SYSTEMATIC DESIGN OF A BLOCKCHAIN-BASED CONTINUOUS AUTHENTICATION MECHANISM TOWARDS VULNERABLE ROAD USER (VRU) SAFETY

A Thesis Presented

By

James Edward Bednar

Submitted to the Francis College of Engineering

University of Massachusetts Lowell

In Partial Fulfillment of the Requirements

for the Degree of Master of Science

Computer Engineering

February 2024

SYSTEMATIC DESIGN OF A BLOCKCHAIN-BASED CONTINUOUS AUTHENTICATION MECHANISM TOWARDS VULNERABLE ROAD USER (VRU) SAFETY

BY

JAMES EDWARD BEDNAR
B.S. WENTWORTH INSTITUTE OF TECHNOLOGY (2021)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
COMPUTER ENGINEERING
UNIVERSITY OF MASSACHUSSETS LOWELL

Signature of Author: [illegible] **Date:** 11/20/2023

Signature of Thesis Chair: [illegible]
Dr. Chunxiao Chigan

Signatures of Other Thesis Committee Members:

Committee Member Signature: [illegible]
Dr. Kavitha Chandra

Committee Member Signature: [illegible]
Dr. David Claudio

SYSTEMATIC DESIGN OF A BLOCKCHAIN-BASED CONTINUOUS AUTHENTICATION MECHANISM TOWARDS VULNERABLE ROAD USER (VRU) SAFETY

BY
JAMES EDWARD BEDNAR

ABSTRACT OF A THESIS SUBMITTED TO THE FACULTY OF THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

FOR THE DEGREE OF
MASTER OF SCIENCE IN COMPUTER ENGINEERING
UNIVERSITY OF MASSACHUSSETS LOWELL
FEBRUARY 2024

Thesis Supervisor: Chunxiao Chigan, Ph.D.
Professor, Department of Electrical and Computer Engineering

Abstract

The recent proposal of next-generation connected and autonomous vehicle (CAV) transportation systems provides an opportunity to evaluate how collaboration between connected vehicles (CV), autonomous vehicles (AV), and roadside unit (RSU) devices can promote improved roadside safety for vulnerable road users (VRU) such as bicyclists and pedestrians. There is a real need for engineering solutions which advance and promote VRU safety, as highlighted by the modern phenomena of fatal AV-to-VRU collisions. The development of a VRU collision avoidance (VCA) system which aids faulty AVs in collision avoidance through collaboration between vehicles and traffic intersection-based RSU devices is of particular interest. Further, the development of a continuous authentication mechanism for RSU devices is of critical importance, as such devices represent valuable targets for cybersecurity threat actors due to their centralized role in the proposed VCA system. This work applies a systematic approach to explore the viability of the proposed VCA system and promote its security through the development of a blockchain-based continuous authentication mechanism. We provide formal definitions for key system components and propose a VCA system reference architecture which specifies a baseline for the development of further technical solutions. We then examine the viability of Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) in the context of vehicular networking to determine their viability for use in a real-time emergency message application. We conclude with the proposal of a blockchain-based continuous authentication mechanism which leverages a smart contract running on a private deployment of the Ethereum blockchain to provide ongoing validation of RSU device operation. These efforts combine to provide a systematic exploration of the problem space to promote improved VRU roadside safety.

Acknowledgments

I would like to express my gratitude to my thesis advisor Dr. Chunxiao (Tricia) Chigan for the opportunity to pursue research in the Secure and Reliable Networking Research Lab during my time at UMass Lowell. Her willingness to help support the exploration of my disparate interests in computer network security, systems engineering, and societally relevant engineering problems afforded me the ability to pursue the research efforts captured within this document.

Thank you to Dr. Kavitha Chandra and Dr. David Claudio for lending their time and talents to my thesis committee. I greatly appreciate the opportunity to have interdisciplinary perspectives present on my thesis committee, as I strongly believe that collaboration between engineering disciplines provides us with the greatest chance to address the most pressing technical challenges of today. Additionally, thank you to Dr. Rolando Herrero and Dr. David Claudio for their technical consultation on IoT system design patterns and queuing delay models, respectively.

Finally, I would like to thank my friends and family for their support throughout this process. To my brother and sister, Brandon and Kayleigh: I am extremely proud of the success that both of you have found in your careers, and I am happy to be the third-best engineer in the family. To my parents: Thank you for giving me the opportunity to move halfway across the country to pursue my goals, even if they tend to change over time. To my fiancée and better half, Jocelyn: Thank you for always supporting me in my academic pursuits, even if they sometimes get in the way of date night. You got all my love.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

100GbE – 100 Gigabit Ethernet
3GPP – 3rd Generation Partnership Project
4G LTE – 4th Generation Long-Term Evolution
5G NR – 5th Generation Near Radio
ACK – Acknowledgement
ADS – Automated Driving System
AMD – Advanced Micro Devices
AU – Application Unit
AV – Autonomous Vehicle
AV-to-VRU – Autonomous Vehicle to Vulnerable Road User
CA – Certificate Authority
CAM – Cooperative Awareness Message
CAN – Controller Area Network
CAV – Connected and Autonomous Vehicle
CLI – Command Line Interface
CoAP – Constrained Application Protocol
CON – Confirmable
CRL – Certificate Revocation List
CV – Connected Vehicle
C-V2X – Cellular Vehicle-to-Everything
DApp – Distributed Application
DBMS – Database Management System
DDT – Dynamic Driving Task
DENM – Decentralized Environmental Notification Message
DoS – Denial of Service
DoT –Department of Transportation
DSRC – Dedicated Short Range Communication
EDA – Event-Driven Architecture
ETSI – European Telecommunications Standards Institute
EVM – Ethereum Virtual Machine
FCC – Federal Communication Commission
FSD – Full Self-Driving
GB – Gigabyte
IEEE – Institute of Electrical and Electronics Engineers
IETF – Internet Engineering Task Force
IoT – Internet of Things
IoV – Internet of Vehicles
IP – Internet Protocol
IPv4 – Internet Protocol version 4
IPv6 – Internet Protocol version 6
ISP – Internet Service Provider
ITS – Intelligent Transportation System
LTS – Long Term Support
M2M – Machine-to-Machine

Mbps – Megabits per second
MID – Message Identifier
MITM – Man-in-the-Middle
ML – Machine Learning
MQTT – Message Queue Telemetry Transport
MRC – Minimal Risk Condition
mTLS – Mutual Transport Layer Security
NHTSA – National Highway Traffic Safety Administration
NON – Non-Confirmable
OASIS – Organization for the Advancement of Structured Information Systems
OBU – On-Board Unit
ODD – Operational Design Domain
OEDR – Object and Event Detection and Response
OTA – Over-the-Air
P2P – Peer-to-Peer
PKI – Public Key Infrastructure
PoA – Proof of Authority
QoS – Quality of Service
REST – Representational State Transfer
RFC – Request for Comment
RMV – Registry of Motor Vehicles
RST – Reset
RSU – Roadside Unit
RTT – Round-Trip Time
SAE – Society of Automotive Engineers
SSP – Secure Service Provider
TA – Trusted Authority
tc – Traffic Control Utility
TCP – Transmission Control Protocol
TLS – Transport Layer Security
TTP – Trusted Third Party
UDP – User Datagram Protocol
URI – Uniform Resource Identifier
V2I – Vehicle-to-Infrastructure
V2V – Vehicle-to-Vehicle
V2X – Vehicle-to-Everything
VA – Validation Authority
VANET – Vehicular Ad-Hoc Network
VCA – Vulnerable Road User Collision Avoidance
VIN – Vehicle Identification Number
VLAN – Virtual Local Area Network
VM – Virtual Machine
VRU – Vulnerable Road User
WAVE – Wireless Access in Vehicular Environments
WLAN – Wireless Local Area Network

Chapter 1 – Introduction

In this chapter we outline our motivation for the design of a vulnerable road user (VRU) collision avoidance (VCA) system. We provide an overview of continuous authentication and highlight the utility of blockchain technology for continuous authentication. We then define a cybersecurity threat model, which guides the design decisions made throughout this thesis. We conclude with an overview of the technical contributions of later chapters.

1.1 Motivation for Vulnerable Road User (VRU) Collision Avoidance System

Current approaches to the design of connected and autonomous vehicle (CAV) transportation systems fail to prioritize the safety of VRUs such as pedestrians and bicyclists. There is an immediate need for engineering solutions which promote VRU roadside safety, as highlighted by the first fatality of a pedestrian due to an autonomous vehicle (AV) to VRU (AV-to-VRU) collision in March 2018 [1] – [3]. While fatal instances of AV-to-VRU collision remain rare, the recent recall of the Tesla Full Self-Driving (FSD) Beta by the National Highway Traffic Safety Administration (NHTSA) due to faulty operating behaviors such as traveling through traffic intersections during a stale yellow traffic light underlines the need for intersection management solutions which promote VRU roadside safety [4], [5].

Given this context, a VCA system which aggregates the sensor data readouts of nearby vehicles and delivers fleet-wide emergency messages upon VRU detection in the roadway would represent a meaningful contribution to VRU roadside safety. We propose the development of a VCA system capable of sensor data readout aggregation through a traffic intersection-based roadside unit (RSU) emergency message application. This VCA system would exist within the broader scope of a metropolitan-area CAV transportation system and consist of entities including nearby connected vehicles (CV), AVs, and a traffic intersection-based RSU device, alongside any

VRUs in the local area. Given the centralized role of the RSU device, we further propose the use of a blockchain-based continuous authentication mechanism to promote security through ongoing evaluation of RSU device operation. We envision the proposed VCA system as a system-of-systems engineering solution for the problem of faulty AV-to-VRU collisions. We define a faulty AV as one which fails to detect a VRU in the roadway through onboard sensors, thus leading to a failure of the in-vehicle safety system and a potentially fatal AV-to-VRU collision.

Overall, this work seeks to explore whether current technologies can enable the creation of a system which facilitates cooperative communication between CVs, AVs, and traffic-intersection-based RSU devices to promote improved safety for VRUs seeking freedom of movement near active roadways. The notional traffic intersection diagram presented below in Figure 1 provides an overview of a potential AV-to-VRU collision scenario. Given the presence of CVs, AVs, and traffic intersection-based RSU devices within the proposed VCA system, we seek to determine answers to the following research questions:

1. How can CVs, AVs, and traffic intersection-based RSU devices cooperatively exchange messages such that faulty AVs become aware of VRUs in the roadway and manage to prevent a potentially fatal collision?
2. How can we leverage emerging technologies such as the Internet of Things (IoT) paradigm, blockchain-based distributed systems, and continuous authentication to enable the creation of a secure system capable of mitigating AV-to-VRU collisions?
3. How can we enable the creation of a resilient system that can continue to operate under partial system failure conditions such as the deauthentication of a traffic intersection-based RSU device or the presence of known cyberattacks?

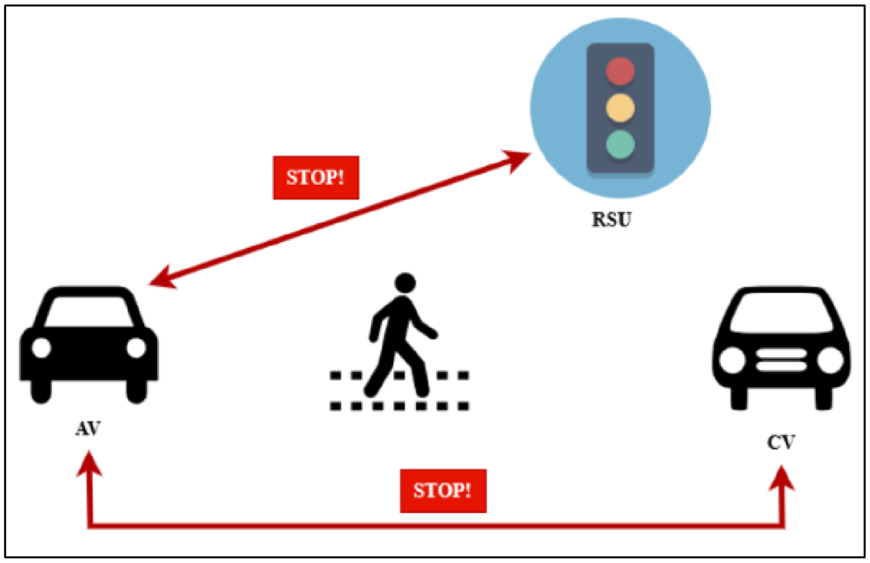


Figure 1: CV and RSU Device Collaborate to Notify Faulty AV of VRU

Throughout this work we apply systems thinking, which seeks to enable improved insight into the construction and performance of complex systems through evaluation of interactions between components, dependent systems, and the environment rather than the decomposition of a given system of interest into its requisite components [6]. As such, we apply a systematic approach to the design and analysis of the proposed VCA system through the definition of a VCA system reference architecture, the development of a baseline CAV emergency message application, and the creation of a blockchain-based continuous authentication mechanism. These related technical contributions combine to form the theoretical basis of a viable engineering solution for the problem of faulty AV-to-VRU collisions.

1.2 Overview of Continuous Authentication

Authentication represents the process through which a user or device proves its identity within the context of a computer system. There exist five standard authentication factors, which include something you know (e.g., password), something you have (e.g., authentication token), something you are (e.g., user biometrics or device fingerprint), somewhere you are (e.g., geographic location), and something you do (e.g., user gesture or device action) [7]. Traditional static authentication involves a user providing one or more authentication factors to gain access to a computer system at the start of a communication session. If a user successfully authenticates into a computer system, then they may gain access to system resources through a separate authorization process, which falls outside the scope of this discussion. Modern computer systems which leverage the IoT or cyber-physical system paradigms often require the ongoing exchange of real-time data without the intervention of a human user. Such systems often require ongoing validation of devices, and therefore require a solution which provides continuous evaluation of authentication status throughout the duration of a communication session [8].

Baig and Eskeland define continuous authentication as an authentication mechanism which continuously monitors user actions throughout the duration of a communication session to determine whether the given entity represents a legitimate user of the system. If it appears that the user is no longer deemed legitimate, then the given entity is deauthenticated from the communication session in an automated manner. Continuous authentication mechanisms are often capable of performing passive reauthentication of entities without external notifications or the need for manual intervention [9]. As such, the implementation of a continuous authentication mechanism would offer security advantages for the proposed VCA system, as the traffic intersection-based RSU device represents an attractive target for cybersecurity threat actors due to its centralized role in the operation of the system.

Gonzalez-Manzano et al. outline a five-step process for the design of continuous authentication mechanisms which includes [10]:

1. Scenario Selection: What scenario will the continuous authentication mechanism be applied to?
2. Device Selection: Which device type(s) will collect authentication data and participate in the process of continuous authentication?
3. Feature Selection: What mechanism(s) will be used to enforce the continuous authentication process?
4. Authentication Enforcement: How will the authentication and deauthentication decisions be enforced by the continuous authentication mechanism?
5. Evaluation Analysis: Do empirical evaluations with a sample dataset show that the proposed mechanism is effective? If so, consider implementation in a production environment. If not, iterate on the proposed continuous authentication mechanism until it is ready for the production environment.

We apply this process to the design of a continuous authentication mechanism for traffic intersection-based RSU devices, as detailed below in Chapter 4.

While many implementations of continuous authentication leverage machine learning (ML) techniques [9] the development of blockchain-based continuous authentication mechanisms also represents an active research area [8]. The current lack of real-world datasets for the proposed VCA system makes the application of ML techniques impractical. As such, we propose an alternative approach which leverages a blockchain-based smart contract to enable the continuous authentication of traffic intersection-based RSU devices. We believe this approach to continuous authentication represents a meaningful contribution to the literature, as presented in detail below in Chapter 4.

1.3 Definition of Cybersecurity Threat Model

Given the cyber-physical nature of the proposed VCA system, the success of our systematic approach requires knowledge of relevant cybersecurity threats to guide our effort. As such, we define the below cybersecurity threat model for the VCA system and its related technical solutions. We start our cybersecurity threat model with the definition of our cybersecurity design goals, followed by an overview of known cybersecurity attacks applicable to vehicular networks.

While CAV transportation systems represent an emerging technology, there exist many known cybersecurity attack vectors against CVs, AVs, and related smart infrastructure components such as RSU devices as highlighted by Chowdhury et al. in [11]. We seek to focus our cybersecurity threat model on the potential damage an external threat actor could cause if they were to gain unauthorized access to the traffic intersection-based RSU device. While we can reasonably assume the availability of the traffic intersection-based RSU device, we cannot assume its integrity due in large part to the centralized role it plays in each individual VCA system instance. As such, the RSU device represents a single point of failure for the VCA system without proper remediation. We therefore seek to ensure that the RSU device is reputable, which would allow us to ignore illegitimate messages through a process such as RSU device identity verification. We further seek to guarantee the confidentiality, integrity, authenticity, and non-repudiation of RSU device messages through the application of technical solutions such as industry-standard cryptography.

We recognize that cybersecurity threat actors widely vary in terms of motivation, resources, and technical skill. Cybersecurity threat actors can range in sophistication from nation-state sponsored advanced persistent threats to amateur script kiddies. We therefore opt not to define a specific type of cybersecurity threat actor and instead focus our efforts on the remediation of known cybersecurity attacks found within vehicular networks. If we can successfully remediate the risks posed by these attacks, then we can reasonably conclude that we have mitigated most of the risk presented by an outside cybersecurity threat actor. While this approach does not account for the remediation of future attack vectors through the discovery of zero-day vulnerabilities, such efforts fall within the domain of defensive cyber operations and therefore lie outside the scope of this work.

Given this context, we seek to mitigate those cybersecurity attacks which have already been demonstrated in the context of real-world vehicular systems from the computer network security perspective. We are particularly interested in technical solutions which help to mitigate attacks on authentication systems, which can lead to unacceptably high levels of delay in time-sensitive vehicular network communications as demonstrated by Talavera et al. in [12]. The known cybersecurity attacks which we seek to mitigate include denial of service (DoS) attacks, impersonation attacks, replay attacks, data falsification attacks, data tampering attacks, eavesdropping attacks, and password and cryptographic key attacks.

DoS attacks seek to interfere with the normal operation of a communication system through the flooding of useless messages, which denies access to services for legitimate actors on the network. Impersonation attacks involve threat actors sending fake messages to gain access to the identity of a legitimate entity on the network. Replay attacks involve threat actors recording and retransmitting valid packets after they have already been transmitted to gain illegitimate access to network services. Data falsification attacks involve the transmission of illegitimate messages, which can have serious impacts for safety-critical systems such as the proposed VCA system. Similarly, data tampering attacks involve the unapproved modification of data by a threat actor. Eavesdropping attacks may involve a threat actor listening in on unencrypted network traffic but may also take the form of man-in-the-middle (MITM) attacks. Finally, password and cryptographic key attacks seek to either gain access to a plaintext password or use stolen cryptographic secrets to gain unauthorized access to network resources. For a more in-depth discussion we direct the reader to the work of Sun et al. which provides an in-depth overview of these cybersecurity attacks on vehicular networks in the context of CAV transportation systems [13]. We provide a summary of these attack types and their impact on the core cybersecurity goals of confidentiality, integrity, availability, authenticity, and non-repudiation below in Table 1.

Table 1: Known Cybersecurity Attacks for Vehicular Networks

Cybersecurity Attack	**Example Attack Method(s)**	**Impact Area(s)**
Denial of Service	Packet flooding, Resource exhaustion	Availability
Impersonation	Stolen credentials, Spoofing	Availability, Integrity
Replay	Packet capture with retransmission	Integrity
Data Falsification	Broadcast of false data packets	Authenticity, Integrity
Data Tampering	Database modification, Database injection	Non-repudiation, Integrity
Eavesdropping	Packet capture of traffic, Stolen credentials	Confidentiality
Password and Key	Password cracking, Stolen credentials	Authenticity, Integrity

We seek to ensure continuous authentication for the traffic intersection-based RSU device alongside a guarantee of confidentiality, integrity, authenticity, and non-repudiation for messages sent by CVs, AVs, and RSU devices present within the system. This means we must determine technical solutions for the implementation of cybersecurity controls such as mutual device authentication, transport layer security (TLS), and continuous authentication of RSU devices within the design of the proposed VCA system. We propose potential solutions for these cybersecurity controls in subsequent chapters of this thesis. In particular, the end of Chapter 3 proposes the implementation of mutual device authentication and TLS for the proof-of-concept CAV emergency message application as future work, while Chapter 4 presents work towards the development of a blockchain-based continuous authentication mechanism. We provide additional insight into the contributions of subsequent chapters in the following section.

1.4 Technical Contributions

This section seeks to provide an overview of the technical contributions of later chapters. We provide an overview of the work towards technical solutions for the VCA system captured in Chapter 2, Chapter 3, and Chapter 4. This overview provides the reader with high-level information related to the technical contributions of each chapter, which aids in understanding the systematic nature of this work.

1.4.1 Contribution of Chapter 2 –VCA System Reference Architecture

Chapter 2 provides an overview of CAV transportation systems and related engineering standards, which enables the systematic development of a VCA system reference architecture. We provide definitions for key components and provide a set of reasonable assumptions which bind our solution space. We conclude with a formal definition of the VCA system reference architecture, which serves as a baseline for the development of relevant technical solutions seen in later chapters.

1.4.2 Contribution of Chapter 3 – CAV Emergency Message Application

Chapter 3 presents a comparative analysis of Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) for use in a real-time CAV emergency message application. MQTT and CoAP represent two of the most popular application layer protocols for machine-to-machine (M2M) and IoT system design. We provide a brief overview of each protocol and test their performance under moderate network loss conditions to determine which provides greater quality of service for real-time message exchange. We then implement a baseline CAV emergency message application, which leverages the more performant protocol to demonstrate the viability of such a solution towards improved VRU roadside safety. We recognize the need for mutual authentication alongside TLS, and therefore propose the implementation of mutual transport layer security (mTLS) as future work.

1.4.3 Contribution of Chapter 4 – Blockchain for Continuous Authentication

Chapter 4 outlines a proposal for the application of blockchain technology in support of continuous authentication for vehicular networks. We introduce the fundamentals of blockchain technology to provide context for the reader. We then provide further information on the Ethereum blockchain and outline the design of the solution. We propose the use of a private deployment of the Ethereum blockchain which leverages a smart contract to enable continuous authentication of traffic intersection-based RSU devices within in the proposed VCA system. We conclude with an execution example of the resulting blockchain-based continuous authentication mechanism for use within the proposed VCA system.

1.5 Authorship

This thesis is based on several interconnected research efforts. Some of these research efforts benefit from the input of external collaborators. The work of Chapter 3 was conducted independently with technical consultation from Dr. Rolando Herrero and Dr. David Claudio on IoT system design patterns and queueing delay models, respectively. The work of Chapter 4 was conducted independently with technical guidance from Dr. Chunxiao Chigan. While the technical contributions of this work benefit from the expertise provided by these faculty members, all the research presented below was conducted independently by the author.

Chapter 2 – VCA System Reference Architecture

In this chapter we introduce background information relevant to the design of a VCA system for use in a next-generation CAV transportation system. We provide an overview of current proposals for CAV transportation system architectures and outline key communication paradigms for vehicular networks. We then provide insight into relevant engineering standards through a comparison of cellular vehicle-to-everything (C-V2X) and Dedicated Short Range Communication (DSRC) technologies, followed by an overview of vehicular automation. We conclude with the definition of key system components, the declaration of assumptions which bind our solution space, and the definition of a VCA system reference architecture which serves as the foundation for the technical contributions of later chapters.

2.1 Introduction

The successful design of the proposed VCA system and its related technical solutions requires prior knowledge of CAV transportation systems. This section provides insight into CAV transportation systems, including an overview of core terminology and current wireless communication paradigms for vehicular networks. This information provides context for the systematic definition of a VCA system reference architecture, which we define below in Section 2.4.

2.1.1 Overview of CAV Transportation Systems

CAV transportation systems represent the confluence of CV and AV technology in a single, unified next-generation transportation system. We adopt the Department of Transportation (DoT) definition of CVs as automotive vehicles which possess onboard wireless communication equipment yet lack an autonomous driving capability [14]. Likewise, we adopt the DoT definition of AVs as automotive vehicles which possess onboard wireless communication equipment alongside some level of autonomous driving capability [15]. Given these definitions, we can define a CAV transportation system as one which consists of CVs, AVs, and smart infrastructure components such as traffic intersection-based RSU devices. CAV transportation systems represent a type of intelligent transportation system (ITS), which often seek to improve the safety and environmental sustainability of transportation systems through the application of emerging technology [16], [17]. One may also categorize CAV transportation systems under the umbrella of Internet of Vehicles (IoV), which seeks to apply IoT design patterns to vehicular networks [18].

CAV transportation systems represent a potential successor to vehicular ad-hoc networks (VANET), which historically represents one of the most active areas of vehicular networking research. Modern vehicular networks often leverage the VANET paradigm to enable point-to-point communication between vehicles and smart infrastructure devices. Vehicular networks typically facilitate wireless communication with three logical units, which include two in-vehicle units and one smart infrastructure unit. The in-vehicle units consist of the application unit (AU) and the on-board unit (OBU), while RSU devices represent the smart infrastructure unit.

The AU consists of software applications related to vehicular networking features and leverages the OBU for node-to-node communications. The OBU represents the in-vehicle hardware component responsible for coordination of wireless communication via onboard radio devices. This enables the OBU to manage the transmission and reception of network data for a single vehicle [19]. The RSU device serves as the roadside smart infrastructure component which facilitates wide-range wireless communication for vehicles in the local area. The RSU device may also host safety applications [20], such as the baseline CAV emergency message application presented below in Chapter 3.

The European CAR-2-CAR Communication Consortium standard applies these logical units to define three vehicular network communication domains: the in-vehicle domain, the ad-hoc domain, and the infrastructure domain. The in-vehicle domain consists of the OBU and AU to facilitate wireless communication for a single vehicle. The ad-hoc domain includes the OBUs of nearby vehicles and any nearby RSU devices to facilitate wireless ad-hoc communication between vehicular nodes and smart infrastructure nodes. The infrastructure domain focuses on RSU devices, which act as edge gateways to enable the local vehicular network to communicate with the broader Internet [21]. Figure 2 provides an overview of the in-vehicle domain, ad-hoc domain, and infrastructure domain in the context of a local vehicular network.

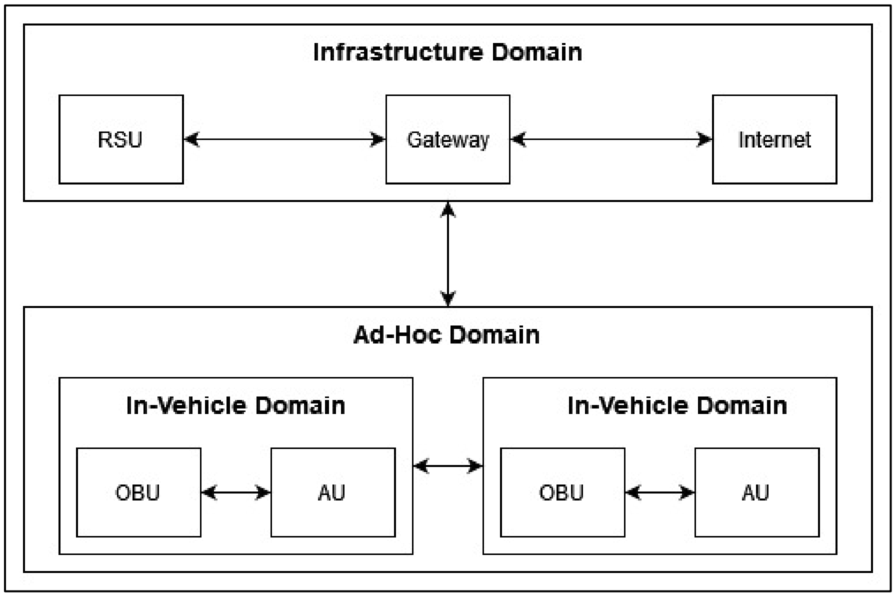


Figure 2: Block Diagram of In-Vehicle, Ad-Hoc, and Infrastructure Domains

2.1.2 V2V, V2I, and V2X Communication Paradigms for Vehicular Networks

The heterogeneous nature of CAV transportation systems requires context on the common types of wireless communication paradigms in vehicular networks. CV technology such as that of

VANETs seeks to enable vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), and vehicle-to-everything (V2X) communication to facilitate ubiquitous vehicular network connectivity [22], [23]. V2V communication allows network-enabled vehicles to directly communicate with each other via OBU devices. The leading proposals for real-world V2V applications would enable improved situational awareness for individual vehicles through networking of the sensor data readouts present throughout the vehicle fleet [24]. V2I communication allows network-enabled vehicles to communicate with smart infrastructure components such as traffic intersection-based RSU devices. Current proposals for the application of the V2I paradigm include the use of wireless communication to facilitate data exchange for real-time traffic monitoring, parking availability notifications, and electronic toll payments [25], [26]. V2X communication allows network-enabled vehicles to communicate with general network entities such as other network-enabled vehicles, smart infrastructure components, or consumer-grade electronic devices such as smartphones [27]. Proposals for V2X solutions tend to focus on the anticipated upside of benefits such as improved road safety and increased traffic efficiency [28], [29].

Given this context and recognizing the various terms for wireless communication paradigms present in the current literature, we opt to focus on the V2X communication paradigm throughout this work. We make this distinction with the understanding that the V2V and V2I paradigms both exist within the broader scope of the V2X paradigm. Figure 3 presents logical V2V, V2I, and V2X network connections in the context of a notional vehicular network. This notional vehicular network operates in ad-hoc mode, such that nodes may join and leave the network at will. This yields characteristics similar to a traditional peer-to-peer (P2P) network architecture.

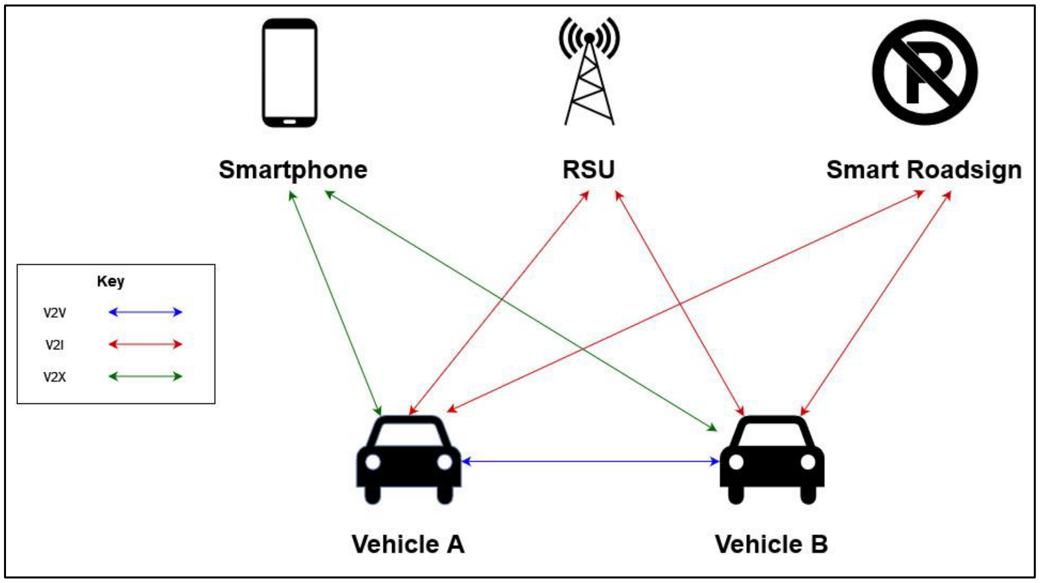


Figure 3: Overview of Logical V2V, V2I, and V2X Network Connections

While traditional vehicular network implementations such as VANETs tend to focus on ad-hoc mode, we opt to focus on network design for infrastructure mode within the scope of this work. Infrastructure mode views communication systems through the lens of the traditional Internet client-server model. We opt to focus on infrastructure mode for the proposed VCA system due in large part to the lower technical complexity of infrastructure mode network applications when compared to similar technical solutions which use ad-hoc mode. As such, we view the RSU device as an edge gateway which enables wireless network connectivity for CVs and AVs in the local area. The local CVs, AVs, and RSU devices form an access network, while the edge gateway provides a routable path to the broader core network. We assume an access-side wireless cellular connection and a core-side wired Ethernet connection, as seen below in Figure 4.

