and data rate	18 dBm, 6Mbps
Jamming durations	from 200 ms to 40000 ms

A GUI was implemented using the Dot Net Framework in order to have visual detection alerts about the hybrid jamming attack while conducting the experiments. The GUI enabled us to observe the transmitter behavior at the receiver side in the following periods:

- *Before the jamming attack*: where the receiver gets a BSM every 100 ms without any interference.

- *During the attack*: where the receiver does not receive any BSM from the transmitter as the latter senses the medium to be busy due to jamming. In this duration, the hybrid jamming detection algorithm counts the missing BSMs, i.e., those that were not received from that particular transmitter.

- *Recovery period*: which occurs directly after the jamming attack ends. During this period of time, the receiver receives a burst of BSMs from the transmitter subjected to such attack. The received bursts sizes were found to be equal to the missing BSMs from the transmitter.

All three period can be seen in Figure 3.5, which is a snapshot of the GUI program.

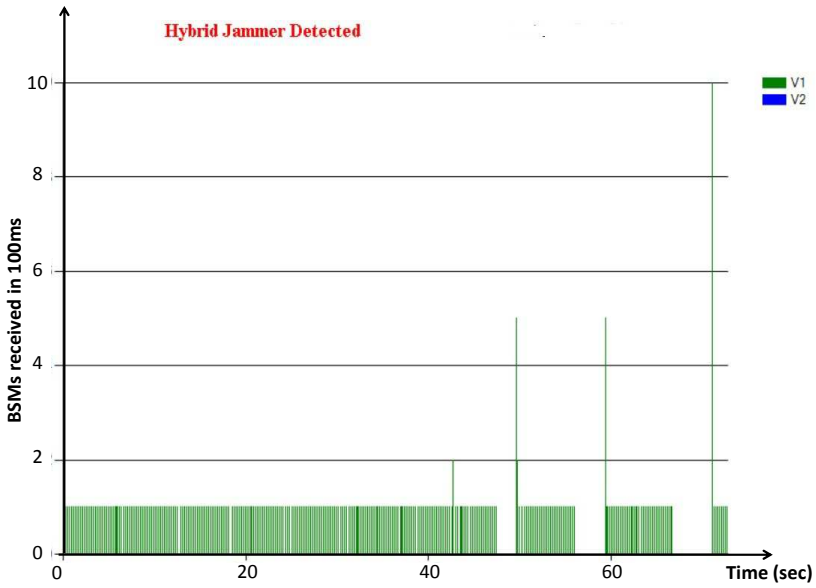

FIGURE 3.5: Hybrid jamming detection GUI

Repeated experiments confirmed that the detection approach detects jamming attacks successfully. Specifically it can detect jamming periods that are greater than $\alpha * 100ms$, and less than or equal to $Q_{size} * 100ms$. Recall that α refers to the expected number of omissions or collisions due to environmental conditions, and Q_{size} is the OBU's queue size.

As the detection algorithm is based mainly on the timing information of the received BSMs, it was noticed that the detection algorithm fails when the receiver and the transmitter were not time-synchronized. This was observed due to frequent GPS signal outages while conducting the lab experiments. Such outages caused the OBUs to have different clock values, and thus the receiver was not able to detect the missing BSMs.

3.5 CONCLUSIONS

This chapter presented a new hybrid jammer capable of failing DSRC safety applications. The jammer exposed queuing behavior that was exploited for an effective attack strategy. Scenarios for stationary and mobile jammers were presented together with their impact on the EEBL safety application. However, we expect that the jammer will have similar implications for other DSRC safety applications. Experiments were conducted using commercial DSRC equipment validated the expected impact of the jammer. An algorithm was presented that allows detection of hybrid jamming. This algorithm can also distinguish between misbehaving nodes and nodes that are impacted by the jammer in such a way that makes them appear to be selfishly misbehaving by current misbehavior detection strategies. The detection algorithm was implemented and tested using the commercial DSRC devices. It was found that the detection algorithm detects the hybrid jamming attacks for different jamming durations, however, it requires that the DSRC devices be time-synchronized, as we had observed that frequent GPS signal outages could lead to detection failure. Thus, the clock synchronization problem and its impact on DSRC safety applications will be discussed in the next chapters.

CHAPTER 4

A Clock Synchronization Algorithm for VANET

Clock synchronization is critical in the operation of wireless communication that assure data consistency and reliability. It can be achieved using either centralized or decentralized approaches [48]. The current configuration of VANET uses GPS as a central controller for clock synchronization. A GPS receiver connected to the OBUs is the main source of OBU's clock data. However, GPS might suffer from frequent GPS signal outages in urban cities or it can be easily maliciously spoofed by inducing a forged GPS signal. Moreover, a conventional GPS receiver cannot detect a spoofed signal [49].

Recall that we had observed GPS signal outages while conducting lab experiments to test the hybrid jammer detection approach described in the previous chapter. In one instance we observed a difference of 2 seconds between the clocks of two different OBUs due to the GPS signal outage. Such outage would cause the hybrid jammer detection approach to fail because it is based mainly on time stamps inside the BSMs.

Given the criticality of timeliness of BSM messages [38], failure of the GPS based clock synchronization has also the potential to cause safety applications to fail. To mitigate the centralized GPS clock synchronization as a single point of failure, one approach is to require all vehicles to participate in time synchronization. This requires vehicles to exchange and cooperatively agree on their respective local clock values.

Agreement can be generally categorized into exact and inexact or approximate agreement. In exact agreement, all non-faulty communicating vehicles are required to agree on the same exact decision. Byzantine Consensus [19] and the Interactive Consistency Problem [22] are the best-known forms of exact agreement, where all non-faulty nodes must agree on a single value. Leader election for example is considered to be one of the exact agreement applications. On the other hand, in inexact agreement, also called Approximate Agreement (AA), vehicles are not

required to reach agreement (to vote) on the same exact value. Rather, they must converge on final values that are within a predefined tolerance.

AA must satisfy Agreement and Validity conditions [50]. The agreement condition requires that all non-faulty clocks halt with voted values that are within a predefined tolerance of each other. The validity condition ensures that the final voted values stay within the range of the initially correct clock values. Most of the AA algorithms published employ rounds of data exchange and require the use of an approximation function F to update the clock values, which are then used in the next round of data-exchange. The objective is to gradually shrink the *diameter* (difference) between local clock values, by providing a sufficient number of rounds of data-exchanges to reach the predefined tolerance value.

In this chapter, a decentralized clock synchronization protocol for VANET is presented. The proposed protocol is based on approximate agreement and does not require extra hardware nor modifications of any standards. The benefits of the proposed clock synchronization algorithm are higher resilience of safety applications to GPS spoofing attacks and when GPS signals may not be available, such as in urban cities.

4.1 RELATED WORK

Clock synchronization protocols are classified into centralized and decentralized approaches [48]. The Global Navigation Satellite System (GNSS) [51] is a centralized clock synchronization protocol that is often used by ad-hoc networks. It has the advantage of simplicity, but requires the use of GPS data in the clock synchronization process. The decentralized approaches to clock synchronization require that all nodes (vehicles) participate in the synchronization process. Much research has been conducted on clock synchronization in ad-hoc networks, with less focus on VANET, as described below.

4.1.1 *Clock Synchronization in Ad-hoc Networks*

A Reference Broadcast Synchronization (RBS) protocol was proposed in [52] that exploits the broadcast property of the wireless communication medium. In this protocol, a transmitter sends a reference packet to all its neighbors. Each receiver records the receiving time of the reference packet according to its local clock and then exchanges the receiving time with other receivers. Based on receivers' observations, the clock offset between the receivers can be easily computed.

The Average Time Synchronization (ATS) protocol was proposed in [53], which uses cluster heads as the key nodes in the network synchronization process. Each cluster head broadcasts a synchronization time packet to all nodes in the cluster. Each node in the cluster replies with the receiving time of the synchronization packet. The cluster head then averages all receiving times to get the current global time. Finally, the cluster head broadcasts a time message that contains the computed global time to every node. Both, the ATS and RBS protocols require a large number of message exchanges, and the synchronization process needs to be restarted whenever a new node with a different time joins the group. Network-wide synchronization is achieved by a similar approach among the cluster heads.

A dynamic wireless mobile network clock synchronization protocol was proposed in [54]. The protocol is based on linear approximate consensus in the presence of Byzantine faults. Periodically, each non-faulty node collects the timestamps sent by its neighbors and updates its clock value with the average of these timestamps. The authors assume the facilitating nodes are not capable of using fake identifiers. As will be demonstrated later, the average convergence function may not be the most efficient function in all conditions. Due to the high dynamic nature and fast network topology changes of VANETs, the aforementioned protocols are not suitable for vehicular ad hoc networks and can take a long time to synchronize.

4.1.2 *Clock Synchronization in VANET*

A clock synchronization protocol for VANET called Converging Time Synchronization (CTS) was proposed in [48]. This protocol is based on a sponsor election

mechanism. One of the network nodes, called *initiator*, asks its neighbors to send their number of synchronized group members, their vehicles IDs (VIDs) and time differences. Based on each neighbor's reply, the node that is synchronized to the largest number of neighbors is elected to be the new sponsor. The new sponsor then broadcasts a clock adjusting message to all nodes to adjust their own time.

In [55], a Hybrid Clock Synchronization (HCS) was proposed. It includes wireless sensors at fixed distances in the road to improve VANET synchronization. The main disadvantage in HCS is that it has large hardware overhead. Both HCS and CTS derive an equation to calculate the elapsed clock synchronization time. They appear to be immune to network topology but they are affected dramatically when traffic density increases. Furthermore, they introduce a large number of messages that are not part of the standards associated with BSMs.

A Time Table Diffusion (TTD) synchronization protocol was proposed in [56] that is immune to network topology and does not require the use of GPS. The protocol exploits the idea of the Time Table Transfer (TTT) protocol [57], where each node collects time information from its neighbors and constructs a time table. This time table is then broadcast to the neighbors to build their own time table and adjust their clock. TTD also has the same disadvantages of HCS and CTS in that it requires a large number of extra messages outside the standards.

4.2 SYSTEM AND FAULT MODELS

Due to the properties of VANET, such as the fast changing, unknown topology, and the number of participating nodes, the problem of distributed clock synchronization is very complex. We therefore take the step of introducing a simple model, representing the key properties of pathological, worst case, scenarios.

4.2.1 *System Model and Notation*

This subsection introduces a simple VANET model to demonstrate the impact of connectivity on clock synchronization. For the sake of clarity, the focus is only on investigating the clock synchronization problem. Therefore, no message losses

or collisions are assumed to occur, e.g., due to the hidden terminal problem or natural phenomena like shadowing. The terms “vehicle” and "node” will be used interchangeably while describing the network model.

Assume N vehicles equipped with non-faulty OBUs travel on a road. Vehicles are moving in the same heading (direction) with fixed speed. The communication capabilities between vehicles are described by a connectivity graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vehicles $\mathcal{V} = \{v_1, v_2, ..., v_N\}$, and $\mathcal{E}$ is the set of undirected edges $e_{i,j}$ for each communicating pair v_i, v_j. Let R be a given transmission range. Then, for each $e_{i,j} \in \mathcal{E}$, the distance between v_i and v_j, denoted by $d_{i,j}$, is less than or equal to R. Therefore, $\mathcal{E} = \{e_{i,j} | i \neq j, d_{i,j} \leq R, \forall i, j = 1, 2, ..., N\}$. Note that the graph will change over time as vehicles move. However, the graph is only used for the problem description and analysis. The graph does not have to be maintained as part of the algorithms below.

Assume that vehicles are divided into two fully connected clusters $\mathcal{V}_{C1}$ and $\mathcal{V}_{C2}$, where $|\mathcal{V}_{C1}| = |\mathcal{V}_{C2}| = N/2$. For simplicity, the distance between any two successive vehicles in the same cluster is fixed, e.g., to represent the road safety distance. Figure 4.1-S0 shows the order of moving vehicles in each cluster.

The distance between clusters, $d_{CL(1,2)}$, is defined as the distance between the leading vehicle v_1 in the left cluster (black) $\mathcal{V}_{C1}$ and the last vehicle v_5 in the right cluster (red) $\mathcal{V}_{C2}$ in the figure. As the cluster distance $d_{CL(1,2)}$ decreases, communication links between the isolated clusters in Figure 4.1-S1 appear. For example, the connectivity graph shown in Figure 4.1-S2 depicts the scenario in which vehicle v_1 of the cluster $\mathcal{V}_{C1}$ and vehicle v_5 of the cluster $\mathcal{V}_{C2}$ receive each other’s messages. In the graph, vertices are labeled by a pair x, y, where x indicates the position of vehicle in its cluster and y is the size of the neighborhood of v_x, i.e., the number of neighbors including itself. For instance, label $1, 6$ represents v_1 as the first vehicle in the cluster with a neighborhood of size 6, composed of five neighbors and itself. As the clusters get closer, the neighborhoods of the closing vehicles increase, as can be seen in Figures 4.1-S3 and S4. It will be shown later that the size of the neighborhoods has implications on the clock synchronization speed.

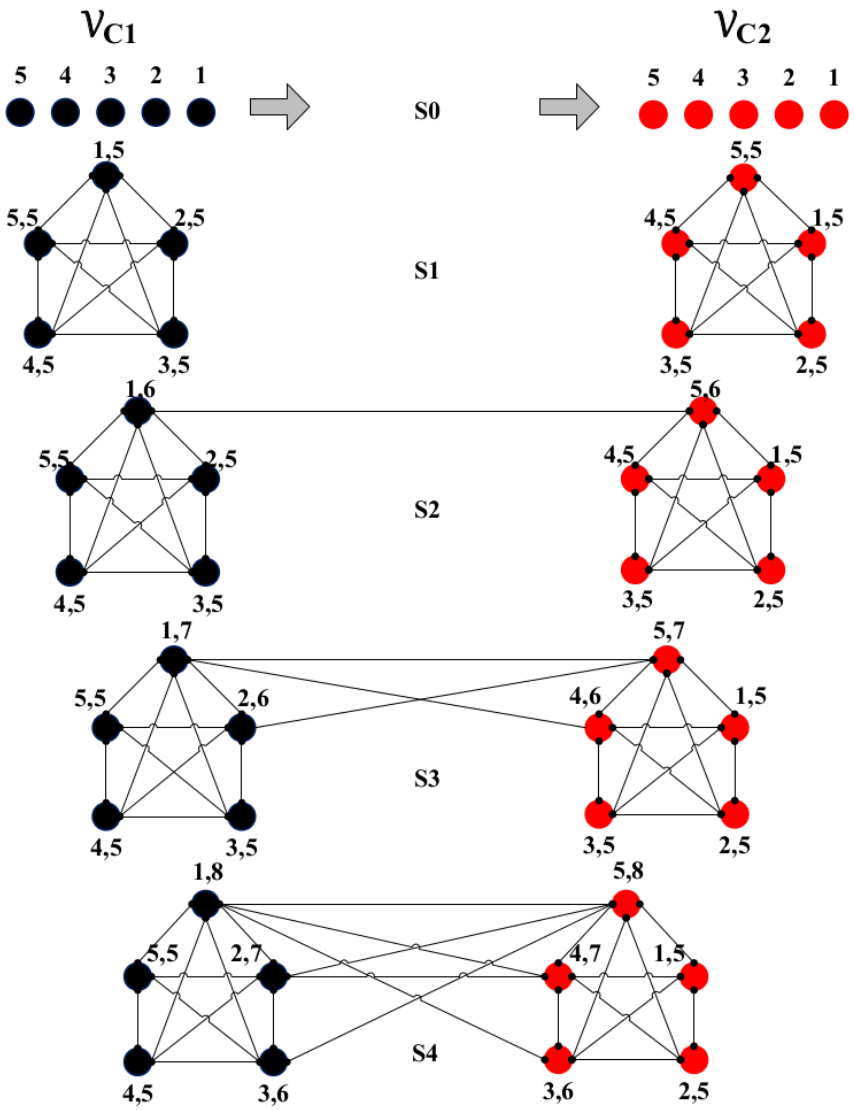

FIGURE 4.1: Two cluster connectivity graphs

Each vehicle v_i broadcasts a BSM every 100ms with a fixed transmission power P. No vehicle is assumed to be misbehaving, i.e., all vehicles follow the 10 BSMs/sec transmission rate, and no vehicle tampers with BSMs. Each vehicle maintains a sorted multiset in which it collects all timestamps, which will be used during a round-based voting process. Specifically, each vehicle v_i collects the *timestamp*, $t_{RV(j)}$, extracted from every BSM received from v_j, together with the corresponding local *reception time*, $t_{rec(j)}$, and saves them as pairs $(t_{RV(j)}, t_{rec(j)})$ in a table. Just before voting at time t_{vote}, these timestamps are adjusted, similar to [54], and included in a voting multiset $\mathbf{V}_i$. Each vehicle v_i has its own local voting multiset $\mathbf{V}_i =$

$\{t_{k(1)}, t_{k(2)}, ..., t_{k(n_i)}\}$, where n_i is the neighborhood size of vehicle v_i and $k(j), j \leq n_i$ is the vehicle from which the time is logged. The time estimation value $t_{k(j)}$ is the clock value of v_j corrected to the time at v_i. Thus, $t_{k(j)} = t_{RV(j)} + (t_{vote} - t_{rec(j)})$.

Let $\mathbf{U}_{all}$ be the multiset of all clock values held by the vehicles in the network. Define $\delta(\mathbf{U}_{all})$ as the global diameter between clock values in the entire network. The global diameter is the difference between the maximum and minimum clock values in the network, i.e., $\delta(\mathbf{U}_{all}) = \max(\mathbf{U}_{all}) - \min(\mathbf{U}_{all})$. On the other hand, the local diameter is the maximum difference between the clock values in multiset $\mathbf{V}_i$ of vehicle v_i, i.e., $\delta(\mathbf{V}_i) = \max(\mathbf{V}_i) - \min(\mathbf{V}_i)$. As was indicated before, the network topology in VANET is likely to change due to the mobility of vehicles. Therefore, the entities defined above, such as edge set, $\mathbf{U}_{all}$, and $\mathbf{V}_i$ may change as well.

Whereas the definitions above assume that the network graph is known, in the application domain of connected vehicles it may be difficult or impossible to precisely specify the graph. This is mainly due to the potential inability to define the exact graph boundaries and the fast changing topologies. For example, in high traffic density the graph may span vehicles over a large geographic area. Within such large area constant topology changes will take place. The definition of $\mathbf{U}_{all}$ in such case is impractical. Therefore, for practical reasons we consider more simplified scenarios. Specifically, we study the graph from the viewpoint of the HV, as it spans over a small geographic area, i.e., its relevant neighborhood. We concentrated also on the impact of the worst cases of clusters merging on convergence speed.

4.2.2 *Fault Model*

Recall that we assume no vehicle is behaving maliciously, e.g., no OBU is manipulating data of BSMs that are broadcast. The source of the problem is assumed to be the absence or manipulation of GPS signals, that is, GPS timing data is omitted or faulty data is injected externally. However, the OBUs themselves behave correctly, as specified. We will study the impact of two types of GPS related faults on clock synchronization in VANET.

The first fault considered is due to GPS signal outage caused by physical obstructions, such as tall buildings, mountains, or tunnels. In the absence of GPS signals,

such outages can cause receivers to have different clock values. This will lead to inconsistencies of BSM time signals sent in BSMs between vehicles. We observed such faults while conducting lab experiments using Arada OBUs during GPS signal outages. As stated before, in one instance, we observed a difference of 2 seconds between the clock values of two OBUs. A probable cause of OBUs to have different time values is the power-up initialization in the absence of GPS signals.

The second fault can be induced by GPS time spoofing attacks. The attacker is assumed to be able to spoof GPS signals in a limited geographical area as described in [49]. Attacks can be conducted near areas that suffer from GPS signal outages, like at the exit of parking garages, or tunnel entrances, as shown in Figure 4.2. Manipulating the time component in the GPS signal allows an attacker to change a GPS receiver's local time.

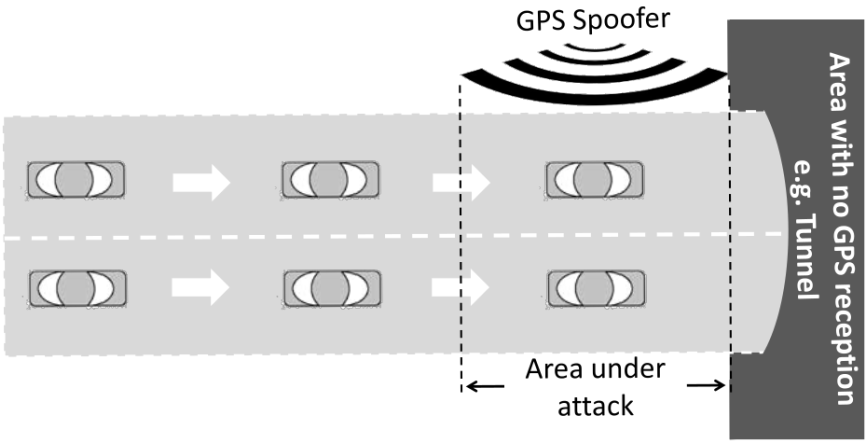

FIGURE 4.2: GPS spoofing attack scenario

The impact of GPS time manipulation will be described based on the scenario shown in Figure 2.8. Due to the hazard, the braking RV emits BSMs containing a hard-braking event during time period T_{brake}. Both faults described above could cause clock differences of more than 500ms between the HV and RV. Recall that the BSM time-to-live should be no more than 500 ms, as stated in [38]. Such time difference would cause the HV to discard important BSMs received from the RV, as it would consider these BSMs to be outdated. This could seriously affect the EEBL, as important BSM events may be discarded. The EEBL executing in the HV will fail

only if it discards all BSMs containing the event from the RV up to time t_{react}, which is the latest time for the HV's driver to react [43].

4.3 PROPOSED CLOCK SYNCHRONIZATION PROTOCOL

The proposed clock synchronization protocol aims to synchronize the clocks of vehicles without having any information about the VANET communication topology. It does not require any extra hardware, nor changes to any standards. Figure 4.3 outlines the protocol, which is executed on each vehicle's OBU. It will be described below from the viewpoint of the HV. The protocol is divided into two main stages, a *Normal Operation Stage* and an *Agreement Stage*.

4.3.1 *Normal Operation Stage*

The Normal Operation Stage initiates the Agreement Stage just before sending a new BSM. This allows the HV to populate the BSM to be broadcast with the agreement clock value based on the freshest information available. The agreement stage is initiated Δ_o time units before the BSM transmission interval *Timer* expires, where Δ_o is the projected overhead of the Agreement Stage computations.

Upon receiving a BSM from a neighbor v_j, the receiving vehicle extracts the BSM content, which includes v_j's status information, its unique vehicle identification (VID), and a *timestamp*. This timestamp is the clock value used by v_j as the result of its agreement algorithm. Each vehicle maintains a table, called *Sync Table*, in which it keeps a record for each neighbor v_j, heading in the same direction, as indicated in the BSM heading field. The record maintains the following information from a neighbor v_j's latest BSM: 1) the VID of v_j, 2) the receiving vehicle's local clock value, $t_{rec(j)}$, when the BSM from v_j was received, 3) the BSM *timestamp*, $t_{RV(j)}$, and 4) the heading of v_j. It should be noted that the heading is saved in case the receiving vehicle changes direction.

If the received BSM is coming from a new neighbor, a new record is created in *Sync Table*, and the neighbor counter is incremented by one. If the BSM comes from an existing neighbor, based on the sender's VID, the data in the existing record

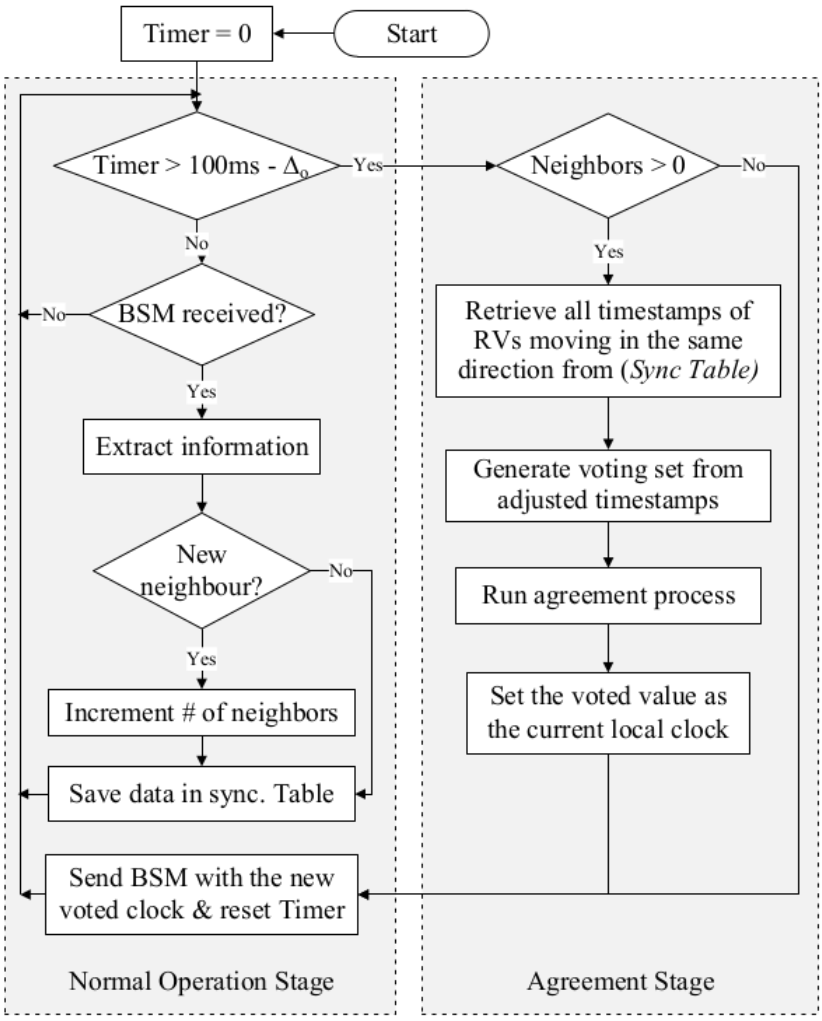

FIGURE 4.3: Proposed clock synchronization protocol

will be updated with the latest BSMs received. If no BSM has been received from a vehicle v_j for a predetermined time interval, the record of that vehicle is deleted and the neighborhood count is decremented. The reception and transmission functions code are show in Section C.3 and Section C.2 respectively.

4.3.2 *Agreement Stage*

The Agreement Stage is shown in the right part of Figure 4.3. If a vehicle has no neighbors, there is no need for agreement and the stage returns control to the Normal Operation Stage, where the BSM will be sent. Otherwise the vehicle retrieves the timestamps $t_{RV(j)}$ of those vehicles with the same heading as its own from the

Sync Table. The vehicle then generates its sorted voting multiset $\mathbf{V}_i$ based on $t_{k(j)}$ values for each neighbor v_j as described in Subsection 4.2.1 to account for the time elapsed on the vehicle running the agreement process. To obtain the new voted value, the general Mean Subsequence Reduced (MSR) convergence function shown in Equation (4.1) is used [63].

$$F_C(\mathbf{V}_i) = Mean[Sel_\sigma(Red^\tau(\mathbf{V}_i))] \tag{4.1}$$

It involves reduction, selection, and computing the mean. The reduction function Red^τ removes the τ largest and smallest elements from multiset $\mathbf{V}_i$. The reduction is not used to account for faulty nodes as in [50], but to reduce the impact of clock values of vehicles joining a synchronized cluster. Otherwise, each vehicle joining the cluster could trigger the synchronization process.

The selection function Sel_σ selects a submultiset of σ elements from the reduced multiset. Several selection functions are considered: 1) *Fault Tolerant MidPoint (FTM)*, which selects the smallest and largest elements for the mean, 2) *Fault Tolerant Average (FTA)*, which selects all elements for the mean, and 3) *Midpoint*, which selects the median element.

If the diameter $\delta(\mathbf{U}_{all})$ is reduced after each single voting round, the protocol is called *single-step convergent* [50]. This allows clock convergence to occur after a finite number of rounds. At the end of each agreement round, the vehicle sets its local clock to the new voted clock value and sends it in the new BSM. Vehicles in the transmission range receive this BSM and will use this clock value in the voting process of their next round.

The agreement process may stops when $\delta(\mathbf{V}_i)$ becomes less than the predefined tolerance Δ_{tol}. As the proposed clock synchronization is an augmentation to the GPS synchronization, the agreement stage is assumed to be always running. The agreement process code is shown in Section C.4.

4.4 SIMULATIONS AND ANALYSIS

The following presentation of results demonstrates the impact of diverse parameters on clock convergence using the simplified pathological VANET scenario described above.

4.4.1 *Assumptions and Parameters*

The simulations, using the Network Simulator Version 3 (NS-3), considered six scenarios, in which N vehicles are moving on a two lane road, such as shown in Figure 4.2. Vehicles are divided into two fully connected clusters moving in the same direction and with identical constant speed. Each cluster consists of $N/2$ vehicles, where the distance between any two successive vehicles is the same. In the following discussion, we will refer to the graphs of a simple network shown in Figure 4.1. However, the simulations conducted use a larger number of nodes.

In the simulation, Scenario S1 represents a graph similar to Figure 4.1-S1, where the two clusters are disjoint, as the distance $d_{CL(1,2)}$ between the clusters is larger than the transmission range R of 300m. Scenario S2 is similar to Figure 4.1-S2, where the first vehicle, v_1, of $\mathcal{V}_{C1}$ and the last vehicle, v_5, of $\mathcal{V}_{C2}$ are able to exchange their status information.

The clock values of vehicles of each cluster are initialized with values consistent with the spoofing attack shown in Figure 4.2, where each vehicle in the area under attack is initialized with the same value. The time difference between the two clusters is initially 30 seconds. This represents half of the maximum value of the timestamp field in a BSM, and is thus the worst-case offset possible. Within each cluster, initial values are random and uniformly distributed from a time interval of 10 seconds. The predefined tolerance Δ_{tol} was chosen to be less than 500ms, which is the *time-to-live* limit in [38]. Whereas a Δ_{tol} of 500ms may seem large for clock synchronization, recall that our goal is not tight synchronization, but fast reaction to events. Thus the EEBL can react to BSMs under such tolerance.

In order to study the influence of traffic density on the clock convergence rate, each of the six scenarios described above are repeated for three different traffic

densities. *Low density* assumed $N = 20$ vehicles with a speed of 70 mph and a distance $d_{i,j}$ between successive vehicles v_i and v_j of 60m. Similarly, *medium density* assumed $N = 40$ with a speed of 35 mph and $d_{i,j} = 30$m, and *high density* assumed $N = 80$ with a speed of 20 mph and $d_{i,j} = 15$m. It should be noted that the parameters in the scenarios are carefully chosen to result in the same overall simulation distance. The above parameters represent a traffic flow of 3600 vehicles per hour in each scenario.

To study the impact of reduction and selection functions on the convergence shown in Equation 4.1, different values for τ and σ are used. Specifically, $Red^{\tau}(\mathbf{V}_i)$, which reduces the τ smallest and largest values of $\mathbf{V}_i$, is simulated for different τ representing reductions of 0, 10%, 20%, and 30%. For each, the four different selection functions Sel_{σ} described in Section 4.3 are used. The simulation of each scenario investigated 1) how the global diameter $\delta(\mathbf{U}_{all})$ changed after each round, and 2) the number of rounds taken by each individual vehicle until its local diameter $\delta(\mathbf{V}_i)$ converged to the predefined tolerance Δ_{tol}. Each simulation ended when the local diameter $\delta(\mathbf{V}_i)$ of each vehicle became less than Δ_{tol} or when the number of rounds exceeded 300 i.e. 30 seconds. The latter was only the case for the non-convergent Midpoint selection functions, as described later.

Each simulation was repeated 50 times and results shown are based on the worst results observed. It is emphasized that the focus of the simulations was on the pathological (worst case) scenarios, not on the best or average behavior. Appendix C, shows the main functions and the global variables of the NS3 code used for the clock synchronization approach.

4.4.2 *Analysis of Simulations*

During analysis, we observed that the number of rounds to reach agreement used by different scenarios was very similar for reductions of 0, 10%, 20% and 30%. However, the behavior shifted, as will be described. Convergence for reductions of 10%, 20% and 30% was single step, whereas $\tau = 0$ showed oscillations during convergence. Due to space restrictions, we mainly focus on the graphs for 30% in Figure 4.4, which shows the number of rounds for different scenarios and selection functions.

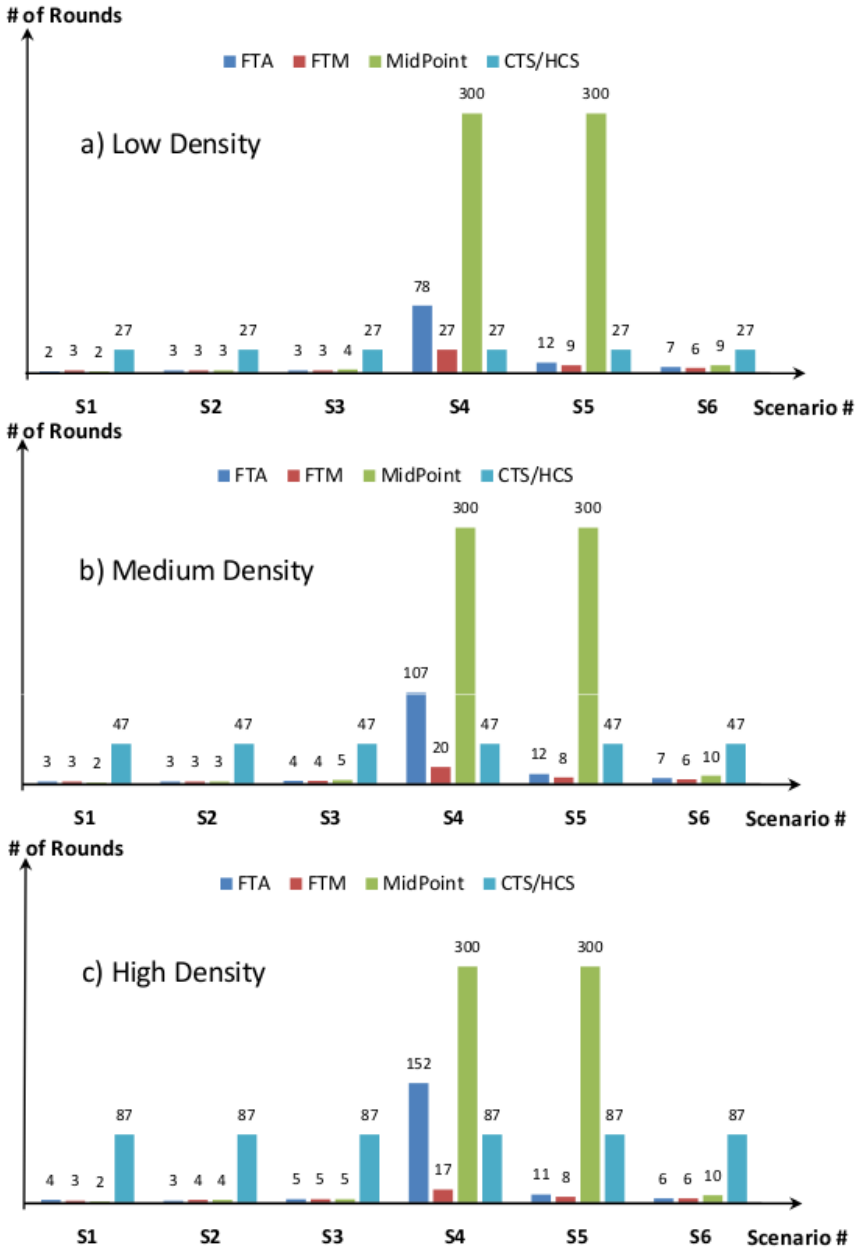

FIGURE 4.4: Impact of selection functions for 30% reduction

Recall that higher numbered scenarios imply higher degree of cluster overlap, as described in Subsection 4.4.1. Furthermore, note that resynchronization would be avoided within a cluster if the values of joining vehicles are eliminated as the result of reduction, in which case only the joining nodes would be synchronized.

The worst case scenario is when only one new value from a different cluster remains in the reduced multiset, because having only one value from the other

cluster prolongs synchronization. Figure 4.4 shows this worst case in Scenario 4 for the three different traffic densities. An example is the case in Figure 4.1-S2, with no reduction, or Figure 4.1-S3, with one reduced value. The same worst case behavior was observed for lower reduction percentages. For example, this shifting behavior can be seen if one compares reduction of 30% in Figure 4.4c with that of 20% in Figure 4.5. In the prior the worst was in Scenario 4, whereas in the latter it was in Scenario 3.

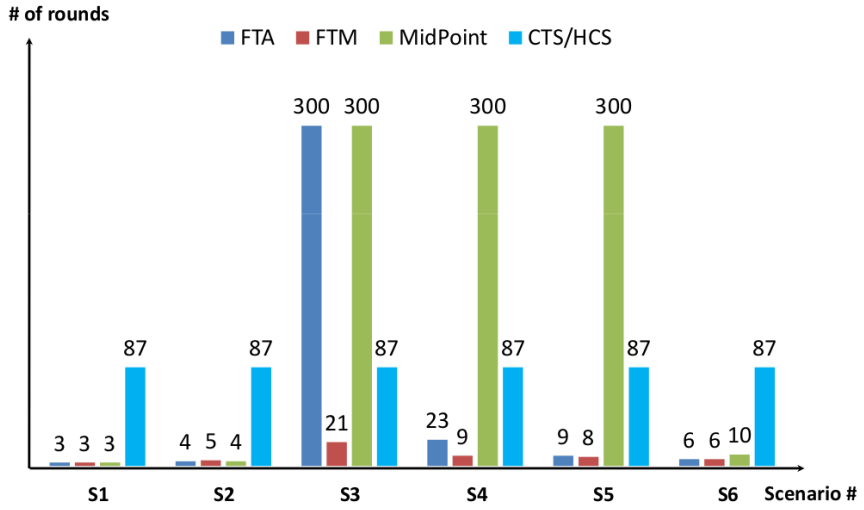

FIGURE 4.5: Impact of selection functions for 20% reduction

With respect to the selection functions in Figure 4.4, it is immediately evident that MidPoint shows the worst convergence in Scenarios 4 and 5. In fact it does not converge at all in these scenarios, where the synchronization process was terminated after 300 rounds, i.e., 30 seconds. In the first three scenarios, the FTA, FTM and MidPoint selections performed nearly identical and outperformed the CTS and HCS. However, in the worst cases Scenarios 4 and Scenario 5, FTM has best performance. This is due to the fact that FTM puts higher weight on the values of the vehicles in the neighbor cluster, thereby speeding up the clock convergence. The scenarios using low, medium and high densities only differ in the number of rounds and FTM performs better as densities increase. Whereas the performance of MidPoint is rather independent of the densities, the FTA used more rounds in denser vehicle distributions. The same was observed for CTS and HCS in all scenarios. The simulations of Figure 4.5 also showed that MidPoint did not converge. It should

be noted that FTA did converge after 517 rounds when the limit on the reduction function was increased.

Based on the results shown, we conclude that the FTM selection function is by far best suited for DSRC safety applications subjected to spoofing attacks in such scenarios.

4.5 CONCLUSIONS

This chapter investigated the impact of synchronization of vehicles, or the lack thereof, on the EEBL safety application. A decentralized clock synchronization protocol capable of mitigating against GPS spoofing attacks was presented. The new protocol is based on approximate agreement and it does not require any extra hardware or message overhead. It was compared against the existing protocols CTS and HCS. Within the proposed protocol, the impact of several selection functions on the convergence rate was also investigated. Simulations using Network Simulator Version 3 (NS-3) showed that the new protocol performs best when using the Fault Tolerant Midpoint selection function, especially in worst case scenarios.

In general, this research introduced a theoretical network model and its benefits to DSRC safety applications. It used simple pathological traffic scenarios that exposed the key issues in GPS time spoofing.

CHAPTER 5

Enhanced Clock Synchronization in the Presence of Singular and Hybrid Fault Modes

In this chapter, we propose an Enhanced Clock Synchronization Algorithm (ECSA) aiming to improve the reliability of the EEBL Safety Application. It is based on approximate agreement, however, it enhances the previous solutions presented in Chapter4 by considering 1) omission faults, 2) malicious faults, and 3) a hybrid fault model consisting of both omission and malicious faults.

Several new agreement algorithms are introduced and their relative worst case performance is analyzed. Reliability improvements will be related to clock convergence speeds, the impact of fault types, and the algorithm parameters. Results are supported by simulations using NS3 and the real-time traffic scenario mobility model SUMO. The simulations show that the proposed algorithm can effectively function under the assumed fault model. The algorithm is used in conjunction with GPS-based methods, with no extra hardware nor modifications of standards.

5.1 RELATED WORK

A summary of existing clock synchronization approaches for wireless ad-hoc network and VANET was presented in Section 4.1. However, as the proposed clock synchronization approach is based on inexact agreement, traditional approximate algorithms will be briefly discussed.

An AA algorithm typically uses a sequence of rounds, in which the difference between the values held by non-faulty nodes is gradually reduced. This difference is often referred to as *diameter*. AA algorithms are based on two conditions: Agreement and Validity. The Agreement condition requires the diameter to reach a predefined threshold. The Validity condition ensures the final agreed values stay strictly within the range of the initial correct values. To reach agreement, all non-faulty vehicles execute the following steps in each voting round:

- Broadcast: Every vehicle transmits its local clock value to all vehicles in its transmission range.
- Collect: Each vehicle saves the received clock values from its neighbors in a voting multiset, including its own clock value.
- Sampling: The vehicles remove (filter) some values from their voting multisets. The mechanism for filtering the multisets depends on the specific agreement algorithm.
- Execution: Each vehicle applies a voting function to its sampled multiset.

Agreement algorithms are generally divided into three families based on the sampling method used.

The first family, called Anonymous or Oblivious, pays no attention to the state or source of the received values. An example is the MSR algorithm [63] described in Section 4.3.2. The MSR uses the general convergence function F_C shown in Equation 4.1, that executes reduction, selection, and computing the mean.

Another example of anonymous algorithms is the Omission Mean Subsequence Reduced (OMSR) agreement algorithm introduced in [23]. In contrast to MSR algorithms, OMSR algorithms do not replace the missing values by any default value, which acts implicitly as a filtering mechanism in the sampling step. Moreover, the reduction function of both MSR and OMSR voting algorithms depend on the maximum number of erroneous values expected by a receiving node due to *Transmissive Symmetric* (TS) and *Transmissive Asymmetric* (TA) faults introduced in the OTH-5 model [23]. A TS fault occurs when the same erroneous message is received by all receivers, whereas a TA fault delivers different erroneous values to different receivers.