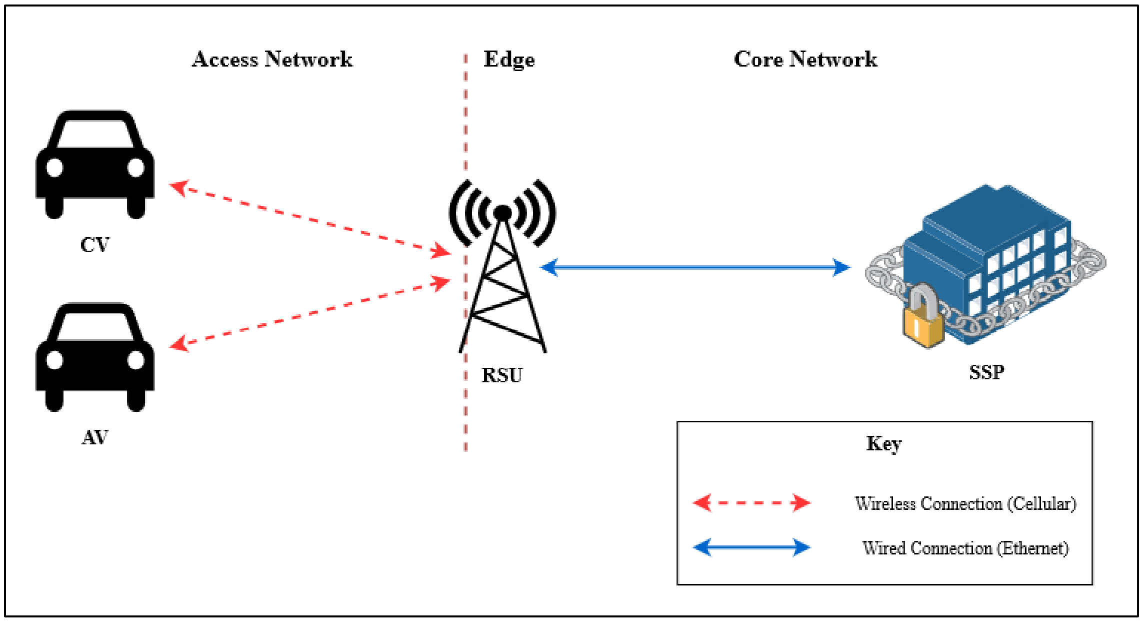


Figure 4: Access Network and Core Network for Proposed VCA System

This network design provides bidirectional access between entities on the access network and the core-side Secure Service Provider (SSP), which functions as a secure back-end datacenter in this context. We recognize Figure 4 abstracts away the intermediate network infrastructure that enables bidirectional communication between the RSU device gateway and the SSP. In practice network traffic may need to traverse multiple hops within the core network via Internet Service Provider (ISP) infrastructure to enable this bidirectional communication. While we only consider a single VCA system in this context, additional VCA system instances would exist at other traffic intersections throughout the metropolitan-area CAV transportation system. We provide further details on the specific functions of the SSP and other key VCA system components below in Section 2.3.1.

2.2 Engineering Standards for CAV Transportation Systems

The successful design and implementation of CAV transportation systems relies on the technical foundation of engineering standards. Engineering standards for mobile wireless communications and autonomous vehicle operation provide the basis for next-generation vehicular networks. As such, this section provides a brief discussion of wireless communication standards for vehicular networks alongside an overview of autonomous vehicle terminology. This information informs the systematic design efforts seen within the later chapters of this work.

2.2.1 Cellular V2X (C-V2X) and Direct Short-Range Communication (DSRC)

Modern vehicular networks include vehicles equipped with consumer-grade Wi-Fi access points, vehicles equipped with cellular data connections, and a select few vehicles equipped with DSRC radio devices [30]. DSRC represents a legacy wireless technology which leverages the Institute of Electrical and Electronics Engineers (IEEE) 802.11p protocol for wireless access in vehicular environments (WAVE). While DSRC provides a viable technical solution, it suffers from low industry adoption rates, with only a few supported vehicles available to consumers despite optimistic DoT market projections [31], [32]. DSRC enables short-range, line-of-sight operation for the V2V, V2I, and V2X communication through the creation of a device-to-device ad-hoc wireless local area network (WLAN) [33]. Despite these features, in 2020 the Federal Communication Commission (FCC) reallocated the wireless spectrum earmarked for ITS communication via DSRC to a combination of unlicensed 5.9 GHz Wi-Fi and ITS safety messages via cellular communication [34]. The FCC made this regulatory change due to low industry adoption rates, with the legacy DSRC wireless spectrum of 5.850 to 5.925 GHz divided into two separate spectrum allocations of 5.850 to 5.895 GHz and 5.895 to 5.925 GHz for unlicensed 5.9 GHz Wi-Fi and ITS safety messages via cellular communication, respectively [35]. The FCC further encouraged the automotive industry to transition to next-generation C-V2X wireless technology as soon as possible [36].

The 3rd Generation Partnership Project (3GPP) C-V2X technical standards represent a leading candidate for next-generation C-V2X wireless technology standardization, which has implications for future vehicular network performance. The current 3GPP C-V2X technical standards provide support for modern cellular technologies such as 4th Generation Long-Term Evolution (4G LTE) and 5th Generation Near Radio (5G NR) [37]. The 3GPP C-V2X Release 17 technical standard defines two modes of operation: C-V2X direct mode and C-V2X indirect mode. C-V2X direct mode enables device-to-device communication for V2V, V2I, and V2X messages in a manner similar to DSRC, while C-V2X indirect mode leverages cellular communication to enable the one-to-many exchange of V2X messages. Initial studies suggest that C-V2X provides a high level of reliability for real-time communications in areas with dense automotive traffic [38], which makes it an ideal candidate for use in ITS safety systems such as the proposed VCA system. We provide a brief comparison of the C-V2X and DSRC technical standards based on the prior work of Garcia et al. [39] and Kenney [40] below in Table 2.

Table 2: Comparison of DSRC and C-V2X Wireless Technology Standards

	DSRC	**C-V2X**	
		Direct	**Indirect**
Technical Standard	IEEE 802.11p	3GPP Release 14	3GPP Release 17
Initial Deployment	2015	2021	2024
Wireless Technology	WLAN	WLAN	Cellular
Wireless Band	5.9 GHz	5.9 GHz	5.9 GHz
Spectrum Allocation	5.850 to 5.925 GHz	5.895 to 5.925 GHz	

We assume the use of C-V2X indirect mode for the design of the VCA system reference architecture and its related technical solutions. We base this design choice on the combination of the effective deprecation of DSRC technology via FFC regulatory action and the ongoing 3GPP C-V2X standardization effort. Furthermore, alternative standardization efforts such as those of the European Telecommunications Standards Institute (ETSI) propose the use of C-V2X for cooperative awareness messages (CAM) and decentralized environmental notification messages (DENM) within vehicular network-based safety systems. CAMs provide periodic sensor data readouts from nearby vehicles, while DENMs facilitate emergency alerts for events such as VRU detection [41]. Given this context, we opt to further focus the design of technical solutions for emergency messages on the exchange of CAMs and DENMs within the scope of the VCA system, as highlighted below in Chapter 3.

2.2.2 SAE J3016 2021 Terminology Standard for Vehicular Automation

While CV technology seeks to leverage computational resources present throughout the vehicle fleet via wireless communication, AV technology seeks to leverage a combination of onboard computational resources and algorithms to enable autonomous driving features on a per-vehicle basis. Recent developments in the availability of vehicular automation features in consumer-grade automobiles has led to a growth in public interest surrounding the field of AV technology. However, the criteria that defines which modes of vehicle operation qualify as autonomous driving remain generally misunderstood by the average consumer [42]. For example, while the Tesla FSD Beta referenced above in Section 1.1 is sometimes referred to as self-driving, it does not meet industry-standard definitions for full AV operation [43]. The DoT suggests the use of the Society of Automotive Engineers (SAE) J3016 2021 terminology standard to define six levels of operation for vehicular automation and associated technical terminology [44]. We note that the SAE J3016 2021 standard does not define safety standards, and instead provides technical definitions and terminology to enable consistent discourse on topics related to vehicular automation technology. Table 3 below provides an overview of the six levels of operation for vehicle automation defined by SAE J3106 2021 as summarized by Koopman in [45].

Table 3: Overview of Vehicular Automation Levels from SAE J3016 2021 Standard

SAE Level	**High-Level Definition**	**Example Feature(s)**
Level 0	No Driving Automation	Blind Spot Warning, Static Cruise Control
Level 1	Driver Assistance	Lane Centering or Adaptive Cruise Control
Level 2	Partial Driver Automation	Lane Centering and Adaptive Cruise Control
Level 3	Conditional Driving Automation	Traffic Jam Management
Level 4	High Driving Automation	Local Geographic AV Operation
Level 5	Full Driving Automation	Global Geographic AV Operation

We further note that the example features highlighted in Table 3 for SAE Level 0 to SAE Level 3 only provide partial vehicle automation capabilities such as blind spot monitoring, adaptive cruise control, and traffic jam management. The example features of SAE Level 4 and SAE Level 5 provide capabilities typically associated with full AV operation at the local geographic (i.e., autonomous operation in a known local area) and global geographic (i.e., autonomous operation anywhere) levels, respectively. Many automotive manufactures offer consumer-grade vehicles which provide features up to SAE Level 2. Mercedes-Benz currently represents the only manufacturer with SAE Level 3 features available in select consumer-grade models [46], while companies such as Waymo and Cruise offer SAE Level 4 taxi services in select test markets such as San Fransico, California [47]. While Tesla FSD Beta features only qualifies as SAE Level 2, its inconsistent and unreliable operation highlights the potential dangers of faulty AV operation despite not technically qualifying as AV operation within the scope of these definitions. The disconnect between SAE Level 2 and SAE Level 3 classification highlights the potential ambiguity of SAE J3016 2021 terminology.

The SAE J3016 2021 standard also defines terminology for those seeking to discuss vehicular automation features in a systematic manner. The standard defines the operational design domain (ODD) as the environmental conditions which the automated driving system (ADS) is intended to operate within, such as weather, geographic area, and other relevant factors. Object and Event Detection and Response (OEDR) defines the aspect of the ADS responsible for environmental monitoring, alongside the planning and execution of responses to environmental stimuli. This includes the monitoring and response of the direct roadway environment, such as any other vehicles or VRUs in the local area. The presence of ADS-based features is usually indicative of a vehicle which possesses SAE Level 3 features or higher [45], [48].

For example, we can state that an AV operates in the ODD and performs OEDR via the ADS, which combine to define the dynamic driving task (DDT). If the ADS fails in the execution of the DDT, then fallback occurs to reach the minimal risk condition (MRC). Fallback represents the mechanism which allows the human driver to intervene if an unexpected event occurs, while the MRC represents the effort which seeks to safely stop the vehicle as part of a fallback operation or related maneuver [45], [48]. Given this terminology, we may redefine a faulty AV as one which fails to recognize an unexpected VRU in the roadway through standard OEDR measures, thus leading to a failure of the ADS in execution of the DDT and a potentially fatal instance of AV-to-AVR collision.

The complexity of the six levels of vehicular automation features and its related terminology underlines the fact the SAE J3016 2021 standard does not provide an intuitive framework for non-technical users. As such, Koopman proposes the creation of a set of vehicle automation modes which provide alternative terminology that may be more useful in standard practice. Koopman [49] proposes the definition of four vehicle operation modes which roughly map to the objectives of SAE J3016 2021: assistive mode, supervised mode, automated mode, and autonomous mode. Under the assistive mode, human drivers leverage modern safety features such as lane centering while maintaining complete control of the vehicle. The supervised mode involves an ADS-like system handling the DDT, while the human driver maintains situational awareness with their eyes on the road. The automated mode requires an ADS-like system to handle the DDT and driving safety features, which does not require the human driver to keep their eyes on the road. However, the human driver may need to intervene with notice in the event of an exceptional safety event, such as a vehicle breakdown. Finally, the autonomous mode represents the operation of a true AV, where the vehicle takes responsibility for the execution of the DDT and all safety features. We provide a breakdown of task responsibility between the driver and vehicle for these alternative vehicle operation modes below in Table 4.

Table 4: Breakdown of Task Responsibility for Vehicle Operation Modes

Operation Mode	**Breakdown of Task Responsibility**		
	Driving	**Driving Safety**	**Other Safety**
Assistive	Driver	Driver	Driver
Supervised	Vehicle	Driver	Driver
Automated	Vehicle	Vehicle	Driver
Autonomous	Vehicle	Vehicle	Vehicle

2.3 Components and Assumptions for VCA System Reference Architecture

The systematic design of the VCA system and its related technical solutions requires a reference architecture to serve as a baseline for further development. In this section we define key system components and declare a set of reasonable assumptions which enables the definition of a

VCA system reference architecture. Definitions for key system components aid in the creation of our reference architecture, while the declaration of reasonable assumptions allows us to bind the solution space so that we may focus on the core goal of improved VRU roadside safety.

2.3.1 Key System Components

The definition of a VCA system reference architecture requires the formal definition of the components which fall within the system scope. The VCA system and its related technical solutions shall consider the following components to be within scope: SSP, CV, AV, RSU, VRU, and cybersecurity threat actor. The definition presented below for the SSP is based on the prior work of Wang [50] and Kitiibwa [51] through the Secure and Reliable Networking Research Lab at the University of Massachusetts Lowell. The CV, AV, RSU, and VRU definitions presented below are based on the prior work of Gholamhosseinian and Seitz [52] on the topic of cooperative intersection management for heterogeneous CV transportation systems. We also provide a general definition of a cybersecurity threat actor based on industry-standard cybersecurity terminology.

2.3.1.1 Secure Service Provider

The SSP represents a private industry organization, government agency, or private-public consortium which coordinates the registration of entities present within a CAV transportation system. An entity may represent a CV, AV, or smart infrastructure component such as an RSU device. The SSP manages data such as vehicle identification numbers (VIN), registration information, performance metrics, and any other relevant data associated with a given entity. While the SSP is not necessarily part of the statewide Registry of Motor Vehicles (RMV) or similar government agency responsible for the administration of automotive registration, we envision the SSP to work closely with RMV-like agencies through shared access to a back-end database management system (DBMS). This DBMS could consist of a legacy relational database solution, or a next-generation distributed ledger-based solution built on top of blockchain technology, as proposed below in Chapter 4.

2.3.1.2 Connected Vehicles

CVs represent vehicles which possess V2X network connectivity but lack the ability to operate in an autonomous driving mode. CVs may also possess an onboard suite of sensors that help to monitor driving conditions. We consider vehicle automation features which fall within the scope of SAE Level 0 to SAE Level 3 to form the working definition of CVs for the purposes of this work. Further, we assume that the CVs present within the proposed VCA system use modern cellular standards such as those defined by the 3GPP C-V2X technical standards.

2.3.1.3 Autonomous Vehicles

AVs represent vehicles which possess V2X network connectivity and possess the ability to operate in an autonomous driving mode. We consider vehicle automation features which fall within the scope of SAE Level 4 or SAE Level 5 to form the working definition of AVs for the purposes of this work. As above, we assume that AVs present within the proposed VCA system use modern cellular standards such as those defined by the 3GPP C-V2X technical standards.

2.3.1.4 Roadside Unit

RSU devices represent the roadside network infrastructure responsible for processing sensor data readouts sent by nearby CVs and AVs. RSU devices send actuation messages on an as-needed basis to nearby CVs and AVs via an onboard emergency message application. These actuation messages vary based on the content of the received sensor data readouts. For example, if sensor data readouts from nearby vehicles indicate the presence of a VRU in the roadway, then the RSU device publishes an emergency stop message to stop nearby vehicles in order to avoid a collision. While RSU devices can be embedded in many different types of roadside infrastructure, we consider the special case of traffic intersection-based RSU device within the scope of the proposed VCA system.

2.3.1.5 Vulnerable Road User

VRUs include roadside users such as pedestrians and bicyclists who are unprotected and at high risk of injury in the event of a vehicular collision. If a VRU unexpectedly enters the roadway, then there exists an increased risk of AV-to-VRU collision in the case of faulty AV operation. We consider the unexpected entry of a VRU in the roadway to be a stimulus which functional CVs and AVs report to the traffic intersection-based RSU device via routine sensor data readouts. This allows us to focus on our systematic design approach for the VCA system as opposed to the low-level sensor and signal processing work which underpins such features.

2.3.1.6 Cybersecurity Threat Actor

Cybersecurity threat actors represent individuals or organizations which seek to tamper, interfere, or otherwise modify the normal operation of the CAV transportation system. Cybersecurity threat actors may be external to an organization or internal to an organization. Further, cybersecurity threat actors range in capability from highly determined and sophisticated parties with access to vast amounts of time, money, and technical skill such as advanced persistent threats to unsophisticated actors seeking to interfere with system operation through the use of open-source cybersecurity toolsets such as so-called script kiddies. We are not concerned with the exact nature of the cybersecurity threat actors which may target the proposed VCA system or the broader metropolitan-area CAV transportation system within the scope of this work. Rather, we define the notion of a cybersecurity threat actor to underline the fact that vehicular networks exist in an adversarial environment where potentially malicious individuals or organizations may seek to interfere with safe and reliable system operation. The possibility of interference by cybersecurity threat actors represents a reality of modern computer system design, and thus demands the proactive consideration of security controls throughout the system lifecycle.

2.3.2 Assumptions for Solution Space

The theoretical nature of the proposed VCA system requires a set of reasonable assumptions to bind our solution space. These assumptions allow us to focus our efforts on the pursuit of system-level engineering contributions within the scope of this work. Further, these assumptions implicitly recognize that the development of the VCA system represents a single aspect of a broader, system-of-systems engineering problem which requires interdisciplinary perspectives to enable the real-world implementation of a safe and reliable CAV transportation system.

We assume that the VCA system has access to an external reputation management system for the CVs, AVs, and RSU devices which make up the VCA system. There are numerous approaches to the design of reputation management systems within the current literature which provide support and justification for this assumption as summarized in [53]. This assumption allows us to use the concept of reputation scores as an input into the blockchain-based continuous authentication mechanism proposed in Chapter 4 without the need to create a novel reputation management system for CAV transportation systems.

We assume physical security for the in-vehicle and smart infrastructure hardware components which facilitate V2X wireless communication within the scope of the VCA system. In other words, we assume that the physical hardware present within CVs, AVs, and RSU devices which facilitate V2X wireless communication are secure from physical tampering and other physical-based attacks. We base this assumption on the prior work of Stajano and Anderson on physical tamper protection for ubiquitous computing [54]. This assumption allows us to focus our efforts on secure computer network design without the need for novel physical security solutions.

We assume that in-vehicle electronic communications such as Controller Area Network (CAN) bus signals are logically separated from external wireless communications. This assumption allows us to focus on the design of system-level solutions for vehicular networks without the need to consider the potential for crosstalk or interference from in-vehicle communications. While the secure integration of in-vehicle communications and external wireless communications represents a viable research area, such efforts fall outside the scope of this work.

We assume access to a secure key management system for the use of public key infrastructure (PKI) cryptography. We further assume that there exists a trusted SSP which acts as the trusted authority (TA) responsible for PKI key distribution and management. This assumption allows us to leverage PKI cryptography for system-wide security features without the need to define a novel key management system.

We assume the SSP acts as an uncompromised TA entity within the scope of the VCA system and the broader metropolitan-area CAV transportation system. In other words, we assume that the SSP will not be compromised by a cyber-based effect, which in turn allows us to assume a high level of integrity and availability for the SSP. This assumption allows us to leverage the SSP as a guaranteed root of trust within the context of our systematic design efforts. As such, we can rely on the cybersecurity capabilities provided by the SSP without the need to define explicit cybersecurity controls for the SSP itself.

In this section we provide definitions for key system components and outline assumptions which bind our solution space. These efforts allow us to focus on our core systematic design goals, as opposed to the development of ancillary cybersecurity controls for system aspects which fall outside the scope of this work. In the next section we apply the information presented up to this point to define a VCA system reference architecture.

2.4 Definition of VCA System Reference Architecture

While the instantiation of a system reference architecture at the start of a systematic design process may seem counterintuitive, we opt for this approach due to a lack of real-world CAV transportation systems to serve as a model architecture for the proposed VCA system. As such, we seek to formerly define a reference architecture for the proposed VCA system. This reference architecture shall provide a foundation for the application of a systematic design process towards the creation of technical solutions which seek to promote improved VRU safety.

2.4.1 VCA System Reference Architecture

We focus our definition of the VCA system reference architecture on three key aspects of the system: a single VCA system instance, the back-end network infrastructure which connects multiple VCA system instances to the SSP, and the distribution of PKI cryptography resources throughout the overall system. The single VCA system instance defines a reference architecture for a single traffic intersection. The back-end network infrastructure defines how multiple VCA system instances connect to the back-end SSP. The distribution of PKI cryptography resources defines how CVs, AVs, and RSU devices interact with the back-end SSP to enable the use of PKI within the overall system-of-systems scope.

The single VCA system instance consists of the CVs and AVs near the traffic intersection-based RSU device. The single VCA system instance may also include one or more VRUs who have unexpectedly entered the traffic intersection. Furthermore, an outside cybersecurity threat actor may attempt to tamper with the operation of a given VCA system instance. While cybersecurity threat actors may attempt to tamper with multiple entities within the VCA system at the same time, within this context we consider the special case of a single threat actor attempting to interfere with the traffic intersection-based RSU device and its standard operation. We present a diagram which captures the reference architecture of a single VCA system instance below in Figure 5.

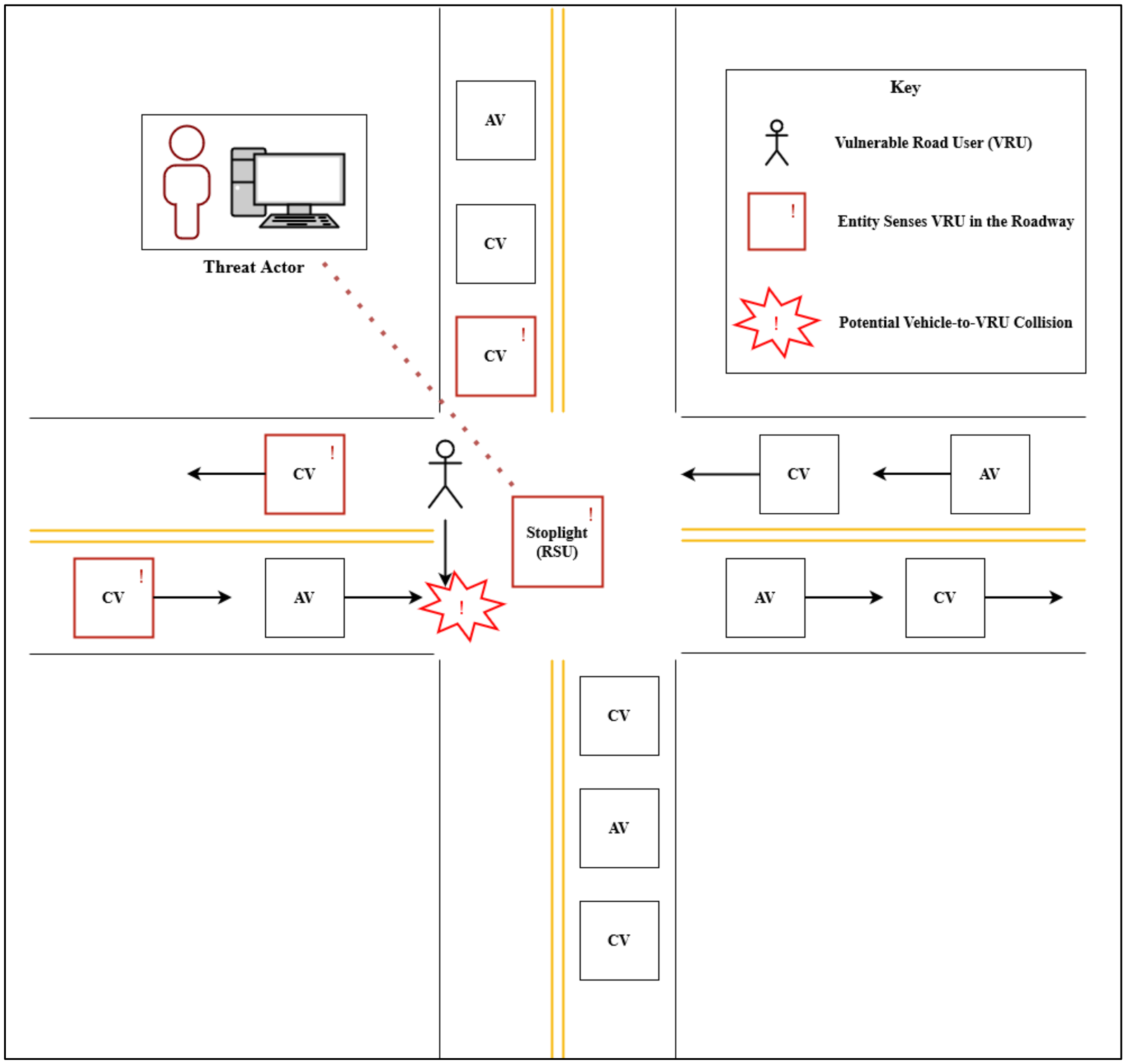

Threat Actor
AV
CV
CV
!
Key
Vulnerable Road User (VRU)
!
Entity Senses VRU in the Roadway
!
Potential Vehicle-to-VRU Collision
CV
!
CV
AV
Stoplight (RSU)
!
CV
!
AV
!
AV
CV
CV
AV
CV

Figure 5: Reference Architecture for Single VCA System Instance

Each VCA system instance connects to the SSP via the back-end network infrastructure. The back-end network infrastructure allows multiple VCA system instances to connect to the SSP via the router functionality provided by the RSU device. In a local context the RSU device coordinates the broadcast of emergency messages based on the contents of routine sensor data readouts from nearby CVs and AVs. The routine sensor data readouts function as CAMs while the emergency messages function as DENMs, as outlined above in Section 2.2.1. In a broader system-of-systems context, the RSU device present within each VCA system instance connects to a back-end router, which enables communication between each of the VCA system instances and the SSP. We envision this back-end router to provide support for 100 Gigabit Ethernet (100GbE) network connections to minimize transmission delay and maximize the transmission rate for data packets. This back-end network infrastructure allows for RSU devices to remain synchronized with the SSP via routine software updates during low-use traffic periods such as overnight. We provide a diagram of the back-end network infrastructure below in Figure 6.

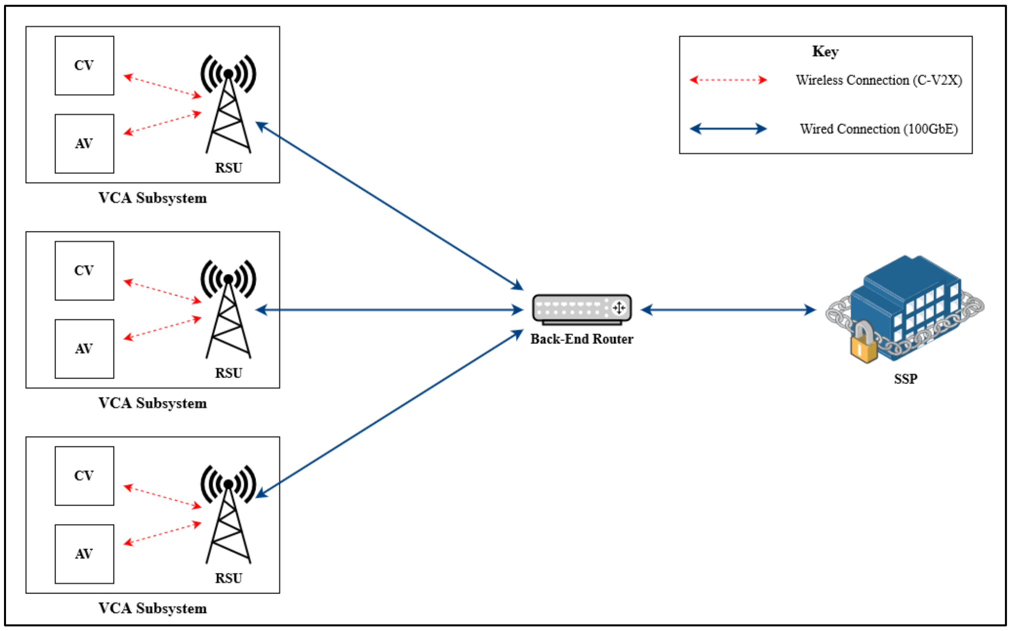


Figure 6: Overview of Back-End Network Infrastructure

The distribution of PKI cryptography resources throughout each VCA system instance and broader CAV transportation system focuses on the use of the SSP as a trusted third party (TTP) responsible for PKI certificate distribution, validation, and revocation for CVs, AVs, and RSU devices present throughout the system. As such, the SSP operates as both the certificate authority (CA) and validation authority (VA), in addition to the certificate revocation list (CRL) issuer. This defines a monopoly-style PKI system where the SSP represents the only TTP and is therefore universally trusted by CVs, AVs, and RSU devices present within the overall system-of-systems context [55]. We recognize this represents a potentially insecure design choice for real-world use cases. However, we opt to leverage the assumption of a secure SSP defined above in Section 2.3.2 to grant us access to PKI cryptography without the need to develop a novel set of cybersecurity controls for the SSP. This allows us to leverage the tools of PKI cryptography while we focus our efforts on the systematic design of technical solutions for the VCA system.

The SSP provides support for TLS certificates for CVs, AVs and RSU devices present within each individual VCA system. Each TLS certificate leverages the X.509 certificate standard as outlined in the Internet Engineering Task Force (IETF) Request for Comment (RFC) 5280 [56]. TLS certificates shall have a certificate expiration time of 24 hours within the VCA system reference architecture. We assume that the issuance of new PKI certificates is accomplished through an out-of-band update mechanism. CVs and AVs are issued client certificates through an over-the-air (OTA) update mechanism during low-use periods, while RSU devices are issued server certificates through an update mechanism which leverages the back-end network infrastructure between RSU devices and the SSP. For example, CVs and AVs may receive new client certificates via OTA updates while garaged overnight, while RSU devices may receive new server certificates during low-use automotive traffic periods.

2.4.2 Standard Format for Sensor Data Readouts

We seek to create a baseline CAV emergency message application within Chapter 3. However, the wireless communication devices associated with various CVs, AVs, and RSU devices may use different data format specifications. As such, we propose the use of a standard format for sensor data readouts sent by vehicles and emergency actuation messages sent by the traffic intersection-based RSU device. We provide an overview of this standard sensor data readout format below in Table 5.

Table 5: Standard Format for Sensor Data Readouts

Payload Entry	Definition	Example Values
ENTITY_ID	String which defines the identity of the CV, AV, or RSU device sending the message.	“Connected_Vehicle” “Autonomous_Vehicle” “Roadside_Unit”
TIMESTAMP	The timestamp of when the message was created by the given CV, AV, or RSU device.	“2023-05-01 12:52:32”
EMERGENCY_STOP	The actuation flag set by the RSU device which causes CVs and AVs to perform an emergency stop due to VRU detection.	“True” or “False”
SLOW_CAUTION	The actuation flag set by the RSU device which directs CVs and AVs to proceed with caution after the detection of a non-VRU roadside obstruction.	“True” or “False”
STOP_VRU	The sensor readout flag set by the CV or AV associated with the sensor data readout which indicates that the given CV or AV has sensed the presence of a VRU in the roadway.	“True” or “False”

2.5 Conclusion

The effective design of the proposed VCA system and its related technical solutions requires prior knowledge of CAV transportation systems, vehicular network communication paradigms, and relevant engineering standards. As such, we provide a brief overview of each of these topics within this chapter. We then apply this information to enable the definition of a VCA system reference architecture. The VCA system reference architecture provides a foundation for the development of related technical solutions, while the standard sensor data readout format enables system-wide interoperability for data exchanges between those entities present within a given VCA system instance.

While this chapter provides a concise overview of relevant technical information, there remain opportunities for future work. Academic researchers, technical standards organizations, and automotive industry representatives must continue to work towards the clear definition of technical standards for next-generation CAV transportation systems. In particular, the terminology standards which apply to the description of vehicular automation features require further refinement to avoid unnecessary confusion for the average consumer. Further, the automotive industry must also work towards the widespread adoption and ubiquitous availability of C-V2X technology in consumer-grade automobiles for the potential benefits of vehicular networks to become a reality. Additional work on reference architectures for CAV transportation systems and related safety systems such as the proposed VCA system represents a viable opportunity for future research and development. While these efforts fall outside the scope of this work, their advancement would increase the real-world viability of the proposed VCA system and its related technical solutions.

Chapter 3 – Development of CAV Emergency Message Application

In this chapter we present work towards the creation of a CAV emergency message application for the proposed VCA system. We first explore the viability of two popular application layer protocols for M2M and IoT communication systems: Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP). We provide an overview of each protocol and outline the unique features which make them viable candidates for use within vehicular networks. We then perform protocol analysis to determine which option exhibits the best performance in terms of average message latency under moderate network loss conditions. We use these results to inform the creation of a baseline CAV emergency message application. We highlight the viability of this solution towards improved VRU roadside safety with a code execution example and conclude with a discussion of opportunities for future work.

3.1 Introduction

MQTT leverages a broker-based publisher-subscriber communication paradigm built on top of the reliable transport provided by Transmission Control Protocol (TCP). Alternatively, CoAP provides lightweight client-server message exchange built on top of the best effort transport provided by User Datagram Protocol (UDP). MQTT and CoAP leverage unique design approaches to enable the exchange of messages in the context of IoT systems, and the analysis of their relative performance represents an active field of research as highlighted by [57] – [60]. MQTT leverages an event-driven architecture (EDA) and provides support for three quality of service (QoS) levels. CoAP leverages a representational state transfer (REST) architecture to facilitate lightweight message exchange via either confirmable (CON) or non-confirmable (NON) traffic. Table 6 below provides a brief overview of the key differences between MQTT and CoAP.

Table 6: Key Differences Between MQTT and CoAP

	Architecture	**Transport Layer**	**Port**	**Communication Paradigm**
MQTT	EDA	TCP	1883	Publisher-Subscriber
CoAP	REST	UDP	5683	Client-Server

We may further examine MQTT and CoAP in the context of the standard Transmission Control Protocol/Internet Protocol (TCP/IP) model. Recall that the TCP/IP model consists of five standardized layers: the application layer, the transport layer, the network layer, the data link layer, and the physical layer. The application layer manages messages specific to a network application running on a given host device. The transport layer enables host-to-host exchange of segments through either reliable connection-oriented TCP or unreliable connection-less UDP. The network layer facilitates the exchange of datagrams across the networks using Internet Protocol (IP). The link layer provides point-to-point forwarding of frames from node-to-node, while the physical layer provides the physical medium which enables bits of data to move across the network [61]. Each layer of the TCP/IP model appends additional information to data from the prior layer through a process known as encapsulation, where unique headers are added by subsequent layers. We provide an overview of the five-layer TCP/IP model below in Figure 7.

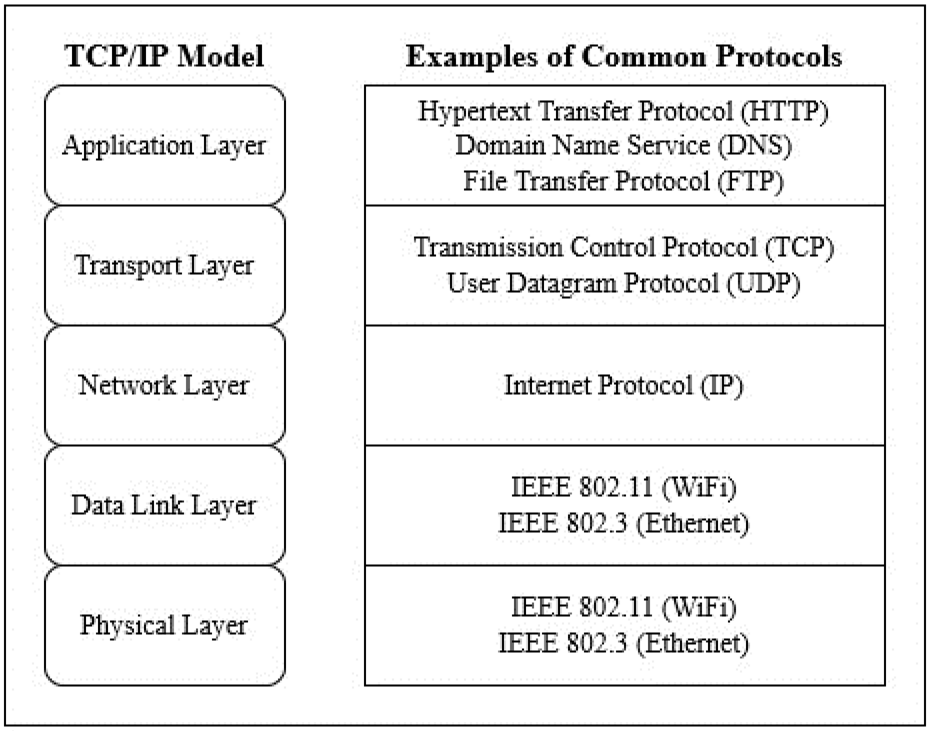


Figure 7: Overview of Five-Layer TCP/IP Model with Common Protocols

While the TCP/IP model underpins the work towards a CAV emergency message application presented in this chapter, we opt to forgo an in-depth discussion of each TCP/IP model layer for brevity. Instead, we encourage the reader to reference the work of Kurose and Ross [62] for an in-depth discussion on the topic. We use the above TCP/IP model to provide context for the MQTT and CoAP protocol stacks in the below subsections. We also revisit the TCP/IP model below in Chapter 4 to provide an analogy between traditional computer network systems and emerging blockchain technology systems.

3.1.1 MQTT: Overview, TCP/IP Stack, and Quality of Service

MQTT provides an EDA protocol which uses the publisher-subscriber paradigm to deliver messages to interested entities within a given communication system. The Organization for the Advancement of Structured Information Systems (OASIS) manages the standards for MQTT, with

MQTT version 3.1.1 (v3.1.1) and MQTT version 5 (v5) representing the most popular versions in use today. We use MQTT v3.1.1 throughout this work, as open source software packages for MQTT v5 only provide partial support for some advanced development features.

MQTT relies on a centralized broker to collect, store, and forward messages such as sensor data readouts to interested parties. MQTT implementations consists of the MQTT broker and one or more MQTT endpoints, which may operate as either clients or servers. The nomenclature of IoT systems uses alternative definitions for clients and servers, where access-side IoT devices represent servers and core-side software applications represent clients. These alternative definitions reflect the fact that core-side software applications often request data from access-side devices within IoT systems, which deviates from the traditional client-server model of the Internet.

MQTT endpoints operate as either publishers or subscribers within the protocol. MQTT endpoints may publish messages to system-wide identifiers called topics. Likewise, MQTT endpoints which seek to receive messages on a given subject can subscribe to its associated topic. Once a given MQTT endpoint subscribes to a topic it will receive all messages published to that topic. This allows for a simple implementation of multicast communication. The OASIS MQTT v3.1.1 standard [63] provides an in-depth overview of the process that enables MQTT endpoints to register as either publishers or subscribers, which we summarize below in Figure 8.

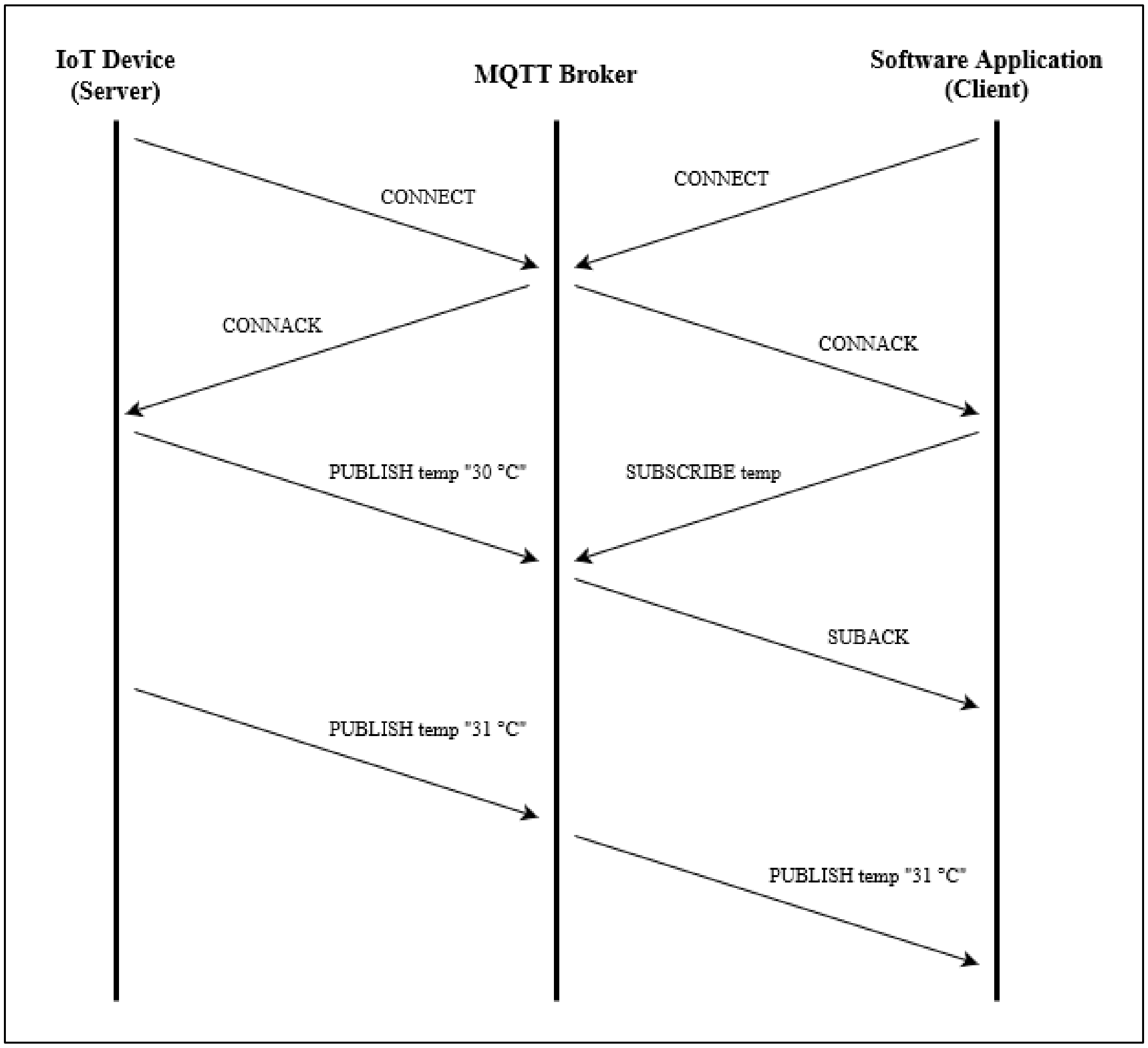


Figure 8: MQTT Connect, Subscribe, and Publish for IoT-enabled Temperature System

Figure 8 captures the process required to configure an IoT-enabled temperature system with the MQTT protocol. First, the software application initiates a connection to the MQTT broker via a CONNECT message. The IoT-enabled temperature sensor soon follows with its own CONNECT message, and the MQTT broker acknowledges these new connections through the issuance of CONNACK messages. Once the IoT-enabled temperature sensor connects to the MQTT broker it starts to publish sensor data readouts to the temp (i.e., temperature) topic. We note that the software application does not start to receive messages until it subscribes to the temp topic via the

SUBSCRIBE and SUBACK message exchange. These message exchanges at the application layer rely on an underlying protocol stack, as seen below in Figure 9.

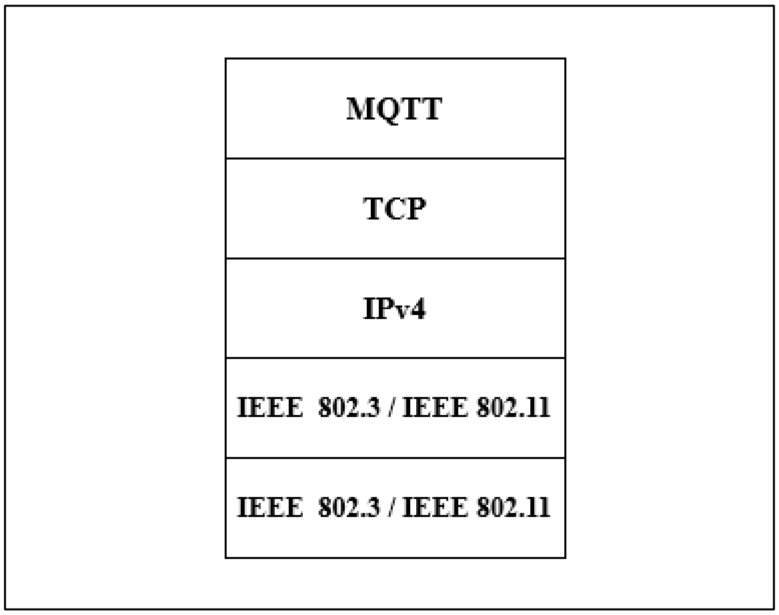


Figure 9: TCP/IP Stack for MQTT

The notional TCP/IP stack for MQTT presented above captures the lower level protocols of the transport layer, network layer, data link layer, and physical layer. While real-world implementations of MQTT tend to leverage Internet Protocol version 6 (IPv6) at the network layer, we opt to use Internet Protocol version 4 (IPv4) within the context of this work. The virtualized network testbed environments presented later in this chapter rely on a point-to-point virtual local area network (VLAN) between two virtual machines (VM) running on the same host device. As such, the selection of IPv4 helps to simplify the creation of this virtual network. Likewise, the data link layer and physical layer protocols seen above are intended to be notional, as many different protocols can be used in a real-world MQTT stack.

While MQTT leverages TCP over port 1883 for reliable transport, the protocol also defines three QoS levels to enable various levels of service guarantee. The three QoS levels for MQTT are QoS 0, QoS 1, and QoS 2. We shall provide a summary of these QoS levels based on the OASIS MQTT v3.1.1 specification [63] and the prior work of Herrero [64]. MQTT QoS 0 provides support for a delivery guarantee of at most once for each message. MQTT QoS 0 does not provide any method for retransmission in the event of message loss. If message loss occurs in transit from the MQTT publisher to the MQTT broker, then the message is not retransmitted and not made available to MQTT subscribers. Figure 10 provides an overview of MQTT QoS 0 without network loss (left) and with network loss (right).

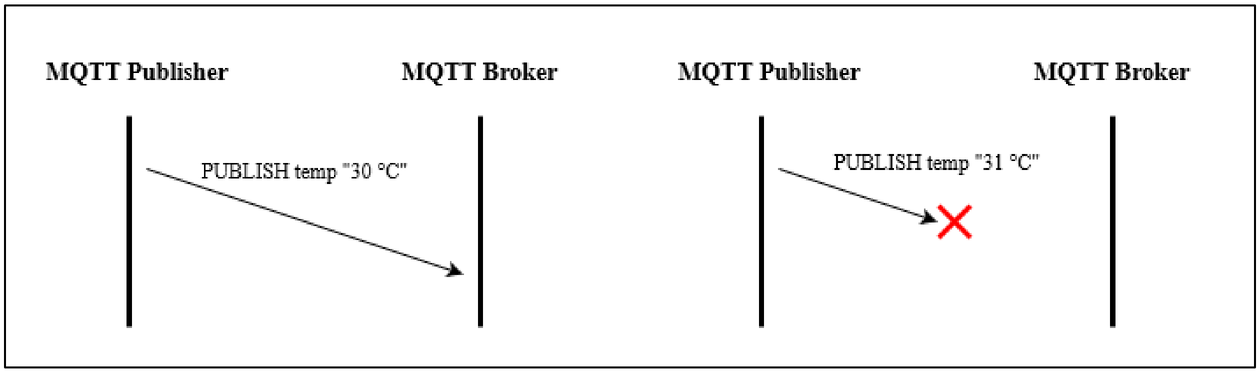


Figure 10: MQTT QoS 0 without Network Loss (left) and with Network Loss (right)

MQTT QoS 1 provides support for a delivery guarantee of at least once for each message. MQTT publishers transmit messages for a given topic via the MQTT broker with a two-way communication mechanism which requires one round-trip time (RTT). The MQTT publisher issues a PUBLISH message to the MQTT broker, with the MQTT broker in turn publishing the message to any MQTT subscribers for the given topic. Once the message has been published to MQTT subscribers, the MQTT broker sends a PUBACK message to the MQTT publisher to confirm that the message has been received and published to the relevant MQTT subscribers. If message loss occurs, then the MQTT publisher will periodically retransmit PUBLISH messages to the MQTT broker until it receives a PUBACK message.

We note that the two-way communication of PUBLISH and PUBACK messages which occurs between the MQTT publisher and the MQTT broker also occurs between the MQTT broker and the MQTT subscriber if the MQTT broker initiates message delivery under MQTT QoS 1. MQTT treats the transmission of individual messages as isolated events, which allows publisher-to-broker and broker-to-subscriber message transmissions to use different QoS levels if deemed appropriate. Figure 11 below provides an overview of the message exchange between MQTT publisher, MQTT broker, and MQTT subscriber operating under MQTT QoS 1.

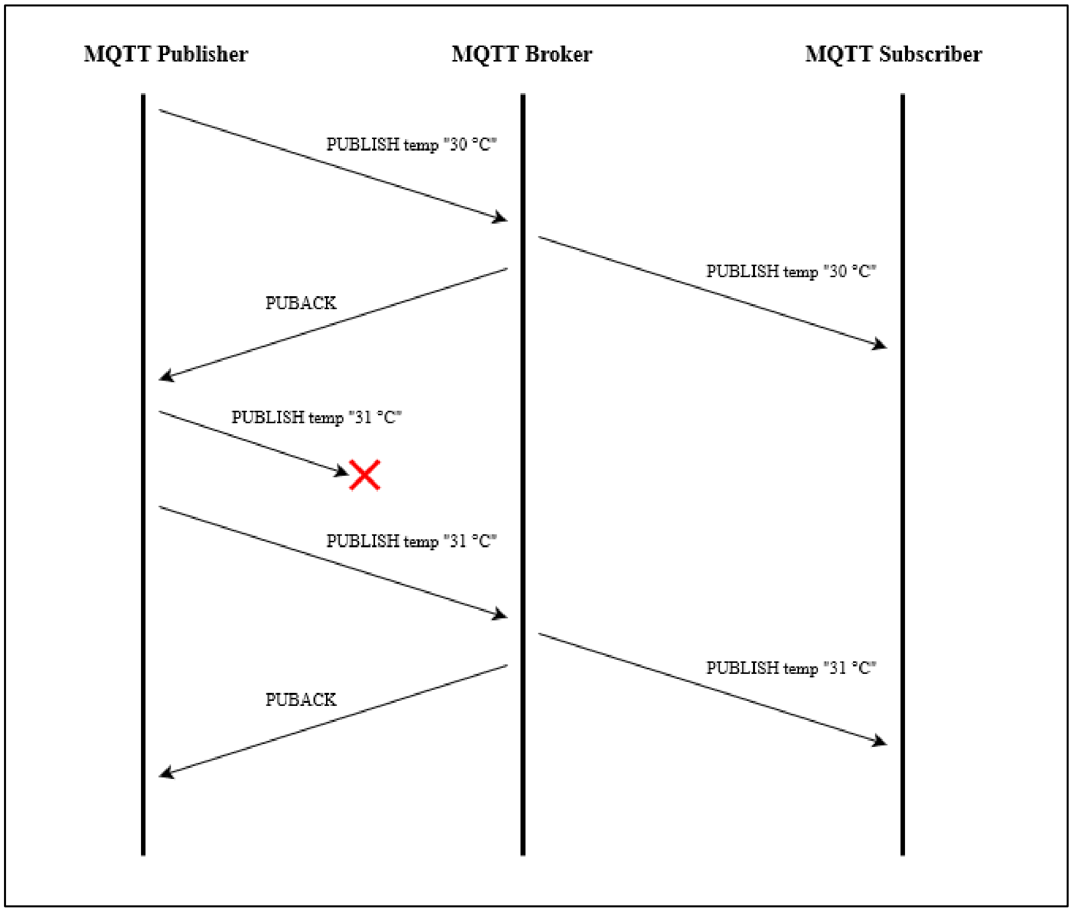


Figure 11: MQTT QoS 1 with Retransmission due to Message Loss

MQTT QoS 2 provides a delivery guarantee of exactly once for each message. MQTT publishers transmit messages for a given topic via the MQTT broker with a four-way communication mechanism which requires two RTT. The MQTT publisher issues a PUBLISH message to the MQTT broker, and the MQTT broker responds with a PUBREC message to acknowledge the request. The MQTT publisher then issues a PUBREL message, which instructs the MQTT broker to release the message for publication. The MQTT broker publishes the message to MQTT subscribers and issues a final PUBREL, which marks the end of the MQTT QoS 2 publication process.

We note the OASIS MQTT v3.1.1 specification [63] for MQTT QoS 2 allows the MQTT broker to start the message publication process to MQTT subscribers after the receipt of either the PUBLISH or PUBREL. Method A enables the MQTT broker to store the message upon receipt of the initial PUBLISH, then publish and discard the message upon receipt of the subsequent PUBREL. Alternatively, Method B enables the MQTT broker to store the message and initiate the publication process upon receipt of the initial PUBLISH, then discard the message upon receipt of the subsequent PUBREL message. We opt to use MQTT QOS 2 with Method A within the scope of this work.

MQTT QoS 2 handles network loss conditions through the retransmission of PUBLISH or PUBREL messages from the MQTT publisher that do not receive responses from the MQTT broker. This method of network loss recovery pairs with the four-way communication mechanism of MQTT QoS 2 to guarantee the delivery of each message exactly once, with no potential for repeat delivery. This approach varies from the service guarantee of MQTT QoS 1, which ensures the delivery of each message at least once with a potential for repeat delivery. Figure 12 below provides an overview of the message exchange between MQTT publisher, MQTT broker, and MQTT subscriber operating under MQTT QoS 2 with Method A.

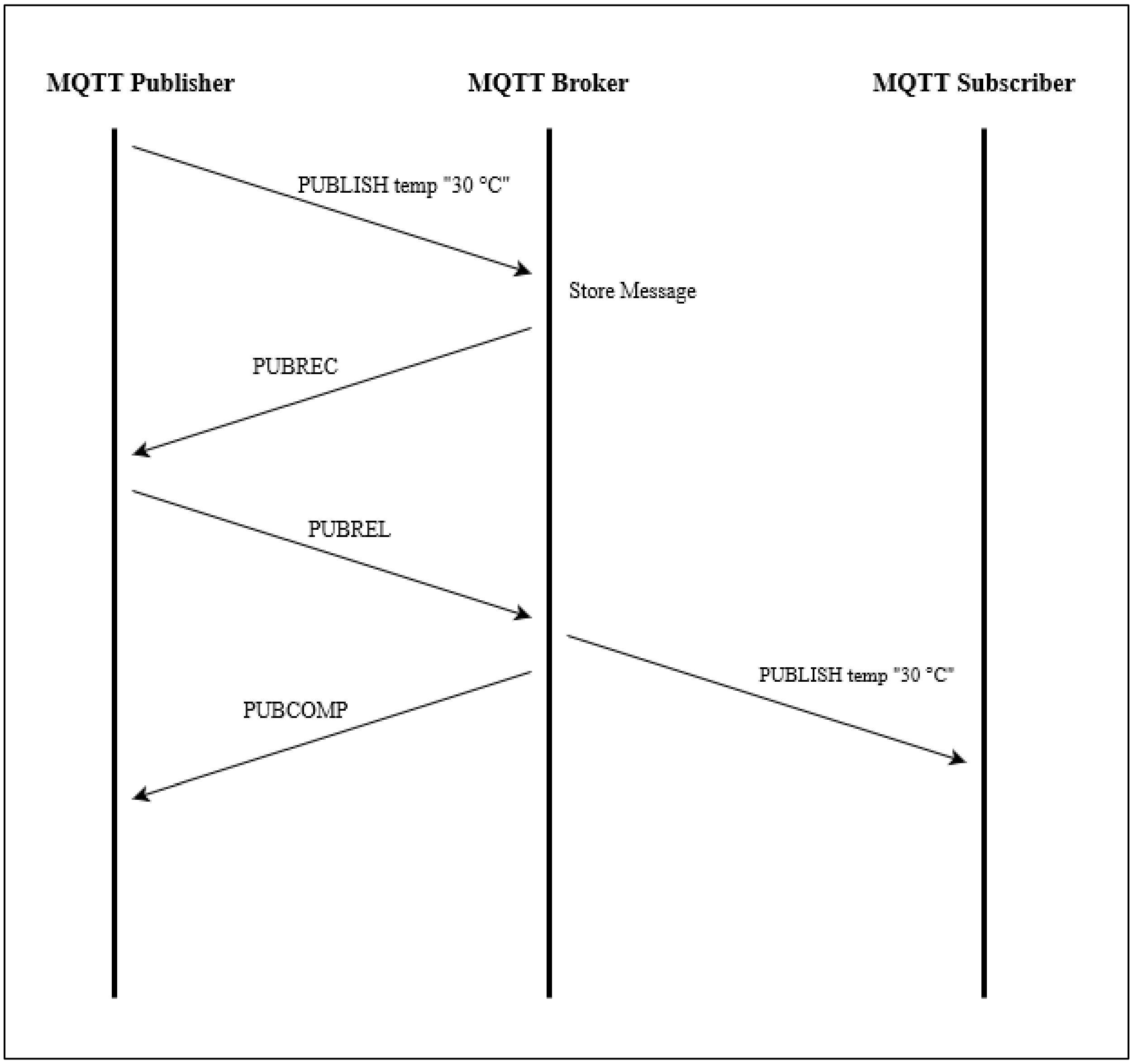


Figure 12: MQTT QoS 2 with Method A

3.1.2 CoAP: Protocol Overview, TCP/IP Stack, and Confirmable/Non-Confirmable

CoAP provides a REST protocol which uses the client-server paradigm to deliver messages between entities within a given communication system. The IETF defines the CoAP protocol in RFC 7252 [65], with multiple extensions available through follow-on RFC specifications. For example, RFC 7959 [66] extends CoAP to support block-wise transfers of structured data for improved latency of larger payloads. This standardization approach allows the core CoAP protocol to remain lightweight while still providing support for additional features.

CoAP leverages the client-server paradigm to facilitate the delivery of messages such as sensor data readouts within a given communication system. CoAP implementations typically consist of one or more CoAP servers and a CoAP client. As with MQTT, CoAP views core-side software applications as clients and access-side IoT devices as servers. Clients and servers use a combination of IP address and port number to exchange request-response style messages via best effort UDP transport.

CoAP provides a stateless protocol, which means that individual messages between client and server may be exchanged without the need for setup messages. CoAP can support the exchange of multiple related messages with optional session tokens. 16-bit message identifiers denote individual CoAP message transactions, while optional 8-bit session tokens may denote a series of related messages [65]. The exchange of CoAP messages at the application layer relies on an underlying protocol stack, as seen below in Figure 13.

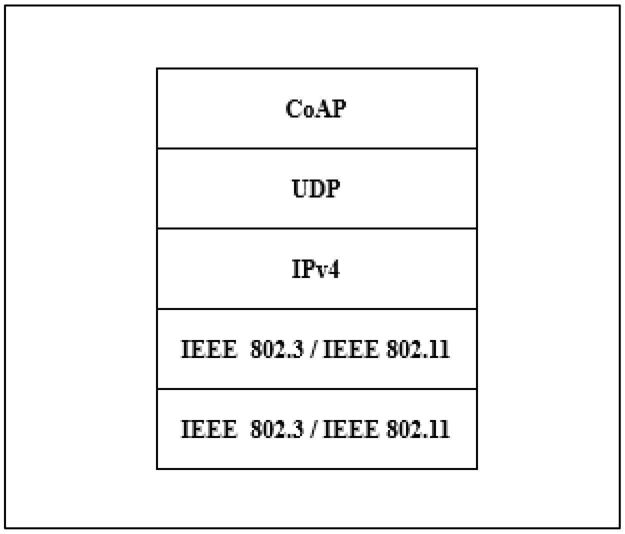


Figure 13: TCP/IP Stack for CoAP

As above, we opt to use IPv4 for CoAP traffic within the context of this work to simplify the creation of a point-to-point VLAN between two VMs on the same host device. Similarly, the data link layer and physical layer protocols listed in Figure 13 represent the many different protocols which may be used in a real-world CoAP protocol stack. We provide additional information on the specific VLAN configuration of the virtualized network testbed below in Section 3.2.1.

The use of UDP transport for CoAP messages enables the overall protocol stack to remain lightweight, but it does not provide the inherent reliability of TCP transport. CoAP addresses this shortfall through a logical two-layer adaptation consisting of the lower-level messages layer and the higher-level requests/responses layer. The messages layer interfaces with UDP to manage asynchronous interactions with CON, NON, acknowledgement (ACK), and reset (RST) messages. The requests/responses layer manages the creation of the appropriate request or response message [66], [67]. This protocol adaptation exists within the CoAP header and maps to the TCP/IP stack as seen below in Figure 14.

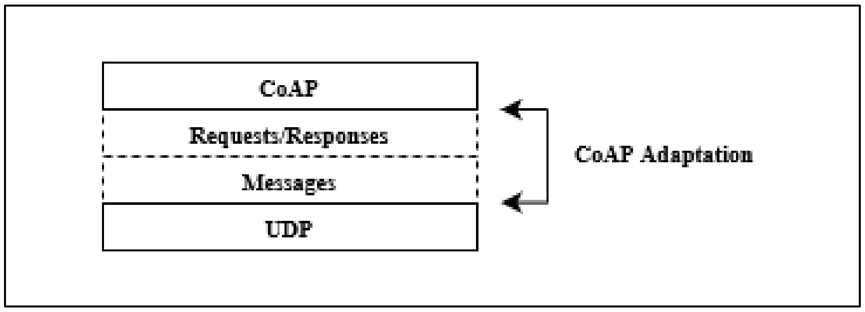


Figure 14: CoAP Adaptation Layers within the TCP/IP Stack

The use of this header adaptation allows CoAP to support two QoS levels in the form of CON traffic and NON traffic. We shall provide an overview of CoAP CON and CoAP NON based on IETF RFC 7252 [65] and the prior work of Herrero [67]. CoAP CON represents the highest QoS level, with support for retransmission of undelivered messages due to network loss conditions. Each CON message includes a message identifier (MID), request or response type, token value, and an optional data payload. MIDs are 16-bit values that enable CoAP to match CON message exchanges and assist in the identification of duplicate messages. The request or response type indicates whether the CON message contains a request or response and defines the nature of the message. For example, a GET request from a CoAP client directs the CoAP server to respond with a data payload, while a 2.05 Content response from a CoAP server indicates the presence of a data payload within the response message. Tokens represent 8-bit values which must match for request/response message exchange, and therefore identify a single message exchange between CoAP client and CoAP server. CON traffic requires the CoAP server to send a CON request message to the CoAP client. If the requested content is available, then the CoAP server may respond with an ACK message with piggyback data payload delivery, as seen below in Figure 15.

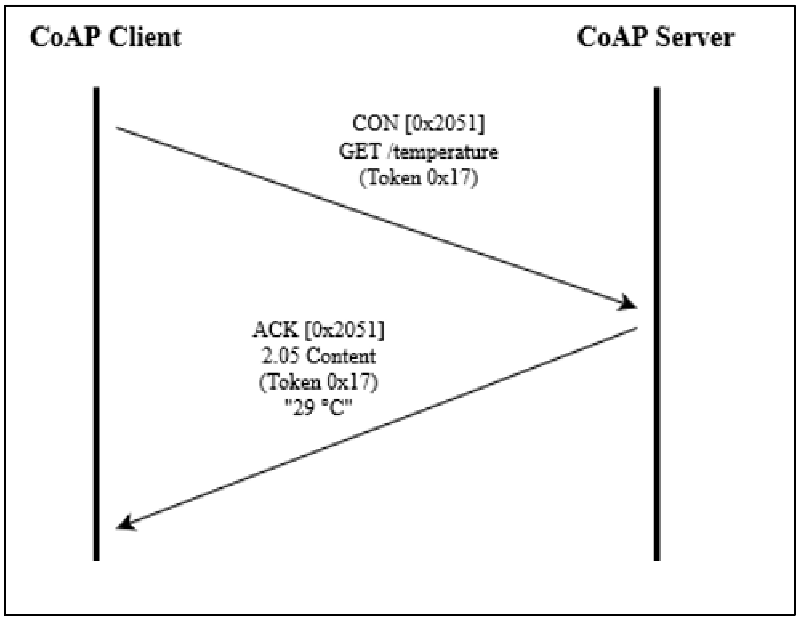


Figure 15: CoAP CON Message Exchange without Network Loss

The example of Figure 15 shows a CoAP client software application requesting sensor readout data from a CoAP server temperature sensor through a CON message exchange. The CON message from the CoAP client requests data associated with the temperature uniform resource identifier (URI) via the GET /temperature request. Likewise, the ACK message from the CoAP server responds with the requested temperature data via the 2.05 Content response and its associated data payload. CoAP CON provides reliable delivery through its ability to recover from network loss, as highlighted below in Figure 16. If a CON request message sent by the CoAP client does not receive an associated ACK response message from the CoAP server, then the CoAP client resends an identical CON request message after a timeout period.

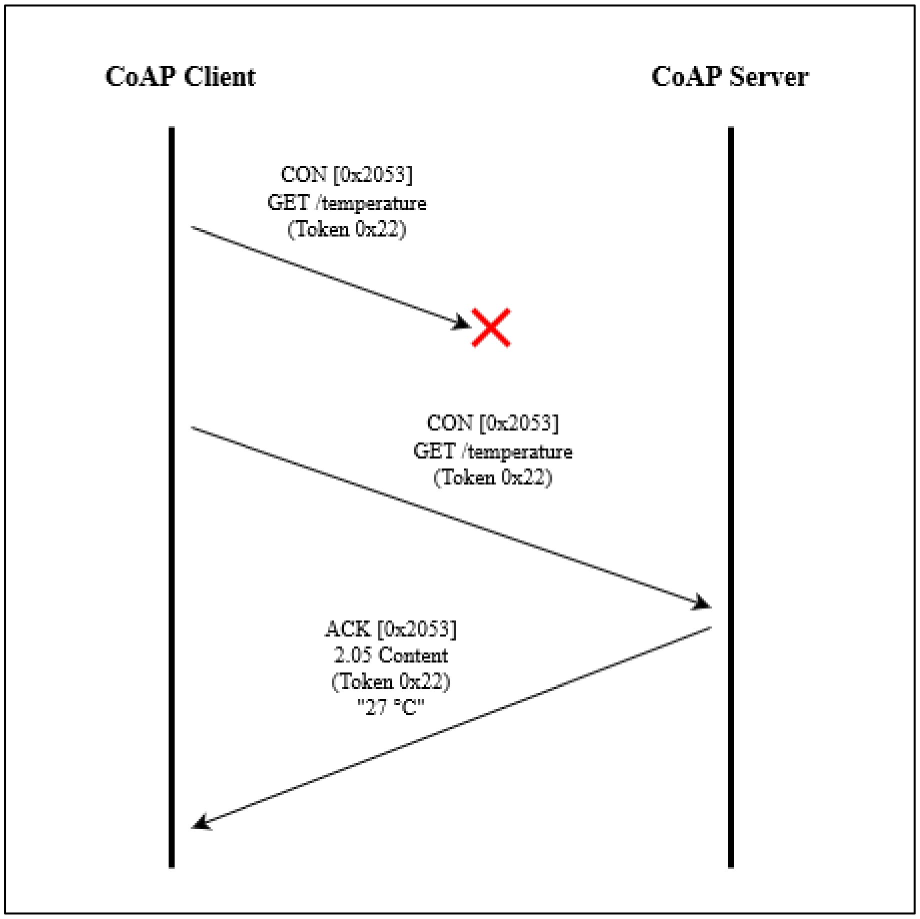


Figure 16: CoAP CON Message Exchange with Network Loss

CoAP NON provides unreliable message delivery, with no support for retransmission of undelivered messages due to network loss conditions. As above, CoAP NON messages include a MID, a request or response type, token value, and an optional data payload. MIDs help to identify duplicate messages for CoAP NON traffic, as they provide unique values for each NON message sent by a CoAP client or CoAP server. The request or response type, token, and optional data payload field provide similar functionality to that of CoAP CON traffic. If a CoAP NON message experiences a delivery disruption due network loss, then recovery of the message is not possible. Figure 17 and Figure 18 below highlight the unreliable nature of CoAP NON message exchange.