In the second family, called Egocentric, nodes prefer values that are closer to their own value. An example is the Mean Subsequence EgoCentric (MSE) algorithm used in [58]. During sampling, each node replaces the missing values, or any value that differs from its own value by more than a threshold ϕ, with its own value. This

family considers the following approximation function [58]:

$$F_C(\mathbf{V}_i) = Mean[Sel_\sigma(\mathbf{V}_i)] \tag{5.1}$$

The last family is called Egophobic. Here nodes prefer values that are further away from their own value. An example is the Mean Subsequence EgoPhobic (MSEP) algorithm [59]. Thus, a node trusts other values more than its own value.

The clock synchronization approach proposed in chapter 4 was based on the MSR family of algorithms. However, this chapter uses a modified version of MSR algorithms in order to reduce the impact of omission and malicious behavior. The modified algorithm is introduced in Section 5.3.1.

5.2 SYSTEM AND ATTACK MODEL

5.2.1 *Attack Model*

Recall that GPS time is used as a centralized source for synchronization in VANET, and that manipulating the time component in the GPS signal might coerce vehicles to interpret received BSMs as outdated if the difference between vehicles exceeds the BSM time-to-live. Such an attack, i.e., causing the safety application to fail by forcing the vehicles to discard their received BSMs, was addressed in the previous chapter under the assumptions that there are no BSM losses, nor are there malicious nodes that manipulate BSM data. However, BSMs might be lost, corrupted or manipulated during the synchronization process.

This chapter considers three fault models in addition to the aforementioned GPS signal faults. The fault models are based on the partitioning of faults in the Omissive/Transmissive Six-Mode fault model [20] shown in Figure 2.12. The first fault model considers the strictly omissive asymmetric (SOA) faults. An SOA fault occurs when a correct message broadcast is received by some but not all receiving vehicles. The SOA faults might occur due to benign reasons such as shadowing or fading of signals.

The second fault model considers transmissive symmetric (TS) faults. A TS fault occurs when the same erroneous value broadcast by a malicious node is received by all receivers. Nodes behaving in this manner are undetectable and aim mainly at delaying or causing the clock convergence process to fail by changing the selection criteria of the agreement process. The maximum number of TS faulty nodes is assumed to be f. We assume that the number of malicious nodes does not exceed the Byzantine threshold of one third [19], i.e. $f < \frac{1}{3}|\mathbf{V}_i|$.

The last fault model is a three-mode fault model that includes the SOA and TS explained previously. The third fault mode is the Single Error Omissive Asymmetric (SEA) that occurs when an erroneous value broadcast is received by some but not by all receivers. The difference between SEA and SOA is the value broadcast, which is erroneous in SEA but correct in SOA. SEA faults have also been addressed in [61, 62].

As the aim of this research is to study worst case scenarios, such attacks and faults are assumed to occur near areas that might be subject to GPS signal outages, like parking garages or tunnel entrances, as shown in Figure 5.1. Malicious nodes, which in this case are distributed, are marked red. Different distributions will be investigated, in order to study their impact on the proposed ECSA.

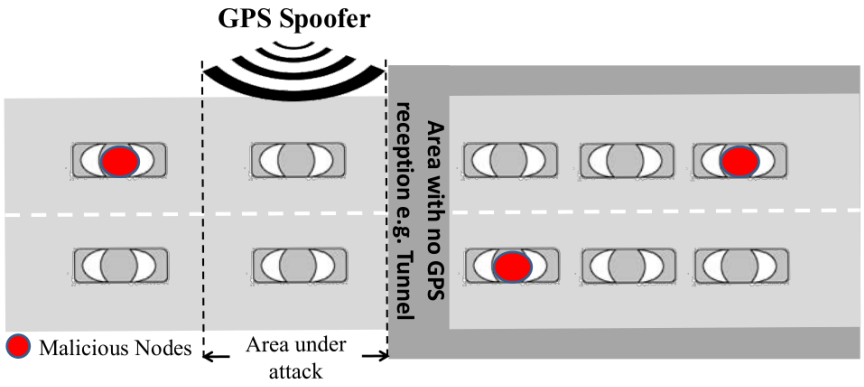

FIGURE 5.1: Attack model

5.2.2 *System Model*

The nature of the highly dynamic topology of VANETs makes the design of a clock synchronization protocol challenging. Several variables affect clock convergence

speed, such as: 1) the traffic density in terms of the number of vehicles and distances between them, 2) the sampling step of the agreement algorithms, 3) the scenarios in which vehicles are merging into a formation, 4) the BSM delivery ratio, and 5) the presence of malicious attacks. Therefore, a simple VANET model was introduced in Section 4.2.1 that investigate the impact of these parameters on clock convergence speed. This model assumes that there are no BSM losses, nor are there malicious nodes that manipulate BSM data. However, now we assume that less than $\frac{1}{3}$ of the total number of nodes N are malicious, to ensure that the Byzantine threshold for a neighborhood is not violated. Therefore, $f < \frac{1}{3}|\mathbf{V}_i|$. Moreover, BSMs might be lost due to environmental conditions such as fading or shadowing. The proposed clock synchronization algorithm synchronizes the local clocks in spite of malicious nodes, but does not attempt to identify them.

5.3 ENHANCED CLOCK SYNCHRONIZATION ALGORITHM

The Enhanced Clock Synchronization Algorithm ECSA is based on iterative voting rounds. Each round executes toward the end of each BSM transmission period. Therefore, the rounds are initiated approximately 100 ms apart from each other. The timestamp in each new BSM broadcast is the voted clock value of the previous round.

At the core of ECSA are agreement algorithms. As described in Section 5.1, different algorithms have been devised to deal with missing and malicious values. By taking advantage of the properties of these algorithms, the next section introduces new ideas to handle the missing values, and making changes to the reduction function, in order to mask the effect of malicious faults.

5.3.1 *Agreement Algorithms*

The agreement algorithms introduced in this subsection are inspired from the general MSR family of algorithms [63]. MSR algorithms follow Equation 4.1. These algorithms are different in their selection functions. One of the selection functions is the Fault-Tolerant Midpoint (FTM), which was shown in chapter 4 to be the best

selection function for merging vehicle scenarios from among a number of selection functions tested. The FTM selects the smallest and largest values from the reduced voting multiset for the mean.

One of the major differences among families of algorithms is in the handling of missing values, e.g., due to BSM omission faults. The first agreement algorithm introduced here is the Mean Subsequence with Fixed Reduction (MSFR). It was inspired by OMSR [23], in that the missing values are not replaced, causing the voting multisets to vary in size. The main difference between OMSR and MSFR is that the MSFR uses a fixed reduction function, whereas the OMSR reduction function depends on the number of faults.

The second agreement algorithm introduced here is the Mean Subsequence Reduced History (MSRH). Recall that each vehicle maintains a record for each neighbor v_j in the *Sync Table* to keep v_j's previous values. The MSRH replaces the missing clock values with the previous values saved in the *Sync Table*.

Mean Subsequence EgoCentric with Reduction (MSER) is the third agreement algorithm introduced. It uses the same preference as the MSE algorithms in [58], i.e. each node replaces the missing values with its own value. However, unlike MSE, it adds a reduction function that removes the extreme values from the voting multisets.

The last agreement algorithm introduced is the Mean Subsequence EgoPhobic with Reduction (MSEPR), which uses the same preference as the MSEP algorithms in [59], in that a node replaces the missing values with values that differ from its own clock value before applying the reduction function. In this agreement algorithm, each node selects the most extreme value from its voting multiset as a replacement for the missing ones.

5.3.2 *ECSA*

Each vehicle executes ECSA, depicted in Figure 5.2. For the sake of simplicity, the algorithm will be described from the viewpoint of the HV. ECSA consists of three major threads. In the first thread, a vehicle continuously receives BSMs broadcast from the surrounding neighbors. In the second thread, the vehicle executes the

agreement algorithm to update its clock value. The last thread broadcasts the new clock value to be used by other vehicles in their agreement process.

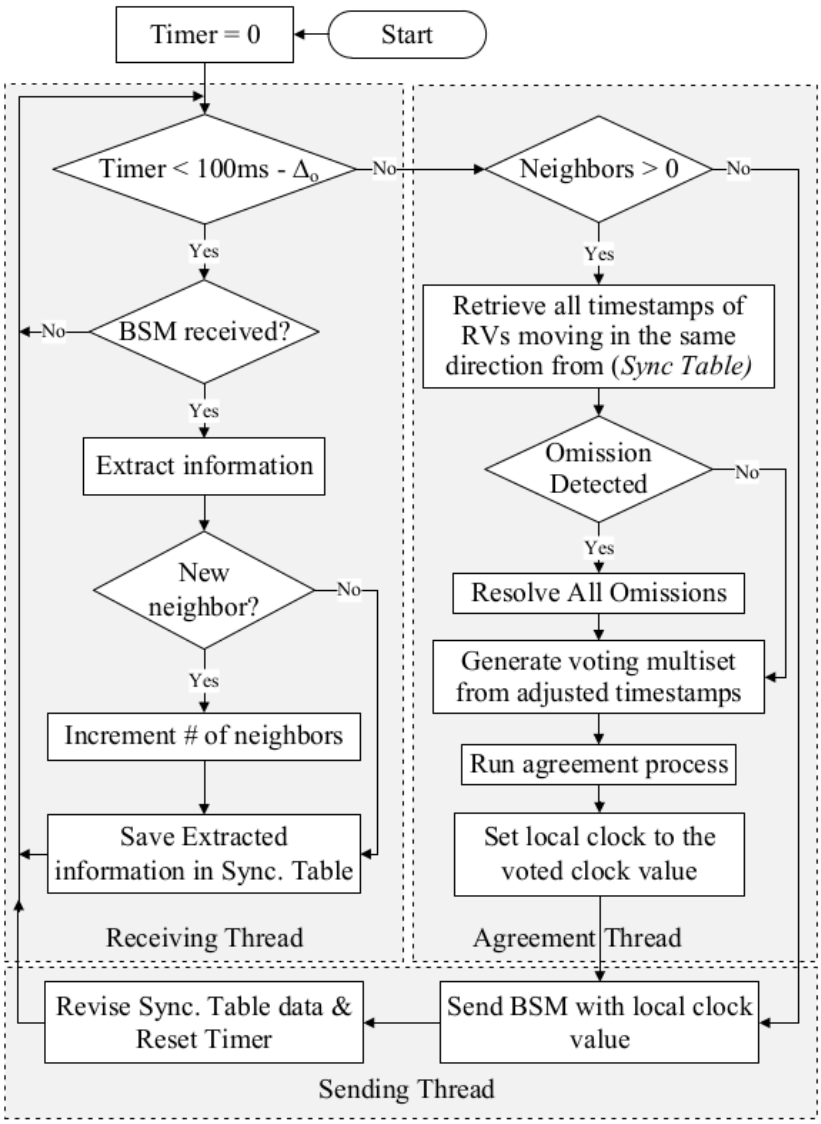

FIGURE 5.2: The Enhanced Clock Synchronization Algorithm (ECSA)

5.3.3 *Receiving Thread*

The receiving thread is depicted in the left part of Figure 5.2. It is responsible for continuously listening to the medium to receive BSMs transmitted from neighboring vehicles. The receiving thread iterations are controlled by a *Timer*, initialized to 0, which controls the time spacing of agreement rounds. The thread executes as long

as $Timer < 100ms - \Delta_o$, before transferring control to the agreement thread. Here Δ_o represents the projected overhead of the agreement thread computations. This allows the agreement thread to calculate the new voted clock value based on the freshest information available.

Upon receiving a BSM from any vehicle v_j, the receiving thread extracts v_j's identification (VID), timestamp and heading (direction). The extracted timestamp is assumed to be the latest voted clock value of vehicle v_j. Each vehicle maintains a record for each of its neighbors in a *Sync Table*. For each BSM received from v_j, this record holds the extracted information and the corresponding HV's local time.

Based on v_j's VID, the receiving thread looks for v_j's record in the *Sync Table*. If the VID is not found in the table, v_j is considered a new neighbor and a new record is created for it. Otherwise, v_j's record is updated by the latest information received. An HV detects an out of range vehicle by continuously monitoring the last BSM received from it. If no BSM has been received from a vehicle v_j for a predetermined period of time, v_j's record is deleted from its *Sync Table* and the neighbor counter is decremented.

The receiving thread initiates the agreement thread Δ_o time units before the *Timer* reaches the BSM transmission period.

5.3.4 *Agreement Thread*

The agreement thread is depicted in the right part of Figure 5.2. Agreement is executed only once in each voting round if the HV has received new BSMs. The agreement thread executes two main steps: The first step is an adjustment to the timestamps $t_{RV(j)}$ from the *Sync Table* as described in Subsection 4.2.1. These adjusted values $t_{k(j)}$ are then sorted in a voting multiset $\mathbf{V}_i$. In the second step, the agreement algorithm is executed to obtain a new clock value. The agreement thread applies one of the agreement algorithms introduced in Section 5.3.1. The behavior of each agreement algorithm will be described in Section 5.4.

At the end of each voting round, HV uses the voted clock value as its new local clock. The voting algorithm is said to be *single-step convergent* if the global diameter $\delta(\mathbf{U}_g)$ is reduced after each voting round [50]. This ensures the clocks

will be synchronized after a finite number of voting rounds. For any vehicle v_i, the agreement thread could be skipped if the local diameter $\delta(\mathbf{V}_i)$ becomes less than a predefined tolerance Δ_{tol}. However, we suggest to keep the agreement thread always running as an augmentation to GPS synchronization.

5.3.5 *Sending Thread*

The sending thread is depicted in the bottom part of Figure 5.2. It is initiated either at the beginning of the agreement thread, if the HV has no neighbors, or the end of the thread, after the local clock was updated with the voted clock value. The sending thread is responsible for creating and sending new BSMs that include the latest vehicle status information and the new voted clock value, which will be used in the voting rounds of other vehicles. Having a new local clock value might affect the *Sync Table* time-data consistency as the receiving time of the latest BSMs for each neighbor is based on the HV's old clock value. Based on HV's old clock value t_{old}, its new voted clock value t_{new}, and the receiving time $t_{HV(j)}$ saved in the *Sync Table*, an adjustment is needed to preserve *Sync Table* data consistency. Specifically, the difference between t_{old} and $t_{HV(j)(old)}$ in each record is calculated and subtracted from t_{new} as shown in Equation 5.2, where $t_{HV(j)(old)}$ refers to the receiving time based on the old clock value and $t_{HV(j)(new)}$ refers to the receiving time based on the new clock value.

$$t_{HV(j)(new)} = t_{new} - (t_{old} - t_{HV(j)(old)}) \tag{5.2}$$

This adjustment keeps the *Sync Table* data up to date and enables the ECSA to track omissions and out of range vehicles.

5.4 SIMULATIONS AND ANALYSIS

The performance of the ECSA in a VANET was evaluated using simulations divided into two parts. The first part was concerned with the mobility model of the moving vehicles, whereas the second part focused on network simulation using the network simulator NS3. Two different mobility models were considered in order to test the

effectiveness of the ECSA using diverse simulation scenarios and parameters. The first model, referred to as Model 1, was based on the *Constant Velocity Mobility Model* of NS3, which is used to simulate the system model described in Section 4.2.1. The second model, referred to as Model 2, was generated using Simulation of Urban MObility (SUMO), which imitates a real VANET environment.

5.4.1 *Simulations and Analysis of Model 1*

According to the system model presented in Section 4.2.1, the simulation model consisted of N vehicles moving on a two-lane unidirectional road with the same static velocity. Vehicles were divided into two fully connected clusters, where each cluster consisted of $N/2$ vehicles. Inter-vehicle distances $d_{i,j}$ were constant and equal to the two seconds safety distance rule, whereas the inter-cluster distance $d_{CL(1,2)}$ was varying as depicted in Figure 4.2.1. Table 5.1 shows the values for different traffic densities (Low, Medium, High) used in the simulations. However, the results shown in this section are based on high traffic density, and only a summary of the other scenarios will be presented.

TABLE 5.1: Traffic density parameters

	Low	Medium	High
N	20	40	80
$d_{i,j}$ (m)	60	30	15
Velocity (m/sec)	30	15	7.5

All vehicles in the simulations were subject to the GPS time spoofing attack described in Subsection 5.2.1. The clock values of vehicles were initialized with random clock values consistent with the GPS time spoofing attack. The time difference between the initial clock values were set to more than BSM *time-to-live*, which is 500ms [38]. Recall that the EEBL safety application should discard outdated messages, which without clock synchronization could lead to failure. Thus, each vehicle executes the ECSA in augmentation to GPS synchronization. Furthermore, recall that agreement is achieved if the local diameter $\delta(\mathbf{V}_i)$ of each vehicle becomes less than Δ_{tol}, which is assumed to be the aforementioned 500ms.

Simulations in this model were conducted 50 times and for each case the worst results for each algorithm and vehicle density were selected for the figures below. The reason for this selection is that we are concerned about worst cases and not averages.

Figure 5.3 depicts the ECSA clock convergence speed. This figure assumes the network encounters no faults, i.e., no omissions or malicious behavior. The scenarios are thus fault free and are referred to as NoFault scenarios. The x-axis of the

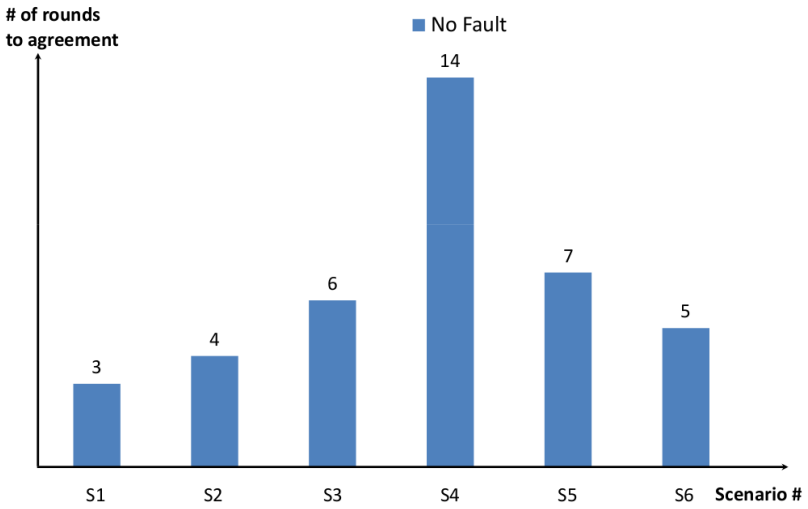

FIGURE 5.3: Convergence speed for different scenarios under GPS time spoofing attack

simulation results in this simulation model (Model 1) represents six scenarios. Each simulation scenario is related to the network model described in Subsection 4.2.1, where communication links between clusters increase for higher numbered scenarios. In contrast to Figure 4.2.1 that shows four scenarios, Figure 5.3 shows all six scenarios with scenario S6 resulting in a fully connected communication topology between the two clusters. The y-axis represents the maximum number of rounds to reach agreement, observed over all 50 simulations. It can be seen that scenario S4, shown in Figure 4.2.1-S4, is the worst case scenario. It took 14 rounds, accounting for 1.4s, to synchronize. This is due to the fact that vehicles in this scenario have a voting multiset with the least number of values from the other cluster after the reduction function was applied. In fact, this is the first scenario in which values from other clusters are considered in voting multisets. The results for lower traffic

densities were very consistent with those of the higher traffic densities in terms of ECSA convergence speed and the worst case behavior.

The simulation scenarios presented above assume that all BSMs were received successfully. Now consider the Strictly Omissive Asymmetric (SOA) faults that cause some BSMs to be omitted due to environmental reasons such as fading or shadowing. According to the IEEE standard [64] Packet Error Rate (PER) of 10% should not be exceeded. Assume that SOA faults will not exceed 10% of the transmitted BSMs during any voting round. Any BSM subjected to SOA faults will thus be received by at least 90% of the RV neighbors.

Figure 5.4 shows the clock convergence speed of the four algorithms, as well as the NoFault case described above. The NoFault case embedded in the figure is used as a reference. SOA faults may affect each vehicle differently, as some may receive a BSM and others may not. As can be seen, the convergence speed of the ECSA using MSRH and MSFR were rather unaffected by such faults. In all but scenario S1 they are identical. MSEPR converged much slower than all others in scenario S3 and MSER in S4. For MSEPR, this was due to the fact that it replaces the missing values with the extreme values, which are then eliminated using the reduction process. In the case of MSER, the convergence was slowed down because a node uses its own value for each missing value, essentially slowing down the propagation of values from the other clusters.

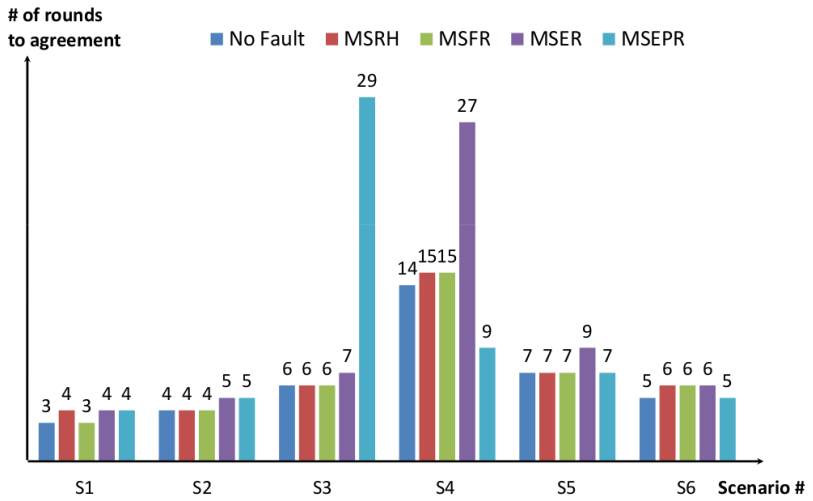

FIGURE 5.4: SOA faults impact on ECSA convergence speed in Model 1

The previous singular fault model, i.e., the SOA model, assumes no malicious behavior. However, a VANET might suffer from *Transmissive Symmetric* (TS) faults caused by malicious nodes. For the singular TS fault model, the simulations assumed that 30% of the vehicles are malicious. These faults can behave in the worst possible case by having the faulty nodes in one cluster produce the lowest extreme values, and the faulty nodes in the other cluster the highest extreme values. This would force one cluster to select and ultimately produce a low voted value, and the other cluster a high voted value. In essence, the faulty nodes can increase the gap in the voted values, which leads to a higher number of rounds for agreement.