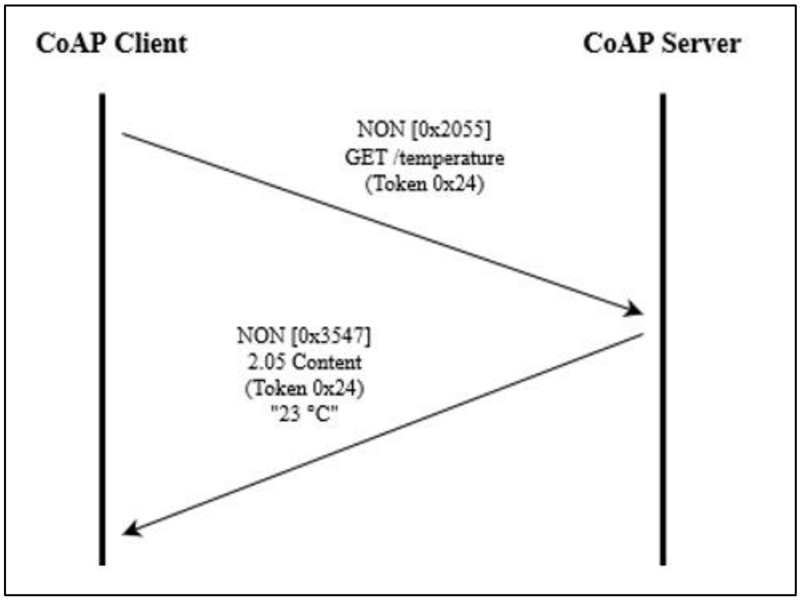


Figure 17: CoAP NON Message Exchange without Network Loss

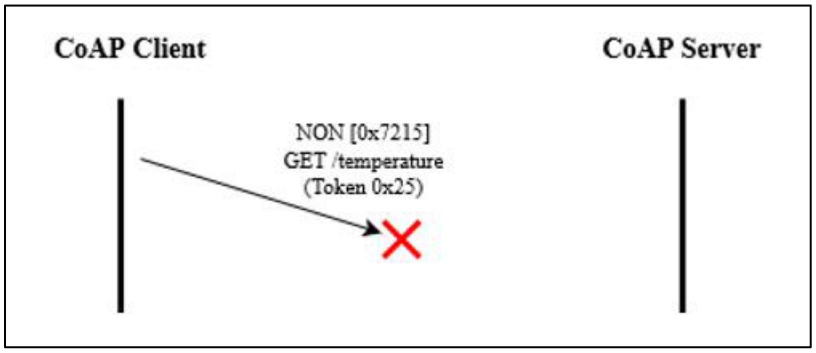


Figure 18: CoAP NON Message Exchange with Network Loss

The CoAP protocol also supports ongoing topic updates from CoAP servers through resource observation as defined in IETF RFC 7641 [68]. Resource observation provides support for both CON and NON traffic, which enables CoAP clients to access ongoing updates from one or more CoAP servers regardless of the chosen QoS level. CoAP observation may be viewed as roughly analogous to MQTT topic subscriptions, where a single registration event grants an observer access to ongoing updates for a given topic. Figure 19 provides an example of CoAP observation for NON traffic between a CoAP client and a CoAP server.

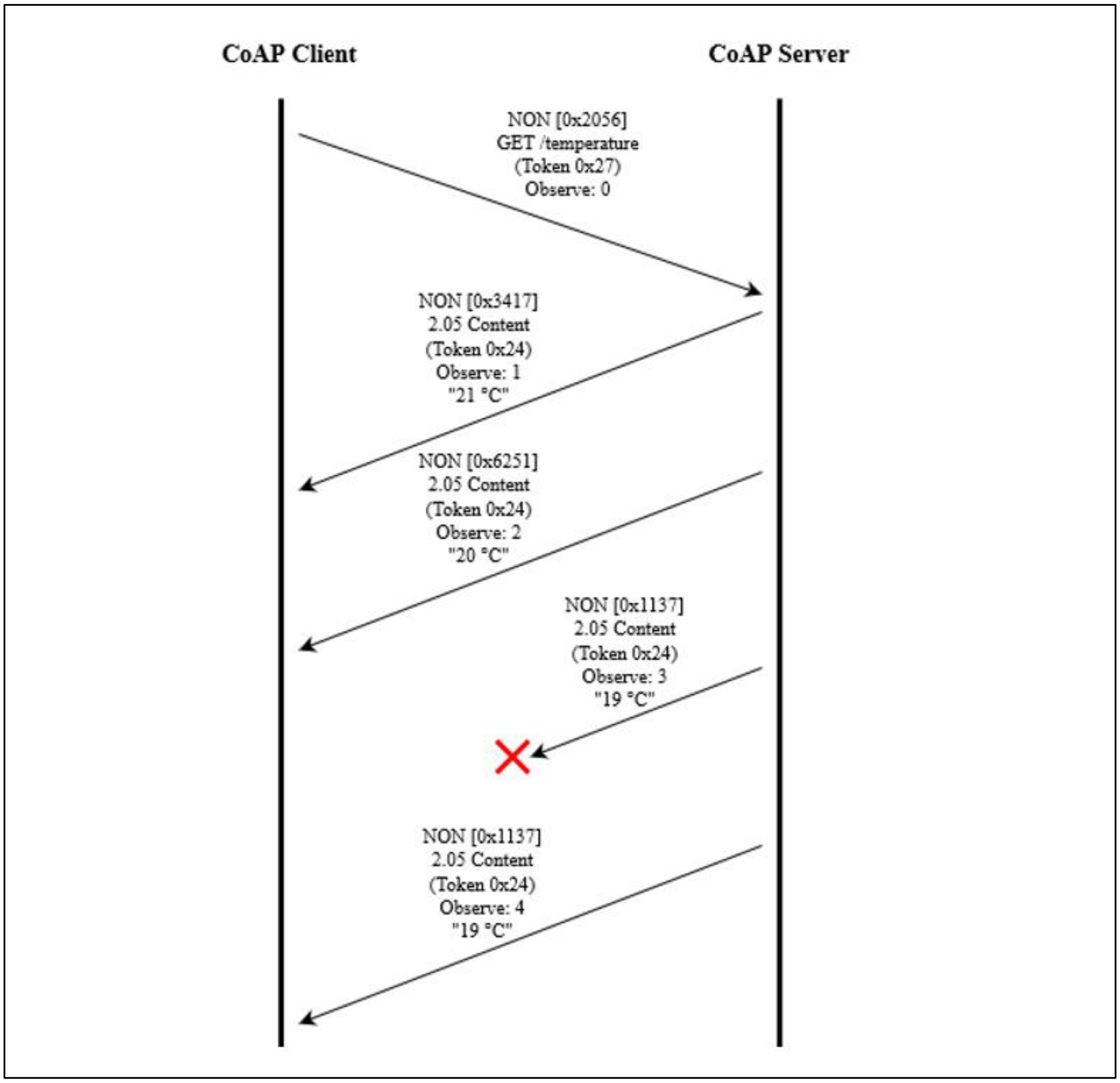


Figure 19: CoAP Observation with NON Traffic

We note that as this example uses the NON traffic, observation messages which experience network loss do not receive retransmission. Such losses do not present a problem for the long-term observation of environmental conditions such as temperature. If the quantity under observation requires reliable transmission of messages, then the use of the CON quality of service represents a better option.

3.2 Protocol Evaluation – Testbed Configuration and Execution Example

This section outlines our efforts to design, implement, and execute a protocol analysis experiment which allows us to evaluate the relative performance of MQTT and CoAP under moderate network loss conditions. We apply the VCA system reference architecture of Figure 5 above to the creation of a virtualized network testbed for MQTT and CoAP traffic. We then detail the process through which we execute MQTT and CoAP traffic on the virtualized network testbed through an execution example.

3.2.1 Virtualized Network Testbed for MQTT and CoAP Traffic

The virtualized network testbed consists of two VMs running the Linux-based Ubuntu 22.04.2 Long Term Support (LTS) desktop operating system via the Oracle VirtualBox v7.0.6 hosted hypervisor. We configure the each of the VMs with computational resources which include 4 virtual cores, 8 gigabytes (GB) of virtual memory, and 50 GB of virtual hard drive space. The underlying computer hardware consists of an Advanced Micro Devices (AMD) Ryzen 7 3700X processor with 16 logical cores and 32 GB of memory, which provides ample overhead for smooth operation of the host device, the Oracle VirtualBox v7.0.6 hosted hypervisor, and the Ubuntu 22.04.2 LTS VMs. We note the exact configuration of our virtualized network testbed may make it difficult for third-parties to exactly replicate our experimental results. While not ideal, we argue that this does not invalidate our efforts, as we seek to understand the relative performance of MQTT and CoAP under moderate network loss conditions with the knowledge that exact protocol performance may vary from network-to-network.

The virtualized network testbed configuration for MQTT consists of multiple MQTT client instances and a single MQTT broker. We instantiate MQTT client instances with the Eclipse Paho v1.5.0 library [69] in Python v3.10.6 for virtualized CV, AV, and RSU device entities. We define the underlying logic associated with these entities via the Paho v1.5.0 Python library and the creation of custom MQTT callbacks based upon the online tutorial provided by Cope in [70]. We opt to leverage the Eclipse Mosquitto v2.0.15 MQTT broker [71] to avoid the need to create a novel MQTT broker from scratch. One VM hosts MQTT client-based CV and AV entities, while the other VM hosts the combined MQTT broker and MQTT client-based RSU device entity. We configure the traffic control (tc) utility [72] to introduce various levels of uniform packet loss at the network interface of VM 1. An overview of the virtualized network testbed for MQTT traffic is provided below in Figure 20.

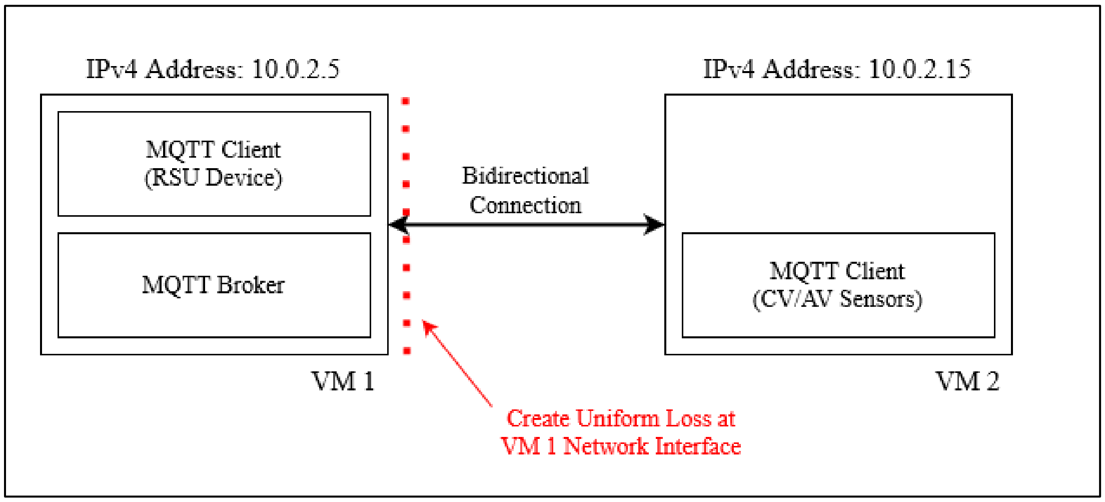


Figure 20: Virtualized Network Testbed for MQTT Traffic

We leverage a similar virtualized network testbed configuration for CoAP traffic which hosts a single CoAP client-based RSU device entity and multiple CoAP server-based CV and AV entities. We instantiate the CoAP client and CoAP server instances via the aiocoap v0.4.7 library [73]. Example code from the aiocoap project in [74] was expanded upon to support the particular use case of this study. The virtualized network testbed for CoAP traffic leverages the same two VMs as the virtualized network testbed for MQTT traffic seen above in Figure 20. In this case, VM 1 encapsulates the CoAP client-based RSU device entity, while VM 2 encapsulates the CoAP server-based CV and AV entities. We note that the virtualized network testbed for CoAP traffic inherits the configuration of the tc utility implemented for the above creation of the virtualized network testbed for MQTT traffic. We present an overview of the virtualized network testbed for CoAP traffic below in Figure 21.

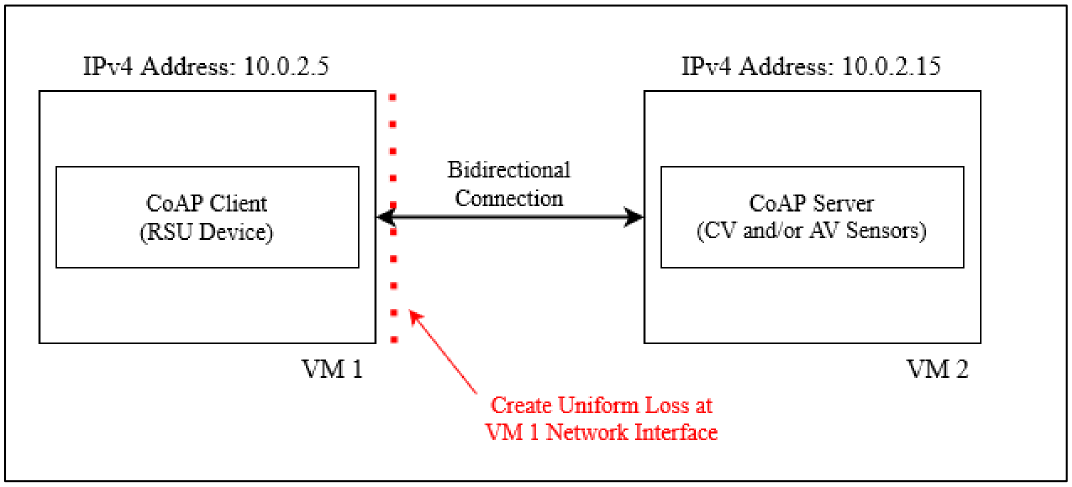
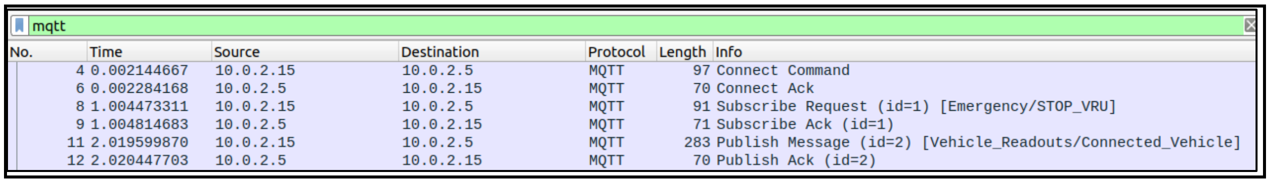


Figure 21: Virtualized Network Testbed for CoAP Traffic

3.2.2 Execution Example for MQTT Traffic

After configuration of the virtualized network testbed for MQTT traffic, we launched instances of Wireshark v3.6.2 on each VM to capture MQTT network traffic. A sample of MQTT network traffic captured via Wireshark v3.6.2 can be seen below in Figure 22.



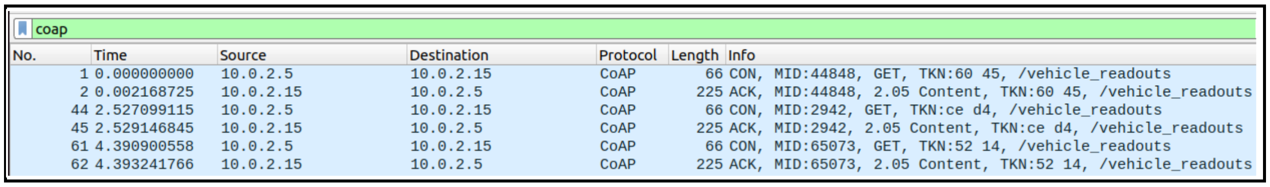
```
mqtt
No.         Time          Source         Destination    Protocol  Length  Info
          4 0.002144667   10.0.2.15      10.0.2.5       MQTT          97 Connect Command
          6 0.002284168   10.0.2.5       10.0.2.15      MQTT          70 Connect Ack
          8 1.004473311   10.0.2.15      10.0.2.5       MQTT          91 Subscribe Request (id=1) [Emergency/STOP_VRU]
          9 1.004814683   10.0.2.5       10.0.2.15      MQTT          71 Subscribe Ack (id=1)
         11 2.019599870   10.0.2.15      10.0.2.5       MQTT         283 Publish Message (id=2) [Vehicle_Readouts/Connected_Vehicle]
         12 2.020447703   10.0.2.5       10.0.2.15      MQTT          70 Publish Ack (id=2)
```

Figure 22: Wireshark Trace for MQTT Traffic

We note that the MQTT network traffic seen above in Figure 22 corresponds to MQTT QoS 1 for a virtualized network testbed with no network loss. Likewise, we note that the PUBLISH message from the MQTT client-based CV entity contains payload data which uses the standard format defined above in Table 5. The data payload is encoded via base64 at the CV entity, then sent across the network and decoded by the RSU device entity. This results in a data payload of 214 bytes and an overall packet size of 283 bytes, which is roughly consistent with prior experimental trials for real-world safety messaging applications in vehicular networks. For example, Qualcomm has shown acceptable network performance for safety messages with a packet size of 193 bytes in real-world experimental trials as detailed in [75]. Additional compression could be performed to further optimize the size of the data payload for PUBLISH messages. However, we opt to forgo these efforts as a slightly larger packet size should have only a minor impact on performance within the scope of the given virtualized network testbed.

3.2.3 Execution Example for CoAP Traffic

After configuration of the virtualized network testbed for CoAP traffic, we launched instances of Wireshark v3.6.2 on each VM to capture CoAP network traffic. A sample of CoAP network traffic captured via Wireshark v3.6.2 can be seen below in Figure 23.



```
coap
No.     Time         Source      Destination  Protocol Length Info
      1 0.000000000  10.0.2.5    10.0.2.15    CoAP        66 CON, MID:44848, GET, TKN:60 45, /vehicle_readouts
      2 0.002168725  10.0.2.15   10.0.2.5     CoAP       225 ACK, MID:44848, 2.05 Content, TKN:60 45, /vehicle_readouts
     44 2.527099115  10.0.2.5    10.0.2.15    CoAP        66 CON, MID:2942, GET, TKN:ce d4, /vehicle_readouts
     45 2.529146845  10.0.2.15   10.0.2.5     CoAP       225 ACK, MID:2942, 2.05 Content, TKN:ce d4, /vehicle_readouts
     61 4.390900558  10.0.2.5    10.0.2.15    CoAP        66 CON, MID:65073, GET, TKN:52 14, /vehicle_readouts
     62 4.393241766  10.0.2.15   10.0.2.5     CoAP       225 ACK, MID:65073, 2.05 Content, TKN:52 14, /vehicle_readouts
```

Figure 23: Wireshark Trace for CoAP Traffic

We note that the CoAP network traffic seen above in Figure 23 corresponds to CoAP CON traffic for a virtualized network configuration with no network loss. Likewise, we note that the ACK 2.05 Content messages provided by the CoAP server-based CV to the CoAP client-based RSU device contain payload data which uses the standard format defined above in Table 5. As with MQTT above, the payload data is encoded via base64 at the CV entity, then sent across the virtualized network and decoded by the RSU device entity. This approach is taken to ensure the size of the payload data and its related encoding has minimal impact on the average latency values seen across the trials conducted for MQTT and CoAP within the scope of this study.

3.3 Performance of MQTT and CoAP Under Network Loss

This section provides an overview of the test case execution and preliminary data analysis for MQTT traffic and CoAP traffic generated via the virtualized network testbed presented in the above section. Given the real-time nature of the proposed CAV emergency message application, we seek to determine which combination of protocol and QoS level yields the best performance in terms of average message latency. The Python code and other relevant artifacts associated with this study are available for reference within a public GitHub repository maintained by the author at [76].

3.3.1 Performance Results for MQTT Traffic Under Network Loss

We execute test cases for MQTT QoS 1 traffic and MQTT QoS 2 traffic for this study. We opt to forgo analysis of MQTT QoS 0 as its lack of service guarantee makes it a poor candidate for use in the CAV emergency message application, as we require both low average message latency and reliable message delivery. The test cases for MQTT QoS 1 and MQTT QoS 2 each consist of network traffic sent from the MQTT client-based CV entity of VM 2 to the MQTT broker and MQTT client-based RSU device of VM 1. We conduct individual trials for 0%, 3%, 5%, and 10%

uniform packet loss injected into the virtualized network testbed by the tc utility. Each individual trial consists of 100 messages to estimate the average message latency for MQTT QoS 1 and MQTT QoS 2. We define average message latency as the overall time it takes for a completed message publication. As such, average message latency for a MQTT QoS 1 message represents the time it takes for a PUBLISH message from the MQTT publisher to receive a PUBACK from the MQTT broker. Likewise, average message latency for a MQTT QoS 2 message represents the time it takes for a PUBLISH message from the MQTT publisher to receive a PUBCOMP message from the MQTT broker.

We present results for MQTT QoS 1 and MQTT QoS 2 in a series of tables and figures below. Table 7 and Table 8 present the median and average message latency values for MQTT QoS 1 traffic and MQTT QoS 2 traffic, respectively. Figure 24 and Figure 25 present box plots which provide insight into the spread of the experimental data. We note that these box plots exclude some outlier data points to ensure the legibility of the figures. These results show that MQTT QoS 1 exhibits lower average message latency than MQTT QoS 2 for all network loss conditions seen within the scope of this study. We provide a more in-depth analysis of the relative performance of MQTT and CoAP below in Section 3.3.3.

Table 7: Median and Average Message Latency for MQTT QoS 1 Traffic

	Uniform Packet Loss (% Loss)			
	0%	**3%**	**5%**	**10%**
Median Latency (ms)	0.367	0.386	0.400	0.405
Average Latency (ms)	0.485	4.534	4.759	21.410

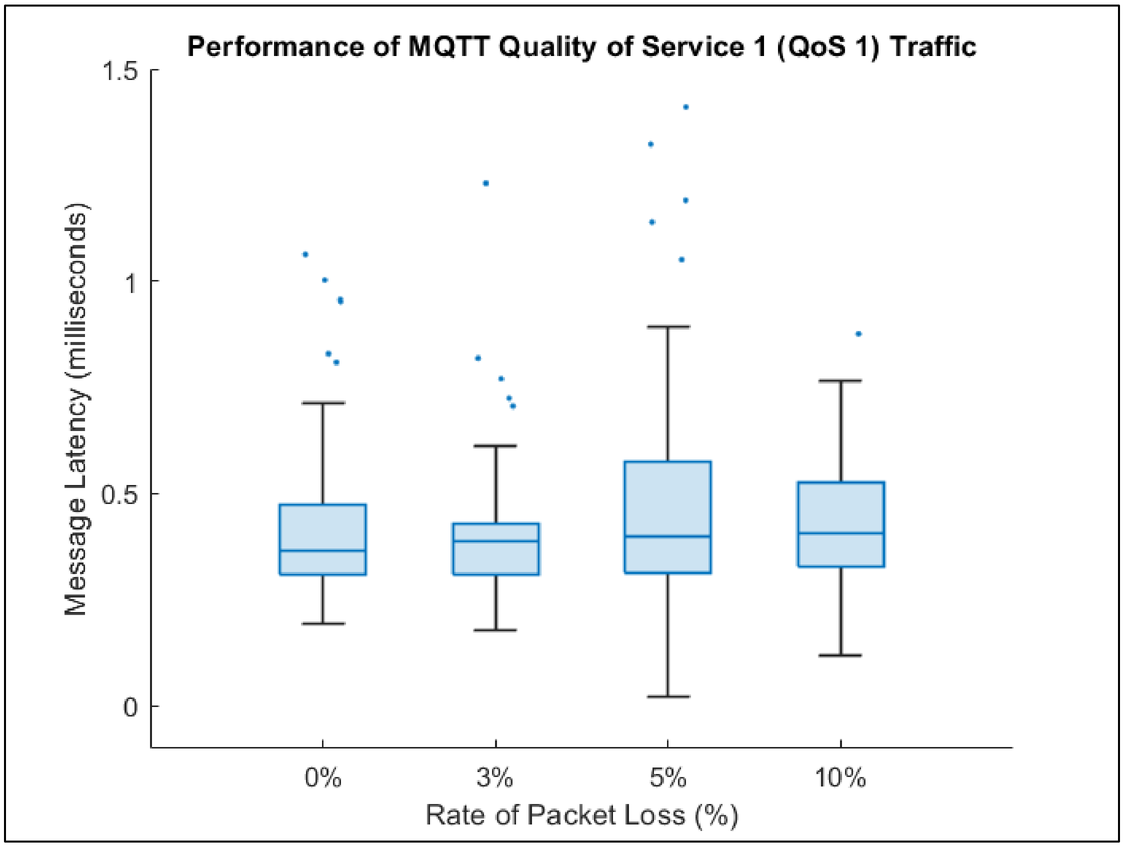


Figure 24: Message Latency of MQTT QoS 1 Traffic

Table 8: Median and Average Message Latency for MQTT QoS 2 Traffic

	Uniform Packet Loss (% Loss)			
	0%	**3%**	**5%**	**10%**
Median Latency (ms)	1.728	1.717	1.750	1.816
Average Latency (ms)	2.003	8.915	11.276	46.232

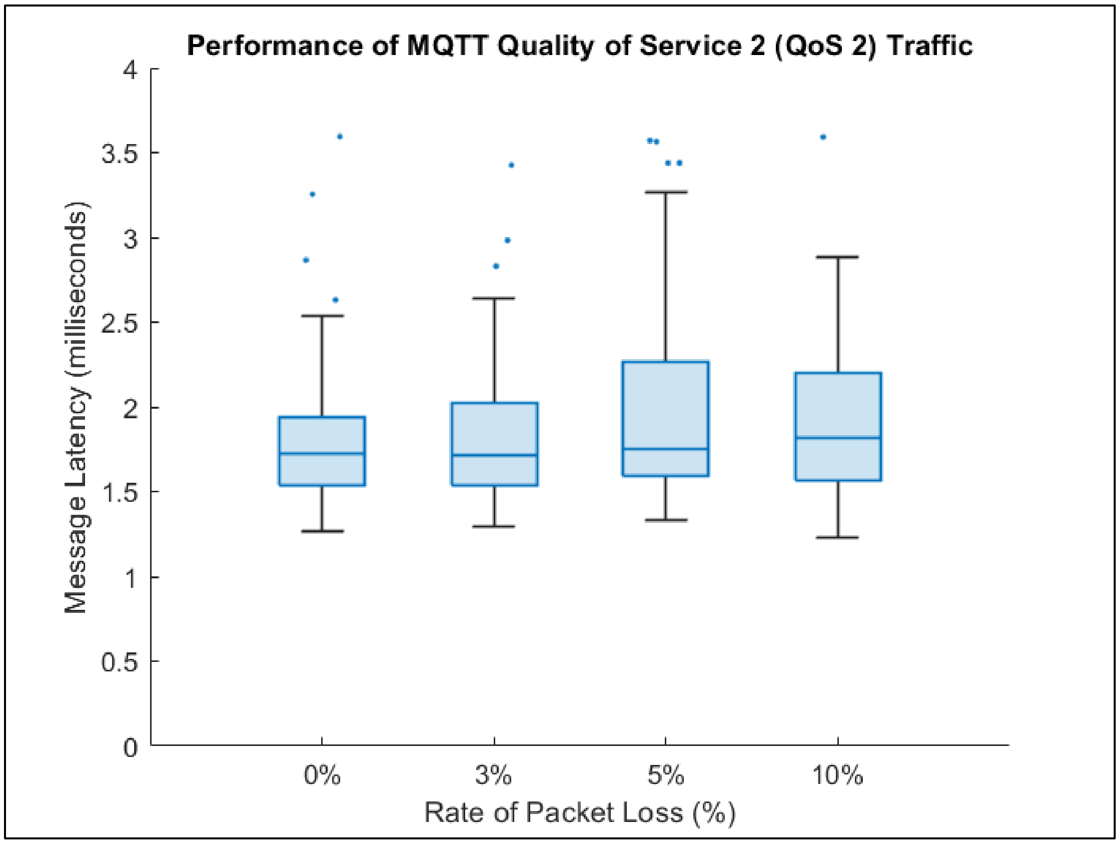


Figure 25: Message Latency of MQTT QoS 2 Traffic

3.3.2 Performance Results for CoAP Traffic Under Network Loss

We execute test cases for CoAP CON traffic and CoAP NON traffic for this study in line with the process outlined above for MQTT traffic. The test cases for CoAP CON and CoAP NON traffic each consist of network traffic sent from the CoAP client-based RSU device entity of VM 1 to the CoAP server-based CV and AV entities of VM 2. Individual trials were conducted for 0%, 3%, 5%, and 10% uniform packet loss injected into the virtualized network testbed by the tc utility. Each individual trial consists of 100 message exchanges consisting of a GET request message from the CoAP client-based RSU device entity operating under observation mode and a 2.05 Content response message from the CoAP server-based CV and AV entities. As above, we define average message latency as the time it takes for a CoAP client request message to receive a response message from the CoAP server.

We present the results for CoAP CON and CoAP NON traffic in a series of tables and figures below. Table 9 presents the median and average message latency values for each level of uniform packet loss for CoAP CON, while Table 10 presents the median and average message latency values alongside the number of infinite loss events for CoAP NON traffic. If a CoAP NON message becomes undeliverable due to network loss, then the message is lost forever. As such, we propose the term infinite latency event to denote situations where network loss leads to an incomplete request/response message exchange. Figure 26 and Figure 27 present box plots to provide additional insight into the spread of the experimental data for CoAP CON and CoAP NON, respectively. As above, these box plots exclude some outlier data points to ensure the legibility of the figures. These results show that CoAP CON traffic exhibits lower average message latency due in large part to the presence of infinite latency events for CoAP NON traffic. We provide an in-depth comparison of MQTT and CoAP performance in Section 3.3.3 below.

Table 9: Median and Average Message Latency for CoAP CON Traffic

	Uniform Packet Loss (% Loss)			
	0%	**3%**	**5%**	**10%**
Median Latency (ms)	2.197	2.192	2.189	2.618
Average Latency (ms)	2.935	54.757	49.265	412.654

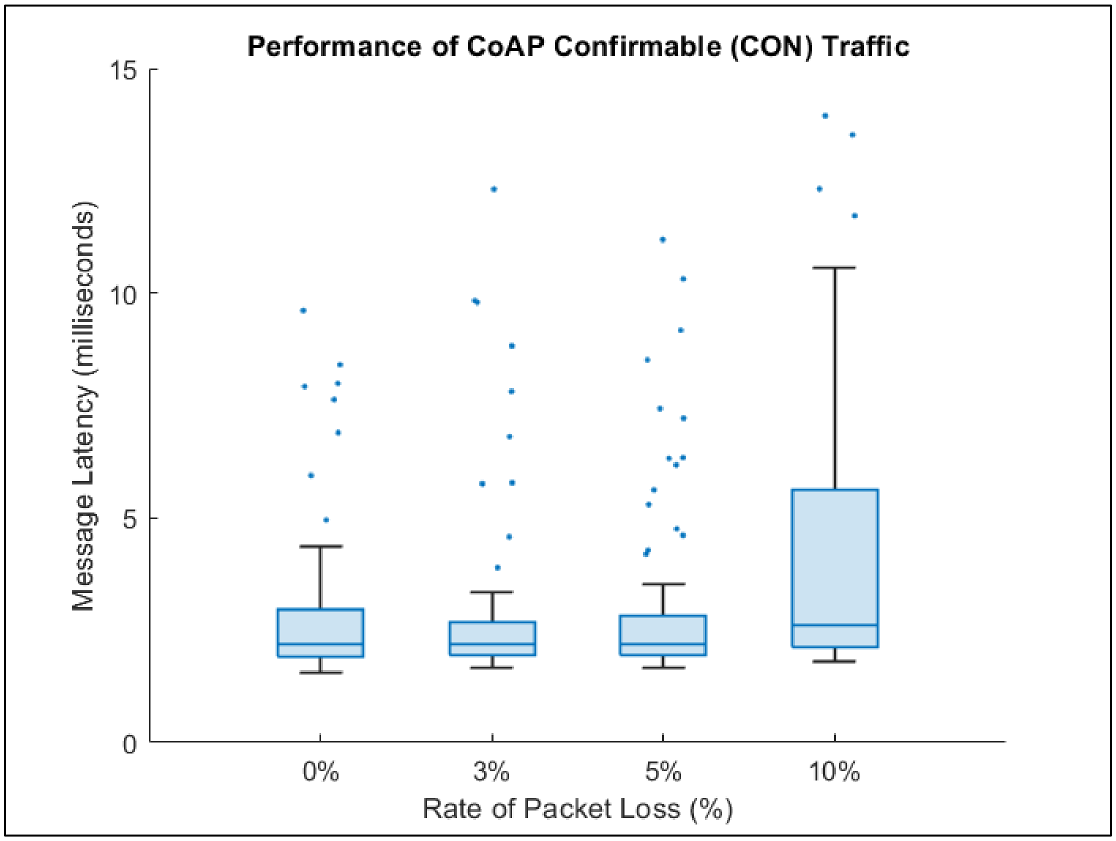


Figure 26: Performance of CoAP CON Traffic

Table 10: Median and Average Message Latency for CoAP NON Traffic

	Uniform Packet Loss (% Loss)			
	0%	**3%**	**5%**	**10%**
Median Latency (ms)	2.050	1.950	1.994	2.004
Average Latency (ms)	2.533	2.030	2.043	1.996
Infinite Latency Events	0	4	7	8

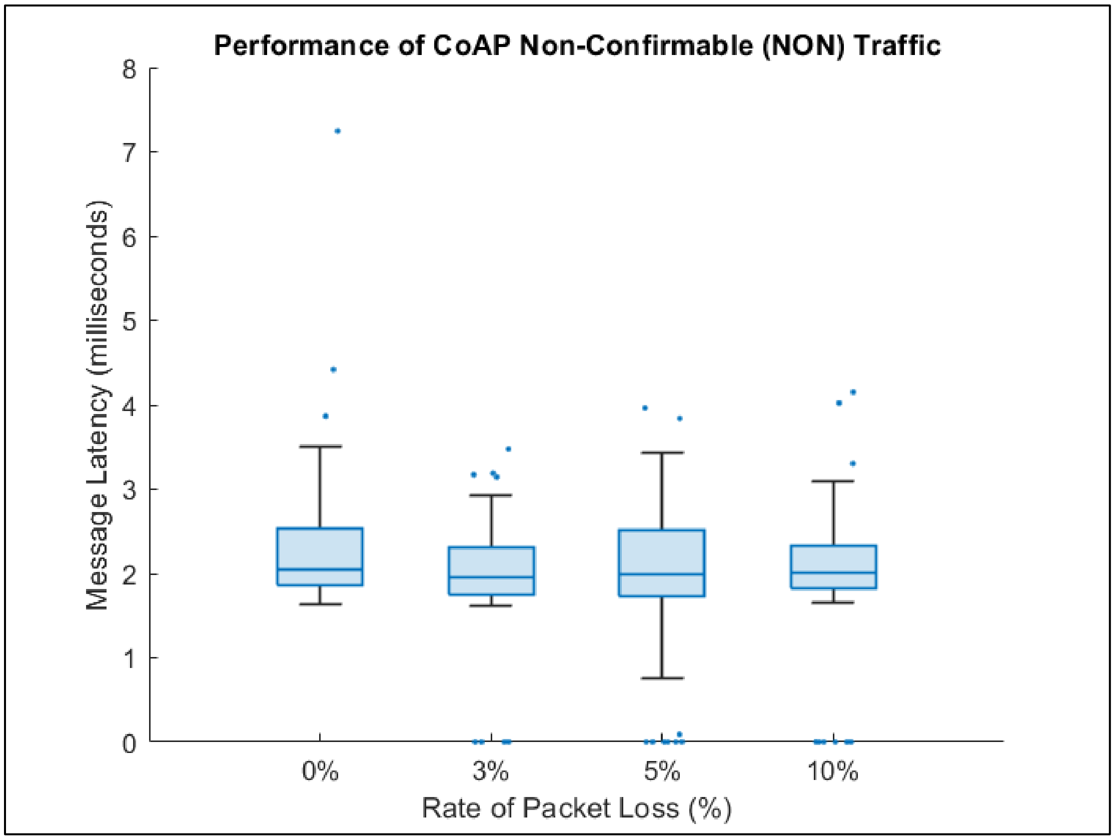


Figure 27: Performance of CoAP NON Traffic

3.3.3 Analysis of Results

We evaluate the data given within the above tables and figures to determine the relative performance of MQTT QoS 1, MQTT QoS 2, CoAP CON, and CoAP NON in terms of average message latency. In particular, we seek to understand the impact of moderate packet loss conditions on average message latency. We define moderate packet loss conditions as the presence of a uniform packet loss rate of 10%.

MQTT QoS 1 presents the lowest average message latency for all network loss conditions seen within the scope of this study. MQTT QoS 1 was found to have an average message latency of approximately 0.49 milliseconds for a lossless network connection, with the average message latency increasing to approximately 21.41 milliseconds for moderate packet loss conditions as seen in Table 7. We note that average message latency tends to increase as uniform packet loss increases, while median message latency remains relatively constant. This indicates that messages with the highest latency do not significantly impact the median latency value. Further, the presence of an average message latency value higher than the median message latency indicates a right-skewed distribution for the experimental data. While the average message latency remains within acceptable operating limits under moderate packet loss conditions, it is much higher than the scale of the box plot seen above in Figure 24.

MQTT QoS 2 presents a higher latency than MQTT QoS 1, with an average message latency of approximately 2.00 milliseconds for a lossless network connection and an average message latency of approximately 46.23 milliseconds for moderate network loss conditions as seen above in Table 8. CoAP CON displays higher average latency for both a lossless network connection and moderate network loss conditions when compared to MQTT QoS 1 and MQTT QoS 2. In particular, the average latency of CoAP CON under moderate network loss conditions is approximately 412.65 milliseconds. This suggests that the extreme latency of CoAP CON under moderate network loss makes it a poor candidate for use in a vehicular network. Similarly, the presence of multiple infinite latency events present within the experimental data of CoAP NON traffic as seen in Table 10 disqualifies this option, as we seek both reliable message delivery and low average message latency.

Given this analysis, we opt to select the use of MQTT QoS 1 for use in the baseline CAV emergency message application. MQTT QoS 1 provides the best performance in terms of average message latency when compared to the other options of MQTT QoS 2, CoAP CON, and CoAP NON. Furthermore, MQTT QoS 1 provides support for reliable message delivery, which is critical in safety systems such as the proposed CAV emergency message application.

3.4 CAV Emergency Message Application with MQTT QoS 1

We recall that the overall goal of this chapter is to implement a baseline CAV emergency message application with the most performant protocol and QoS combination found via the above study. In this section we seek to apply MQTT QoS 1 to the design and implementation of a baseline CAV emergency message application which emulates vehicular network traffic with VRU detection. Furthermore, we seek to estimate the worst-case message latency to determine whether our baseline solution provides an acceptable level of performance when compared to relevant engineering standards.

3.4.1 Overview of CAV Emergency Message Application

The use of MQTT QoS 1 provides additional benefits outside of representing the most performant option from the above study. In particular, we can leverage the EDA architecture and publisher-subscriber communication paradigm provided by MQTT QoS 1 to implement a simple version of multicast communication. We assume that each CV and AV in the local area around a traffic intersection-based RSU device operates as a MQTT client instance. Furthermore, we assume that the RSU device which hosts our baseline CAV emergency message application, which we refer to as the RSU application in this context, functions as a MQTT client instance and is co-located with the MQTT broker on the same computational platform. We can leverage the creation of topics such as Vehicle_Readouts/Connected_Vehicles and Vehicle_Readouts/Autonomous_Vehicles to enable the RSU application to subscribe to the sensor data readouts of CVs and AVs which publish to their respective topics. Likewise, we can create an Emergency/STOP_VRU topic to enable the RSU application to notify all CVs and AVs in the local area of the unexpected presence of a VRU in the roadway. If we translate these concepts into a system diagram, then we end up with a network topology for the baseline CAV emergency message application which looks like Figure 28 below.

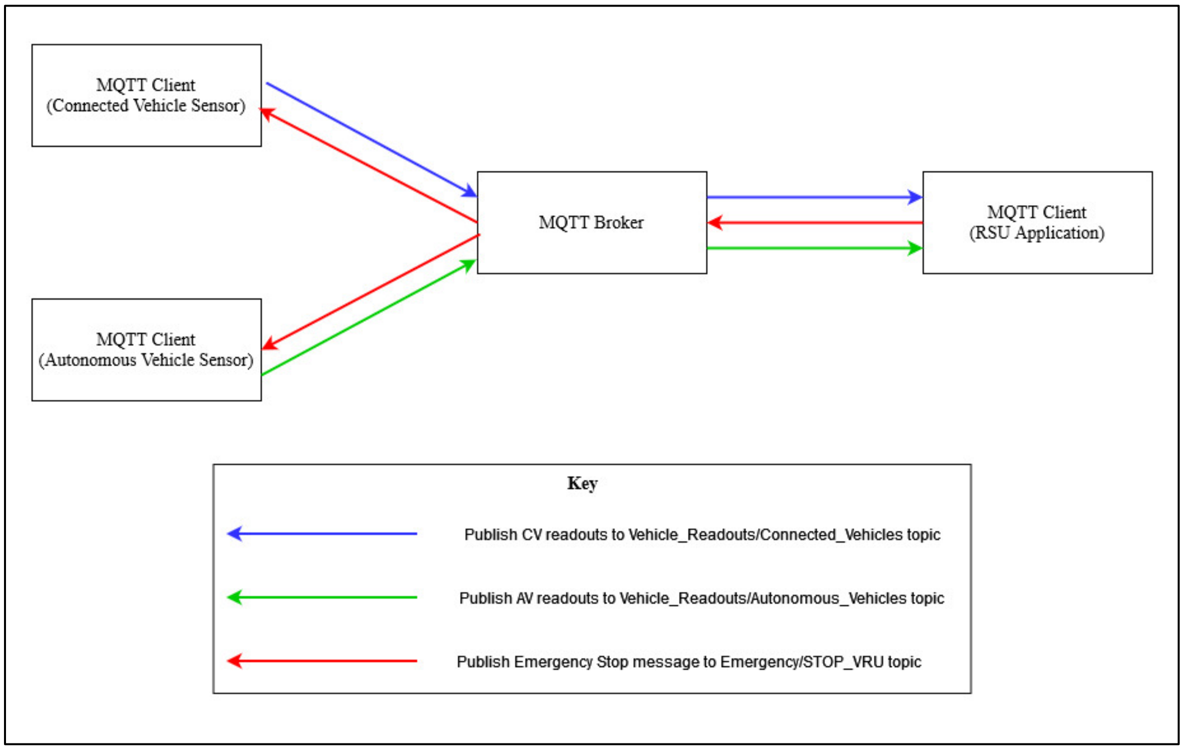
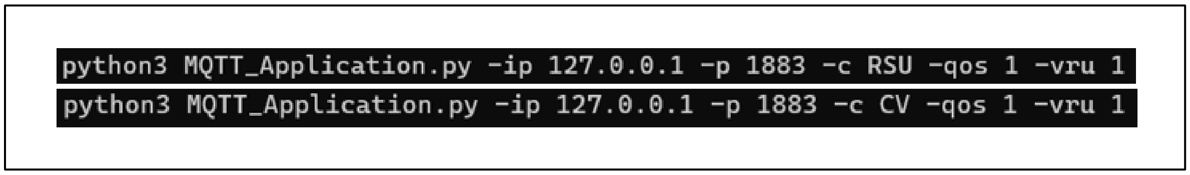

MQTT Client
(Connected Vehicle Sensor)
MQTT Broker
MQTT Client
(RSU Application)
MQTT Client
(Autonomous Vehicle Sensor)
Key
Publish CV readouts to Vehicle_Readouts/Connected_Vehicles topic
Publish AV readouts to Vehicle_Readouts/Autonomous_Vehicles topic
Publish Emergency Stop message to Emergency/STOP_VRU topic

Figure 28: System Topology for CAV Emergency Message Application

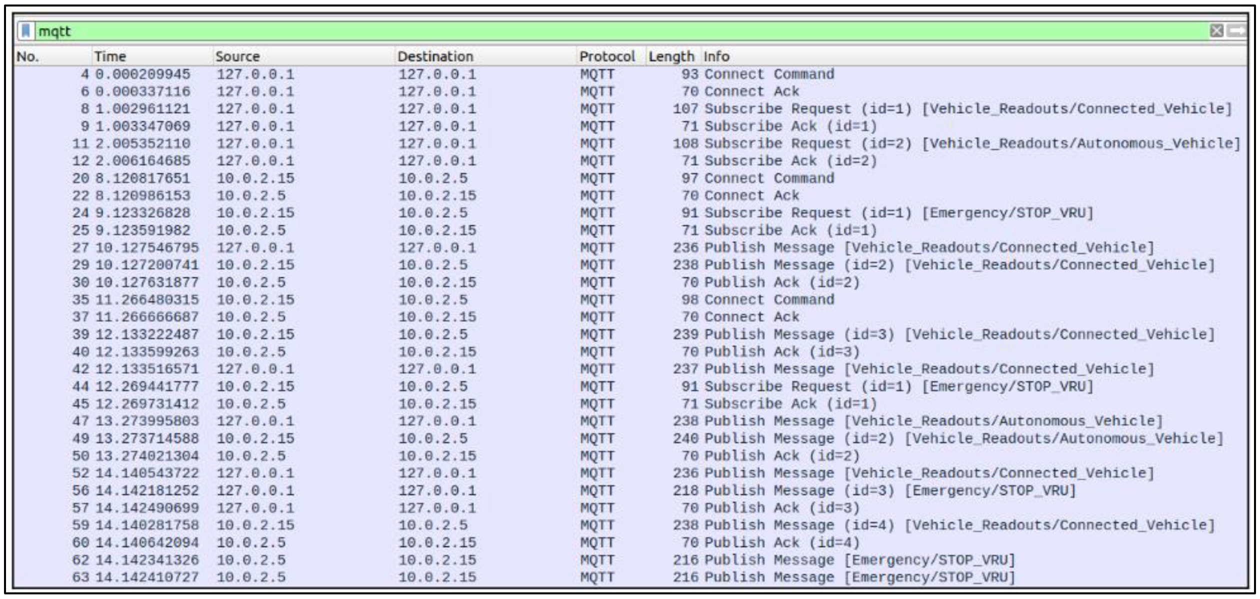
We can apply the system topology of Figure 28 to emulate the proposed CAV emergency message application in software. In particular, we can expand upon the Python code created for the MQTT QoS 1 performance test outlined above to enable the MQTT client-based RSU application to send multicast messages to all vehicles when it receives a vehicle readout with the STOP_VRU flag set to True, as outlined in the standardized sensor data readout format provided above in Table 5. We further opt to create a command line interface (CLI) which parses user inputs for use in the codebase of the baseline CAV emergency message application. An example of the CLI parameters for the MQTT client-based RSU application running MQTT QoS 1 with the potential of VRUs present in the roadway is given below in Figure 29. Similarly, an example of the CLI parameters for a MQTT client-based CV sensor running MQTT QoS 1 is given below in Figure 29. We note that the underlying Python code enables the creation of either CVs or AVs for vehicle emulation, depending on the preference of the end-user.



```
python3 MQTT_Application.py -ip 127.0.0.1 -p 1883 -c RSU -qos 1 -vru 1
python3 MQTT_Application.py -ip 127.0.0.1 -p 1883 -c CV -qos 1 -vru 1
```

Figure 29: CLI Input Parameters for RSU Application (above) and CV Sensor (below)

We provide access to the Python code for the baseline CAV emergency message application at the aforementioned public GitHub repository available at [76]. We opt to omit an in-depth overview of the Python code itself for brevity. Instead, we provide an execution example via the Wireshark trace seen below in Figure 30.



```
mqtt
No.     Time          Source      Destination  Protocol Length Info
      4 0.000209945   127.0.0.1   127.0.0.1    MQTT       93 Connect Command
      6 0.000337116   127.0.0.1   127.0.0.1    MQTT       70 Connect Ack
      8 1.002961121   127.0.0.1   127.0.0.1    MQTT      107 Subscribe Request (id=1) [Vehicle_Readouts/Connected_Vehicle]
      9 1.003347069   127.0.0.1   127.0.0.1    MQTT       71 Subscribe Ack (id=1)
     11 2.005352110   127.0.0.1   127.0.0.1    MQTT      108 Subscribe Request (id=2) [Vehicle_Readouts/Autonomous_Vehicle]
     12 2.006164685   127.0.0.1   127.0.0.1    MQTT       71 Subscribe Ack (id=2)
     20 8.120817651   10.0.2.15   10.0.2.5     MQTT       97 Connect Command
     22 8.120986153   10.0.2.5    10.0.2.15    MQTT       70 Connect Ack
     24 9.123326828   10.0.2.15   10.0.2.5     MQTT       91 Subscribe Request (id=1) [Emergency/STOP_VRU]
     25 9.123591982   10.0.2.5    10.0.2.15    MQTT       71 Subscribe Ack (id=1)
     27 10.127546795  127.0.0.1   127.0.0.1    MQTT      236 Publish Message [Vehicle_Readouts/Connected_Vehicle]
     29 10.127200741  10.0.2.15   10.0.2.5     MQTT      238 Publish Message (id=2) [Vehicle_Readouts/Connected_Vehicle]
     30 10.127631877  10.0.2.5    10.0.2.15    MQTT       70 Publish Ack (id=2)
     35 11.266480315  10.0.2.15   10.0.2.5     MQTT       98 Connect Command
     37 11.266666687  10.0.2.5    10.0.2.15    MQTT       70 Connect Ack
     39 12.133222487  10.0.2.15   10.0.2.5     MQTT      239 Publish Message (id=3) [Vehicle_Readouts/Connected_Vehicle]
     40 12.133599263  10.0.2.5    10.0.2.15    MQTT       70 Publish Ack (id=3)
     42 12.133516571  127.0.0.1   127.0.0.1    MQTT      237 Publish Message [Vehicle_Readouts/Connected_Vehicle]
     44 12.269441777  10.0.2.15   10.0.2.5     MQTT       91 Subscribe Request (id=1) [Emergency/STOP_VRU]
     45 12.269731412  10.0.2.5    10.0.2.15    MQTT       71 Subscribe Ack (id=1)
     47 13.273995803  127.0.0.1   127.0.0.1    MQTT      238 Publish Message [Vehicle_Readouts/Autonomous_Vehicle]
     49 13.273714588  10.0.2.15   10.0.2.5     MQTT      240 Publish Message (id=2) [Vehicle_Readouts/Autonomous_Vehicle]
     50 13.274021304  10.0.2.5    10.0.2.15    MQTT       70 Publish Ack (id=2)
     52 14.140543722  127.0.0.1   127.0.0.1    MQTT      236 Publish Message [Vehicle_Readouts/Connected_Vehicle]
     56 14.142181252  127.0.0.1   127.0.0.1    MQTT      218 Publish Message (id=3) [Emergency/STOP_VRU]
     57 14.142490699  127.0.0.1   127.0.0.1    MQTT       70 Publish Ack (id=3)
     59 14.140281758  10.0.2.15   10.0.2.5     MQTT      238 Publish Message (id=4) [Vehicle_Readouts/Connected_Vehicle]
     60 14.140642094  10.0.2.5    10.0.2.15    MQTT       70 Publish Ack (id=4)
     62 14.142341326  10.0.2.5    10.0.2.15    MQTT      216 Publish Message [Emergency/STOP_VRU]
     63 14.142410727  10.0.2.5    10.0.2.15    MQTT      216 Publish Message [Emergency/STOP_VRU]
```

Figure 30: Execution Example of CAV Emergency Message Application

Evaluation of the Wireshark trace given in Figure 30 shows that the execution example starts with the establishment of a MQTT broker connection for the MQTT client-based RSU application, the MQTT client-based CV sensor, and the MQTT client-based AV sensor, respectively. After a successful broker connection, the RSU application subscribes to the Vehicle_Readouts/Connected_Vehicle and Vehicle_Readouts/Autonomous_Vehicle topics, while the CV and AV each subscribe to the Emergency/STOP_VRU topic. The CV and AV then proceed to publish sensor data readouts to their respective topics, which arrives at the RSU application through subscription delivery by the MQTT broker. This process continues until the RSU application receives a sensor data readout with the STOP_VRU flag set to True, which represents the detection of a VRU in the roadway. Upon VRU detection the RSU application publishes an emergency stop message to the Emergency/STOP_VRU topic, which arrives at the CV and AV through subscription delivery by the MQTT broker. Once the CV and AV receive the emergency stop message, they both attempt to perform an emergency stop maneuver to avoid a collision with the VRU. Within the context of this example the emergency stop maneuver is represented by the termination of the software processes associated with the CV sensor and AV sensor.

Overall, this baseline CAV emergency message application provides a starting point for the development of more detailed traffic management scenarios. Next steps may include the implementation of mTLS to provide a proof-of-concept for the use of mutual authentication in the context of a CAV transportation system. Likewise, the Python code underlying the current baseline solution may be further expanded upon to include more robust features such as group decisions in support of emergency message transmission or the inclusion of a reputation score management solution. This would help to enable the operation of the baseline solution in an ad-hoc mode, which is more representative of a real-world vehicular network.

3.4.2 Estimate of Worst-Case Message Latency for CAV Emergency Message Application

The real-time nature of the baseline CAV emergency message application demands low latency message exchange between vehicular sensors and the traffic intersection-based RSU device. The need for low latency message exchange is underlined by the fact that the first recorded fatal AV-to-VRU collision involved a faulty AV which detected the presence of the VRU in the roadway 5.6 seconds before impact, yet only started to brake 1.3 seconds before fatally striking the VRU due to onboard software issues [1], [77]. If an external safety system such as the proposed CAV emergency message application is made available in such a scenario, then it could help to minimize the odds of occurrence for such fatal accidents.

Given this context, we argue that the theoretical estimation of the worst-case message latency for the proposed CAV emergency message application is of critical importance. Theoretical confirmation that the end-to-end network delay falls within acceptable limits would provide a meaningful step towards the realization of a real-world instance of the proposed CAV emergency message application. We know that end-to-end delay represents the time it takes for a given message to travel over the network from one node to another node. As such, we can define the end-to-end delay between a given vehicular sensor and the traffic intersection-based RSU device via the standard equation defined by Kurose and Ross in [78]:

$$d_{CV-RSU} = d_{proc} + d_{trans} + d_{prop} + d_{queue} \tag{1}$$

where d_{CV-RSU} is the end-to-end delay between a given vehicular sensor and the traffic intersection-based RSU device in seconds, d_{proc} is the processing delay in seconds, d_{trans} is the transmission delay in seconds, d_{prop} is the propagation delay in seconds, and d_{queue} is the queuing delay in seconds. In practice, processing delay and propagation delay are generally considered negligible, so we may ignore them here. This allows us to simplify the given equation for end-to-end delay between a given vehicular sensor and the traffic intersection-based RSU device as:

$$d_{CV-RSU} = d_{trans} + d_{queue} \tag{2}$$

where the end-to-end delay, transmission delay, and queueing delay are defined above. It takes two end-to-end message exchanges, or one RTT, for a single MQTT QoS 1 message exchange as highlighted above in Figure 11. As such, we may define the theoretical worst-case message latency for the CAV emergency message application as:

$$RTT_{max} = 2(d_{trans}) + d_{queue} \tag{3}$$

where two instances of transmission delay and one instance of queuing delay at the RSU device define the theoretical worst-case message latency for a given CV sensor to MQTT broker message. While there exists the possibility for residual latency between the MQTT broker and the RSU device, we assume this latency to be zero, as both entities exist on the same computational platform. We can therefore leverage the experimental results for MQTT QoS 1 average message latency under moderate network loss conditions as given above in Table 7 to define the transmission delay term of Equation 3 as:

$$2(d_{trans}) \approx 21.410 \text{ ms} \tag{4}$$

which allows us to effectively estimate the transmission delay term for the theoretical worst-case message latency.

We note that the current 3GPP NR 5G C-V2X technical standard requires the total message latency of applications to remain under 25 ms [79]. As such, we seek to characterize queueing delay for the proposed CAV emergency message application to ensure the RSU device, which operates in this context as a network router, can support a sufficient number of vehicles without unacceptable levels of performance degradation. We can model the queuing delay of the RSU device operating as a network router through the application of the general G/G/s queuing model as given by Hopp and Spearman in [80], which defines the following equation for average queuing delay:

$$W_q = \left(\frac{c_a^2 + c_s^2}{2}\right)\left(\frac{\rho^{\sqrt{2(s+1)}-1}}{(1-\rho)s}\right)\frac{1}{\mu} \tag{5}$$

where W_q is the average queuing delay in seconds, c_a is the coefficient of variation for interarrival times, c_s is the coefficient of variation for service times, ρ is the utilization of the servers, s is the number of servers present in the system, and μ is the service rate for the system in entities per second. The coefficient of variation for a given random variable is defined as the standard deviation divided by the mean and serves as a general measure of the variability seen within the associated process [81]. We opt to use the general queuing model provided by Equation 5, which assumes an infinite waiting room and no blocking, in an effort to calculate a metric for the queuing delay that may be seen at the RSU device under various levels of arrival rate variability for message packets sent by nearby vehicles. This approach allows us to gain a general sense of the impact of queuing delay on the proposed CAV emergency message application without the creation of a more complex queue simulation. Given this context, we can further define the utilization of servers as:

$$\rho = \frac{\lambda}{s\mu} \tag{6}$$

where λ is the arrival rate into the system in entities per second, μ is the service rate for the system in entities per second, and s is the number of servers present in the system as defined above.

We seek to characterize the theoretical worst-case queuing delay for the RSU device operating as a network router through the application of Equation 5 and Equation 6. As we anticipate the proposed CAV emergency message application to use the 3GPP 5G NR C-V2X series of technical standards [39], we may leverage the suggested parameters of these technical standards to define input values for our queueing model. The current 3GPP 5G NR C-V2X standards require a minimum transmission rate of R = 50 megabits per second (Mbps) for safety message applications and a delay of at least 5 milliseconds between safety message transmissions [82], [83]. Likewise, prior experimental results for emergency message application prototypes in C-V2X vehicular network testbeds have used a safety message size of L = 193 bytes (i.e., 1544 bits), which is roughly consistent with the packet size seen in the above experimental results of Section 3.3 [75]. Therefore, we can define the following static input parameters for our queuing model:

$$s = 1 \text{ server} \tag{7}$$

$$\lambda_{CV} = 200 \text{ messages per second} \tag{8}$$

$$n = 1, 2, 3, \dots, 160 \text{ CV sensors} \tag{9}$$

$$\lambda_{sys} = \lambda_{CV} \cdot n \text{ messages per second} \tag{10}$$

$$\mu_{RSU} = \frac{R}{L} = \frac{50\ Mbps}{1544\ bits} \approx 32{,}383 \text{ messages per second} \tag{11}$$

where $s = 1$ server denotes a single RSU device operating as a network router, $\lambda_{CV} = 200$ messages per second results from the 5G NR C-V2X standards requirement of 5 ms between safety messages, λ_{sys} messages per second represents the overall arrival rate of messages for n total CV sensors, and $\mu_{RSU} = 32{,}383$ messages per second derives from the publicly available datasheet of a modern C-V2X RSU device and current 5G NR C-V2X technical standards [84], [85]. In addition to these static input parameters, we seek to determine the potential impact of arrival rate variability on average queuing delay. As such, we define the following values for the coefficient of variation for interarrival times and coefficient of variation for service times:

$$c_a = [1.00, 1.25, 1.50] \tag{12}$$

$$c_s = 1.00 \tag{13}$$

where $c_a = 1.00$, $c_a = 1.25$, and $c_a = 1.50$ indicate moderate (low) interarrival variability, moderate (high) interarrival variability, and high interarrival variability of messages to the RSU device, respectively. Likewise, $c_s = 1.00$ indicates moderate (low) service variability of the RSU device in its operation as a network router. The various levels of arrival rate variability given by Equation 12 combined with the moderate (low) service rate variability given by Equation 13 allow us to approximate the behavior of the RSU device as a G/M/1 queue, such that we assume the router capabilities of the RSU device service a single packet at a time with exponentially distributed service times. We apply this complete set of input parameters through the creation of a MATLAB computation of queuing delay, which yields the results seen below in Figure 31 and Table 11. We provide open access to this MATLAB code through the aforementioned GitHub repository maintained by the author at [76].

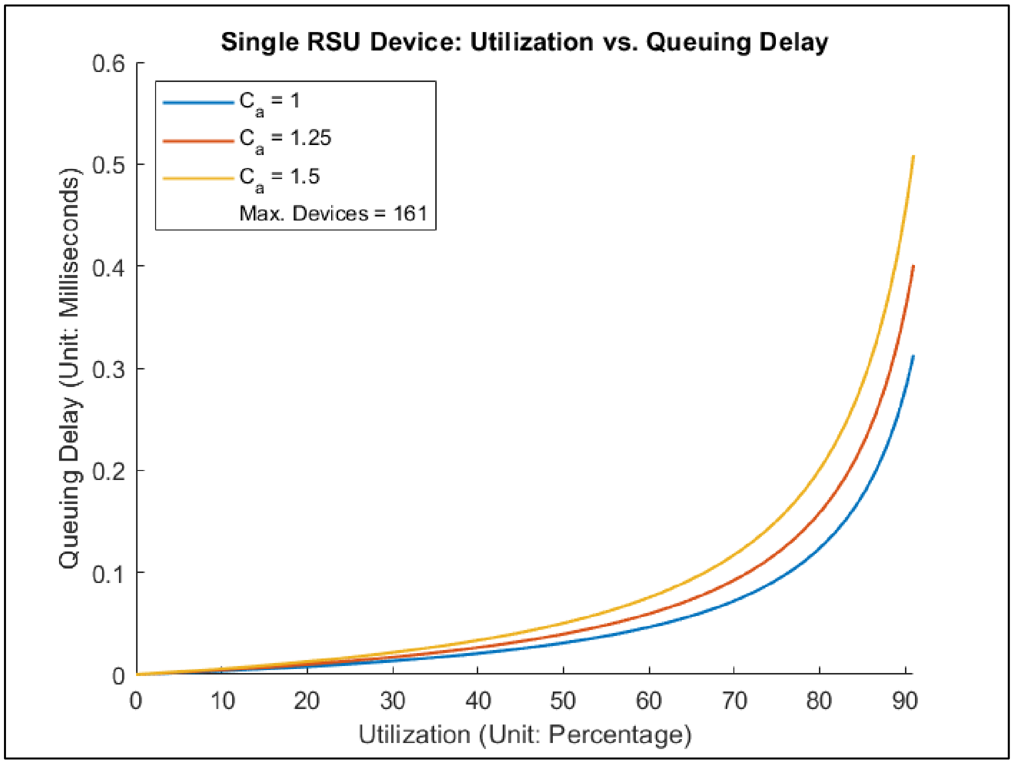


Figure 31: Utilization Versus Queuing Delay with Arrival Rate Variability

Table 11: Characterization of Queuing Delay for Variable Arrival Rates

		Arrival Rate Variability (C_a)		
Utilization (ρ)	**Vehicle Quantity**	**Moderate (Low) ($c_a = 1$)**	**Moderate (High) ($c_a = 1.25$)**	**High ($c_a = 1.50$)**
$\rho = 0.4941$	81	0.0302	0.0386	0.0490
$\rho = 0.5435$	89	0.0368	0.0471	0.0597
$\rho = 0.5929$	97	0.0450	0.0576	0.0731
$\rho = 0.6423$	105	0.0555	0.0710	0.0901
$\rho = 0.6917$	113	0.0693	0.0888	0.1126
$\rho = 0.7411$	121	0.0884	0.1133	0.1437
$\rho = 0.7905$	129	0.1165	0.1493	0.1894
$\rho = 0.8399$	137	0.1621	0.2076	0.2633
$\rho = 0.8894$	145	0.2482	0.3180	0.4033
$\rho = 0.9388$	153	0.4734	0.6066	0.7693
$\rho = 0.9820$	160	1.6844	2.1581	2.7371
$\rho = 0.9882$	161	2.5801	3.3057	4.1926

Analysis of Figure 31 highlights the trend of queuing delay trending towards exponential growth as the utilization rate approaches 100%. This is especially apparent from the point of 70% utilization rate onwards and demonstrates the fact that higher arrival rate variability correlates with a larger queuing delay, as highlighted by the yellow and red trend lines for moderate (high) and high arrival rate variability, respectively. As the utilization rate approaches 100%, the queuing delay values explode to multiple milliseconds. In particular, we note that the absolute worst-case scenario of 98.82% utilization rate with high arrival rate variability corresponds to a queueing delay of 4.193 milliseconds for 161 vehicles simultaneously connected to the RSU device. This result suggests that a maximum of 161 vehicles sending the maximum permissible message rate of 200 messages per second (see Equation 8) may be connected to the RSU device before the system collapses due to the emergence of infinite queuing delay upon the connection of the 162nd vehicle.

Given this theoretical result for worst-case queuing delay under high arrival rate variability we can characterize the theoretical worst-case latency for the CAV emergency message application using the definition Equation 3 and the round-trip latency defined by Equation 4, such that:

$$RTT_{max} = 2(d_{trans}) + d_{queue} = 21.410 \text{ ms} + 4.193 \text{ ms} \quad (14)$$

$$RTT_{max} = 25.603 \text{ ms} \quad (15)$$

which violates the upper-bound set by the current 3GPP NR 5G C-V2X technical standard, which requires the total message latency of applications to remain under 25 milliseconds as outlined above. While this indicates that the theoretical worst-case message latency does not satisfy the requirements of current technical standards, it is worth noting that the likelihood that a given RSU device must manage simultaneous connections from 161 vehicles sending the maximum permissible message rate of 200 messages per second is exceedingly low. Therefore, we can reasonably conclude that the real-world worst-case message latency will not exceed the 25 millisecond threshold, as the combined transmission delay of Equation 4 and queuing delay values of Table 11 fall below the 25 millisecond threshold for all values outside of this extreme outlier.

Given the critical nature of the safety features provided by the proposed CAV emergency message application, future transportation system designers may wish to apply a strict bound on the number of vehicles which are allowed to connect to a given RSU device in order to satisfy the requirement for a total message latency of less than 25 milliseconds. For example, the above results of Table 11 suggest that a maximum of 160 vehicles sending the maximum permissible message rate under high arrival variability may connect to a given RSU device before the impact of queueing delay drives the total message latency above the 25 millisecond limit. Such a use case highlights the potential for engineering analysis to guide policy decisions for next-generation CAV transportation system design.

3.5 Conclusion

In this chapter we present an overview of the MQTT and CoAP application layer protocols. We perform a computer-based experiment to determine the effect of network loss on message latency, and we find that each protocol exhibits high average message latency in the presence of moderate packet loss. Analysis of the experimental data for MQTT QoS 1, MQTT QoS 2, CoAP CON, and CoAP NON indicates MQTT QoS 1 to represent the best option for the baseline CAV emergency message application. Upon selection of MQTT QoS 1, we implement a baseline CAV emergency message application and highlight the utility of this solution towards improved VRU roadside safety through an execution example. We also perform a theoretical estimate of the worst-case message latency and determine that the performance satisfies the upper bound for message latency set forth by current engineering standards under most realistic operating conditions.