Different distributions of faults have been tried to understand their impact on the ECSA's convergence speed. Figure 5.5 depicts different fault distributions for the low traffic density scenarios. Figure 5.5a shows the *Center* distribution, where the malicious nodes (marked with yellow) of each cluster are concentrated in the area closest to the other cluster. Figure 5.5b displays the *One-End* distribution, where the malicious nodes of each cluster are concentrated at one end of their cluster. The *Scattered* distribution is shown in Figure 5.5c, where the malicious nodes are randomly distributed over both clusters.

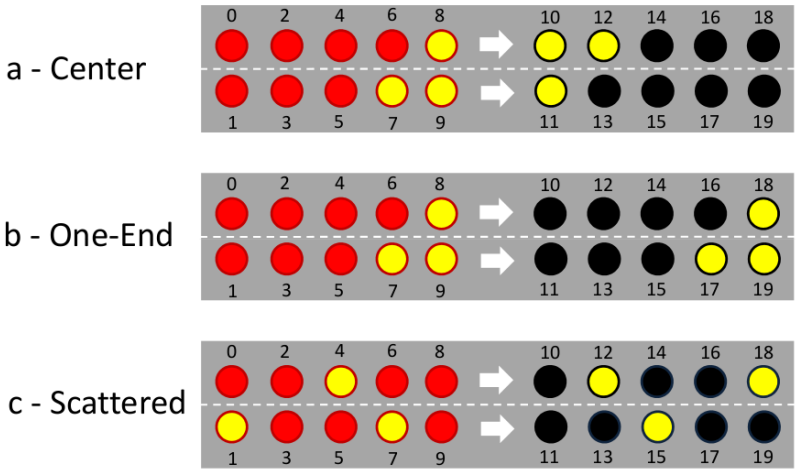

FIGURE 5.5: Different distributions of malicious faults

Figure 5.6 shows the ECSA convergences speeds for the TS fault distribution in Figure 5.5. ECSA can handle such malicious activity as long as the malicious nodes percentage does not exceed the reduction function limit applied to the voting

multiset. However, simulations showed that the malicious nodes were able to slow down clock convergence speeds in comparison to the non-malicious fault scenarios. It can be seen that the *Center* distribution has the worst impact on ECSA, whereas the *One-End* has the least impact. This is due to the fact that the nodes located in the center of the simulation area are responsible for the clock convergence between merging clusters. Since the malicious nodes in the center behave in the worst manner by broadcasting the extreme values, their values are eliminated by the reduction function, thus delaying convergence.

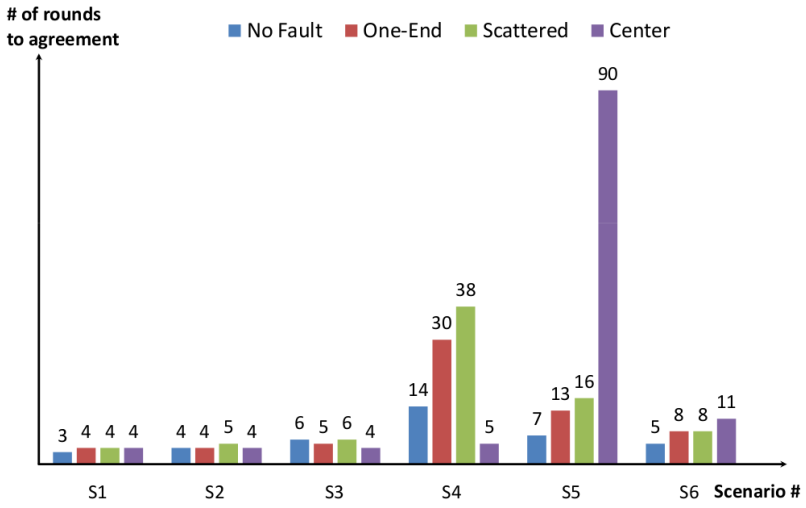

FIGURE 5.6: TS faults impact on ECSA convergence speed in Model 1

Now consider the hybrid fault model for the scattered distribution. Recall that this model consists of SEA, SOA, and TS faults. A receiving node can experience an omission due to SOA or SEA faults. In addition, a node might receive an erroneous value from an SEA faulty node or experience an omission from the same node. From the perspective of a receiving node, the total number of SEA and SOA faults, denoted by N_{SEA} and N_{SOA} respectively, account for both transmissive and missing values. If $N_{SEA(a)}$ and $N_{SEA(o)}$ denote the number of erroneous and missing erroneous values respectively, then $N_{SEA} = N_{SEA(a)} + N_{SEA(o)}$. Since we assume at most 10% omissions, due to the fact that the Packet Error Rate (PER) should not exceed 10% [64], $N_{SEA(o)} + N_{SOA} \leq 0.1\ N_{BSM}$, where N_{BSM} is the total number of BSMs generated per round. Additionally, it is assumed that $N_{SEA(a)} + N_{TS} \leq 0.3\ N_{BSM}$, where 30% is the reduction applied to a voting multiset and N_{TS} is the total number

of TS faults. This implies that $N_{SEA} + N_{TS} \leq 0.3\ N_{BSM} + N_{SEA(o)}$, which leads to

$$N_{SEA} + N_{TS} \leq$$

$$(0.3 + 0.1)\ N_{BSM} - N_{SOA} = 0.4\ N_{BSM} - N_{SOA}$$

and

$$N_{TS} \leq 0.4\ N_{BSM} - (N_{SOA} + N_{SEA})$$

Figure 5.7 shows the hybrid fault model impact on the convergence speed of the ECSA using different agreement algorithms. Both MSFR and MSEPR failed to converge before the set simulation round limit of 300, which accounts for 30s. This is because the impact of TS on the agreement process depends on the replacement preference of the omitted clock values and the total omissions, i.e., $N_{SOA} + N_{SEA(o)}$. Thus, not replacing the missing values as in MSFR, or replacing the missing values with extreme values as in MSEPR, allows the TS fault percentage to overwhelm the assumed reduction limit. The convergence time using MSRH or MSER also increased

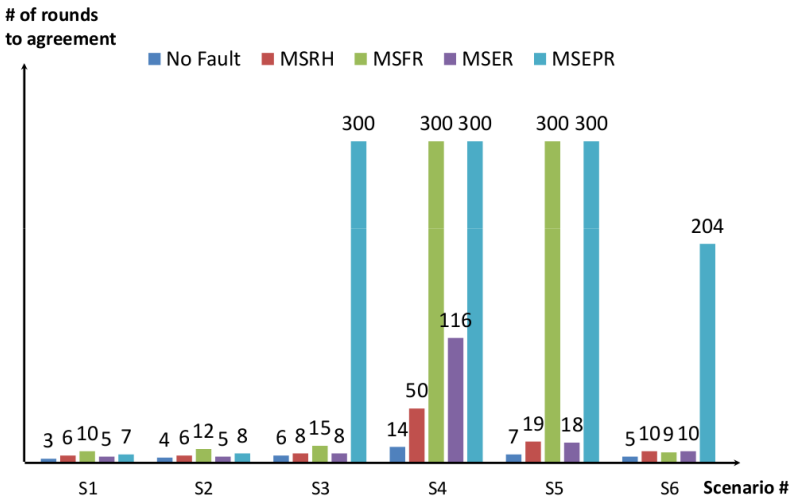

FIGURE 5.7: The ECSA convergence speed using the hybrid fault model for scattered malicious node distribution

significantly over No-Fault, most noticeable in scenario S4.

5.4.2 *Simulations and Analysis of Model 2*

In this simulation model denoted by Model 2, we coupled the traffic simulation tool SUMO with network simulation tool NS3. The reason for selecting SUMO as a road simulation tool lies in its strong capability of traffic mobility generation that imitates the real VANET environment. The chosen simulation area was the two-lane unidirectional Brooklyn-Battery Tunnel, renamed Hugh L. Carey Tunnel.

Unlike the previous model, the inter-spacings between vehicles are not static and vehicle velocities were randomly selected around the tunnel speed limit. Specifically, the vehicle velocities limit were 15 m/sec ±1.5 m/sec. The real traffic density of the tunnel considered in this model is 38 vehicles per minute, based on traffic volumes presented in [65]. However, different numbers of vehicles were simulated in this tunnel, i.e., 20, 40 and 80 vehicles, using the same traffic density of the tunnel. These parameters created different voting multiset sizes for each vehicle. Figure 5.8 shows the clock convergence speed of the ECSA subjected to GPS time spoofing attack only. No other faults such as omissions, or malicious vehicles were assumed.

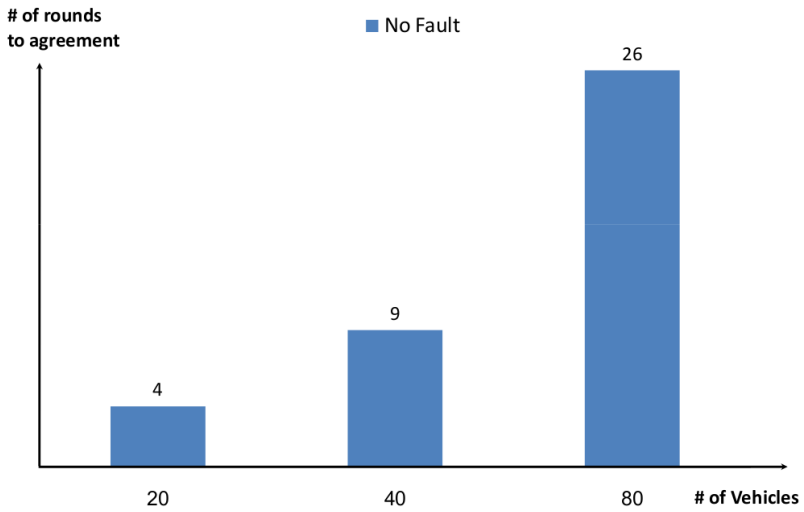

FIGURE 5.8: Convergence speed of ECSA under GPS time spoofing attack using SUMO simulation

Vehicles in this simulation model are subject to the fault model introduced in Subsection 5.2.1, which was also applied to Model 1. Figure 5.9 shows the impact of the SOA faults on the ECSA convergence speed using different agreement algo-

rithms, as well as the NoFault case shown in Figure 5.8. Recall that 80 vehicles were used in the simulation scenarios of Model 1, whereas 20, 40, and 80 vehicles were used in Model 2. Now consider the performance of different agreement algorithms in case of, 1) the worst case scenario of Model 1, shown in Figure 5.4, and 2) the simulation results using 80 vehicles in Model 2 shown in Figure 5.9. It can be seen that the simulation results are consistent in both models in such cases. The main difference is that MSEPR produces faster convergence in Model 2 compared with Model 1. This is due to the dynamic nature of simulation Model 2, where the worst case scenario is not guaranteed as in Model 1.

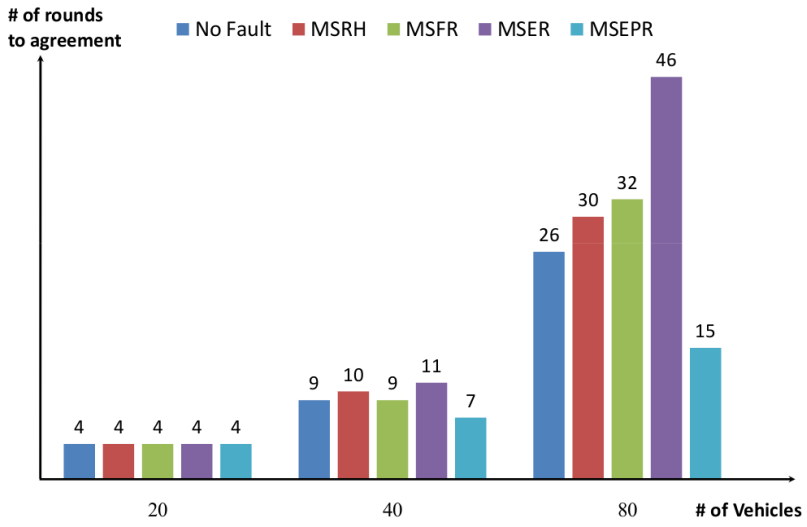

FIGURE 5.9: SOA faults impact on ECSA convergence speed in Model 2

The second singular fault model in Model 2 considers TS faults only. In this model, only the scattered distribution of malicious nodes is considered. Recall that vehicle neighborhood size might change during the simulation. However, no more than 30% of the TS faults are allowed to exist in the voting multiset for any vehicle at any time during the simulation. Figure 5.10 shows the impact of the scattered malicious nodes distribution on the ECSA along with the no fault model. As can be seen, the impact of malicious behavior is worse compared to the worst case in Model 1 shown in Figure 5.6.

Finally, consider the hybrid fault model for Model 2. Figure 5.11 shows the hybrid fault model impact on different agreement algorithms used in the ECSA. The

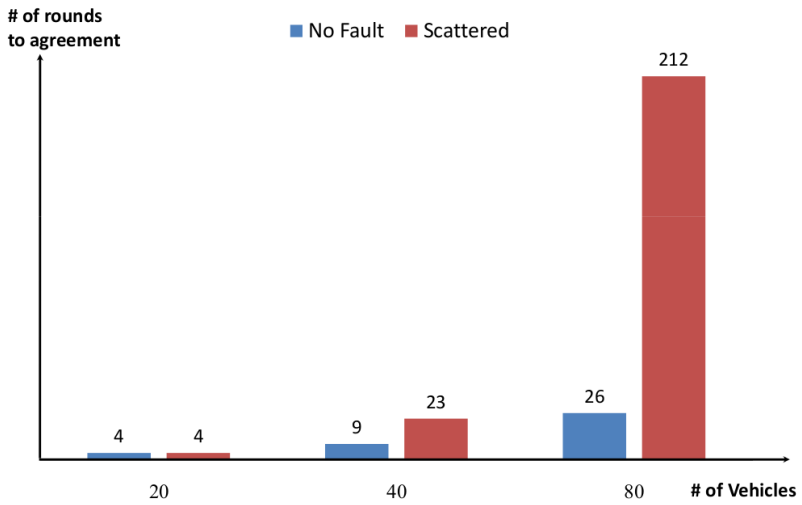

FIGURE 5.10: The impact of singular TS fault model on ECSA convergence speed in Model 2

observation is that the omission faults strengthen the impact of TS faults compared with the results shown in Figure 5.9 and Figure 5.10. It can also be noticed that the simulation results of this Model shown in Figure 5.11 are consistent with those in Model 1 shown in Figure 5.7. Specifically, both MSFR and MSEPR failed to converge before 300 rounds, whereas, both MSRH and MSER successfully converged before such bound, even though it took considerable time to do so in the case of high traffic density.

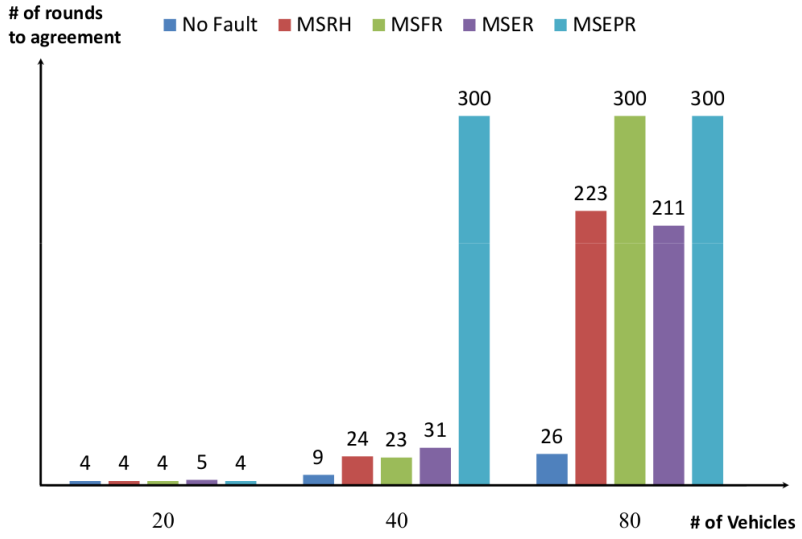

FIGURE 5.11: Hybrid fault model impact on ECSA convergence speed in Model 2 using scattered malicious node distribution

Table 5.2 summarizes the impact of different fault models on both simulation models presented above. Recall that the focus of the simulations was on the pathological (worst case) scenarios, and not on the best or average behavior. This table shows the worst number of rounds to reach agreement observed in each simulation scenario. The column labeled Model 1 shows the results among the six simulation scenarios in the first simulation model, whereas the column under Model 2 is the result using 80 vehicles in the second simulation model. The table further indicates that MSRH and MSER agreement algorithms perform better in comparison to other agreement algorithms.

TABLE 5.2: Performance comparison (in rounds) of the agreement algorithms under Model 1 and Model 2

	Model 1	Model 2
No SOA Nor TS	14	26
Strictly Omissive Asymmetric Faults		
MSFR	14	39
MSRH	14	30
MSER	29	54
MSEPR	23	16
Transmissive Symmetric Faults		
Scattered Distribution	38	212
Hybrid Fault Model		
MSFR	Exceeded 300	Exceeded 300
MSRH	55	239
MSER	213	191
MSEPR	Exceeded 300	Exceeded 300

5.5 CONCLUSIONS

This chapter addressed the clock synchronization problem in vehicular ad-hoc network (VANET). The proposed Enhanced Clock Synchronization (ECSA) is not meant to be a tight clock synchronization protocol, but to tolerant GPS time spoofing

attacks in the presence of omission and malicious faults. It is an augmentation to the GPS synchronization protocol to enhance the reliability of DSRC safety applications. The effectiveness of ECSA was demonstrated using the electronic emergency brake light (EEBL) safety application, where time spoofing attacks would have otherwise caused safety applications to disregard seemingly outdated messages, thus causing application failure.

The ECSA utilized different agreement algorithms based on approximate agreement (AA). The performance of these algorithms was analyzed by simulations under two different mobility models using NS-3 and SUMO. The analysis and the simulation results showed the stability and effectiveness of the ECSA over different simulation scenarios. ECSA offers higher clock synchronization reliability while not requiring any extra hardware nor producing additional message overhead that is prevalent in most agreement algorithms.

chapter 6

Clock Synchronization with Connected and Autonomous Vehicles

Tomorrow's transportation system is expected to include a mix of autonomous and connected vehicles. Such mixed vehicle model presents challenges in their coexistence, due to very different sensing technologies. In order to reduce accidents, safety applications have been introduced, which rely on wireless communication. Examples for connected vehicles are the DSRC Intersection Collision Avoidance and Intersection Movement Assist safety application. Autonomous vehicles should have the same communication capabilities as connected vehicles in addition to their onboard sensors. This is to overcome hazard detection limitations of line-of-sight sensors, especially in situations with blocked visibility such as in the aforementioned collision avoidance applications.

As concluded in Chapter 4, that DSRC safety applications require that vehicles are time-synchronized. Otherwise, safety applications might discard the received information as it could be considered to be outdated [38]. Recall that, time synchronization in VANET is based on GPS time signals, which migh suffer from signal reflections and short-term outages in urban areas. For example, the chance of receiving a clear GPS signal in urban areas such as Hong Kong, Tokyo, or New York is less than 20% [66]. Furthermore, vehicles may be subjected to GPS time spoofing attacks, where an attacker injects wrong GPS time stamps in the GPS signal, thereby causing a clock synchronization problem. Such faults may cause safety applications, such as the ICA, to discard received messages perceived outdated.

In this chapter, we assumed a mixed traffic model with user-operated connected vehicles and autonomous vehicles. In such environment, we can show that an Autonomous Vehicle (AV) with its typical line of sight sensing capability using lidar, radar, and/or cameras, may not be able to avoid intersection crashes similar to ICA. For example, sensing a crossing RV might be delayed in areas where large objects or buildings block the line of sight. It will be shown that augmenting the AV with

V2V communication capability can help overcome these limitations as it provides valuable information beyond line of sight, in addition to the onboard sensors used.

Moreover, we introduce a Clock Synchronization Algorithm for VANET that overcomes GPS time faults. The proposed algorithm is based on the clock synchronization approach presented in Chapter4 and aims to mitigate the ICA safety application failure. The proposed algorithm considers omission faults, where messages may be lost or are corrupted. Such faults may occur due to environmental conditions like shadowing and fading of the signal. The proposed algorithm uses a new approximate agreement algorithm that has the capability to handle omissions. The performance of the algorithm was evaluated using NS3 simulations combined with Simulation of Urban MObility (SUMO).

6.1 RELATED WORK

Several approaches to solve the clock synchronization problem in ad-hoc networks have been proposed. As described in Section 4.1, they are classified into two categories: centralized and decentralized [48]. The centralized approaches use a central clock source such as GPS [51], whereas the decentralized approaches are based on an iterative agreement process, in our case using approximate agreement.