While the contributions of this chapter represent an initial step towards the implementation of a CAV emergency message application, the exist many opportunities for future work. Our analysis found that CoAP performs worse than MQTT within the context of this study. This result appears to disagree with the prior work of [86] which suggests that CoAP CON provides better performance than MQTT QoS 1 within the scope of edge-based vehicle service provisioning workloads. The differences in outcome between this experiment and the results presented in [86] may be due to differences in experimental approach, as the authors of [86] created a real-world vehicular network testbed while we applied a virtualized network testbed. As such, further investigation of the relative performance of MQTT and CoAP may be warranted. Additionally, the baseline CAV emergency message application requires additional development, especially as it pertains to the implementation of security features such as mutual authentication and mTLS, if such a solution is to be implemented in the context of a real-world vehicular network.

Chapter 4 – Continuous Authentication for RSU Devices

In this chapter we provide an overview of blockchain technology and highlight its relevance to the technical problems presented by continuous authentication mechanisms. We provide an overview of the Ethereum blockchain and Solidity smart contracts, which combine to serve as the basis for the proposed blockchain-based continuous authentication mechanism. We then present a blockchain-based continuous authentication mechanism which leverages a Solidity smart contract in conjunction with a private deployment of the Ethereum blockchain running the proof of authority (PoA) consensus algorithm to enforce continuous authentication for RSU devices in conjunction with a reputation score system. We conclude with an execution example of a proof-of-concept implementation of our blockchain-based continuous authentication mechanism, followed by a discussion of its limitations and opportunities future work.

4.1 Introduction

The concept of modern blockchain technology was first proposed in 2008 by the pseudonymous author Satoshi Nakamoto in a white paper which outlined the creation of a digital currency called Bitcoin [87]. Bitcoin combined concepts from prior technologies such as P2Pnetworking, distributed databases, and cryptographic hash functions to provide a public, immutable, and distributed ledger capable of authoritatively tracking all transactions on the network. Popular discourse often lumps cryptocurrency networks such as Bitcoin with the underlying blockchain technology. However, the nature of blockchain technology lends itself to many practical uses in areas outside of cryptocurrencies, such as supply chain management, secure healthcare systems, and identity management [88]. As such, the work presented here does not cover cryptocurrencies. We instead provide an overview of blockchain technology and outline how it can be applied to the development of continuous authentication mechanisms.

4.1.1 Overview of Blockchain Technology

While the concept of blockchain was first introduced by Satoshi Nakamoto in 2008, the underlying technology has undergone a significant maturation process over the past fifteen years. Modern blockchain technology combines numerous different technologies, such as cryptography, distributed databases, and P2P networks, into a unified solution capable of providing support for the distributed and autonomous execution of computing tasks. These computing tasks can be as simple as the authoritative maintenance of values on a distributed balance sheet (e.g., Bitcoin) or as complicated as the execution of a distributed application (DApp) across multiple network nodes.

Blockchain technology provides many desirable characteristics for distributed systems, such as decentralization, trust, transparency and non-repudiation [89]. These system characteristics are facilitated by the core aspects of blockchain technology: blocks of transactions, the blockchain itself, and the distributed nature in which network nodes add new blocks of transactions to the blockchain. The inherent complexity of blockchain technology means that we cannot possibly cover all the aspects of the field within the scope of this work. As such, we opt to provide a brief summary of relevant concepts within the below subsections. Alternatively, a more in-depth overview of core blockchain technology concepts can be found in textbooks such as those authored by Rehmani in [90] and Lantz and Cawrey in [91].

4.1.1.1 Blocks of Transactions and the Blockchain

Blockchain technology leverages blocks as the core data structure, which captures an immutable record of the data provided in transactions. We adopt the NIST definition of blockchain technology as ‘distributed digital ledgers of cryptographically signed transactions that are grouped into blocks’ [92, p. 1] within the scope of this work. Each block consists of a header and a series of transaction records. Most blockchains use the block header to capture key pieces of information such as the cryptographic hash value of the previous block, a time stamp, a single-use number known as a nonce, and the Merkle root hash value. The hash value of a given block helps to identify both the block and its related transactions on the blockchain, while the nonce provides a one-time value that is typically used as an input to help generate the hash value. Similarly, the timestamp denotes when in time the block was created, while the Merkle root hash value allows observers to confirm that a given transaction exists within the broader context of the Merkle tree associated with a given block. Blockchains often use a Merkle tree or similar data structure to enable the recursive hashing of multiple transactions into a single block hash, as seen below in Figure 32. This approach helps to ensure that no transaction can undergo modification without modification of the block header, which helps to guarantee the immutable nature of each block and its related transactions [93].

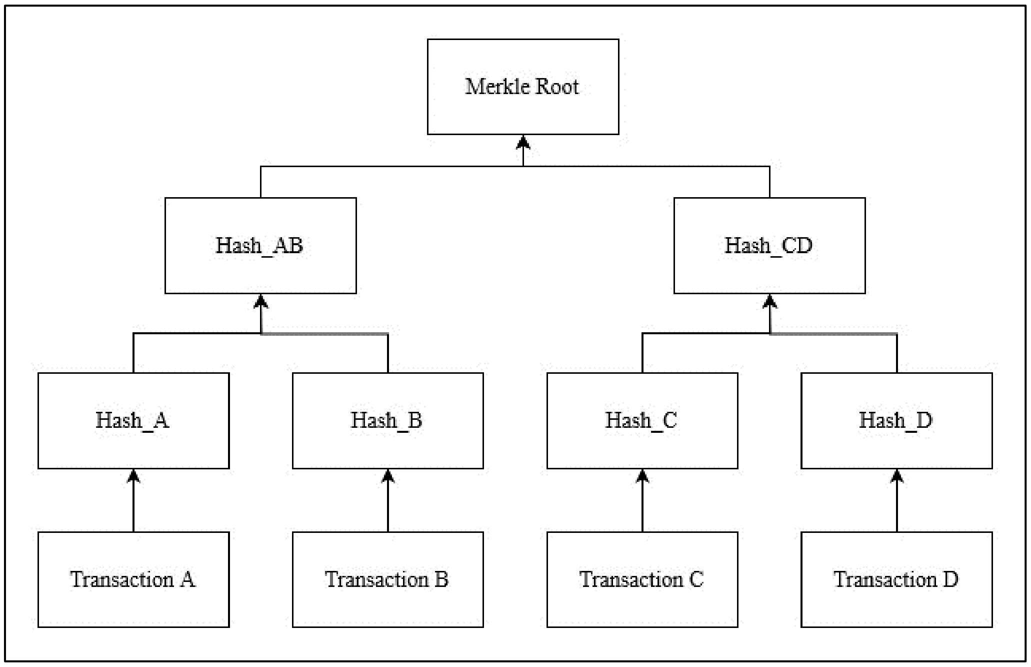


Figure 32: Example of Merkle Root Structure for Multiple Transactions

Transactions may include arbitrary data that the developer of the blockchain or associated DApp may want to capture and record in a public nature. As such, the transactions present within a given block vary based on the actions that have been carried out by the network nodes of the given blockchain deployment. Each transaction within a block represents a public record, such that the information captured within the transaction is available in plaintext to anyone who inspects the blockchain records [93]. This has important implications for the design of a blockchain-based continuous authentication mechanism. In particular, the fact that transaction data is stored on the blockchain in plaintext makes the secure storage of cryptographic secrets such as private keys extremely difficult, if not practically impossible. We therefore argue that the effective implementation of a blockchain-based continuous authentication mechanism must use a combination of on-chain and off-chain authentication factors. We elaborate on this concept below in Section 4.3. We provide a diagram of a representative blockchain, with block header and block transaction components outlined for each block, below in Figure 33.

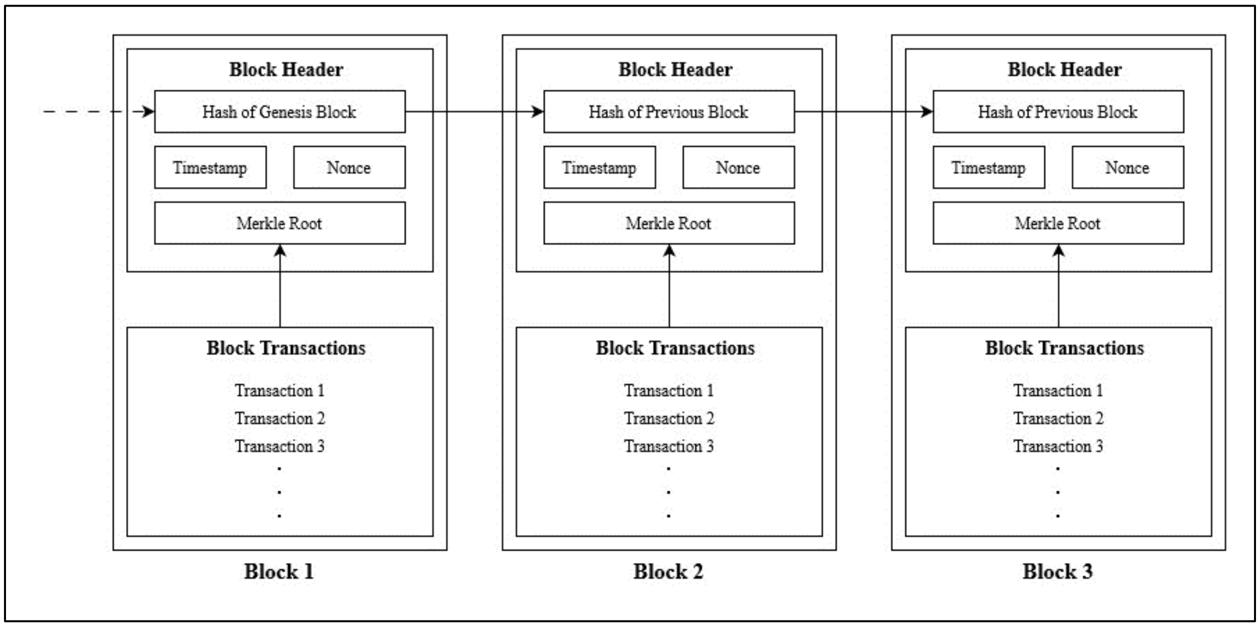


Figure 33: Overview of Process for Adding Blocks to the Blockchain

4.1.1.2 P2P Networking and Block Mining

Blockchain technology leverages P2P network architecture to enable each node to contribute resources such as computing power to the overall blockchain network. Each node consists of a computer running the core software for the given blockchain. Nodes may operate as either a user of the blockchain network or a provider of the blockchain network, without the need to pass through any intermediate authorities. Full nodes represent blockchain participants which have access to full routing, transaction storage, and block mining capabilities. Alternatively, lightweight nodes provide a basic set of functions, such as transaction history, without the need for the high levels of computational performance typically associated with full nodes. Full nodes may add new blocks of verified transactions to the blockchain through a process known as block mining. Block mining occurs at a set frequency, such as every 15 seconds in the case of Ethereum, and uses one of a number of consensus protocols, as outlined below in Section 4.1.1.3. Once a block has been successfully mined by a given node, it is propagated to other network nodes through gossiping, as is often the case in P2P network implementations [93].

4.1.1.3 Consensus Protocols: PoW, PoS, and PoA

Consensus protocols define the distributed algorithms through which full nodes add new blocks to the blockchain. While there exist many different types of consensus protocols, we opt to focus on three popular options within the scope of this work, which include Proof of Work (PoW), Proof of Stake (PoS), and PoA. We provide a brief overview of these three common consensus algorithms based on the prior work of Xiao et al. in [94].

The PoW consensus protocol, sometimes referred to as the Nakamoto consensus protocol, represents one of the key technical contributions of the original Bitcoin whitepaper. PoW enables the creation of new blocks through the use of computational power to solve an increasingly difficult mathematical puzzle. As such, the odds of a given minder successfully mining the next block are directly proportional to the amount of computational power which they contribute to the consensus process. If a given node successfully mines a block, then it receives an incentive reward such as cryptocurrency. PoW seeks to prevent successful Sybil attacks, which involve a cybersecurity threat actor gaining over half (i.e., 51% or more) of the network's computational power in an effort to hijack the blockchain. However, the computationally intensive nature of PoW leads to potential scaling issues, especially as it relates to the energy inefficiency of first-generation blockchains such the Bitcoin network.

The PoS consensus protocol seeks to address the problem of energy inefficiency in an effort to enable the development of more sustainable blockchain solutions. PoS involves mining nodes contributing a stake of cryptocurrency or similar network token to the consensus process. As with PoW, if a given node successfully mines a block, then it receives an incentive reward such as a network token. As such, the odds of a given miner successfully mining the next block are proportional to the amount of network tokens which they stake towards the consensus process. This approach leverages token ownership to mitigate the likelihood of a Sybil attack, as a cybersecurity threat actor would need to control over half of the network tokens to successfully hijack the blockchain.

The PoA consensus protocol takes an alternative approach under the assumption that mining nodes are granted the ability to create blocks through a mandatory certification process. This allows for nodes to effectively participate in the network without the need for large amounts of monetary resources or computational power. Mining nodes submit to a mandatory certification process, and receive a unique identifier once verified to the network. Proof of this verification process is the key requirement for participation in the network, and mining nodes take turns serving as the authority responsible for adding new blocks. As such, PoA mining nodes do not require an incentive structure to promote contributions to the network. If a given mining node is determined to be operating in an untrustworthy manner, then a majority vote from the other mining nodes may remove this potentially rouge node from the network. This makes the PoA consensus protocol especially useful for blockchain deployments which can effectively verify the identity of each node, such as private blockchain deployments.

4.1.1.4 Public, Private, and Consortium Blockchain Deployments

There exist three types of deployments for blockchain technology, which consist of public blockchain deployment, private blockchain deployment, and consortium blockchain deployment. Each of these deployment options provide a series of benefits and drawbacks, depending on the use case. We provide a brief overview of each of these deployment options based on the prior work of Bhutta et al. in [95].

Public blockchain deployments, sometimes called permissionless blockchain, defines a system where nodes do not require any permission to join the network. As such, any node may join the network, which can be beneficial for systems which seek to provide true decentralization, such as the Bitcoin network. This allows for new blocks to be proposed, published, and validated by any node which opts to join the network and manages to contribute sufficient resources. While public blockchain deployments benefit from decentralization, they typically suffer from low transaction throughput due to the computationally intensive consensus mechanisms which are often required to mine new blocks.

Private blockchain deployment, sometimes called permissioned blockchain, defines a system where nodes require explicit permission to join the network. As such, only nodes that have been granted a prior invitation or set of valid credentials from the entity which controls the network may join the system. While this approach sacrifices the benefits of decentralization, it is generally considered more secure than public blockchain deployment, as only known participants can join the system. Further, private blockchain deployments can often support high levels of transaction throughput due to the fact that alternative consensus mechanisms with a lower computational requirement can be used on the basis of shared trust between authorized nodes.

Consortium blockchain deployment, sometimes called federated blockchain, defines a system which combines aspects of both public blockchain deployment and private blockchain deployment. While the blockchain network itself is permissioned, a number of different organizations may add nodes to the system. As such, all parties share a base level of trust, yet it may be difficult for one party to verify the trustworthiness of any given node under ownership by another party. This approach often provides a transaction throughput speeds which are lower than private blockchain deployments, yet higher than public blockchain deployments.

4.1.1.5 Programmable Blockchain: Ethereum and Smart Contracts

While the Bitcoin network represents the first iteration of blockchain technology, there exist many different types of blockchains in use today. We opt to focus our attention within the scope of this work on the Ethereum blockchain, as it represents a viable option for use in the proposed blockchain-based continuous authentication mechanism for RSU devices due to its support of programmability through smart contracts. Ethereum, which was first proposed in late 2013 by Vitalik Buterin in a white paper [96], provides a distributed platform for the deployment of DApps. The core technical concepts underlying the Ethereum blockchain were later expanded upon in the highly technical Ethereum yellow paper by Wood [97] and the more readable Ethereum beige paper by Dameron [98]. We note that Ethereum previously supported the PoW consensus protocol [99], currently leverages the PoS consensus protocol [100], and provides ongoing support for the PoA consensus protocol in the context of private blockchain deployments [101].

Ethereum provides support for programmable blockchain features through the use of a distributed computational platform known as the Ethereum Virtual Machine (EVM), which leverages the P2P nature of blockchain networks to enable the creation of a distributed state machine, as opposed to the traditional distributed ledger of the first-generation Bitcoin network. Ethereum developers may create and execute DApps through the creation of smart contracts, which provide self-executing logic in the form of program code which exists on the EVM across the distributed network. Ethereum smart contracts are often written in high-level programming languages such as Solidity and compiled into low-level bytecode for execution via the EVM. The inception of programmable blockchains underlines the evolution of blockchain technology from a static distributed ledger system to a dynamic distributed computational platform. Further, the programmable nature of smart contracts enables the implementation of complex autonomous behaviors for Ethereum network nodes, which helps to facilitate the practical adoption of blockchain technology through an expansion of practical applications [102].

While the initial invention of smart contracts sought utility in the autonomous execution of financial transactions, we believe the characteristics of this technology make it useful for the autonomous execution of authentication decisions as well. Smart contracts provide autonomy, self-sufficiency, and a level of decentralization. As such, once a smart contract is deployed to the Ethereum network it is capable of operating autonomously for an indefinite period of time across the distributed blockchain network [102]. Kemmoe et al. [103] provide insight into the lifecycle of a smart contract through the definition of five key steps:

1. Developers create the logic of the smart contract and compile the source code into the appropriate bytecode.
2. Developers publish the smart contract via bytecode to the blockchain platform, where it will be stored and deployed on the blockchain network. Most blockchains, such as Ethereum, require that smart contracts remain read-only after deployment, such that the only way to change the logic of the smart contract is to edit the source code and redeploy a new instance of the revised bytecode.
3. Blockchain network nodes start to send transactions to the smart contract via its associated network address. Unprocessed transactions remain stored in a pool of transactions until processed by the smart contract.
4. The smart contract executes and validates transactions based on the pre-programmed logic defined by the underlying bytecode.
5. The smart contract adds validated transactions to the next block to be appended to the blockchain. If the validated transaction alters the internal variables of the smart contract, then those updated variables are considered the initial variables for the next transaction. This process of receiving, validating, and appending transactions via new blocks to the blockchain (i.e., Step 3 to Step 5) continues in an autonomous fashion for the lifetime of the smart contract.

We leverage the lifecycle of smart contracts outlined by this process to enable the creation of a blockchain-based continuous authentication mechanism for RSU devices, as outlined below in Section 4.3.

4.1.1.6 Blockchain Technology Stack: TCP/IP Model Analogy

The nature of blockchain technology lends itself to an analogous comparison with the TCP/IP model of computer networking provided above in Figure 7. While the TCP/IP model of computer networking consists of a stack of five distinct layers, the blockchain technology model proposed by Wu et al. in [93] consists of a stack of four distinct layers, which include the application layer, the consensus layer, the network layer, and the data layer. We provide an overview of this blockchain technology model via the below Figure 34. Each of these layers combine to enable the development of DApps, which help to expand the practical use cases of blockchain technology.

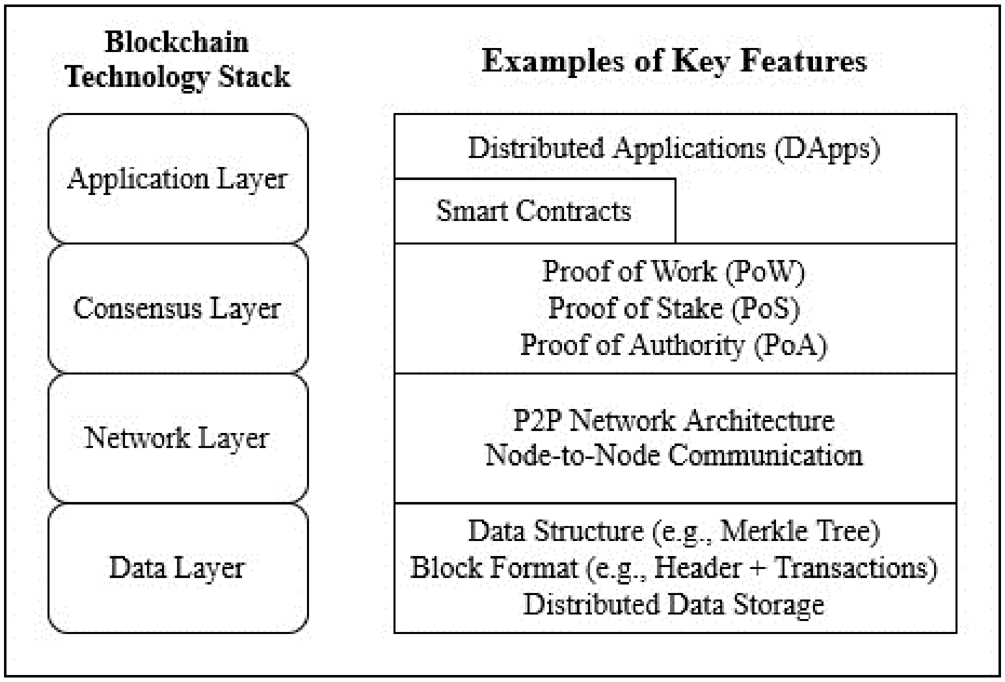


Figure 34: Overview of Blockchain Technology Stack with Key Features

Figure 34 above highlights the different capabilities provided by each layer of the blockchain technology stack. The application layer provides the programmable functionality of the blockchain, as enabled through supplemental technologies such as smart contracts. The consensus layer enables the P2P nodes found throughout a given network to append new transaction data to the blockchain through the use of consensus protocols such as PoW, PoS, and PoA. The network layer enables network entities such as mining nodes and blockchain users to interact with the overall blockchain through the P2P network paradigm. The data layer enables the efficient storage of transaction data through the use of data structures such as Merkle tree, which provides users with universal access to blockchain transactions while supporting non-repudiation [93].

4.2 Prior Uses of Blockchain Technology for Authentication Mechanisms

The viability of blockchain technology for authentication has been demonstrated in numerous manners throughout the prior literature. As referenced above in Section 1.2, prior research suggests that blockchain technology may be useful for continuous authentication in situations where a viable data set for ML-based approaches is not readily available [9], as is the case for the proposed VCA system and its related technical solutions. Given this context, we provide an overview of prior examples of the application of blockchain technology for both static authentication and continuous authentication within this section.

The Bubbles of Trust project provides a blockchain-based decentralized authentication mechanism which creates virtual zones called bubbles around groups of devices, which function as a sort of blockchain-enabled high trust zone through the application of the Ethereum blockchain [104]. Similarly, Yang et al. propose the use of a blockchain-based distributed control system with an authentication mechanism enabled by the PoA consensus protocol in support of secure electrical microgrid deployments [105]. These represent two of the more notable applications of blockchain technology for authentication within the context of IoT and M2M systems, respectively.

There also exist a number of prior blockchain-based authentication scheme proposals for VANETs and C-V2X vehicular. Lu et al. propose the creation of a blockchain-based privacy-preserving authentication (BPPA) scheme for VANETs, which leverages the Merkle Patricia Tree data structure and a permissioned blockchain deployment to extend the traditional blockchain architecture to enable the creation of a static authentication mechanism [106]. Lui et al. leverage blockchain technology in support of cooperative authentication between a single RSU device and a combination of both trusted vehicles and untrusted vehicles in the context of vehicular edge computing [107]. Such efforts highlight the viability of a blockchain-based authentication mechanism in the context of vehicular networks.

Further, there exists a number of high-quality survey papers which provide useful information on the core concepts related to the creation of blockchain-based authentication mechanisms for vehicular networks. Jabbar et al. provide an overview of the various use cases for blockchain technology within the ITS domain, including blockchain-based authentication for vehicular networks [108]. Mollah et al. provide an overview of blockchain technology as it pertains to the IoV paradigm, including a discussion of the benefits blockchain may provide for the security of vehicular networks [109]. Bagga et al. provide an overview of authentication protocols for vehicular networks, which briefly discuss the viability of blockchain-based authentication mechanisms [110].

These prior research contributions and survey papers combine to provide a baseline level of knowledge for the development of a blockchain-based continuous authentication mechanism for RSU devices within the proposed VCA system. We note that while much of the prior literature discusses the practical and theoretical applications of blockchain technology in the context of static authentication mechanisms, there appears to be very little work on blockchain-based continuous authentication mechanisms for RSU devices. This trend suggests a potential gap in the prior literature and may support a future claim of novelty for the blockchain-based continuous authentication mechanism which we outline in the following section. However, we recognize that a more comprehensive and systematic literature review is required before we can definitively claim novelty for our technical solution.

4.3 Ethereum-Based Smart Contract for Continuous Authentication of RSU

We now seek to apply the information presented above to the design of a blockchain-based continuous authentication mechanism for RSU devices in support of improved VRU safety. We recall that our core goal is to enable the continuous authentication of RSU devices, as such devices represent a single point of failure within the context of the proposed VCA system and its related CAV emergency message application. The current approaches to RSU device authentication involves a one-time, static authentication procedure with a TTP such as the SSP. While static authentication is useful for ensuring that a given entity is trustworthy at the start of a session, it is not sufficient for ongoing, real-time authentication enforcement throughout the duration of a communication session. As such, we seek to highlight the viability of a blockchain-based continuous authentication mechanism through the application of an Ethereum-based smart contract.

4.3.1 Utility of Ethereum Blockchain for Smart Contracts

We opt to leverage the Ethereum blockchain as the basis for our smart contract for continuous authentication of RSU devices. The Ethereum blockchain is inherently programmable through the use of Solidity smart contracts, as outlined above in Section 4.1.1.5. This makes

Ethereum an ideal platform for the development of a blockchain-based continuous authentication mechanism. While alternative platforms such as the Hyperledger blockchain provide better performance than Ethereum in terms of transaction latency, we opt to use the Ethereum blockchain as it provides built-on support for the PoA consensus mechanism [111], [112]. The use of the Ethereum blockchain also provides us with access to a broad range of open-source development tools, which vastly exceed the options currently available for Hyperledger due in large part to its typical use in enterprise-grade private blockchain deployments. We highlight the collaborative nature of the Ethereum development community through the use of an open-source textbook on Ethereum smart contract development provided by Antonopoulous and Wood at [113], which helps to inform the development of our smart contract for continuous authentication of RSU devices.

4.3.2 Smart Contract for Continuous Authentication of RSU Devices

We recall the reference architecture for a single VCA system instance defined above in Figure 5 of Section 2.4.1. We assume the CV, AV, and RSU device entities present within a given VCA system instance represent permissioned entities through the prior distribution of PKI certificates by the SSP. In other words, we assume that all CVs, AVs, and RSU devices present within the scope of a given VCA system instance undergo a static authentication process before the execution of the proposed blockchain-based continuous authentication mechanism. We define the separate processes of static authentication and continuous authentication via Figure 35 below.

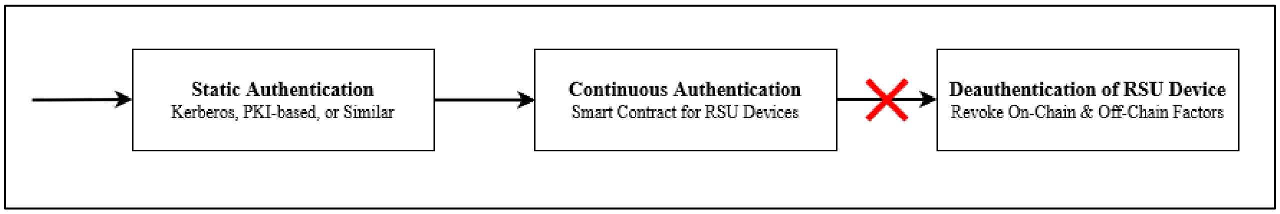

Static Authentication
Kerberos, PKI-based, or Similar
Continuous Authentication
Smart Contract for RSU Devices
Deauthentication of RSU Device
Revoke On-Chain & Off-Chain Factors

Figure 35: Overview of Authentication Lifecycle for RSU Device

Given this context, we may opt to use a private Ethereum blockchain deployment where only permissioned CV, AV, and RSU device entities alongside the SSP are allowed to participate in the smart contract. The SSP performs an initial static authentication through the enrollment of CV, AV, and RSU devices into the permissioned system, as highlighted above in Figure 35. Recall that our core focus remains the continuous authentication of traffic intersection-based RSU devices. The smart contract shall therefore focus on the continuous authentication of RSU devices. As such, we consider CVs and AVs as sensors which provide sensor data readouts to the RSU device, which itself undergoes the process of continuous authentication through the ongoing execution of the smart contract. We believe that the effective implementation of a continuous authentication mechanism for the traffic intersection-based RSU device requires the use of a system aspect which characterizes the trustworthiness of a given RSU device on an ongoing basis. As such, we propose the use of reputation scores as a way to determine whether a given RSU device is operating as a trustworthy actor which deserves ongoing access to the system.

The prior literature supports the use of reputation scores as a viable system aspect for the ongoing verification of trustworthy RSU device behavior. For example, the work of Cui et al. applies reputation scores to enable the continuous verification of messages within a vehicular network [114]. Similarly, the work of Liu et al. uses reputation scores to enable the development of a blockchain-based cooperative authentication mechanism [107]. We note that these prior efforts differ from our present effort, as they focused on the application of reputations scores for continuous message verification and cooperative device authentication, respectively, while we focus on continuous authentication for RSU devices. Further, while Liu et al. apply blockchain technology to aid cooperative device authentication, they do not explore the use of smart contracts in the context of a vehicular network authentication system. Given this context, we can reasonably assume that the proposed smart contract for continuous authentication of RSU devices has access to a proven mechanism for reputation score management.

We now shift our focus to the definition of the blockchain-based distributed system which underpins the execution of the smart contract for continuous authentication of RSU devices. We assume that the underlying network nodes consist of five traffic intersection-based RSU devices operating as full nodes alongside the SSP operating as both a full node and the sole owner of the smart contract. We note that each traffic intersection-based RSU device exists within the context of a VCA system instance. While a real-world deployment of this distributed system may consist of a greater number of full nodes, we opt to constrain this notional system deployment to five RSU devices for simplicity. We provide an overview of the network architecture for this distributed system below in Figure 36, which expands upon the initial back-end network infrastructure for the parallel deployment of multiple VCA system instances first defined above in Figure 6 of Section 2.4.1. We note that while the notional system deployment applied to the below execution example consists of five RSU devices present within individual VCA system instances, the system architecture defined below in Figure 36 consists of only three RSU devices and their associated VCA system instances. We make this modification to ensure the legibility of each of the system aspects captured within the below figure.

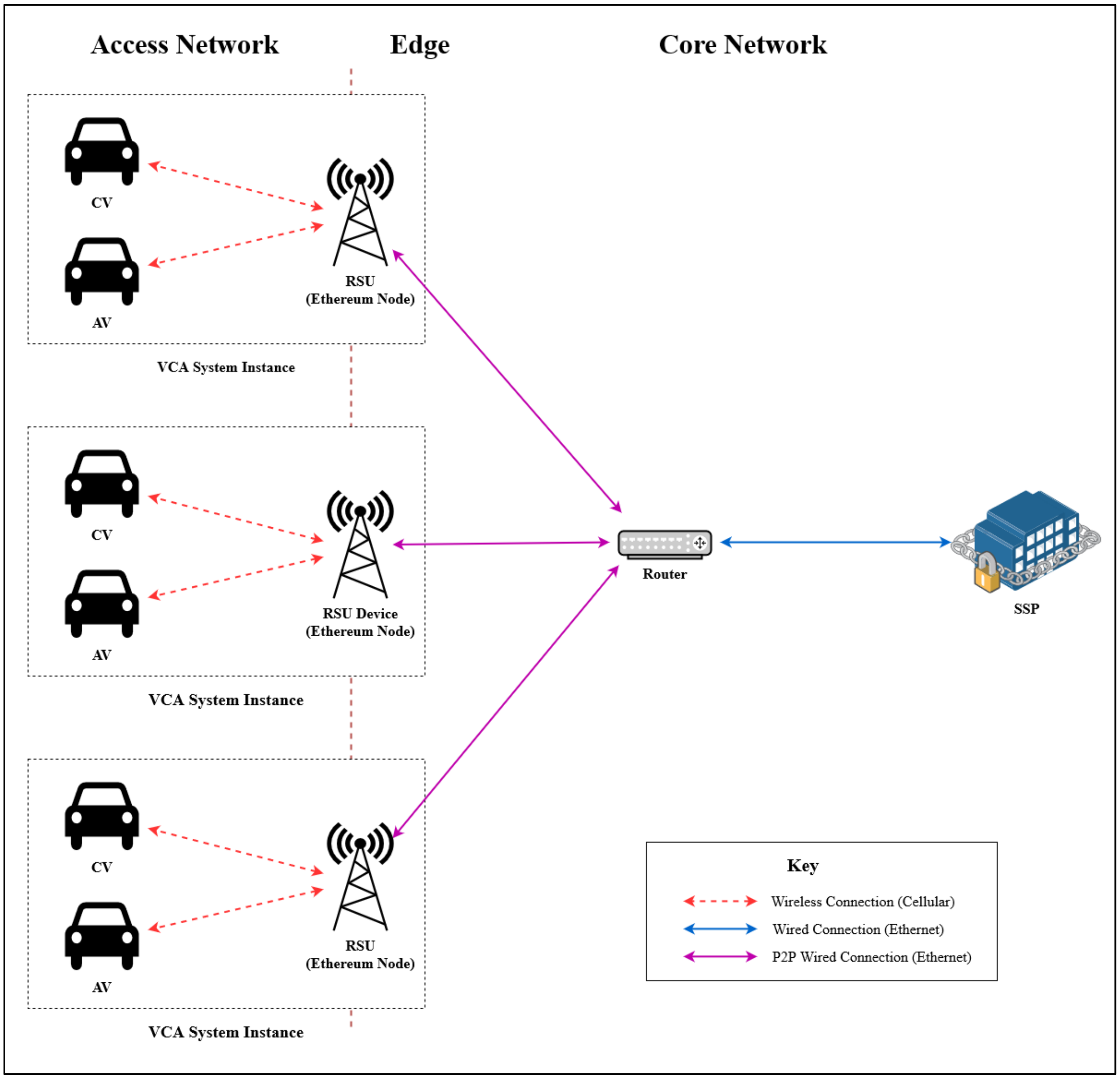


Figure 36: System-of-Systems Network Architecture for Private-Mode Ethereum Deployment

We note that each traffic intersection-based RSU device represents a known and permissioned entity within the system. This allows us to leverage the PoA consensus protocol for the addition of new blocks to the private Ethereum blockchain. We recall that PoA differs from PoW and PoS such that participant nodes in a PoA-based blockchain system take turns mining blocks based on their authority within the network, as opposed to the amount of computational resources or network tokens which they contribute to the network. In this context, the five traffic intersection-based RSU devices operate as authority nodes which have prior approval to take turn mining blocks, as granted through the initial static authentication process facilitated by the SSP, as outlined above in Figure 35.