Approximate agreement is conducted using a sequence of iterations, i.e.,rounds, in which the nodes' clock values converge to approximately the same value. In each round, every node broadcasts its current clock value to its neighbors. A node then updates its own value by applying a convergence function to the received values, which is then used in the next round of voting. An example of an agreement function is MSR [63], described in Section 4.3.2, that applies the three main functions: reduction, selection, and mean. However, this research uses an agreement algorithm based on the OMSR was proposed in [23] and discussed in Section 5.1.

6.2 MOTIVATIONAL FIELD TEST

To test the reliability of V2V communication for the ICA safety application in an intersection where large buildings limit line of sight, a field experiment was conducted at the University of Idaho campus using vehicles equipped with Arada Locomate Classic OBUs [4]. Figure 6.1 shows the field experiment location and setup, with the target intersection identified by a circle in the northwest location. Two vehicles were used in the experiment, one traveling westbound with a stop sign at the intersection, and the other northbound in a straight path. The vehicle marked by a red dot was the RV traveling on the crossing path with a stop sign of the HV marked by a green dot, with starting distances to the target intersection identified. The line of sight between the vehicles was obstructed by the building shown. Table 6.1 shows the

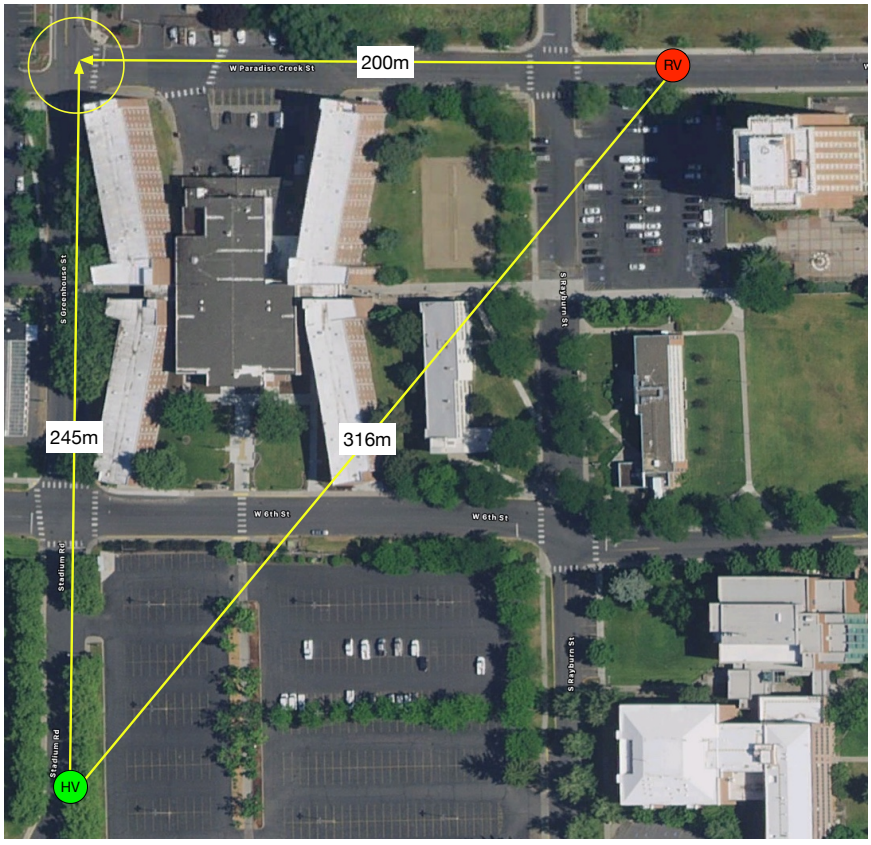

FIGURE 6.1: Field experiment Location

OBU configuration used for the field test.

Figure 6.2 shows the BSMs received by the HV per second in the absence of malicious act for a typical test case. The figure shows the number of BSMs received by

TABLE 6.1: Field test parameters

OBU Model	Arada Systems LocoMate Classic
Number of OBUs	2
Vehicles speed	5 m/sec
BSM generation	10 packets/s
Channel	Safety Channel 172
Transmitter power	21 dBm
Data rate	6Mbps

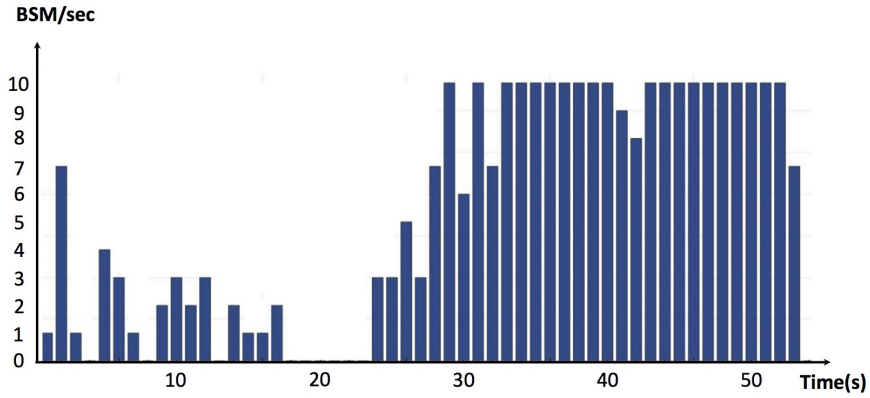

FIGURE 6.2: Received BSMs during the field experiment

the northbound HV from the westbound RV as they both approach the intersection. The starting locations of both vehicles at time zero are the colored dots shown in Figure 6.1. In this specific case initially BSM reception was low or absent during the first 23 seconds, and increased afterwards. It should be mentioned that the experiments were conducted during real traffic condition, where unrelated vehicles were in front of the RV, and light rain resulting in high humidity and wet asphalt.

In all cases it was found that DSRC suffered from omissions due to the buildings impacting communication. However, the BSM delivery rates were sufficient for the ICA to function properly. It should be noted that the reception of only one BSM is sufficient for ICA to function. In similar field experiments in open space such omissions were not observed at such distances.

In intersections with visual obstruction by large objects, such as shown in the northwest of the scenario in Figure 6.1, an AV without DSRC communication may not have sufficient time to detect an RV approaching the intersection. Thus, without line of sight, a crash may not be avoidable, as will be demonstrated at the end of Sec-

tion 6.4. This strongly suggests that DSRC communication capability be required for both human-driven vehicles and AVs to avoid possible intersection crash scenarios. In the latter case this is to compensate for the lack of line of sight.

6.3 CLOCK SYNCHRONIZATION ALGORITHM CONSIDERING OMISSIONS

In this section, we propose a clock synchronization algorithm inspired by the algorithm introduced in Section 4.3. The new algorithm can handle GPS signal outages and GPS time spoofing attacks in the presence of Strictly Omission Asymmetric (SOA) faults. The latter implies that one or more vehicles do not receive a BSM. The algorithm aims to enhance the ICA safety application reliability in the presence of such faults. Figure 6.3 presents the algorithm, which is divided into three main threads, i.e., BSM collection, agreement, and transmission thread.

In the BSM collection thread, the algorithm collects the BSMs received from the neighboring vehicles. Upon receiving a BSM, the transmitter vehicle ID and its timestamp are retrieved. Recall that the timestamp is the creation time of the BSM at the transmitter side. Each vehicle maintains a table called *Sync Table* to save the extracted information. The *Sync Table* has one record for each neighbor. A record contains the latest extracted information received, combined with the receiving time based on the receiver's local clock value. This record can be identified by the neighbor's vehicle ID.

The collection thread iterations are controlled by a *Timer*, initialized to 0, which controls the time spacing of agreement rounds. The thread executes as long as $Timer < 100ms - \Delta_o$, where Δ_o represent the agreement thread computation overhead. The agreement thread is initiated after *Timer* is expired. It is assumed that each vehicle has at least one neighbor to be able to run the agreement process. The agreement thread is executed only once per voting round. It consists of four main steps.

The first step of the agreement thread is data retrieval. The stored timestamps are retrieved and missing values are detected. The missing values can be easily detected

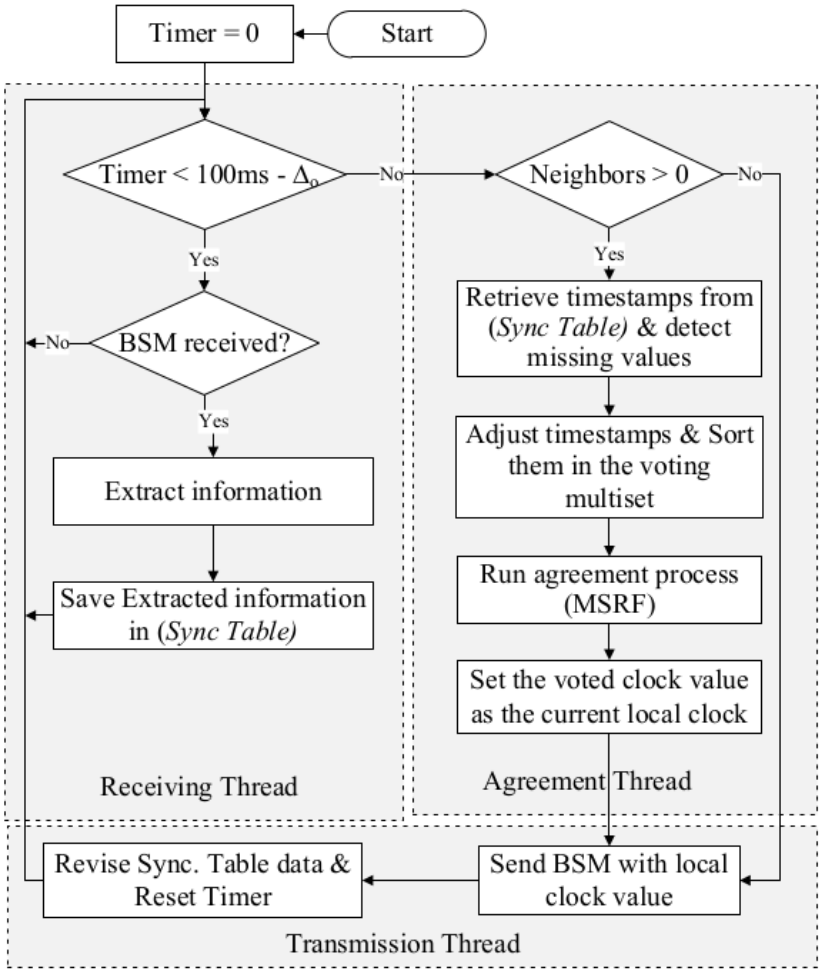

FIGURE 6.3: VANET clock synchronization algorithm

as each vehicle keeps the receiving time of the neighbors BSMs. Inspired by the OMSR agreement algorithm, the missing values are not replaced with any default value.

The second step performs adjustment and sorting. The retrieved times are adjusted similarly to [54]. The purpose of this step is to use fresh timestamps at the time agreement is initiated, denoted by t_{vote}. Any timestamp $t_{RV(j)}$ for any neighbor v_j is adjusted by the time difference between the receiving time of the BSM, denoted by $t_{HV(j)}$, and time t_{vote}. The adjusted timestamp t_j is thus computed as

$$t_j = t_{RV(j)} + (t_{vote} - t_{HV(j)}) \tag{6.1}$$

These timestamps are then sorted to form the final voting multiset.

The third step is the application of MSFR, which uses the MSR convergence function, however, unlike OMSR, it applies the reduction function Red^{τ} with fixed τ in order to remove the smallest and largest values from the voting multiset, producing the reduced multiset. Different reduction percentages will be analyzed in Section 6.4. The proposed algorithm applies the FTM selection function Sel_{σ} to the reduced voting multiset. FTM obtains the mean of the largest and smallest value selected from the reduced multiset. FTM was determined to be the best selection function for scenarios where vehicles are merging [60]. The mean is considered as the voted clock value. Finally, the voted clock is set to be the new local clock.

The transmission thread is executed next. It is responsible for 1) broadcasting the new BSM containing the local clock value to be used by the vehicle's neighbors in their agreement process, and 2) updating the *Sync Table*. Recall that neighbor information is saved in the *Sync Table* combined with the receiving time $t_{HV(j)}$ of this information. Thus, the *Sync Table* must be revised to account for the local clock value derived in the agreement process. This keeps the *Sync Table* up to date and enables the clock synchronization algorithm to track omissions. The new $t_{HV(j)}$ value is calculated and updated in the *Sync Table* for each record using Equation 5.2 described in Subsection 5.3.5.

6.4 SIMULATIONS AND ANALYSIS

This section shows the effectiveness of the proposed clock synchronization algorithm for different traffic densities using simulations and field-experiments.

6.4.1 *Simulation Results*

Simulations are divided into two main parts. The first part is concerned with the mobility model generation for the moving vehicles. It was conducted using Simulation of Urban MObility (SUMO) that emulates a real world VANET environment. The second part discusses network simulations using the network simulator NS3.

The simulations consisted of N vehicles moving on intersecting two-lane unidirectional roads. Vehicles were divided into two clusters, where each cluster was moving in the cross path direction of the other cluster. Vehicle speeds were random and bounded by the road speed limit. The leading vehicles of each cluster were configured to arrive the intersection point at the same time, effectively simulating a potential crash scenario. A BSM transmission range of 300m was assumed in an unobstructed environment, e.g., no buildings or other obstacles were simulated. Some BSMs could be lost due to phenomena like fading or shadowing, resulting in SOA faults [23]. Specifically, 10% of the transmitted BSMs during simulations were assumed to be omissions causing SOA faults, which is consistent with the Packet Error Rate (PER) of [64].

In the simulations, all vehicles of the cluster led by the RV were subjected to GPS time spoofing attacks. Their clock values were initialized with random values consistent with such an attack. Time spacings between initial values were chosen to be more than the BSM time-to-live [38], which might force the ICA safety application to discard the BSMs perceived as outdated. Each vehicle in the simulation executed the proposed clock synchronization algorithm introduced in Section 6.3.

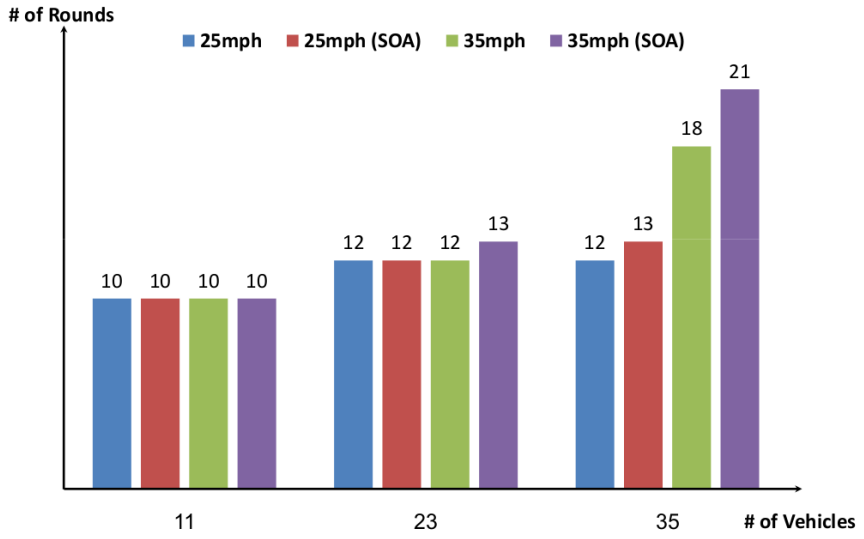

FIGURE 6.4: SOA impact on clock convergence speed [in rounds]

Figure 6.4 depicts the clock convergence speed for the proposed algorithm in the absence and presence of SOA faults for different speeds. The notation (SOA) added

to the ledger identifies those simulations that assumed the SOA fault model. The x-axis of the figure refers to the number of vehicles implemented in the simulation. The y-axis represents the number of rounds consumed until the agreement was reached. Each round accounts for a 100ms time span, which is the BSM transmission interval. Recall that agreement is achieved when the clock values of all communicating vehicles are within less than 500ms, i.e., the BSM time-to-live. In the case of 11 vehicles, synchronization was achieved in 10 rounds for all speeds and fault scenarios. This equal convergence speed is due to all vehicles being within the communication range of each other, i.e., the communication graph was fully connected within the same cluster. As traffic density increased, the impact of partially connected clusters caused agreement to take more rounds.

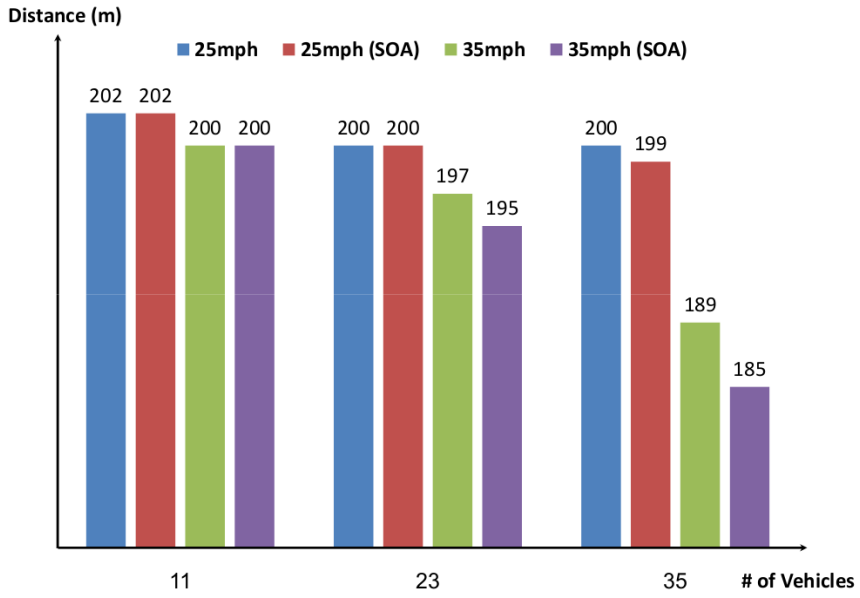

FIGURE 6.5: SOA impact on distance to crash [in meters]

Figure 6.5 shows the impact of clock convergence on the remaining distance to the crash point. The ICA scenario can be visualized as a triangle with right angle, where each cathetus represents the traveling path of a vehicle and a hypotenuse is equal to the line of sight distance between vehicles. If one assumes the maximal BSM transmission distance of 300m (hypotenuse), then the point where the leading vehicle of both clusters start to communicate is approximately 212m (cathetus). The y-axis of the figure indicates the distance to the crash point after the clocks have been

synchronized to less than 500ms. For each scenario in Figure 6.4, Figure 6.5 shows the remaining distance to the crash point. As can be seen, the more rounds are needed for synchronization the shorter the remaining distances to the crash point. It should be noted that even if scenarios have the same number of rounds, the distances may be different depending on the speed of the vehicles.

Recall that the MSFR agreement algorithm applies fixed reduction percentages to the voting multiset during the agreement process. Different reduction percentages were applied and their impact on the proposed algorithm and the ICA safety application was studied. Specifically, reductions of 0%, 15% and 30% were used in the simulations. In this chapter, reductions are not used to deal with erroneous values as in [23], but to allow fast synchronization of vehicles merging a synchronized cluster. An example of merging is a vehicle joining a synchronized cluster from side street. If the number of merging vehicles is below the reduction value, then one-round synchronization is achieved.

Figure 6.6 shows the impact of reduction on convergence speed, and Figure 6.7 shows respective remaining distances to the crash point. As can be seen, higher reduction percentages increase the number of rounds to converge, hence reducing the remaining distance to the crash point. The intuition is that the higher reduction removes more values from the voting multiset, in essence discarding more values from the other cluster that could otherwise have higher impact on the convergence speed.

Let's consider the effectiveness of ICA. For the speed of 25mph, all distances in Figure 6.7 are sufficient to alert the driver early enough to react and stop the vehicle before the crash point, as will be described in the case of 30% reduction. This is the most interesting scenario, as it leaves only 123m to impact. Assuming a reaction time of 1s (11.2m for 25mph) and considering almost 19m braking distance on wet asphalt (see Table 2.1), this leaves a safety buffer of about 95m. Looking at Figure 6.1, such distance allows for plenty of additional time to satisfy R_{safety} (see Figure 2.10).

The results of the figures above are for the ICA safety application. In fact the impact of reduction on clock convergence speed depends on the safety application. For example, the results presented in [60] for the EEBL safety application showed that

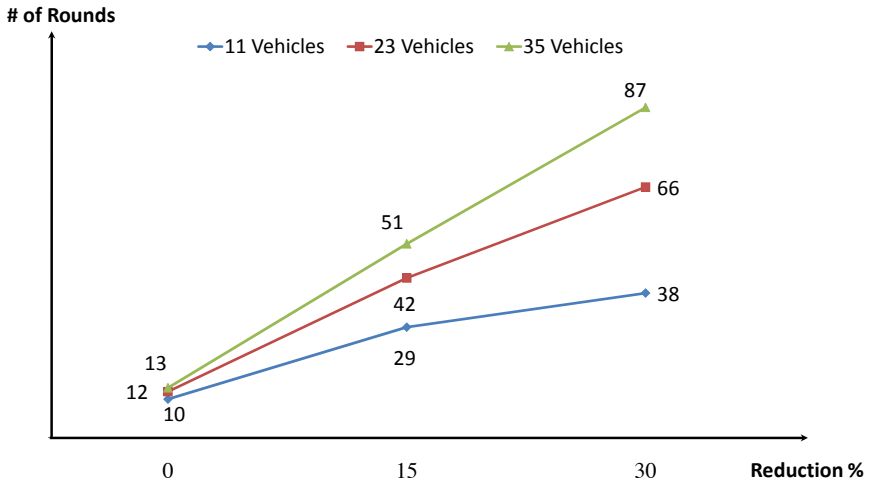

FIGURE 6.6: Impact of reduction percentages on convergence speed at 25mph

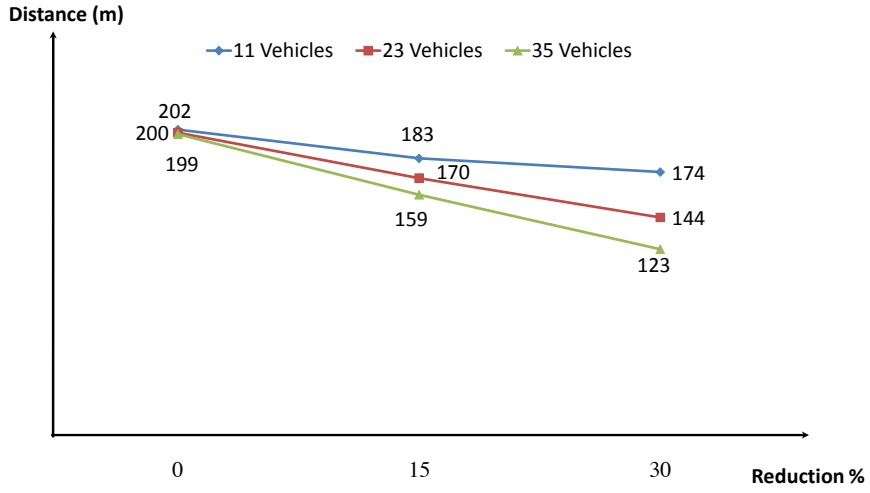

FIGURE 6.7: Impact of reduction percentages on convergence speed at 25mph

convergence speed did not deteriorate for higher reduction percentages. The reason behind this behavior is the relative speed of clusters in the two safety applications. For the ICA the speed at which clusters approach each other is significantly higher than that of EEBL, where the relative speeds of the clusters differed only slightly.

6.4.2 *Field-Test Analysis*

First we show the necessity of AV's to have DSRC capability. Let's assume the test scenario shown in Figure 6.8, where the northbound (green) vehicle is an AV operating without DSRC capability. Given the geometry of the intersection of the

test site, the AV will have no visibility of the RV until 18m to the crash point, as its view is obstructed by the building. However, from Table 2.1 we can see that the brake distance alone is 18.3m. Accounting for some time for the AV to detect the RV, we can conclude that a crash will not be avoidable.

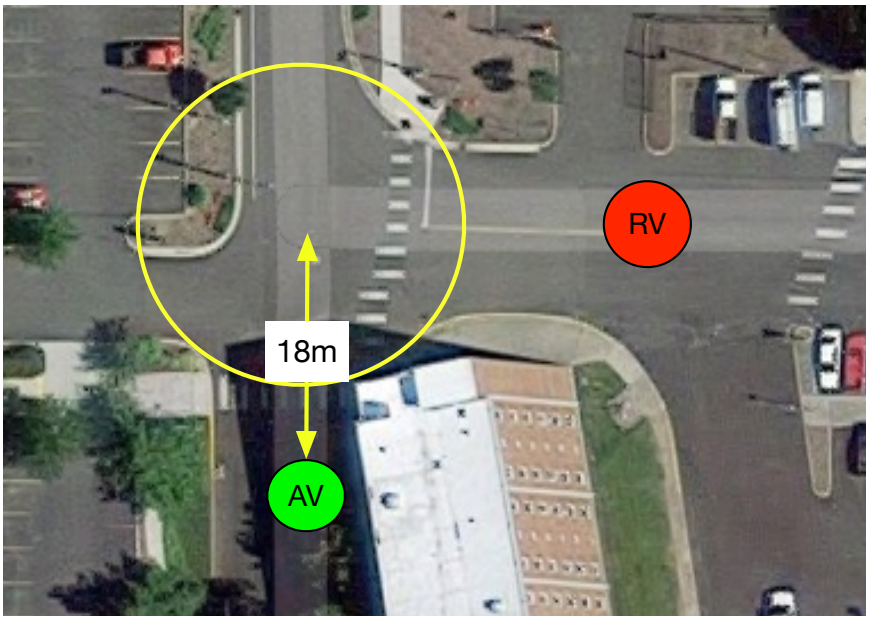

FIGURE 6.8: Collision avoidance of scenario with AV without DSRC

Now consider the field test of Figure 6.1, and assume the vehicle has DSRC capability. In a two vehicle scenario, as shown, it would take only two rounds to synchronize clocks. With only a fraction of the BSMs received in Figure 6.2 there is ample time for the ICA to alert the driver with plenty time to react. Even if we had the same number of vehicles as in the simulated scenarios, a collision would have been avoided. This can be deducted by establishing an upper bound from the worst case scenario in Figure 6.6, which showed 87 rounds before convergence for the largest vehicle cluster and greatest reduction. Taking the BSM reception ratios from the field test, this translates into a convergence time of approximately 7 seconds (or 77m) before reaching the crash point. This leaves plenty of time to react and brake.

Finally, if the green vehicle in Figure 6.1 would have been a DSRC-enabled AV, the results would have been even better than that of an HV, as its reaction time could be arguably superior to that of a human.

6.5 CONCLUSIONS

This chapter addressed the impact of clock synchronization problems on the Intersection Collision Avoidance safety application subjected to GPS outages or GPS time spoofing. It has been shown that such faults can cause safety applications to fail as messages from attacked vehicles appear outdated and are discarded. A mixed traffic model was assumed with user-operated connected vehicles and autonomous vehicles. To mitigate the faults, a decentralized clock synchronization algorithm based on approximate agreement was introduced, with the additional capability of dealing with omission faults. The algorithm was studied using the Intersection Collision Avoidance safety application. Its effectiveness was analyzed using NS3 and SUMO simulations, as well as field tests using vehicles equipped with commercial DSRC devices. The simulation results show that clock synchronization was achieved in a timely manner for different traffic densities and speeds, in spite of GPS time spoofing and omissions. The proposed algorithm does not incur additional message overhead, as it only uses the standard basic safety messages.

With respect to autonomous vehicles, the field tests demonstrated that DSRC capability for autonomous vehicles is crucial. In the test cases autonomous vehicles without such communication were not able to avoid crash scenarios.

CHAPTER 7

Conclusions and Future Work

This research considered VANETs and several safety applications that aim to reduce road accidents. Several fault types, ranging from benign to malicious, and their impact on the reliability of DSRC safety applications were studied. The faults considered were the result of wireless jamming and GPS time spoofing attacks. Four different mitigation strategies aiming to improve safety applications reliability were proposed. Such strategies can be implemented into the safety application design. They were tested through field tests, lab experiments, and simulations sing NS3 and SUMO. Test results showed the effectiveness of the proposed approaches. The algorithms do not require extra hardware overhead nor do they require modifications of existing standards.

As our first contribution, we introduced a new hybrid jammer attack that combines the properties of constant and deceptive jammers in addition to characteristics resembling random jammers. It exposed queuing behavior that was exploited for effective attack strategies. Two attack scenarios for stationary and mobile jammers were presented together with their impact on the EEBL safety application, however, we expect that the jammer will have similar implications for other DSRC safety applications. Experiments were conducted with commercial DSRC equipment, validating the expected impact of the jammer.

As a second contribution, a hybrid jamming detection algorithm was presented as a mitigation strategy for the above attack. The proposed algorithm was implemented inside the receiving thread of the Arada Locomate Classic OBU in order to be tested. It can distinguish between legitimate nodes subjected to the hybrid jamming attack and misbehaving nodes. The test results showed the effectiveness of the proposed algorithm as long as the OBUs were time-synchronized.

The third contribution was inspired from the previous one, where clock synchronization problems were noticed during testing the hybrid jamming detection algorithm. In this contribution, we introduced a decentralized clock synchronization

protocol for VANET, capable of mitigating GPS signal outages and GPS spoofing attacks. It is based on approximate agreement, where all vehicles participate in the synchronization process. Among several selection functions, which were implemented in the agreement algorithm, Fault Tolerant Midpoint was identified to perform best, especially in worst case scenarios. The results presented were supported by simulations using Network Simulator Version 3 (NS-3). The proposed protocol was superior compared with existing protocols like CTS and HCS, in terms of clock convergence speed, extra hardware overhead, and compliance to standards.

As a fourth contribution, an enhanced clock synchronization algorithm (ECSA) for the previous clock synchronization process was presented. ECSA has the capabilities to handle GPS spoofing attacks in the presence of omissions and malicious faults. It is also based on approximate agreement, where new agreement algorithms were introduced and their behavior were investigated. Results were supported by simulations using NS-3, based on two different mobility models. The theoretical analysis and the simulation results show the stability and effectiveness of the ECSA over different simulation scenarios.

As a fifth and final contribution, we introduced a clock synchronization algorithm that considers a different safety application in a mixed traffic model with user-operated connected vehicles and autonomous vehicles. Specifically, the algorithm aims to mitigate the GPS spoofing attack in the Intersection Collision Avoidance safety application in the presence of omission fault. Its effectiveness was analyzed using NS3 and SUMO simulations, as well as field tests using vehicles equipped with commercial DSRC devices. Field experiments shows that the DSRC capability for autonomous vehicles is crucial, or they might not be able to avoid intersection crash scenarios, especially when the view is blocked. Furthermore, the algorithm proved effective in mitigating GPS time spoofing attacks, even in environments where communication was affected by large buildings.

7.1 FUTURE WORK

This research can be the basis for several extensions. First, regarding wireless jamming attacks, which are considered to be common and easy to implement, it may be useful to employ counter measures upon detection. For example, for constant jammers various techniques such as frequency hopping and channel surfing may be considered, and their impact on reliability could be analyzed.

Second, we have proposed several clock synchronization algorithms where the main goal is not tight synchronization, but fast synchronization to increase safety application reliability. It would be interesting to consider a tighter clock synchronization tolerance for the proposed protocol. As shown in Chapter 6, the behavior of the clock convergence changed by changing the safety application and changing the crash scenario. Thus, other safety applications should be considered and their behavior should be investigated.

Third, we have assumed the use of fixed reduction percentages during the agreement process. It might be beneficial to consider adaptive reduction percentages. This adaptive reduction might depend on several parameters such as fault types, like in [23], and also might depend on the size of the vehicles' neighborhood.

Finally, all the approaches introduced in this dissertation referred to DSRC as the key technology. The basic concepts will however apply to wireless technology in general. Therefore we suggest to investigate how the attacks and mitigation strategies relate to cellular technologies like LET-Advanced or 5G.

Bibliography

[1] *Traffic Safety Facts: Crash Stats, U.S. Department of Transportation*, National Highway Traffic Safety Administration, DOT HS 812 326, August 2016.

[2] *Traffic safety facts: Crash stats âĂŤ Critical reasons for crashes investigated in the National Motor Vehicle Crash Causation Survey*, 2015.

[3] *Technology Applications for Traffic Safety Programs: A Primer*, National Highway Traffic Safety Administration, DOT HS 811 040, September 2008.

[4] Arada Systems, *www.aradasystems.com*

[5] *Standard Specification for Telecommunications and Information Exchange Between Roadside and Vehicle Systems - 5 GHz Band Dedicated Short Range Communications (DSRC) Medium Access Control (MAC) and Physical Layer (PHY) Spec.*, ASTM E2213-03, 2010.

[6] *Amendment of the Commission's Rules Regarding Dedicated Short-Range Communication Services in the 5.850-5.925 GHz Band (5.9 GHz Band)*, Federal Communications Commission FCC 03-324, 2004.

[7] H. Alturkostani, *Mitigation Strategies for Safety Applications in Vehicular Ad Hoc Networks Subjected to Jamming* University of Idaho, Moscow, USA, April 2016.

[8] *Vehicle Safety Communications-Applications (VSC-A) Final Report*, DOT HS 811 492 A. U.S. DoT, NHTSA. September 2011.

[9] *Crash Factors in Intersection-Related Crashes: An On-Scene Perspective* U.S. Department of Transportation, National Highway Traffic Safety Administration, DOT HS 811 366, September 2010.

[10] J. B. Kenney, *Dedicated Short-Range Communications (DSRC) Standards in the United States*, Proceedings of the IEEE, vol. 99, no. 7, pp. 1162-1182, 2011.

[11] *Dedicated Short Range Communications (DSRC) Message Set Dictionary. Society of Automotive Engineers* SAE J2735, November 2009.

[12] *Analysis of Fatal Motor Vehicle Traffic Crashes and Fatalities at Intersections, 1997 to 2004* U.S. Department of Transportation, National Highway Traffic Safety Administration, DOT HS 810 682, February 2007.

[13] *Characterization Test Procedures for Intersection Collision Avoidance Systems Based on Vehicle-to-Vehicle Communications* U.S. Department of Transportation, National Highway Traffic Safety Administration, DOT HS 812 223, December 2015.

[14] *A Policy on Geometric Design of Highways and Streets*, 6th Edition, American Association of State Highway and Transportation Officials (AASHTO), 2011.

[15] *802.11-2007 - 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* IEEE Std 802.11, 12 June 2007.

[16] *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 Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements* IEEE Std 802.11e, 2005.

[17] C. Brandauer, G. Iannaccone, C. Diot, T. Ziegler, S. Fdida and M. May, *Comparison of tail drop and active queue management performance for bulk-data and Web-like Internet traffic*, Proc. Sixth IEEE Symposium on Computers and Communications, Hammamet, 2001, pp. 122-129.

[18] L. Hendriks, *Effects of transmission queue size buffer and scheduling mechanisms on the IEEE 802.11p beaconing performance* Proc. Twente Student Conf. IT, pp. 1-10, 2011.

[19] M. Pease, T. Shostak, L. Lamport, *Reaching Agreement in the Presence of Faults* Journal of the ACM, (1980), 27(2), 228-234.

[20] PAUL J. WEBER, *Dynamic Reduction Algorithms for Fault Tolerant Convergent Voting With Hybrid Faults*, Michigan Technological University, 2006.

[21] F.J. Meyer and D.K. Pradhan, *Consensus with Dual Failure Modes*, Proc. 17th Fault-Tolerant Computing Symp., July 1987, pp. 48 - 54.

[22] P. Thambidurai, Y-K, Park, *Interactive Consistence with Multiple Failure Modes*, 7th Reliable Distributed Systems Symposium, (1988), 93-100.

[23] M. H. Azadmanesh and R. M. Kieckhafer, *Exploiting omissive faults in synchronous approximate agreement* in IEEE Transactions on Computers, vol. 49, no. 10, pp. 1031-1042, Oct 2000.

[24] H. Hasrouny, A.E. Samhat, C. Bassil, A. Laouiti, *Vanet Security Challenges and Solutions: A survey*, Vehicular Communications, Elsevier, vol. 7, pp. 7-20, 2017.

[25] S. Bittl, A. A. Gonzalez, M. Myrtus, H. Beckmann, S. Sailer and B. Eissfeller, *Emerging attacks on VANET security based on GPS Time Spoofing*, 2015 IEEE Conference on Communications and Network Security (CNS), Florence, 2015, pp. 344-352.

[26] X. Sun, X. Lin and P. H. Ho, *Secure Vehicular Communications Based on Group Signature and ID-Based Signature Scheme* 2007 IEEE International Conference on Communications, Glasgow, 2007, pp. 1539-1545.

[27] M. S. Al-kahtani, *Survey on security attacks in Vehicular Ad hoc Networks (VANETs)* 2012 6th International Conference on Signal Processing and Communication Systems, Gold Coast, QLD, 2012, pp. 1-9.

[28] Xu W, Trappe W, Zhang Y, Wood T, *The feasibility of launching and detecting jamming attacks in wireless networks*, In: Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2005, pp 46-57.

[29] K. Grover; A. Lim; Q. Yang, *Jamming and Anti-jamming Techniques in Wireless Networks: A Survey* Int. J. Ad Hoc and Ubiquitous Computing, Vol. 17, No. 4, 2014.

[30] K. Pelechrinis, et. al., *Denial of Service Attacks in Wireless Networks: The Case of Jammers* Communications Surveys & Tutorials, IEEE, Vol.13, No.2, pp.245-257, 2^{nd} Quarter 2011.

[31] C. Alocious, H. Xiao and B. Christianson, *Analysis of DoS attacks at MAC Layer in mobile adhoc networks* 2015 International Wireless Communications and Mobile Computing Conference (IWCMC), Dubrovnik, 2015, pp. 811-816.

[32] M. Li, S. Salinas, P. Li, J. Sun, and X. Huang, *MAC-Layer Selfish Misbehavior in IEEE 802.11 Ad Hoc Networks: Detection and Defense* IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 14, NO. 6, JUNE 2015.

[33] P. Kyasanur and N. Vaidya, *Selfish MAC layer misbehavior in wireless network* IEEE Trans. on Mobile Computing, vol. 4, no. 5, pp. 502-516, Sep. 2005.

[34] Z. Lu, W. Wang, and C. Wang, *On order gain of backoff misbehaving nodes in CSMA/CA-based wireless networks* in Proc. IEEE Conf. Comput. Commun., San Diego, CA, USA, Mar. 2010, pp. 1-9.

[35] L. Toledo, and X. Wang, *Robust detection of MAC layer denial-of-service attacks in CSMA/CA wireless networks* IEEE Trans. Inf. Forensics Security, vol. 3, no. 3, pp. 347-358, Sep. 2008.

[36] T. Zhou, R. R. Choudhury, and P. Ning, *P2dap-sybil attacks detection in vehicular ad hoc networks* IEEE J. Sel. Areas Commun., vol. 29, no. 3, pp. 582-594, Mar. 2011.

[37] H. Yu, P. B. Gibbons, and M. Kaminsky, *Sybillimit: A near-optimal social network defense against sybil attacks* IEEE/ACM Trans. Netw., vol. 18, no. 3, pp. 885-898, Jun. 2010.

[38] X. Ma, X. Yin, and K.S. Trivedi, *On the Reliability of Safety Applications in VANETs* Invited paper, International Journal of Performability Engineering Special Issue on Dependability of Wireless Systems and Networks, 8(2), March 2012.

[39] M. Raya, J. Hubaux, and I. Aad, *Domino: A system to detect greedy behavior in ieee 802.11 hotspots* in Proc. ACM 2nd Int. Conf. Mobile Syst., Appl. Serv., Boston, MA, USA, Jun. 2004, pp. 84-97.

[40] M. N. Mejri and J. Ben-Othman, *Entropy as a new metric for denial of service attack detection in vehicular ad-hoc networks*, in Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems, 2014.

[41] X. Ma, X. Yin, and K.S. Trivedi, *On the Reliability of Safety Applications in VANETs* Invited paper, International Journal of Performability Engineering Special Issue on Dependability of Wireless Systems and Networks, 8(2), March 2012.

[42] H. Alturkostani, A. Chitrakar, R. Rinker, and A. Krings, *On the Design of Jamming-Aware Safety Applications in VANETs* Cyber and Information Security Research Conference (CISR 2015), Oak Ridge National Laboratory, Tennessee, USA, April 2015.

[43] A. Serageldin, H. Alturkostani and A. Krings, *On the Reliability of DSRC Safety Applications: A Case of Jamming*, in IEEE International Conference on Connected Vehicles and Expo (ICCVE), Las Vegas, NV, pp. 501-506, 2013.

[44] W. B. Johnson, *Design and Analysis of Fault-Tolerant Digital Systems* Addison-Wesley Publishing Company, New York, 1989.

[45] IEEE Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-Channel Operation, IEEE Std 1609.4TM, 2010.

[46] *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 Amendment 6: Wireless Access in Vehicular Environments*, IEEE Std 802.11p, 2010.

[47] A. Nguyen, et.al., *Solution of detecting jamming attacks in vehicle ad hoc networks* Proc. 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems (MSWiMO13), ACM, New York, 405-410, 2013.

[48] S.Wang, A. Pervez, M. Nekovee, *Converging time synchronization algorithm for highly dynamic vehicular ad hoc networks (VANETs)* 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP), 2010.

[49] T. E. Humphreys, B. M. Ledvina, M. L. Psiaki, B. W. O Hanlon, and P. M. Kintner, Jr., *Assessing the spoofing threat: development of a portable GPS civilian spoofer* in Proceedings of the ION GNSS Meeting. Savannah, GA: Institute of Navigation, 2008.

[50] Satish M. Srinivasan, Azad H. Azadmanesh, *Data aggregation in partially connected networks* Computer Communications, Volume 32, March 2009, Pages 594-601.

[51] R. Scopigno and H. A. Cozzetti, *GNSS Synchronization in Vanets* 3rd International Conference on New Technologies, Mobility and Security, Cairo, 2009, pp. 1-5.

[52] J. Elson, L. Girod and D. Estrin, *Fine-grained time synchronization using reference broadcasts* the Fifth Symposium on Operating Systems Design and Implementation, pp. 147-163, (2002).

[53] L. Li, Y. Liu, H. Yang, H. Wang, *A Precision Adaptive Average Time Synchronization Protocol in Wireless Sensor Networks* in: Proceedings of the IEEE International Conference on Information and Automation, Zhangjiajie, China, 2008.

[54] C. Li, Y. Wang and M. Hurfin, *Clock Synchronization in Mobile Ad Hoc Networks Based on an Iterative Approximate Byzantine Consensus Protocol* 2014 IEEE 28th International Conference on Advanced Information Networking and Applications, Victoria, BC, 2014, pp. 210-217.