The use of the PoA consensus mechanism pairs with the use of the reputation score system aspect to enable accountability for RSU devices. For example, if a potentially untrustworthy RSU device attempts to add an illegitimate block to the current blockchain, then the other RSU devices would mark the potentially untrustworthy RSU device for removal from the group of authority nodes. This action would prevent potentially untrustworthy RSU device from mining future blocks until a remediation process may occur. If a potentially untrustworthy RSU device is removed from the overall system, then a fail-over process would occur to enable the continued processing of impacted CV and AV sensor data readouts by a nearby trustworthy RSU device. We provide an overview of this automatic fail-over process within Figure 37 as given on the following page. We note that while the removal of a network node from the mining pool acts as a sort of deauthentication from the mining process, such an event would be considered a separate process from the continuous authentication mechanism for RSU devices itself, as the remediation of potentially untrustworthy mining nodes occurs at the consensus layer while the continuous authentication mechanism for RSU devices occurs as the application layer of the blockchain technology stack, as seen above in Figure 34 of Section 4.1.1.6.

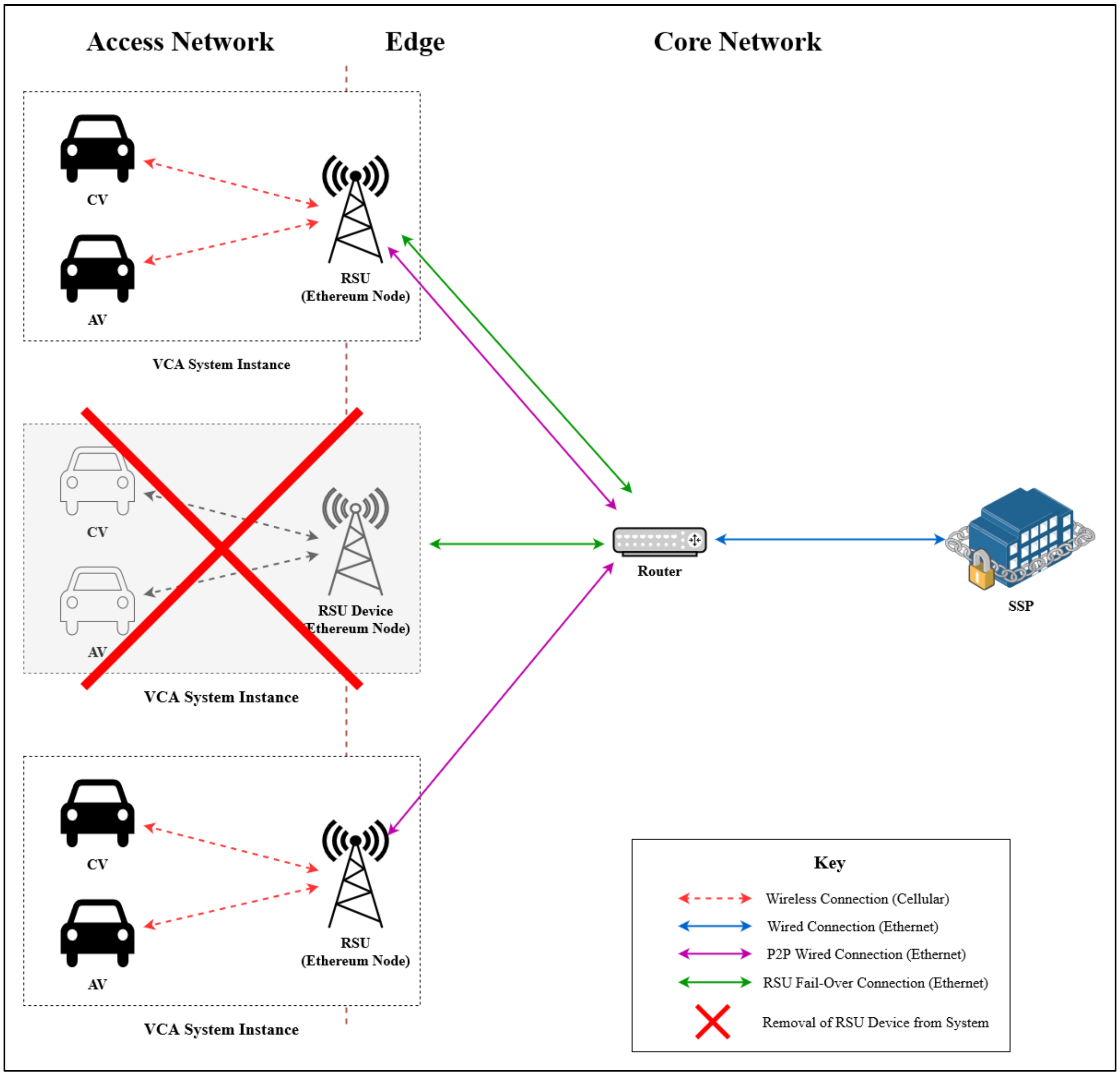


Figure 37: Automatic Fail-Over Connection Upon Removal of RSU Device

The smart contract for continuous authentication of RSU devices leverages the underlying distributed system to facilitate the ongoing authentication of RSU devices through a combination of an on-chain authentication factor and an off-chain authentication factor. We use a combination of on-chain and off-chain authentication factors to account for the fact that all data on the blockchain is public, and it is poor security practice to publish cryptographic secrets to public databases. The on-chain authentication factor consists of a reputation score which automatically updates based on the behavior of a given traffic intersection-based RSU device. For example, if a given RSU device attempts to perform an undesirable action such as the publication of an illegitimate emergency message, then the smart contract automatically decreases the reputation score associated with that RSU device. Alternatively, if a given RSU device performs a desirable action such as validating legitimate transactions, then the smart contract automatically increases the reputation score associated with that RSU device.

We use a notional reputation score mechanism which varies from 0 to 100 to represent the least trustworthy state and most trustworthy state of a given RSU device, respectively. Each RSU device starts with a notional reputation score of 100 and receives either increases of 5 for the execution of desirable actions or decreases of 20 for the execution of undesirable actions. The maximum value for the notional reputation score is 100, and the deauthentication process occurs if the notional reputation score for a given RSU device falls below the threshold of 60. The reputation score of a given RSU device represents its on-chain authentication factor, while an authentication token associated with the execution of a broader SSP-hosted application represents its off-chain authentication factor. If the reputation score for a device falls below the threshold of 60, then the Ethereum smart contract automatically revokes its on-chain authentication factor through the emission of a signal to the SSP-hosted application, which directs the revocation of the associated off-chain authentication token for the given RSU device. We provide an overview of the logic which the smart contract executes in the form of a timeline diagram below in Figure 38.

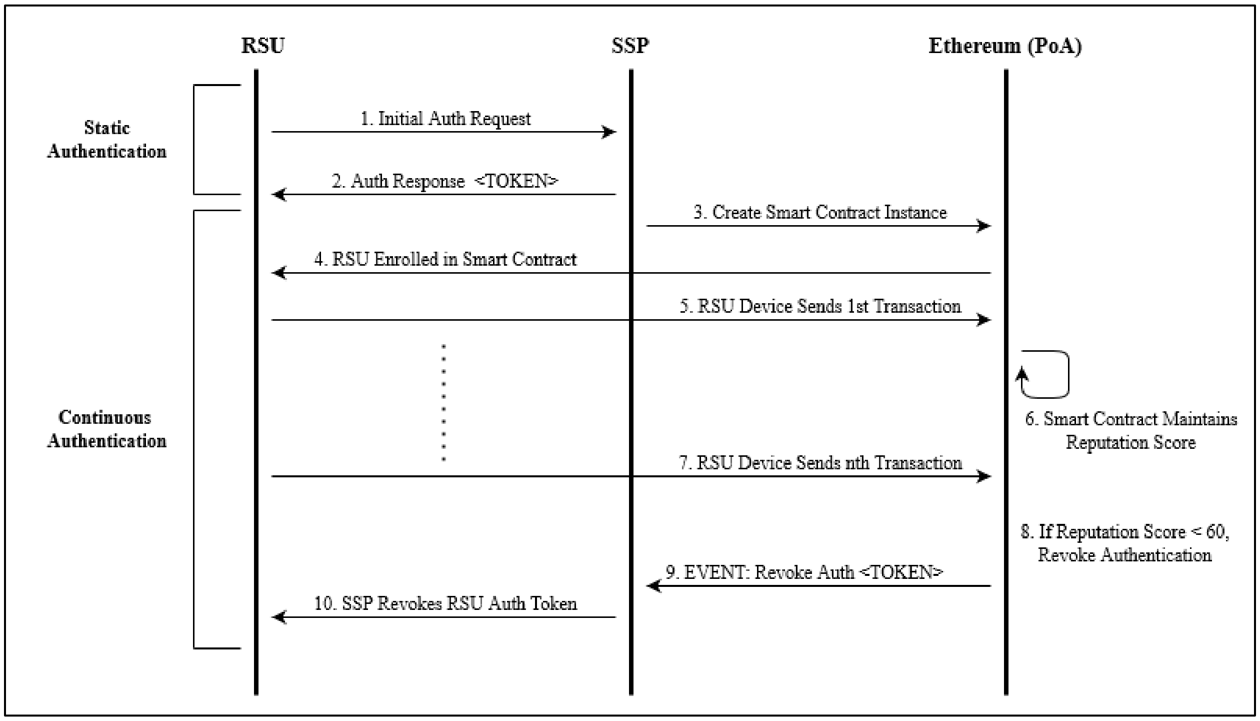

RSU
SSP
Ethereum (PoA)
Static Authentication
Continuous Authentication
1. Initial Auth Request
2. Auth Response <TOKEN>
3. Create Smart Contract Instance
4. RSU Enrolled in Smart Contract
5. RSU Device Sends 1st Transaction
6. Smart Contract Maintains Reputation Score
7. RSU Device Sends nth Transaction
8. If Reputation Score < 60, Revoke Authentication
9. EVENT: Revoke Auth <TOKEN>
10. SSP Revokes RSU Auth Token

Figure 38: Timeline Diagram of Smart Contract for Continuous Authentication

The above timeline diagram of Figure 38 provides an overview of the continuous authentication mechanism for RSU devices which is executed by the smart contract. The overall process of continuous authentication for RSU devices as executed by the smart contract can be summarized by the following steps:

1. The RSU device requests an initial authentication token from the SSP.
2. The SSP responds to the RSU, granting it initial static authentication through the issuance of an authentication token.
3. Once the SSP has the granted maximum number of RSU devices granted initial static authentication (e.g., five RSU devices in the above notional case), the SSP instantiates an instance of the smart contract for continuous authentication of RSU devices on a

private deployment of the Ethereum blockchain running the PoA consensus mechanism. This makes the SSP the sole owner of the given smart contract instance.

4. The RSU devices are enrolled into the smart contract instance when it is instantiated by the SSP. Each RSU device act as an authority node capable of mining blocks to append to the blockchain.
5. The RSU devices send initial transaction(s) to the smart contract in the form of sensor data readouts received from nearby vehicles, RSU actuation messages, and other relevant actions.
6. The smart contract autonomously maintains the reputation score of each RSU device. When an RSU device sends a transaction to the smart contract, the smart contract processes it, determines whether the action of the RSU device is appropriate or inappropriate, and adjusts the reputation score of the RSU device according to the pre-programmed logic of the smart contract.
7. A given RSU device sends the nth transaction to the smart contract, which represents the final transaction sent by the RSU device before its associated reputation score drops below the allowable threshold.
8. Upon processing of the nth transaction of the given RSU device, the smart contract determines that the reputation score of the RSU device has fallen below the allowable threshold of 60. This triggers the smart contract to revoke the on-chain authentication of the RSU device, which results in the emission of an event signal to the associated application hosted by SSP.

9. The SSP receives the event signal from the smart contract, which contains the Ethereum address of the untrustworthy RSU device. The SSP deauthenticates the RSU device through the revocation of its off-chain authentication token.
10. The untrustworthy RSU device is deauthenticated by the SSP, which prevents it from sending any other transactions to the smart contract.

We note that the deauthentication of an untrustworthy traffic intersection-based RSU device negatively impacts the ability of the associated VCA system instance to promote improved VRU safety. As such, if a given RSU device is subject to removal from the system, then the given RSU device should fail open such that it forwards any sensor data readouts received from nearby CVs and AVs to a nearby trustworthy RSU device for processing via execution of the smart contract. We provide an overview of this open fail-over process within Figure 37 above for reference. While this approach to the removal of a given RSU device from the system represents a viable workaround, the technical implementation of this solution falls outside the scope of the current effort. We instead adopt the assumption that the deauthentication of an RSU device results in the desired open fail-over behavior to ensure that the impacted VCA system instance may continue to operate as intended.

4.4 Execution Example of Continuous Authentication for RSU Devices

As in Chapter 3, we opt to provide an execution example of the smart contract for continuous authentication of RSU devices in lieu of a theoretical discussion of the underlying codebase. The Solidity code associated with this execution example can be found in the aforementioned public GitHub repository at [76]. We leverage a series of open-source blockchain development tools provided by the Truffle Suite for this execution example. The Truffle Suite consists of two main modules: the Truffle module, which provides a set of tools for Ethereum smart contract development, and the Ganache module, which provides a virtualization environment for the local private deployment of the Ethereum blockchain to aid in the validation of smart contracts. We use the documentation provided by the Truffle Suite to guide this execution example, which can be found online at [115].

We download and install the current version of Truffle Suite, which consists of the Truffle v5.11.5, Ganache v7.9.1, Solidity v0.5.16, Node v20.8.0, and Web3.js v1.10.0 software packages. We also install the Ganache GUI v2.7.1 client to aid in the visualization of blockchain transactions. We note that the Truffle v5.11.5 and Ganache v7.9.1 packages represent the core contributions of the Truffle Suite, while the Solidity v0.5.16 package provides a Solidity complier to translate the smart contract code into EVM bytecode. The Node v20.8.0 package combines with Web3.js v1.10.0 package to provide necessary JavaScript dependencies for the core Truffle Suite packages.

We begin the execution example with the initialization of a Windows command prompt instance with administrator privileges on our local host machine running the Windows 11 Home Build 22621 operating system. We use this command prompt instance to perform the truffle init command, which creates a new Truffle project as seen below in Figure 39.

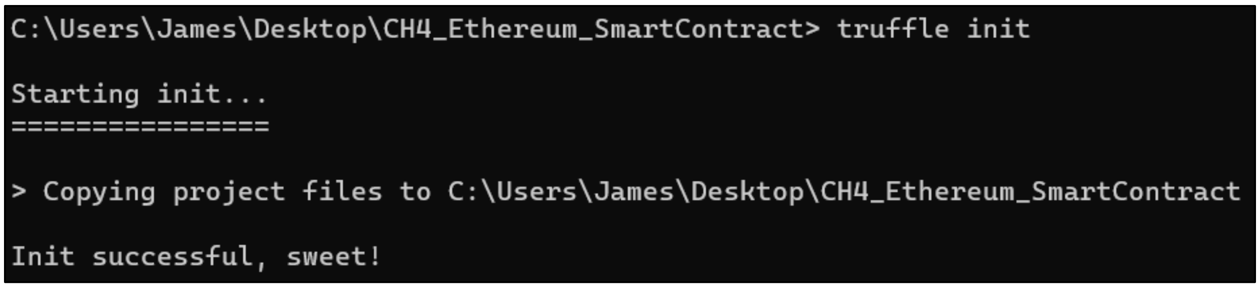
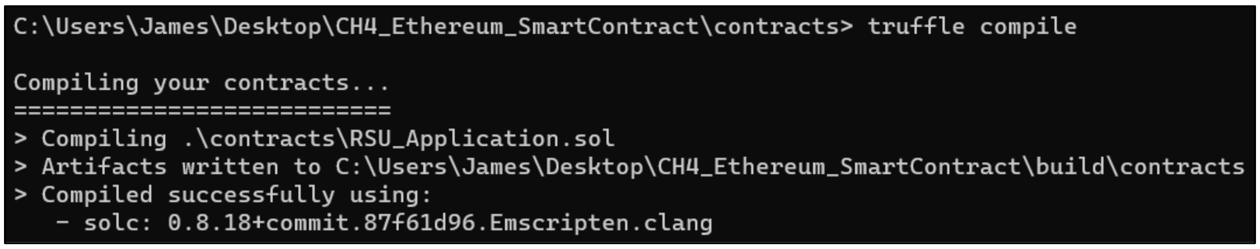

```
C:\Users\James\Desktop\CH4_Ethereum_SmartContract> truffle init

Starting init...
================

> Copying project files to C:\Users\James\Desktop\CH4_Ethereum_SmartContract

Init successful, sweet!
```

Figure 39: Initialization of Truffle Project for Smart Contract

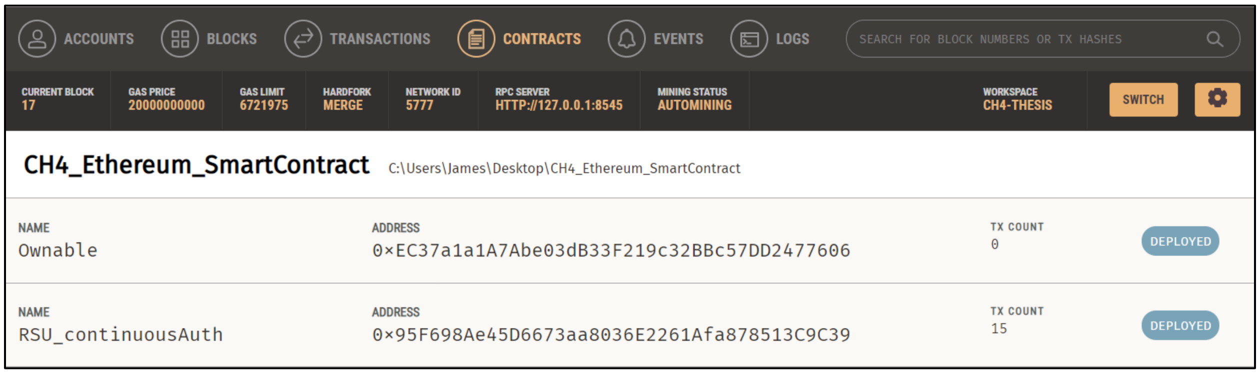
We then import the RSU_Application.sol smart contract file found within our public GitHub repository at [76] into the /contracts/ directory associated with our Truffle project. This allows us to translate the high-level Solidity code of the smart contract into low-level EVM bytecode through the use of the Solidity compiler, which we accomplish through the use of the truffle compile command as seen below in Figure 40. We note that compilation of the smart contract requires the Solidity v0.8.18 compiler, which in turn requires a minor modification to the configuration file associated with our Truffle project.



```
C:\Users\James\Desktop\CH4_Ethereum_SmartContract\contracts> truffle compile

Compiling your contracts...
===========================
> Compiling .\contracts\RSU_Application.sol
> Artifacts written to C:\Users\James\Desktop\CH4_Ethereum_SmartContract\build\contracts
> Compiled successfully using:
   - solc: 0.8.18+commit.87f61d96.Emscripten.clang
```

Figure 40: Compilation of Smart Contract with Solidity Compiler

After successful compilation of the smart contract into EVM bytecode we create a basic migration script, which allows the Truffle v5.11.5 package to deploy the smart contract bytecode to a local instance of the Ganache v7.9.1 virtualized blockchain, which provides a private deployment of the Ethereum blockchain with 10 mining nodes running the PoW consensus protocol. While the smart contract for continuous authentication solution proposed above in Section 4.3.2 calls for the use of the PoA consensus protocol, we use the PoW consensus protocol within this context as the Truffle Suite does not yet provide native support for the PoA consensus protocol. As such, we deploy the smart contract to the local Ganache blockchain instance with the understanding that the focus of this execution example is the demonstration of the core smart contract functionality at the application layer of the blockchain technology stack, as opposed to the already well understood consensus protocols available at the lower-level consensus layer. Given this context, we launch the Ganache GUI v2.7.1 client and import our Truffle project configuration file, which allows us to deploy the smart contract bytecode to the Ganache virtualized blockchain via the truffle migrate command. The execution of this command deploys the RSU_continuousAuth smart contract to the Ganache virtualized blockchain, as seen below in Figure 41.



CURRENT BLOCK 17 | GAS PRICE 20000000000 | GAS LIMIT 6721975 | HARDFORK MERGE | NETWORK ID 5777 | RPC SERVER HTTP://127.0.0.1:8545 | MINING STATUS AUTOMINING | WORKSPACE CH4-THESIS

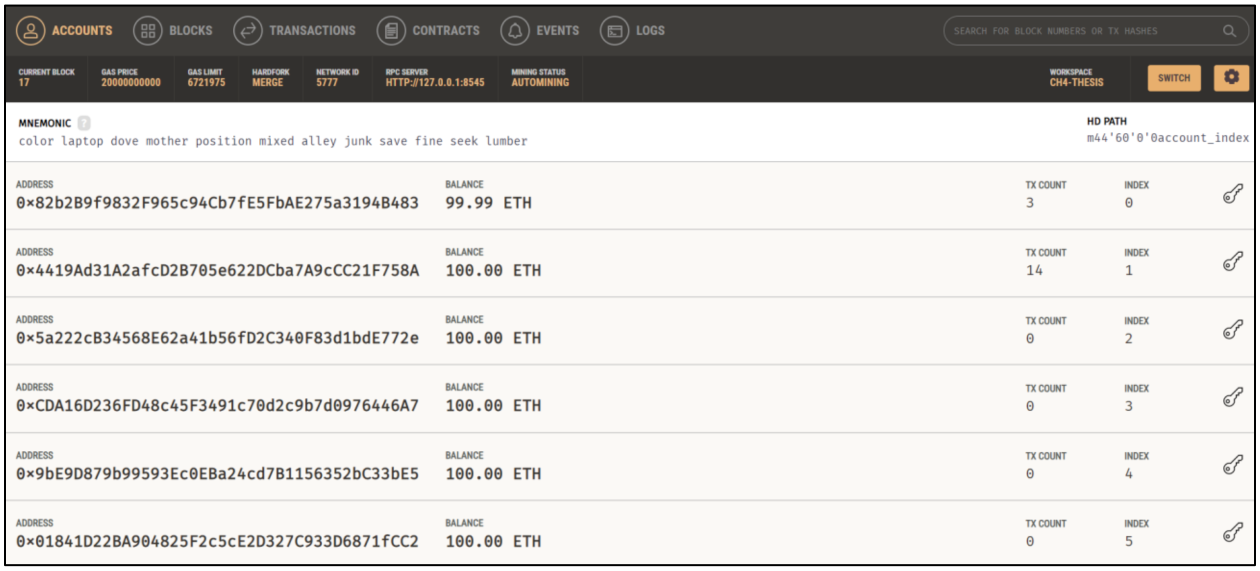
CH4_Ethereum_SmartContract C:\Users\James\Desktop\CH4_Ethereum_SmartContract

NAME	ADDRESS	TX COUNT	
Ownable	0×EC37a1a1A7Abe03dB33F219c32BBc57DD2477606	0	DEPLOYED
RSU_continuousAuth	0×95F698Ae45D6673aa8036E2261Afa878513C9C39	15	DEPLOYED

Figure 41: Deployment of Smart Contracts to Ganache Virtualized Blockchain

We note that the Ownable smart contract seen above in Figure 41 provides a mechanism for the Ethereum node which deploys the smart contract to become its sole owner, thus providing a basic level of security against tampering by external parties. The RSU_continuousAuth smart contract leverages inheritance through the object-oriented nature of the Solidity programming language to gain access to the functionality of the Ownable smart contract. This enables Ethereum node associated with the SSP to become the sole owner of the RSU_continuousAuth smart contract upon deployment, which helps to prevent external cybersecurity threat actors from attempting to spoof contract ownership.

We demonstrate SSP ownership of the smart contract through deployment at the Ethereum node associated with address index 0, as seen below in Figure 42. We also note that the Ethereum nodes associated with address index 1 through address index 5 seen below in Figure 42 represent the five RSU devices acting as mining nodes on the blockchain. This aligns with the network architecture provided above in Figure 36 and allows us to emulate the proposed system-of-systems network architecture using the RSU devices present within each VCA system instance as mining nodes in this context.



CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
17	20000000000	6721975	MERGE	5777	HTTP://127.0.0.1:8545	AUTOMINING	CH4-THESIS

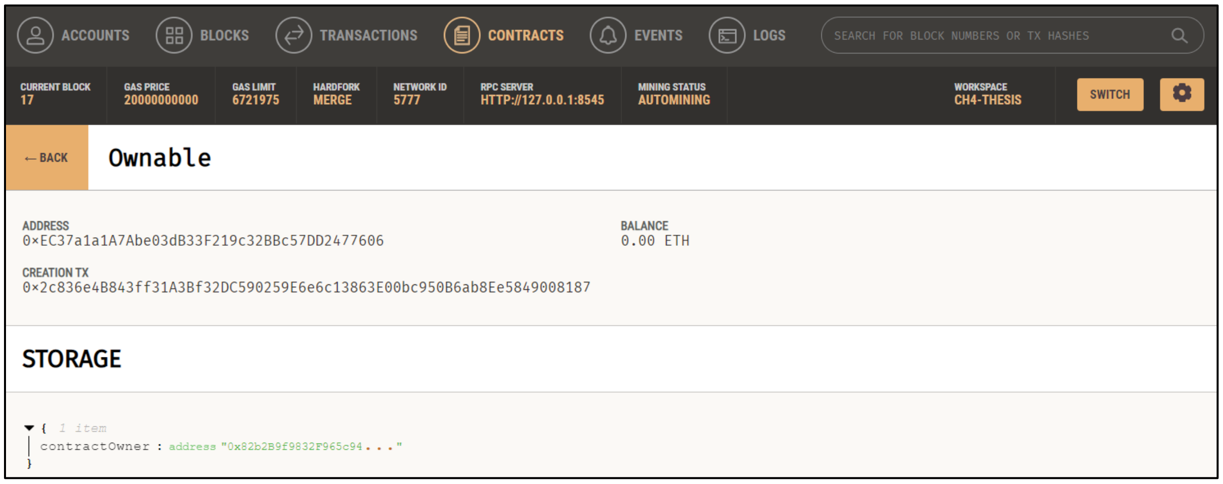
MNEMONIC
color laptop dove mother position mixed alley junk save fine seek lumber

HD PATH
m44'60'0'0account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×82b2B9f9832F965c94Cb7fE5FbAE275a3194B483	99.99 ETH	3	0
0×4419Ad31A2afcD2B705e622DCba7A9cCC21F758A	100.00 ETH	14	1
0×5a222cB34568E62a41b56fD2C340F83d1bdE772e	100.00 ETH	0	2
0×CDA16D236FD48c45F3491c70d2c9b7d0976446A7	100.00 ETH	0	3
0×9bE9D879b99593Ec0EBa24cd7B1156352bC33bE5	100.00 ETH	0	4
0×01841D22BA904825F2c5cE2D327C933D6871fCC2	100.00 ETH	0	5

Figure 42: Overview of Ethereum Addresses for SSP (Index 0) and RSU Devices

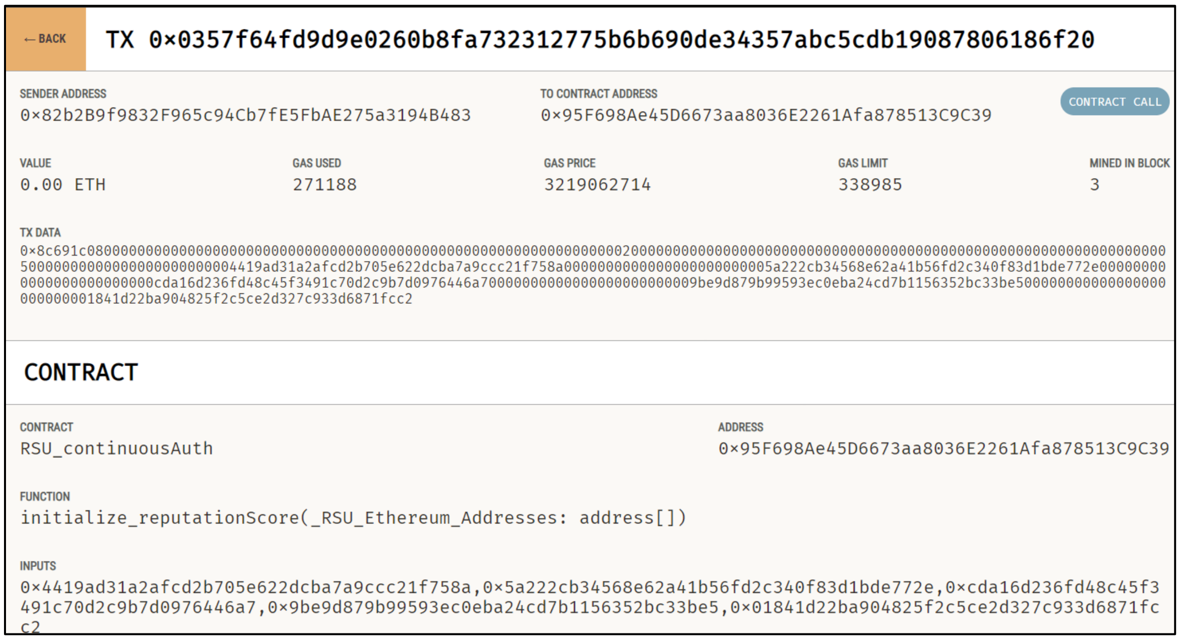
This approach allows us to demonstrate the execution of Step 3 of the smart contract timeline given in Figure 38 above, with the assumption that Step 1 and Step 2 occur before the deployment of the smart contract due to the fact that the SSP grants prior permission to RSU devices with an initial round of static authentication before gaining access to the smart contract aspect of the overall system. We highlight SSP ownership of the RSU_continuousAuth smart contract instance through the storage value associated with the RSU_continuousAuth smart contract, as seen below in Figure 43.



```
CURRENT BLOCK 17
GAS PRICE 20000000000
GAS LIMIT 6721975
HARDFORK MERGE
NETWORK ID 5777
RPC SERVER HTTP://127.0.0.1:8545
MINING STATUS AUTOMINING
WORKSPACE CH4-THESIS

Ownable

ADDRESS
0×EC37a1a1A7Abe03dB33F219c32BBc57DD2477606
BALANCE
0.00 ETH
CREATION TX
0×2c836e4B843ff31A3Bf32DC590259E6e6c13863E00bc950B6ab8Ee5849008187

STORAGE

▼ { 1 item
  contractOwner : address "0x82b2B9f9832F965c94..."
}
```

Figure 43: Demonstration of SSP Ownership of the RSU_continuousAuth Smart Contract

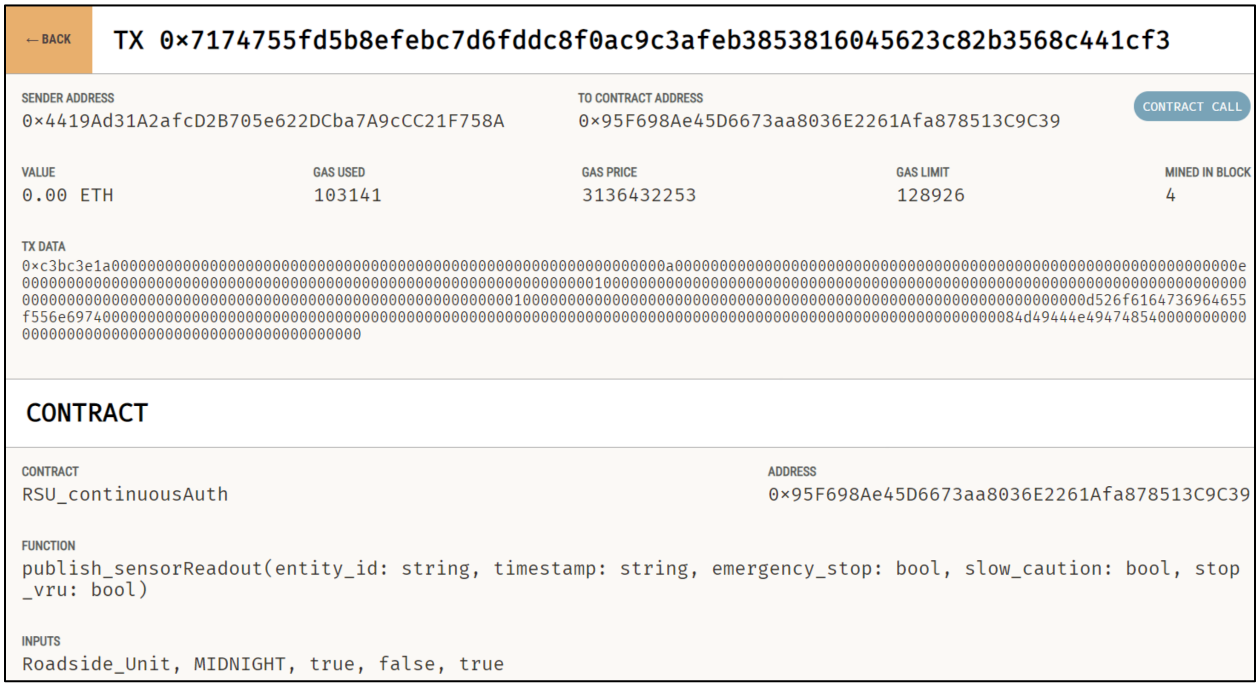
The establishment of the smart contract instance enables us to generate sample transactions. We leverage the development tools of the Truffle v.5.11.5 package to interact with the smart contract. As such, we initialize reputation score method of the smart contract to set the initial reputation score of each of the five RSU devices to 100, as given below in Figure 44.



```
TX 0×0357f64fd9d9e0260b8fa732312775b6b690de34357abc5cdb19087806186f20

SENDER ADDRESS
0×82b2B9f9832F965c94Cb7fE5FbAE275a3194B483
TO CONTRACT ADDRESS
0×95F698Ae45D6673aa8036E2261Afa878513C9C39

VALUE      GAS USED   GAS PRICE    GAS LIMIT   MINED IN BLOCK
0.00 ETH   271188     3219062714   338985      3

TX DATA
0×8c691c0800000000000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000000000000000
5000000000000000000000000004419ad31a2afcd2b705e622dcba7a9ccc21f758a0000000000000000000000005a222cb34568e62a41b56fd2c340f83d1bde772e00000000
0000000000000000cda16d236fd48c45f3491c70d2c9b7d0976446a70000000000000000000000009be9d879b99593ec0eba24cd7b1156352bc33be500000000000000000
000000001841d22ba904825f2c5ce2d327c933d6871fcc2

CONTRACT

CONTRACT
RSU_continuousAuth
ADDRESS
0×95F698Ae45D6673aa8036E2261Afa878513C9C39

FUNCTION
initialize_reputationScore(_RSU_Ethereum_Addresses: address[])

INPUTS
0×4419ad31a2afcd2b705e622dcba7a9ccc21f758a,0×5a222cb34568e62a41b56fd2c340f83d1bde772e,0×cda16d236fd48c45f3
491c70d2c9b7d0976446a7,0×9be9d879b99593ec0eba24cd7b1156352bc33be5,0×01841d22ba904825f2c5ce2d327c933d6871fc
c2
```

Figure 44: Initialization of Reputation Scores for RSU Devices

The initialization of reputation scores for RSU devices corresponds to the execution of Step 4 of the smart contract timeline given above in Figure 38, with each of the five RSU devices enrolled in the smart contract through the assignment of an initial reputation score. This allows us to demonstrate the creation of transactions for individual RSU devices, which corresponds to Step 5 of the smart contract timeline given above in Figure 38. We opt to focus on the generation of transactions for the RSU device associated with address index 1 to demonstrate the core functionality of the smart contract. In particular, we generate an invalid RSU device message as seen below in Figure 45, which leads to a decrease of the reputation score for the given RSU device from 100 to 80.