[55] D. Sam and C. Raj, *A time synchronized Vehicular Ad Hoc Network (HVANET) of roadside sensors and vehicles for safe driving* Journal of Computer Science, vol. 10, no. 10, pp. 1617-1627.

[56] K. Medani , M. Aliouat and Z. Aliouat, *High Velocity Aware Clocks Synchronization Approach in Vehicular Ad Hoc Networks* Springer International Publishing, 978-3-319-19578-0, pp. 479-490.

[57] Reza Khoshdelniat, Moh Lim Sim, Hong Tat Ewe, and Tan Su Wei, *Time Table Transfer Time Synchronization in Mobile Wireless Sensor Networks* vol. 5, PIERS Proceedings, Beijing 2009.

[58] M. H. Azadmanesh, and A. W. Krings, *Egocentric voting algorithms*, IEEE Transactions on Reliability, vol. 46, no. 4, pp. 494-502, Dec. 1997.

[59] M. H. Azadmanesh, and L. Zhou, *Egophobic voting algorithms*, Journal of Computers & Applications, vol. 25, no. 4, pp. 236-246, 2003.

[60] S. Hussein, A. Krings and A. Azadmanesh, *VANET clock synchronization for resilient DSRC safety applications* 2017 Resilience Week (RWS), Wilmington, DE, USA, 2017, pp. 57-63.

[61] K. Driscoll, B. Hall, H. Sivencrona, P. Zumsteg, *Byzantine fault tolerance, from theory to reality*, Intl. Conf. on Computer Safety, Reliability, and Security, Sept. 2003, pp. 235 âĂŞ 248.

[62] M. Paulitsch et. al., *Coverage and the use of cyclic redundancy codes in ultra-dependable systems*, Intl. Conf. on Dependable Systems and Networks, June 2005, pp. 346 âĂŞ 355.

[63] R. M. Kieckhafer and M. H. Azadmanesh, *Reaching approximate agreement with mixed-mode faults* in IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 1, pp. 53-63, Jan 1994.

[64] 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," in ISO/IEC/IEEE 8802-11:2012(E) (Revison of ISO/IEC/IEEE 8802-11-2005 and Amendments), pp.1-2798, Nov. 21 2012.

[65] 2016 New York City Bridge Traffic Volumes, New York City Department of Transportation (NYCDOT), PTDT17D00.E02, February 2018.

[66] X. Zhang, Q. Wang and D. Wan, *Map Matching in Road Crossings of Urban Canyons Based on Road Traverses and Linear Heading-Change Model* in IEEE Transactions on Instrumentation and Measurement, vol. 56, no. 6, pp. 2795-2803, Dec. 2007.

[67] *Arada LocoMate User's Guide* Version 1.23, 2012.

APPENDIX A

Field Test

A field experiment was conducted in order to study the jamming impact on V2V communication. For this purpose three vehicles were equipped with Arada LocoMate Classic OBUs from Arada Systems [4]. An additional Arada LocoMate Classic OBU was configured to be a jammer with programmable data rates. Vehicles were moving at 15 m/sec speed and 50 m distance between any two successive vehicles in a 2 lane road. The jammer was located at a fixed position in the middle of the test area. During the experiment, the OBUs were configured to log all transmitted and captured BSMs in a packet capture (pcap) file. Then we investigated these pacp files using Wireshark, a free open source packet analyzer. This program allowed us to analyze the experiment very carefully and study the impact of the jammer. Figure A.1 shows the location of the experiment, the vehicle's moving order and the jammer position.

A.1 ARADA LOCOMATE OBU COMMANDS

Arada Systems provides the default application *getwbsstxrxencdec* for its locomate OBUs and RSUs, that is used to transmit and receive BSMs. This integrated application can be configured with different parameters and supports different message sets.

The following command was used in the field test experiment to allow theOBU to transmit and receive BSMs and log them.

$getwbsstxrxencdec\ -s\ 172\ -t\ BSM\ -a\ 1\ -o\ TXRX\ -X\ TXRXLOG\ -r\ 6.0\ -j\ 18\ -d\ 100$

The option *-s* signifies using a service channel and it is followed by the channel number which is CH172. Similarly, option *-t* represents the message type to be a BSM. Option *-a* is used to identify the service channel access mode. The value 1 represents alternating mode, whereas 0 represents a continuous mode. Option *-o* is

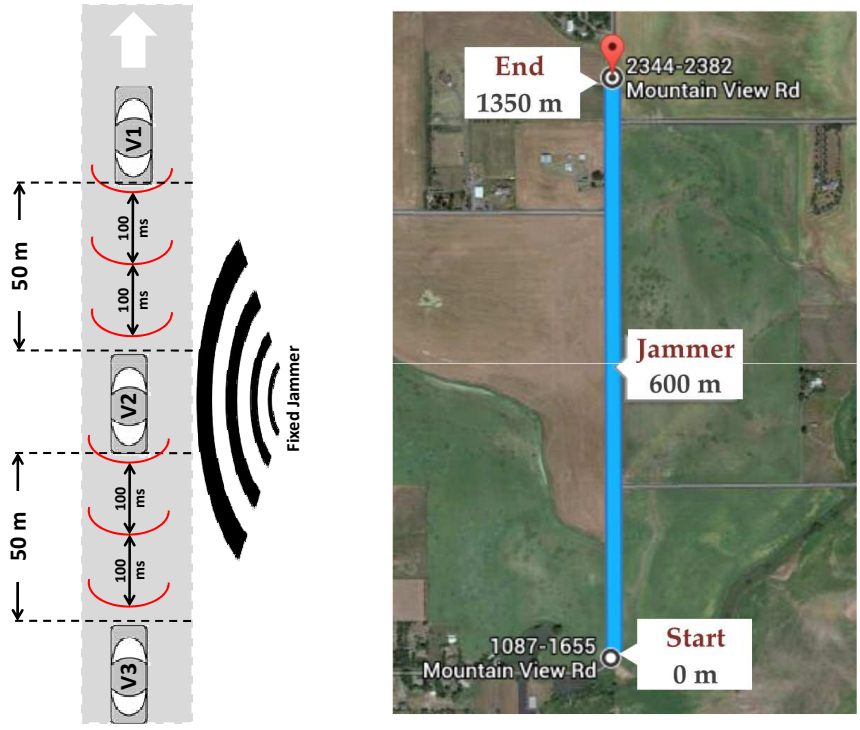

FIGURE A.1: Experiment setup

used to indicate the transmission mode. In this case is it is both transmission and reception. The option *-X* refers to the logging option. Here we logged all transmitted and captured BSMs. Option *-r* indicates the data rate and *-j* the transmission power. Finally, *-d* indicates packet delay in milliseconds. Tables A.1, A.2 and A.3 shows the Arada locomate application parameters and values.

TABLE A.1: Common options [67]

Parameter	**Description**
-m	Mac Address [xx:xx:xx:xx:xx:xx]
-s	Service Channel
-b	TxPkt Channel
-w	Service Type [Provider/User]
-t	Message Type [BSM/ PVD/ RSA/ ICA/ SPAT/ MAP/ TIM]

continued ...

. . . continued

Parameter	Description
-e	Security Type [Plain/Sign/Encrypt]
-D	Certificate Attach Interval in millisec should be in multiple of packet delay
-l	Output log filename, (specify path ending with / for pcap format)
-P	Prefex of certificate files)
-o	Tx/Rx Options [TXRX/ NOTX/ NORXALL/ NORX/ TXRXUDP/ NOTXRX]
-X	Logging Options [TXRXLOG/ TXLOG/ RXLOG/ NOLOG]
-g	sign certificate type [certificate/digest_224/digest_256/certificate_chain]
-p	BSM Part II Packet interval (n BSM Part I messages)
-v	Path history number [2 represents BSM-PH-2, 5 represents BSM-PH-5]
-	Vehicle_Type (value as per DE_VehicleType)
-y	psid value (any decimal value)
-d	packet delay in millisec
-q	User Priority 0/1/2/3/4/5/6/7
-j	txpower in dBm
-M	Model Deployment Device ID
-T	Temporary ID control (1 = random, 0 = fixed upper two bytes)
-S	Safety Supplement (wsmp-s) <0:disable / 1:enable>
-L	Vehicle Length in cm
-W	Vehicle Width in cm

continued . . .

. . . continued

Parameter	Description
-r	data rate 0.0, 3.0, 4.5, 6.0, 9.0, 12.0, 18.0, 24.0, 27.0, 36.0, 48.0,54.0mbps
-n	no argument, and selects no gps device available
-f	Type xml or csv for logging in XML or CSV format. Type pcaphdr for only pcap header logging and pcap for full packet logging
-F	frameType for TIM Packet 0-unknown(default) 1-advisory 2-roadSignage 3-commercialSignage
-A	Active Message Status
-B	Port Address for RSU receive from UDP Server
-R	Repeat rate for WSA frame (Number of WSA per 5 seconds) Repeatrate is included in WSA-Header only if enabled from /proc/wsa_repeatrate_enable
-G	Repeat rate for TA frame (Number of TA per 5 seconds) TA is available only if TA channel [-c option] is given
-I	IP service Enable 1= enable 0 = disable
-O	Timeout for receiving udp data , in seconds

Table A.2: Provider options [67]

Parameter	Description
-z	Service Priority
-a	Service Channel Access [1:Alternating, 0:Continuous]
-c	Specify Channel Number to Transmit TA
-i	TA Channel Interval [1:cch int, 2:sch int]

Table A.3: User options [67]

Parameter	Description
-u	User Request Type [1:auto, 2:unconditional(not wait for WSA from provider), 3:none]
-x	Extended Access <0:alternate /1:continuous>

A.2 ARADA LOCOMATE OBU JAMMER COMMAND

The following commands are the instructions to execute the Jammer application. To start the jammer use the *start_tx99* application with proper frequency, mode, rate, power and configuration values, e.g.

*Start_tx*99 *f* 5860 *m* 1 *r* 6000 *p* 18 *c* 0

The option *-f* signifies the starting frequency of the jammer and in this case it is for CH172. Option *-m* is used to identify the service channel access mode. *-r* indicate the data rate and *-p* indicated the transmission power. Option *-c* is used to indicate the configuration type as data mode and single carrier.

To stop the continuous transmit run the *stop_tx99* application without any arguments, e.g.

*stop_tx*99

APPENDIX B

Transmitter/Receiver OBU Code

B.1 TRANSMITTER

B.1.1 *GPS Information Extraction*

Listing B.1: Global Variables

```
/* This function reads the GPS information,
 * to fill BSM's information such as:
 * Positioning information,
 * Timing information,
 * and Vehcile's speed
 * If the GPS data is not available,
 * default values are selected.
 */
int getGPSInfo() {
  char ch = '1';
  uint64_t *lat, *lng, *alt;
  int status = 0;
  gpssockfd = gpsc_connect(NULL);
  //Check if the GPS is running or not
  if (gpssockfd < 0) {
    printf("gpstime: gpsc is not running...%d\n", gpssockfd);
    return -1;
  }
  //Memory allocation to save GPS information
  memset(&wsmgps, 0, sizeof(GPSData));
  status = write(gpssockfd, &ch, 1);
  if (status < 1) {
    syslog(LOG_INFO, "gpstime: write error %d (err %d)!!\n", status
        , errno);
    return -1;
  }
```

```
26  status = read(gpssockfd, &wsmgps, sizeof(GPSData));  //read from
       gpsc
27  if (status < sizeof(GPSData)) {
28    syslog(LOG_INFO, "gpstime: read error %d (err %d) exp %d!!\n",
         status,
29    errno, sizeof(GPSData));
30    gpsc_close_sock();
31    gpssockfd = -1;
32    sleep(3);
33  }
34  //Read GPS data
35  //If GPS is not connected,
36  //The system will not work at all
37  //If the GPS data is not available,
38  the function selects default values
39  if (wsmgps.actual_time == GPS_INVALID_DATA || wsmgps.actual_time
       == 0.0) {
40    printf("gpstime:wrg at=%lf fix=%d\n", wsmgps.actual_time,
         wsmgps.fix);
41    gpsc_close_sock();
42    gpssockfd = -1;
43    sleep(3);
44  }
45  if(wsmgps.actual_time != 0.0)
46    GPS_Acutal_Time = wsmgps.actual_time;
47  if (wsmgps.latitude == 0)
48    latitude_val = 900000001;
49  else
50    latitude_val = (long) ((wsmgps.latitude) * 10000000);
51  if (wsmgps.longitude == 0)
52    longitude_val = 1800000001;
53  else
54    longitude_val = (long) ((wsmgps.longitude) * 10000000);
55  if (wsmgps.speed != GPS_INVALID_DATA) {
56    transmission_speed[0] = (uint8_t) (((uint32_t) (wsmgps.speed *
         50)
57        & 0xFF00) >> 8);
```

```
58      transmission_speed[1] = (uint8_t) (((uint32_t) (wsmgps.speed *
            50)
59           & 0x00FF));
60      transmission_speed[0] = transmission_speed[0] | 0xE0;
61    } else {
62      transmission_speed[0] = ((8191 & 0xFF00) >> 8);
63      transmission_speed[1] = ((8191 & 0x00FF));
64      transmission_speed[0] = transmission_speed[0] | 0xE0;
65    }
66    return 0;
67  }
```

B.1.2 *BSM Transmitter*

Listing B.2: Global Variables

```
/* Function to build the WSM Request packet.
 * This function encapsulate the BSM related
 * info in to WSM request packet
 */
int buildWSMRequestPacket() {
  //Local variables
  int j;
  asn_enc_rval_t rvalenc;
  asn_dec_rval_t temp;
  uint32_t intg32;
  uint64_t intg64;
  uint16_t intg16;
  /* BSM Data structure declaration */
  BasicSafetyMessage_t *bsm;
  ////////////////////////
  //Memory allocation
  VehicleType_t *vehicleType = (VehicleType_t *) calloc(1,sizeof(
     VehicleType_t));
  vehicleType->buf = (uint8_t *) calloc(1, sizeof(uint8_t));
  vehicleType->size = sizeof(uint8_t);
  vehicleType->buf[0] = vehicle__type;
  VehicleIdent_t *vi = (VehicleIdent_t *) calloc(1, sizeof(
     VehicleIdent_t));
  vi->vehicleType = vehicleType;
  VehicleStatus_t *status = (VehicleStatus_t *) calloc(1, sizeof(
     VehicleStatus_t));
  status->vehicleIdent = vi;
  /* WSM Channel and Tx info */
  wsmreq.chaninfo.channel = Service_Channel;
  wsmreq.chaninfo.rate = Data_Rate;
  wsmreq.chaninfo.txpower = Tx_Power;
  wsmreq.version = 1;
  wsmreq.security = 1;
  wsmreq.psid = 10;
```

```
32  wsmreq.txpriority = 1;
33  memset(&wsmreq.data, 0, sizeof(WSMData));
34  /* BSM related information */
35  bsm = (BasicSafetyMessage_t *) calloc(1, sizeof(*bsm));
36  bsm->status = status;
37  /* allocate memory for buffer which is used to store, what type
        of message it is*/
38  bsm->msgID.buf = (uint8_t *) calloc(1, sizeof(uint8_t));
39  bsm->msgID.size = sizeof(uint8_t);
40  /* Choose what type of message you want to transfer */
41  bsm->msgID.buf[0] = DSRCmsgID_basicSafetyMessage;
42  /* Allocate the memory for the blob buffer, which is used to
        store the data */
43  bsm->blob1.buf = (uint8_t *) calloc(1, 38 * sizeof(uint8_t));
44  bsm->blob1.size = 38 * sizeof(uint8_t);
45  bsm->blob1.buf[0] = count % 127;
46  count++;
47  //Set Vehicle temp ID
48  intg32 = htobe32(temp_id);
49  memcpy(bsm->blob1.buf + 1, &intg32, 4);
50  //Set the two bytes for the Dsecond
51  if(Old_GPS_Acutal_Time != GPS_Acutal_Time)
52  {
53    double fractpart, intpart;
54    fractpart = modf (wsmgps.actual_time , &intpart);
55    uint32_t s = (uint32_t)intpart % 60 * 1000;
56    uint32_t us = fractpart * 1000 ;
57    t_Dsecond = s + us ;
58    printf("<GPS actual Time =%lf and Dsecond = %d>\n", wsmgps.
          actual_time , t_Dsecond);
59    repeated = 0;
60  }
61  else
62  {
63    repeated++;
64    double extra = (double)(repeated * TransmissionRate) / 1000.0;
65    double fractpart, intpart;
```

```
66      fractpart = modf (wsmgps.actual_time , &intpart);
67      uint32_t s = (uint32_t)intpart % 60 * 1000;
68      uint32_t us = (fractpart + extra) * 1000 ;
69      t_Dsecond = s + us;
70      printf("<GPS actual Time =%lf  and Dsecond = %d>\n",
71        wsmgps.actual_time + ((double)(repeated * TransmissionRate) /
              1000.0), t_Dsecond);
72    }
73    bsm->blob1.buf[6] = t_Dsecond % 256;
74    bsm->blob1.buf[5] = t_Dsecond / 256;
75    Old_GPS_Acutal_Time = GPS_Acutal_Time;
76    //Set Positions - 14 bytes
77    //Set the four bytes for the latitude
78    intg32 = htobe32(latitude_val);
79    memcpy(bsm->blob1.buf + 7, &intg32, 4);
80    //Set the four bytes for the longitude
81    intg32 = htobe32(longitude_val);
82    memcpy(bsm->blob1.buf + 11, &intg32, 4);
83    //Set the four bytes for the altitude
84    intg16 = htobe16(wsmgps.altitude);
85    memcpy(bsm->blob1.buf + 15, &intg16, 2);
86    //Set the four bytes for the Postional Accuracy
87    intg32 = htobe32(0xFFFFFFFF);
88    memcpy(bsm->blob1.buf + 17, &intg32, 4);
89    //Set Motion - 12 bytes
90    //Set the two bytes for the Speed
91    intg16 = htobe16(wsmgps.speed);
92    memcpy(bsm->blob1.buf + 21, &intg16, 2);
93    //TODO: Set the two bytes for the Heading
94    intg32 = htobe32(0);
95    memcpy(bsm->blob1.buf + 23, &intg32, 2);
96    //TODO: Set the one byte for the Angle
97    memcpy(bsm->blob1.buf + 25, &intg32, 1);
98    //TODO: Set the seven bytes for the Acceleration
99    intg64 = htobe64(0);
100   memcpy(bsm->blob1.buf + 26, &intg32, 7);
101   //Control - 2 bytes
```

```
102   //TODO: Set the two bytes for the Status
103   memcpy(bsm->blob1.buf + 33, &intg32, 2);
104   //Basic - 3 bytes
105   //TODO: Set the three bytes for the acceleration
106   memcpy(bsm->blob1.buf + 35, &intg32, 3);
107   ////////////////////////
108   gettimeofday(&tv, NULL);
109   printf("<logtime seconds = %llu microseconds = %d>\n", (uint64_t)
          tv.tv_sec, (uint32_t)tv.tv_usec);
110   ////////////////////////
111   rvalenc = der_encode_to_buffer(&asn_DEF_BasicSafetyMessage, bsm,&
          wsmreq.data.contents, 1000); /* Encode your BSM in to WSM
          Packets */
112   xer_fprint(stdout, &asn_DEF_BasicSafetyMessage, bsm);
113   if (rvalenc.encoded == -1) {
114     fprintf(stderr, "Cannot encode %s: %s\n", rvalenc.failed_type->
            name,strerror(errno));
115   } else {
116     wsmreq.data.length = rvalenc.encoded;
117     asn_DEF_BasicSafetyMessage.free_struct(&
            asn_DEF_BasicSafetyMessage, bsm,0);
118   }
119   return 1;
120 }
```

B.2 RECEIVER

B.2.1 *BSM Reception*

Listing B.3: Global Variables

```
1  /* Receiving Function
2   * Runs continuously to receive any BSM from surrounding neighbors.
3   * It runs the hybrid jamming detection algorithm,
4   * which can differentiate between misbehaving vehicles,
5   * and vehicles subjected to hybrid jamming attack.
6   * It calls a misbehaving detection mechanism if a vehicle behavior
        is suspected.
7   */
8  void *rx_client(void *data) {
9    //Local variables
10   int ret = 0;
11   int i = 0;
12   int Vindex = 0;
13   int Diff1 = 0;
14   int Diff2 = 0;
15   asn_dec_rval_t decVal;
16   int first = 0;
17   pid = getpid();
18   double fractpart, intpart;
19   uint32_t s, us;
20   double extra;
21   char ch = '1';
22   int status = 0;
23   //Infinity loop
24   while (1)
25   {
26     //Get the received BSM
27     ret = rxWSMMessage(pid, &rxmsg);
28     gettimeofday(&tv, NULL);
29     //Read GPS Data
30     gpssockfd = gpsc_connect(NULL);
31     if (gpssockfd < 0) {
32       printf("gpstime: gpsc is not running...%d\n", gpssockfd);
```

```
33        return -1;
34      }
35      memset(&wsmgps, 0, sizeof(GPSData));
36      status = write(gpssockfd, &ch, 1);
37      if (status < 1) {
38        syslog(LOG_INFO, "gpstime: write error %d (err %d)!!\n",
             status, errno);
39        return -1;
40      }
41      status = read(gpssockfd, &wsmgps, sizeof(GPSData));  //read
           from gpsc
42      GPS_Acutal_Time = wsmgps.actual_time;
43      //Receiving and printing the new BSM
44      if (ret > 0)
45      {
46        printf("<logtime seconds = %llu microseconds = %06d> \n", (
             uint64_t)tv.tv_sec, (uint32_t)tv.tv_usec);
47        asn_DEF_BasicSafetyMessage.free_struct(&
             asn_DEF_BasicSafetyMessage, bsmLog,0);
48        bsmLog = (BasicSafetyMessage_t *) calloc(1, sizeof(*bsmLog));
49        decVal = ber_decode(0, &asn_DEF_BasicSafetyMessage, (void **)
              &bsmLog,ind.data.contents, 1000); // Decode
50        count++;
51        xer_fprint(stdout, &asn_DEF_BasicSafetyMessage, bsmLog);
52        VehicleID = (*(bsmLog->blob1.buf + 1) << 24) | (*(bsmLog->
             blob1.buf + 2) << 16)
53            | (*(bsmLog->blob1.buf + 3) << 8)  | (*(bsmLog->blob1.buf
                  + 4));
54        Vindex = CheckVehicleID();
55
56        double diff12 = tv.tv_sec + (double)tv.tv_usec/1000000.0;
57        if(Old_GPS_Acutal_Time != GPS_Acutal_Time)
58        {
59          fractpart = modf (wsmgps.actual_time , &intpart);
60          s = (uint32_t)intpart % 60 * 1000;
61          us = fractpart * 1000 ;
62          HVDsecond = s + us ;
```

```
63          repeated = 0;
64        }
65        else if(repeated < 200/TransmissionRate -1)
66        {
67          repeated++;
68          extra = (double)(repeated * TransmissionRate) / 1000.0;
69          fractpart = modf (wsmgps.actual_time , &intpart);
70          s = (uint32_t)intpart % 60 * 1000;
71          us = (fractpart + extra) * 1000 ;
72          HVDsecond = s + us;
73        }
74        Old_GPS_Acutal_Time = GPS_Acutal_Time;
75        NewBSMDsecond = (*(bsmLog->blob1.buf + 5) << 8) | (*(bsmLog->
              blob1.buf + 6));
76        SavedBSMDsecond = (*(VNTbsmLog[Vindex]->blob1.buf + 5) << 8)
              | (*(VNTbsmLog[Vindex]->blob1.buf + 6));
77        //Count the missing BSMs
78        if(*(bsmLog->blob1.buf) > *(VNTbsmLog[Vindex]->blob1.buf) +
              1)
79          VNTPktLost[Vindex] += abs(*(bsmLog->blob1.buf) - *(
                VNTbsmLog[Vindex]->blob1.buf)) - 1;
80        else if(*(bsmLog->blob1.buf) < *(VNTbsmLog[Vindex]->blob1.buf
              ))
81          VNTPktLost[Vindex] += abs(*(bsmLog->blob1.buf) - *(
                VNTbsmLog[Vindex]->blob1.buf) + 127) - 1;
82        //Check for misbehaving behavior
83        if(NewBSMDsecond < SavedBSMDsecond)
84          Diff1 = NewBSMDsecond - SavedBSMDsecond + 60000;
85        else
86          Diff1 = NewBSMDsecond - SavedBSMDsecond;
87        Diff2 = HVDsecond - NewBSMDsecond;
88        if(Diff2 < 0)
89          Diff2 = 0;
90        //Updating the VNT table
91        //for the first packet only
92        VNTLastPktSec[Vindex] = tv.tv_sec;
93        VNTLastPktuSec[Vindex] = tv.tv_usec;
```

```
94        VNTPktReceived[Vindex]++;
95        asn_DEF_BasicSafetyMessage.free_struct(&
             asn_DEF_BasicSafetyMessage, VNTbsmLog[Vindex],0);
96        VNTbsmLog[Vindex] = (BasicSafetyMessage_t *) calloc(1,
             sizeof(*bsmLog));
97        decVal = ber_decode(0, &asn_DEF_BasicSafetyMessage, (void **)
              &VNTbsmLog[Vindex],ind.data.contents, 1000);
98        if(Diff1 >= (TransmissionRate - (TransmissionRate / 5)))
99        {
100         if(VNTMissingBSMs[Vindex] < N_env_error)
101         {
102           VNTMissingBSMs[Vindex] = 0;
103         }
104         else
105         {
106           //Check for hybrid jamming attack
107           if(Diff2 / 100 <= VNTMissingBSMs[Vindex] && Diff2 / 100 >
                  1)
108           {
109             printf("<<<<Hybrid Jammer Attack Detected>>>>\n");
110             printf("<Expected Missing BSM = %03d>\n",VNTMissingBSMs
                   [Vindex]);
111             VNTMissingBSMs[Vindex] = VNTMissingBSMs[Vindex] - 1;
112           }
113           else if(Diff2 / 100 == 0)
114           {
115             VNTMissingBSMs[Vindex] = 0;
116           }
117         }
118         printf("<Number of BSMs lost till now = %04d>\n",VNTPktLost
                [Vindex]);
119       }
120       else
121       {
122         if(NewBSMDsecond != SavedBSMDsecond)
123           printf("<Call Misbehaving Detection Technique>\n");
124       }
```

```
125         ////////////////////////
126       }
127       else CheckMissingBSM();
128     }
129 }
```

B.2.2 *Missing BSMs Check*

Listing B.4: Global Variables

```
/* Function to count missing BSMs from existing neighbors.
 * Based on the information saved in the VNT,
 * this function can calculates the missing BSMs since the last BSM
     received.
 */
void CheckMissingBSM()
{
  int i = 0;
  if(VNTCounter > 0)
  {
    //Get the current time
    CurrentPktSec = tv.tv_sec;
    CurrentPktuSec = tv.tv_usec;
    for(i = 0; i < VNTCounter; i++)
    {
      //Calculate the time passed since the last BSM was received
      DiffSec = CurrentPktSec - VNTLastPktSec[i];
      DiffuSec = CurrentPktuSec - VNTLastPktuSec[i];
      if((DiffSec * 1000000 + DiffuSec) > (100000 + VNTMissingBSMs[
          i] * 100000))
      {
        //Increament the BSM_missing counter by 1
        //if the time passed since the last BSM received is > 100ms
        VNTMissingBSMs[i]++;
        if(VNTMissingBSMs[i] > N_env_error)
        {
          printf("<Number of Missing BSM from Vehicle 0X%08X is %03
              d>\n", VNT[i],VNTMissingBSMs[i]);
        }
        //Vehicle is considered out of range and deleted from the
            VNT
        if(VNTMissingBSMs[i] > 100)
        {
          DeleteVehicle(i);
```

```
31              i--;
32            }
33          }
34        }
35    }
36 }
```

APPENDIX C

Clock Synchronization Simulation Code in NS3

C.1 GLOBAL VARIABLES

Listing C.1: Global Variables

```
1  //Max number of allowed packets in the simulation
2  #define MaxNumofPackets 100000
3  //Predefined Clock tolerance
4  #define Global_Tolerance 500
5  //Number of Nodes in the simulation
6  #define NodeCount 80
7  //Two clusters are simulated as in the proposed model
8  #define NumOfClusters 2
9  //Distance between the clusters
10 #define ClustersSeparator 320
11 //Log file name
12 char LogFile[200] = "LowDensity-S1-30%Reduction.txt";
13 //Trace file name
14 std::string tracebase = "scratch/Animation_Test";
15 //Pointer for animation interface
16 AnimationInterface * pAnim = 0;
17 //Animate or not
18 bool Animate = false;
19 //Print investigation messages to check simulation
20 bool PrintMessages = false;
21 //Selection function 1-FTA, 2-FTM, 3-MidPoint
22 int Algorithm = 2;
23 //Reduction Percentage
24 double ReductionPercentage = 30;
25 //Distance between two successive vehicles
26 int mindistance = 15;
27 int minSpeed = 15;
28 //Seeds used for experiment
```

```
29 #define RandValue 1
30 #define MaxSeed 100
31 //Variable used to change the random time seed
32 int TimeSeed = RandValue;
33 //Synchronization table entries
34 //1 - Current receiving time
35 //2 - Receiver Dsecond
36 //3 - Offset between receiver and transmitter
37 //4 - Vehicle Heading
38 long SyncronizationTable[NodeCount][NodeCount][5] = {0};
39 //Some BSM Fields
40 uint16_t Dsecond[NodeCount] = {0};
41 uint64_t Lat[NodeCount] = {0x1BDCF4FE};
42 uint64_t Long[NodeCount] = {0xBA47D7D};
43 uint8_t Packet_Number[NodeCount];
44 //Counter for synchronization iterations for each vehicle
45 uint32_t NodeSyncIteration[NodeCount] = {0};
46 //Removed nodes Array
47 uint8_t RmovedNodesArray[NodeCount] = {0};
48 //# of number of active nodes surrounding each vehicle
49 uint32_t CountActiveNodes[NodeCount] = {0};
50 //This variable used to indicate frequency of running sync.
    algorithm
51 int IterationSeparator = 1;
52 //Calculates average number of iteration in each trial
53 double AverageNumberOfIterations = 0;
54 //NS3 logging type
55 NS_LOG_COMPONENT_DEFINE ("WifiSimpleOcb");
56 //Mobility model variable
57 Ptr<ConstantVelocityMobilityModel> cvmm[NodeCount];
58 //Stop synchronization flag
59 bool StopFlag[NodeCount] = {false};
60 //Voting multi-set for each vehicle
61 uint16_t Time_Table[NodeCount][NodeCount] = {0};
62 uint16_t ActiveNodesArround[NodeCount][NodeCount] = {0};
63 const double Max_Dsecond_Diff = 60000;
64 uint32_t Min_Dsecond = 0, Max_Dsecond = 0;
```

```
65 //Total number of iterations in each experiment
66 int NumberOfIterations = 1;
67 //Max number of iterations in all experiment
68 uint32_t MaxNumberofIterations = 0;
69 //Stop synchronization flag
70 bool StopConvergence = false;
```

C.2 BSM GENERATION AND TRANSMISSION

Listing C.2: BSM Generation and Transmission Code

```
/* BSM transmission function
 * This function is responsible for calling the agreement process,
 * and transmitting the new BSM with the new local clock value.
 * This function fills the BSM with the required information such
    as:
 * BSM sequence number,
 * Vehicle ID,
 * Positioning information,
 * and the new Dsecond value.
 * Finally it sends the BSM and reschedules itself to run after the
     timer expires.
 */
void GenerateBSM (Ptr<Socket> socket, uint32_t pktSize, uint32_t
   pktCount, Time pktInterval )
{
  uint8_t *buffer = new uint8_t[pktSize];
  //Get the vehicle ID
  uint8_t V_index = socket->GetNode ()->GetId ();
  //Run agreement function to calculate the new clock value
  RunVoting(V_index,pktCount);
  /******Filling the Payload*******/
  //the first byte for the BSM sequence number
  buffer[0] = Packet_Number[V_index] % 127;
  //Four bytes for the ID
  buffer[1] = 10;
  buffer[2] = 1;
  buffer[3] = 1;
  buffer[4] = V_index + 1;
  //Positioning information
  //Spread Lat and Long over four bytes each
  buffer[7] = Lat[V_index] >> 24;
  buffer[8] = Lat[V_index] >> 16;
  buffer[9] = Lat[V_index] >> 8;
  buffer[10] = Lat[V_index] & 0xFF;
```

```
32    buffer[11] = Long[V_index] >> 24;
33    buffer[12] = Long[V_index] >> 16;
34    buffer[13] = Long[V_index] >> 8;
35    buffer[14] = Long[V_index] & 0xFF;
36    //Increament packets number by 1
37    Packet_Number[V_index]++;
38    Lat[V_index] = Lat[V_index] + 110;
39    /******End Filling the Payload*******/
40    //Set the Dsecond value
41    buffer[5] = Dsecond[V_index] >> 8;
42    buffer[6] = Dsecond[V_index] & 0xFF;
43    //Send the BSM
44    socket->Send (Create<Packet> (buffer, pktSize));
45    //Schedule this process to run after the timer expires
46    Simulator::Schedule (pktInterval, &GenerateBSM, socket, pktSize,
          pktCount - 1, pktInterval);
47  }
```

C.3 BSM RECEPTION

Listing C.3: BSM Reception Code

```
1  /* BSM receiving function
2   * This function is responsible for receiving the BSMs fro
       surrounding neighbors.
3   * It continuously running the receiving socket.
4   * After receiving a BSM, it records the receiving time for this
       BSM.
5   * Extract vehicle ID and Dsecond from the received BSM.
6   * Record the extracted information into the synchronization table
       combined with receiving time of this BSM.
7   * Finally it marks the vehicles moving in the same heading to
       consider them in the agreement process.
8   */
9  void ReceiveBSM (Ptr<Socket> socket)
10 {
11     while (Ptr<Packet> packet = socket->Recv ())
12     {
13     uint64_t ms = Simulator::Now().GetMilliSeconds();
14     //Copy BSM data into a temporary buffer
15     uint8_t *buffer = new uint8_t[packet->GetSize()];
16     packet->CopyData (buffer, packet->GetSize());
17     //Get the Receiver index
18     uint8_t  Rec_Index= socket->GetNode ()->GetId ();
19     //Read BSM data
20     //Get the Sender Index
21     uint8_t  Sender_Index = buffer[4] - 1;
22     //Record the receiving time of this BSM in the synchronization
          table
23     SyncronizationTable[Rec_Index][Sender_Index][0] = ms;
24     //Extract Dsecond from the Received BSM and save it in the
          synchronization table
25     uint16_t temp_Dsecond = buffer[5] << 8 | buffer[6];
26     SyncronizationTable[Rec_Index][Sender_Index][1] = temp_Dsecond;
27     //Calculate the time offset between Transmitter and Receiver
          and save it in the synchronization table
```

```
28      SyncronizationTable[Rec_Index][Sender_Index][2] = Dsecond[
            Rec_Index] - temp_Dsecond;
29      //Check for vehicle heading in order to elminate vehicles
30      //in other headings
31      if((cvmm[Rec_Index]->GetVelocity().x / -1 < 0
32      && cvmm[Sender_Index]->GetVelocity().x / -1 < 0)
33      || (cvmm[Rec_Index]->GetVelocity().x / -1 > 0
34      && cvmm[Sender_Index]->GetVelocity().x / -1 > 0))
35      {
36        //Check for vehicles ahead or behind and mark them for future
              use
37        if(cvmm[Sender_Index]->GetPosition().x > cvmm[Rec_Index]->
              GetPosition().x)
38          SyncronizationTable[Rec_Index][Sender_Index][3] = 1;
39        else
40          SyncronizationTable[Rec_Index][Sender_Index][3] = 2;
41      }
42      else
43      {
44        //Mark vehicles moving on another direction
45        SyncronizationTable[Rec_Index][Sender_Index][3] = -1;
46      }
47      }
48  }
```

C.4 AGREEMENT PROCESS

Listing C.4: Agreement Process Code

```
/* Agreement process
 * This process is responsible for executing the areement process,
 * and set the new value to the local clock value.
 * It first extract the saved timestamps from the synchronization
    table and Sort them.
 * Run the MSR proces.
 * Finally check if the agreement is acheived or not.
 */
void RunVoting(uint8_t V_index, uint32_t pktCount)
{
  //Local variables and initialization
  uint8_t ActiveNode = 0;
  uint8_t NotActiveNode = 0;
  StopConvergence = true;
  uint16_t OldDsecond = 0;
  int removedNodes = 0;
  double TimeAve = 0;
  double D2 = 0, D3 = 0;
  uint32_t MaxD2 = 0 , MaxD3 = 0, MaxD4 = 0;
  uint64_t ms = Simulator::Now().GetMilliSeconds();
  StopFlag[V_index] = true;
  //Copying the synchronization table (timestamps entri)
  //into the votin multiset
    for(int i = 0; i < NodeCount; i++)
    Time_Table[V_index][i] = SyncronizationTable[V_index][i][1];
    Time_Table[V_index][V_index] = Dsecond[V_index] ;
    OldDsecond = Time_Table[V_index][V_index];
  //Check vehicles within the predeifned clock predeifned tolerance
    for(int m = 0; m < NodeCount; m++)
    {
    if(Time_Table[V_index][m] != 0)
        {
          if(abs(OldDsecond - Time_Table[V_index][m]) >
              Global_Tolerance)
```

```
33          StopFlag[V_index] = false;
34          }
35      }
36    //Sorting time table
37    for(int m = 1; m <= NodeCount; m++)
38    {
39      for(int n = 0; n < NodeCount - m; n++)
40      {
41        if(Time_Table[V_index][n] > Time_Table[V_index][n+1])
42        {
43            uint32_t temp = Time_Table[V_index][n];
44          Time_Table[V_index][n] = Time_Table[V_index][n+1];
45          Time_Table[V_index][n+1] = temp;
46        }
47      }
48    }
49    //Count number of neighbours vehicles in range
50    for(int j = 0; j < NodeCount; j++)
51    {
52      if(ActiveNodesArround[V_index][j] != 0)
53        ActiveNode++;
54      else
55        NotActiveNode++;
56    }
57    CountActiveNodes[V_index] = ActiveNode;
58    if(ActiveNode >= 4)
59    {
60      //Calculate number of nodes that will be reduced from the
          voting multiset
61      removedNodes = ceil(ReductionPercentage * ((double)ActiveNode
          /100.0));
62      RmovedNodesArray[V_index] = removedNodes;
63      ////////////////////////////////
64      //Increament iteration counter for each vehicle
65      //To calculate how many round taken to acheive the agreement
66      if(!StopFlag[V_index])
67        NodeSyncIteration[V_index]++;
```

```
68      ////////////////////////////////
69      //Run the Fault Tolerant Average (FTA) selection function
70      if(Algorithm == 1)
71      {
72        int countedvalue = 0;
73        for(int j = NotActiveNode + removedNodes; j < NodeCount -
              removedNodes; j++)
74        {
75            TimeAve += Time_Table[V_index][j];
76          countedvalue++;
77        }
78        TimeAve = round(TimeAve / countedvalue);
79      }
80      ////////////////////////////////
81      //Run the Fault Tolerant Midpoint (FTM) selection function
82      else if(Algorithm == 2)
83      {
84          TimeAve = Time_Table[V_index][NotActiveNode + removedNodes]
85                + Time_Table[V_index][NodeCount - removedNodes - 1];
86        TimeAve = round(TimeAve / 2.0);
87      }
88      ////////////////////////////////
89      //Run the Median/MidPoint selection function
90      else if(Algorithm == 3)
91      {
92          if(ActiveNode % 2 == 0)
93            TimeAve = round((double)Time_Table[V_index][NotActiveNode
                  + ActiveNode/2 - 1] / 2.0
94                  + (double)Time_Table[V_index][NotActiveNode +
                        ActiveNode/2] / 2.0);
95        else
96            TimeAve = Time_Table[V_index][NotActiveNode + (ActiveNode
                  + 1)/2];
97      }
98      //Store the new voted value as the current Dsecond
99      Dsecond[V_index] = TimeAve;
100   }
```

```
101   else
102   {
103     NodeSyncIteration[V_index]++;
104   }
105   //Dsecond value must not exceed 60 sec.
106   Dsecond[V_index] = Dsecond[V_index] % 60000;
107   ////////////////////////////////
108     //int ConvergedNodes = 0;
109     StopFlag[V_index] = true;
110   //Count number of synchronized vehicles in each vehicle
        neighbours
111   for(int m = NotActiveNode + removedNodes; m < NodeCount -
        removedNodes; m++)
112   {
113     if(abs(Dsecond[V_index] - Time_Table[V_index][m]) >
          Global_Tolerance)
114     {
115        StopFlag[V_index] = false;
116     }
117   }
118   //If vehicles are synchronized then stop the synchronization
        process
119   for(int j = 0; j < NodeCount; j++)
120   {
121     if(!StopFlag[j])
122     {
123       StopConvergence = false;
124       break;
125     }
126   }
127   //Print investigation messages in order to check synchronization
        process
128   if(PrintMessages)
129   {
130     int vCount = 0;
131     for(int m = 0; m < NodeCount; m++)
132     {
```

```
133       if(ActiveNodesArround[V_index][m] != 0)
134         vCount++;
135     }
136     std::cout <<"Node " << V_index + 0 << " Dsecond = " <<
           OldDsecond << " Active nodes = " << vCount << "\n";
137     printTimeTable(V_index);
138     std::cout <<"Node " << V_index + 0 << " Dsecond = " << Dsecond[
           V_index] << "\n";
139   }
140   //Calculte the total diameter between vehicles's clock
141   if(V_index == 0)
142   {
143     Min_Dsecond = Dsecond[0];
144     Max_Dsecond = Dsecond[0];
145     for(int i = 1; i < NodeCount; i++)
146     {
147         if(Min_Dsecond > Dsecond[i])
148           Min_Dsecond = Dsecond[i];
149       if(Max_Dsecond < Dsecond[i])
150           Max_Dsecond = Dsecond[i];
151     }
152     if(PrintMessages)
153         std::cout <<"Difference = " << Max_Dsecond - Min_Dsecond <<
              "\n";
154   }
155   //Stop synchronization process if number of iterations excceds
        300 round
156   //Or synchronization is acheived
157   if((StopConvergence && V_index == 0) || NodeSyncIteration[V_index
        ] >= 300)
158   {
159     //Check if the number of iterations exceeds the 300 rounds
160     if(NodeSyncIteration[V_index] >= 300)
161         TimeSeed = MaxSeed + 1;
162     //Stop the current convergence experiment
163     StopConvergenceFn();
164     //Reset all variables in order to run new experiment
```

```
165       //with different random numbers
166       Reset();
167     }
168     ////////////////////////////////
169     //Clear voting multiset
170       for(int z = 0; z < NodeCount; z++)
171       {
172       Time_Table[V_index][z] = 0;
173       }
174 }
```