← BACK

TX 0×7174755fd5b8efebc7d6fddc8f0ac9c3afeb3853816045623c82b3568c441cf3

SENDER ADDRESS
0×4419Ad31A2afcD2B705e622DCba7A9cCC21F758A

TO CONTRACT ADDRESS
0×95F698Ae45D6673aa8036E2261Afa878513C9C39

CONTRACT CALL

VALUE	GAS USED	GAS PRICE	GAS LIMIT	MINED IN BLOCK
0.00 ETH	103141	3136432253	128926	4

TX DATA
```
0×c3bc3e1a00000000000000000000000000000000000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000000e
000000000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000d526f6164736964655
f556e697400000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000084d49444e494748540000000000
00000000000000000000000000000000000000
```

CONTRACT

CONTRACT
RSU_continuousAuth

ADDRESS
0×95F698Ae45D6673aa8036E2261Afa878513C9C39

FUNCTION
publish_sensorReadout(entity_id: string, timestamp: string, emergency_stop: bool, slow_caution: bool, stop_vru: bool)

INPUTS
Roadside_Unit, MIDNIGHT, true, false, true

Figure 45: Smart Contract Transaction with Invalid RSU Device Message

We then send a number of valid RSU device messages, which we omit here for brevity. This corresponds to Step 6 of the smart contract timeline given above in Figure 38, where RSU devices send transactions while the smart contract autonomously maintains the reputation score of each RSU device. We terminate the execution example with the transmission of two invalid messages from the RSU device similar to that of Figure 45 above, which forces the deauthentication of the RSU device once its reputation score falls below the threshold of 60. This process corresponds to Step 7, Step 8, and Step 9 of the smart contract timeline given above in Figure 38, where the RSU device sends its final transaction, the smart contract determines that the reputation score of the RSU device has fallen below the threshold of 60, and the smart contract revokes the on-chain authentication through removal of the RSU device from the pool of Ethereum mining nodes, respectively. The revocation of the on-chain authentication of the RSU device corresponds to the emission of an event signal by the smart contract, which an off-chain application hosted by the SSP then uses to fully deauthenticate the RSU device. This action corresponds to Step 10 of the smart contract timeline given above in Figure 38. We provide an overview of the on-chain authentication revocation event as seen in the Ganache GUI client below in Figure 46.

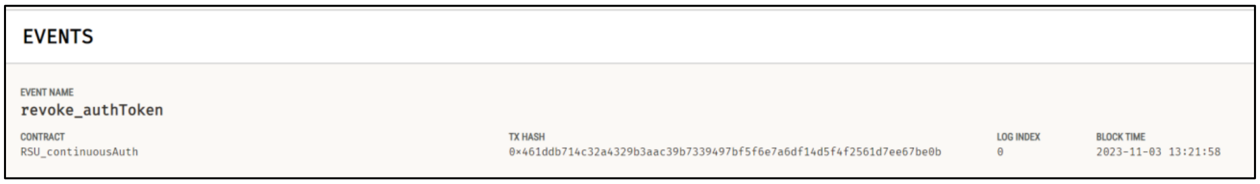

```
EVENTS

EVENT NAME
revoke_authToken
CONTRACT                TX HASH                                                             LOG INDEX   BLOCK TIME
RSU_continuousAuth      0×461ddb714c32a4329b3aac39b7339497bf5f6e7a6df14d5f4f2561d7ee67be0b  0           2023-11-03 13:21:58
```

Figure 46: Emission of Event Signal for On-Chain Authentication Revocation

This execution example provides an overview of the smart contract functionality required to enable continuous authentication of the RSU device. This execution example further highlights the baseline solution implementation for the theoretical smart contract. Overall, we believe this initial effort provides a foundation for future development related to the implementation of blockchain-based continuous authentication for RSU devices.

4.5 Conclusion

In this chapter we present an overview of blockchain technology as it pertains to the development of a blockchain-based continuous authentication mechanism. We then highlight the utility of Ethereum-based smart contracts for the execution of autonomous computing tasks, which enables the development of a smart contract for continuous authentication of RSU devices. We outline an initial proposal for a blockchain-based continuous authentication mechanism for RSU devices and highlight the viability of this smart contract-based solution through an execution example.

Despite the initial contributions presented within this chapter there exist many opportunities for future work. The current work does not consider the potential computational constraints that may be present in such a blockchain-based solution. In particular, the above presentation of the smart contract for continuous authentication of RSU devices does not analyze the transaction throughput which the smart contract or underlying private-mode Ethereum network may be able to support. Similarly, the current work does not analyze the potential storage requirements associated with such a solution. These gaps should be addressed in order to enable the deployment of the proposed solution in a real-world context.

Further development of the proposed smart contract should also consider the implementation of a more realistic reputation score system alongside the current blockchain-based continuous authentication mechanism. This effort could be undertaken alongside a systematic security analysis to verify the security of the proposed blockchain-based continuous authentication mechanism. Finally, the integration of the separate codebases for the CAV emergency message application and the blockchain-based continuous authentication mechanism presented above could provide the foundation for an end-to-end analysis of the proposed VCA system.

Chapter 5 – Conclusion

We recall that the goal of this work is to apply a systematic design approach to explore the viability of the proposed VCA system and promote its security through the development of a blockchain-based continuous authentication mechanism. The development of the proposed VCA system is of particular interest given the modern phenomenon of fatal AV-to-VRU collisions. Further, The development of a continuous authentication mechanism for the traffic intersection-based RSU devices present in each VCA system instance is of critical importance, as such devices represent valuable targets for cybersecurity threat actors due to their centralized role within the system. The contributions of Chapter 2, Chapter 3, and Chapter 4 each provide initial contributions towards the goal of improved VRU safety while also leaving room for future work.

Chapter 2 provides a systematic review of current engineering standards as they pertain to the design of CAV transportation systems. We leverage the foundation provided by these engineering standards to define a reference architecture for the proposed VCA system, alongside a standard sensor data readout format for emergency message exchange between CVs, AVs, and RSU devices. Opportunities for future work include the ongoing refinement of engineering standards for the emerging industry surrounding CAV transportation systems. Further, additional work towards the definition of reference architectures for CAV transportation systems could help to promote the safety of VRUs seeking freedom of movement near these systems.

Chapter 3 applies the VCA system reference architecture to the development of a baseline CAV emergency message application. We perform a trade study to determine whether MQTT or CoAP represent viable options for use in the development of a baseline CAV emergency message application. We determine that MQTT QoS 1 provides the best performance in terms of average message latency, and we apply this result to the development of a baseline CAV emergency message application in Python. We also apply the concepts of end-to-end network delay and queueing theory to determine a theoretical maximum bound for end-to-end message latency for the CAV emergency message application. This effort shows that worst-case message latency should remain under the current requirement of 25 milliseconds for most realistic operating conditions. Opportunities for future work include additional software development on the baseline CAV emergency message application, alongside the pursuit of a real-world experiment to provide verification of our theoretical estimate for worst-case message latency.

Chapter 4 leverages the systematic design work of prior chapters to enable the development of a blockchain-based continuous authentication mechanism. We provide a brief overview of blockchain technology alongside a discussion of the practical uses of smart contracts. We then propose the creation of a smart contract for continuous authentication of RSU devices. This solution combines on-chain and off-chain authentication factors with the autonomous execution of smart contracts to achieve continuous authentication for RSU devices. We convert our theoretical smart contract design into a baseline solution through the use of the Solidity programming language and the Truffle Suite, which we highlight via an execution example. Opportunities for future work include the addition of a more robust reputation score system alongside a systematic security analysis. Further, the end-to-end integration of the CAV emergency message application and the blockchain-based continuous authentication mechanism could enable an end-to-end analysis of the proposed VCA system.

Overall, this work provides a systematic exploration of the problem space of faulty AV-to-VRU collisions. We apply a systematic design approach to the safe and secure design of the VCA system towards improved VRU safety. This provides a broad foundation for additional research directions, which we hope to explore through the pursuit of future work.

References

[1] National Transportation Safety Board (NTSB), "Highway Accident Report: Collision Between Vehicle Controlled by Developmental Automated Driving System and Pedestrian," Nov. 2019.

[2] Griggs, T., & Wakabayashi, D. (2018, March 20). *How a Self-Driving Uber Killed a Pedestrian in Arizona*. The New York Times. Retrieved January 26, 2023, from https://www.nytimes.com/interactive/2018/03/20/us/self-driving-uber-pedestrian-killed.html

[3] Smiley, L. (2022, March 8). *'I'm the Operator': The Aftermath of a Self-Driving Tragedy*. Wired Magazine. Retrieved January 26, 2023, from https://wired.com/story/uber-self-driving-car-fatal-crash/

[4] National Highway Traffic Safety Administration (NHTSA), "Part 573 Safety Recall Report 23V-085," Feb. 2023.

[5] M. Khayatian *et al.*, "A Survey on Intersection Management of Connected Autonomous Vehicles," *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 4, pp. 1–27, Aug. 2020. doi:10.1145/3407903

[6] N. Ullah Ibne Hossain, V. L. Dayarathna, M. Nagahi, and R. Jaradat, "Systems Thinking: Review and Bibliometric Analysis," *Systems*, vol. 2020, no. 8, Jul. 2020. doi:10.3390/systems8030023

[7] M. Chapple, D. Seidl "Chapter 8: Identity and Access Management," in *CompTIA Security+ Study Guide: Exam SY0-601*, 8th ed.

[8] F. Hussain Al-Naji and R. Zagrouba, "A survey on continuous authentication methods in Internet of Things environment," *Computer Communications*, vol. 163, pp. 109–133, Nov. 2020. doi:10.1016/j.comcom.2020.09.006

[9] A. F. Baig and S. Eskeland, "Security, Privacy, and Usability in Continuous Authentication: A Survey," *Sensors*, vol. 2021, no. 21, Sep. 2021. doi:10.3390/s21175967

[10] L. Gonzalez-Manzano, J. M. Fuentes, and A. Ribagorda, "Leveraging User-related Internet of Things for Continuous Authentication: A Survey," *ACM Computing Surveys*, vol. 52, no. 3, Jun. 2019. doi:10.1145/3314023

[11] A. Chowdhury, G. Karmakar, J. Kamruzzaman, A. Jolfaei, and R. Das, "Attacks on Self-Driving Cars and Their Countermeasures: A Survey," *IEEE Access*, vol. 8, pp. 207308–207342, Nov. 2020. doi:10.1109/ACCESS.2020.3037705

[12] E. Talavera, A. Díaz Álvarez, and J. E. Naranjo, "A Review of Security Aspects in Vehicular Ad-Hoc Networks," *IEEE Access*, vol. 7, pp. 41981–41988, Mar. 2019. doi:10.1109/ACCESS.2019.2907861

[13] X. Sun, F. R. Yu, and P. Zhang, “A Survey on Cyber-Security of Connected and Autonomous Vehicles (CAVs),” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6240–6259, Jun. 2021. doi:10.1109/TITS.2021.3085297

[14] Intelligent Transportation Systems Joint Program Office, “Connected Vehicles: Benefits, Roles, Outcomes,” Office of the Assistant Secretary for Research and Technology (OST-R), https://www.its.dot.gov/research_areas/WhitePaper_connected_vehicle.htm (accessed Jun. 7, 2023).

[15] Intelligent Transportation Systems Joint Program Office, “Automation White Paper,” Office of the Assistant Secretary for Research and Technology (OST-R), https://www.its.dot.gov/research_areas/pdf/WhitePaper_connected_vehicle.pdf (accessed Jun. 7, 2023).

[16] FHWA Highway Safety Programs, “Intelligent Transportation Systems Safety,” Federal Highway Administration, https://highways.dot.gov/safety/other/intelligent-transportation-systems-safety (accessed Jun. 7, 2023).

[17] Alternative Fuels Data Center, “Intelligent Transportation System Technologies,” Vehicles Technology Office, https://afdc.energy.gov/conserve/intelligent_transportation.html (accessed Jun. 7, 2023).

[18] S. Sharma and B. Kaushik, “A survey on internet of vehicles: Applications, security issues & solutions,” *Vehicular Communications*, vol. 20, Dec. 2019. doi:10.1016/j.vehcom.2019.100182

[19] N. H. Hussein, C. T. Yaw, S. P. Koh, S. K. Tiong, and K. H. Chong, “A Comprehensive Survey on Vehicular Networking: Communications, Applications, Challenges, and Upcoming Research Directions,” *IEEE Access*, vol. 10, pp. 86127–86180, Aug. 2022. doi:10.1109/ACCESS.2022.3198656

[20] C. M. Silvia, B. M. Masini, G. Ferrari, and I. Thibault, “A Survey on Infrastructure-Based Vehicular Networks,” *Mobile Information Systems*, vol. 2017, Aug. 2017. doi:10.1155/2017/6123868

[21] CAR 2 CAR Communication Consortium, “Overview of the C2C-CC System - Version 1.1,” *CAR 2 CAR Communication Consortium Manifesto*, Aug. 2007.

[22] M. Lee and T. Atkison, “VANET applications: Past, present, and future,” Vehicular Communications, vol. 28, Oct. 2021.

[23] G. Luo, H. Zhou, N. Cheng, Q. Yuan, J. Lin Li, F. Yang, and X. Shen, “Software-Defined Cooperative Data Sharing in Edge Computing Assisted 5G-VANET,” IEEE Transactions on Mobile Computing, vol. 20, no. 3, Mar. 2021.

[24] A. Demba and D. P. F. Moller, "Vehicle-to-Vehicle Communication Technology," IEEE International Conference on Electro/Information Technology (EIT), May 2018.

[25] T. Yeferny and S. Hamad, "Vehicular Ad-hoc Networks: Architecture, Applications and Challenges," International Journal of Computer Science and Network Security, vol. 20, no. 2, Feb. 2020.

[26] P. Choudhary and S. Umang, "A Literature Review on Vehicular Adhoc Network for Intelligent Transport," 2nd International Conference on Computing for Sustainable Global Development (INDIACom), 2015.

[27] H. Ullah, N. Gopalakrishnan Nair, A. Moore, C. Nugent, P. Muschamp, and M. Cuevas, "5G Communication: An Overview of Vehicle-to-Everything, Drones, and Healthcare Use-Cases," *IEEE Access*, vol. 7, Apr. 2019.

[28] M. A. Naeem, X. Jia, M. A. Saleem, W. Akbar, A. Hussain, S. Nazir, and K. M. Ahmad, "Vehicle to Everything (V2X) Communication Protocol by Using Vehicular Ad-Hoc Network," *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, Jan. 2020.

[29] S. Benkirane and M. Benaziz, "Performance Evaluation of IEEE 802.11p and IEEE 802.16e for Vehicular Ad hoc Networks using simulation tools," IEEE 5th International Congress on Information Science and Technology (CiSt), Oct. 2018.

[30] Office of the Assistance Secretary for Research and Technology, "V2X Communications for Transportation: An Overview," United States Department of Transportation, https://www.transportation.gov/research-and-technology/v2x-communications-transportation-overview (accessed Jun. 9, 2023).

[31] H. Feld, "Why Your Vehicle's Wireless Technology Stays Stalled In The Past," Forbes, https://www.forbes.com/sites/forbestechcouncil/2022/04/07/why-your-vehicles-wireless-technology-stays-stalled-in-the-past/ (accessed Jun. 9, 2023).

[32] The Intelligent Transportation Society of America (ITS America), "The Impact of a Vehicle-to-Vehicle Communications Rulemaking on Growth in the DSRC Automotive Aftermarket (FHWA-JPO-17-487)," *U.S. Department of Transportation - National Highway Traffic Safety Administration*, Oct. 2016.

[33] Y. L. Morgan, "Notes on DSRC & WAVE Standards Suite: Its Architecture, Design, and Characteristics," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 4, pp. 504–518, May 2010. doi:10.1109/SURV.2010.033010.00024

[34] Federal Communications Commission, *FCC 20-164*, Nov. 2020.

[35] Federal Communications Commission, "Use of the 5.850-5.925 GHz Band (86 FR 23281)," Federal Register, https://www.federalregister.gov/documents/2021/05/03/2021-08802/use-of-the-5850-5925-ghz-band (accessed Jul. 24, 2023).

[36] W. Wiquist, “FCC MODERNIZES 5.9 GHz BAND FOR WI-FI AND AUTO SAFETY,” Federal Communications Commission, https://docs.fcc.gov/public/attachments/DOC-368228A1.pdf (accessed Jul. 24, 2023).

[37] M. Harounabadi, D. Mohammad Soleymani, S. Bhadauria, M. Leyh, and E. Roth-Mandutz, “V2X in 3GPP Standardization: NR Sidelink in Release-16 and Beyond,” *IEEE Communications Standards Magazine*, vol. 5, no. 1, Mar. 2021. doi:10.1109/MCOMSTD.001.2000070

[38] M. J. Khan *et al.*, “Advancing C-V2X for Level 5 Autonomous Driving from the Perspective of 3GPP Standards,” *Sensors*, vol. 23, no. 4, Feb. 2023. doi:10.3390/s23042261

[39] M. H. Castañeda Garcia *et al.*, “A Tutorial on 5G NR V2X Communications,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1972–2026, Feb. 2021. doi:10.1109/COMST.2021.3057017

[40] J. B. Kenney, “Dedicated Short-Range Communications (DSRC) Standards in the United States,” *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, Jun. 2011. doi:10.1109/JPROC.2011.2132790

[41] S. Chen *et al.*, “Vehicle-to-Everything (V2X) Services Supported by LTE-Based Systems and 5G,” *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, Jul. 2017. doi:10.1109/MCOMSTD.2017.1700015

[42] A. Roy, “How the Language of Self-Driving Is Killing Us,” The Drive, https://www.thedrive.com/article/20495/how-the-language-of-self-driving-is-killing-us (accessed Jun. 9, 2023).

[43] B. Visnic, “Tesla’s FSD recall impacts AV industry,” SAE Mobilus, https://www.sae.org/news/2023/03/tesla-fsd-recall (accessed Jul. 28, 2023).

[44] SAE International, “Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles (Ground Vehicle Standard J3016_202104),” *SAE Mobilus*, Sep. 2021. doi:10.4271/J3016_202104

[45] P. Koopman, “SAE J3016 User Guide,” Carnegie Mellon University, https://users.ece.cmu.edu/~koopman/j3016/index.html (accessed Jul. 28, 2023).

[46] A. Tarantola, “Mercedes is the first certified Level-3-autonomy car company in the US,” Engadget, https://www.engadget.com/mercedes-first-certified-level-3-autonomy-car-company-us-201021118.html (accessed Jul. 28, 2023).

[47] T. Thadani and J. B. Merrill, “California just opened the floodgates for self-driving cars,” The Washington Post, https://www.washingtonpost.com/technology/2023/08/10/san-francisco-robotaxi-approved-waymo-cruise/ (accessed Nov. 3, 2023).

[48] P. Koopman, “L120 Overview of Automated Vehicle Terminology and J3016 Levels,” YouTube, https://www.youtube.com/watch?v=Kykb75_41hY (accessed Jul. 28, 2023).

[49] P. Koopman, “A User’s Guide to Vehicle Automation Modes,” Edge Case Research, https://edgecaseresearch.medium.com/a-users-guide-to-vehicle-automation-modes-4bdd49b30dc0 (accessed Jul. 28, 2023).

[50] Wang, Z. (2018) Secure and Reliable Connected and Autonomous Vehicle Design. Doctoral dissertation. University of Massachusetts Lowell.

[51] S. Kitiibwa, “Cyber Security Issues and Solutions for Effective CAV System Communication Toward Reliable Autonomous Driving,” thesis, ProQuest, 2020

[52] A. Gholamhosseinian and J. Seitz, “A Comprehensive Survey on Cooperative Intersection Management for Heterogeneous Connected Vehicles,” *IEEE Access - IEEE Vehicular Technology Society Section*, vol. 10, Jan. 2022.

[53] R. Hussain, J. Lee, and S. Zeadally, “Trust in VANET: A Survey of Current Solutions and Future Research Opportunities,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, May 2021. doi:10.1109/TITS.2020.2973715

[54] F. Stajano and R. Anderson, “The Resurrecting Duckling: security issues for ubiquitous computing,” *IEEE Computer Magazine*, vol. 35, no. 4, Apr. 2002. doi:10.1109/MC.2002.1012427

[55] C. Kaufman, R. Perlman, and R. Perlner, “Chapter 10 - Trusted Intermediaries,” in *Network Security: Private Communications in a Public World*, 3rd ed, M. Speciner, Ed. Addison-Wesley Professional, 2022

[56] IETF Network Working Group, “RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc5280 (accessed Jul. 28, 2023).

[57] C. Bayılmıs, M. A. Ebleme, Ü. Çavusoglu, K. Küçük, and A. Sevin, “A survey on communication protocols and performance evaluations for Internet of Things ,” *Digital Communications and Networks* , vol. 8, no. 6, pp. 1094–1104, Dec. 2022. doi:10.1016/j.dcan.2022.03.013

[58] D. Silva, L. I. Carvalho, J. Soares, and R. C. Sofia, “A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA ,” *Applied Sciences*, vol. 2021, no. 11, May 2021. doi:10.3390/app11114879

[59] V. Seoane, C. Garcia-Rubio, F. Almenares, and C. Campo, “Performance evaluation of CoAP and MQTT with security support for IoT environments,” *Computer Networks*, vol. 197, Oct. 2021. doi:10.1016/j.comnet.2021.108338

[60] M. Bansal and Priya, “Performance Comparison of MQTT and CoAP Protocols in Different Simulation Environments,” *Inventive Communication and Computational Technologies, Proceedings of ICICCT 2020*, pp. 549–560, Sep. 2020. doi:10.1007/978-981-15-7345-3_47

[61] J. F. Kurose and K. W. Ross, “1.5 Protocol Layers and Their Service Models,” in *Computer Networking: A Top-Down Approach*, 8th ed, Pearson, 2020

[62] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2020.

[63] OASIS Standards Track, “MQTT Version 3.1.1,” OASIS Open, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf (accessed Aug. 7, 2023).

[64] R. Herrero, “5.3.1 MQTT,” in *Fundamentals of IoT Communication Technologies,* 1st ed., Springer, 2021.

[65] Internet Engineering Task Force (IETF) “RFC 7252: The Constrained Application Protocol (CoAP),” IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc7252 (accessed Aug. 8, 2023).

[66] Internet Engineering Task Force (IETF) “RFC 7959: Block-Wise Transfers in the Constrained Application Protocol (CoAP),” IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc7959 (accessed Aug. 8, 2023).

[67] R. Herrero, “5.2.4 CoAP,” in *Fundamentals of IoT Communication Technologies,* 1st ed., Springer, 2021.

[68] Internet Engineering Task Force (IETF) “RFC 7641: Observing Resources in the Constrained Application Protocol (CoAP),” IETF Datatracker, https://datatracker.ietf.org/doc/html/rfc7641 (accessed Aug. 8, 2023).

[69] Eclipse Foundation, “Paho Python Client - Documentation,” *Eclipse Paho MQTT Client*. [Online]. Available: https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php. [Accessed: 01-May-2023].

[70] S. Cope, “Beginners Guide To The MQTT Protocol,” *Steve's Internet Guide: Practical Guide to MQTT and Mosquitto*. [Online]. Available: http://www.steves-internet-guide.com/mqtt/. [Accessed: 03-May-2023].

[71] Eclipse Foundation, “Mosquitto Documentation,” *Eclipse Mosquitto MQTT Broker*. [Online]. Available: https://mosquitto.org/documentation/. [Accessed: 01-May-2023].

[72] Canonical Ltd., “Traffic Control (tc) Manpage,” *Ubuntu Manuals.* [Online]. Available: https://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html. [Accessed: 01-May-2023].

[73] C. Amsüss, “The aiocoap API,” *aiocoap - The Python CoAP library.* [Online]. Available: https://aiocoap.readthedocs.io/en/latest/api.html. [Accessed: 01-May-2023].

[74] C. Amsüss, “Usage Examples,” *aiocoap - The Python CoAP library*. [Online]. Available: https://aiocoap.readthedocs.io/en/latest/examples.html. [Accessed: 03-May-2023].

[75] Qualcomm Technologies, Inc., “C-V2X Technical Performance – Frequently Asked Questions 80-PE732-67 Rev A.,” Oct. 2019.

[76] J. Bednar , “VCA System Thesis - GitHub Repository ,” GitHub https://github.com/ jbednar20 VCA_System_Thesis (accessed Nov. 3, 2023).

[77] A.J. Hawkins, “Serious safety lapses led to Uber’s fatal self-driving crash, new documents suggest, *The Verge*, 06-Nov-2019. [Online]. Available: https://www.theverge.com/2019/11/6/ 20951385/uber-self-driving-crash-death-reason-ntsb-documents/. [Accessed: 04-May-2023].

[78] J.F. Kurose and K.W. Ross, “Section 1.4 – Delay, Loss, and Throughput in Packet-Switched Networks,” in *Computer Networking: A Top-Down Approach*, 7th ed., Pearson, 2017.

[79] Technical Specification Group Services and System Aspects, “Section 5.5.2 – Automated cooperative driving for short distance grouping – Potential requirements [PR.5.5-001],” in *Study on enhancement of 3GPP Support for 5G V2X Services (Release 16)*, 3GPP TR 22.886 V16.2.0, 3rd Generation Partnership Project, Dec. 2018.

[80] W. J. Hopp and M.L. Spearman, “8.6 Variability Interactions – Queueing,” in *Factory Physics - Foundations of Manufacturing Management*, 2nd ed, M. L. Spearman, Ed. Irwin McGraw-Hill, 2001, pp. 264–273

[81] W. J. Hopp and M.L. Spearman, “8.3.1 Measures and Classes of Variability,” in *Factory Physics - Foundations of Manufacturing Management*, 2nd ed, M. L. Spearman, Ed. Irwin McGraw-Hill, 2001, pp. 252

[82] Technical Specification Group Services and System Aspects, “Section 5.22.1.1 – Intersection safety information provisioning for urban driving – General,” in *Study on enhancement of 3GPP Support for 5G V2X Services (Release 16)*, 3GPP TR 22.886 V16.2.0, 3rd Generation Partnership Project, Dec. 2018.

[83] Technical Specification Group Services and System Aspects, “Section 5.2.2 Automated cooperative driving for short distance grouping – Potential requirements [PR.5.5-003a],” in *Study on enhancement of 3GPP Support for 5G V2X Services (Release 16)*, 3GPP TR 22.996 V16.2.0, 3rd Generation Partnership Project, Dec. 2018.

[84] 5G Automotive Association (5GAA), “List of C-V2X Devices,” *5GAA Technical Report,* Nov. 2021.

[85] Commsigna, Inc., “ITS-RS4 Product Brief v10.1,” Nov. 2020.

[86] Z. Laaroussi, R. Morabito, and T. Taleb, “Service Provisioning in Vehicular Networks through Edge and Cloud: An Empirical Analysis,” *2018 IEEE Conference on Standards for Communications and Networking (CSCN)*, Oct. 2018.

[87] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, Oct. 2008. Accessed: Nov. 5, 2023. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[88] S. Al-Megren *et al.*, "Blockchain use cases in digital sectors: A review of the literature," *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Jul. 2018. doi:10.1109/cybermatics_2018.2018.00242

[89] B. Cao *et al.*, "Blockchain Systems, Technologies, and Applications: A Methodology Perspective," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 353–385, Sep. 2023. doi:10.1109/comst.2022.3204702

[90] M. H. Rehmani, *Blockchain Systems and Communication Networks: From Concepts to Implementation*, 1st ed. Springer, 2022.

[91] L. Lantz and D. Cawrey, *Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Distributed Applications*, 1st ed. O'Reilly, 2020.

[92] D. Yaga, P. Mell, N. Roby, and K. Scarfone, "NISTIR 8202: Blockchain Technology Overview," *National Institute of Standards and Technology Publications*, Oct. 2018. doi: nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf

[93] M. Wu *et al.*, "A Comprehensive Survey of Blockchain: From Theory to IoT Applications and Beyond," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8114–8154, Jun. 2019. doi:10.1109/jiot.2019.2922538

[94] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, "A Survey of Distributed Consensus Protocols for Blockchain Networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, Jan. 2020. doi:10.1109/comst.2020.2969706

[95] M. N. Bhutta *et al.*, "A Survey on Blockchain Technology: Evolution, Architecture and Security," *IEEE Access*, vol. 9, pp. 61048–61073, Apr. 2021. doi:10.1109/access.2021.3072849

[96] V. Buterin, "Ethereum Whitepaper," Ethereum Project, https://ethereum.org/en/whitepaper/ (accessed Nov. 5, 2023).

[97] G. Wood, "Ethereum Yellow Paper," Ethereum Project, https://ethereum.github.io/yellowpaper/paper.pdf (accessed Nov. 5, 2023).

[98] M. Dameron, "Ethereum Beigepaper," GitHub, https://github.com/chronaeon/beigepaper/blob/master/beigepaper.pdf (accessed Nov. 5, 2023).

[99] "Proof-of-Work (POW)," Ethereum Project, https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/ (accessed Nov. 5, 2023).

[100] “Proof-of-Stake (POS),” Ethereum Project, https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed Nov. 5, 2023).

[101] P. Szilágyi , “EIP-225: Clique proof-of-authority consensus protocol,” Ethereum Improvement Proposals, https://eips.ethereum.org/EIPS/eip-225 (accessed Nov. 5, 2023).

[102] S. Wang *et al.*, “An Overview of Smart Contract: Architecture, Applications, and Future Trends,” *2018 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2018. doi:10.1109/ivs.2018.8500488

[103] V. Y. Kemmoe, W. Stone, J. Kim, D. Kim, and J. Son, “Recent Advances in Smart Contracts: A Technical Overview and State of the Art,” *IEEE Access*, vol. 8, pp. 117782–117801, Jun. 2020. doi:10.1109/access.2020.3005020

[104] M. T. Hammi, B. Hammi, P. Bellot, and A. Serhrouchni, “Bubbles of Trust: A decentralized blockchain-based authentication system for IoT,” *Computers & Security*, vol. 78, pp. 126–142, Sep. 2018. doi:10.1016/j.cose.2018.06.004

[105] J. Yang, J. Dai, H. B. Gooi, H. D. Nguyen, and A. Paudel, “A Proof-of-Authority Blockchain-Based Distributed Control System for Islanded Microgrids,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 11, pp. 8287–8297, Jan. 2022. doi:10.1109/tii.2022.3142755

[106] Z. Lu, Q. Wang, G. Qu, H. Zhang, and Z. Liu, “A Blockchain-Based Privacy-Preserving Authentication Scheme for VANETs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* , vol. 27, no. 12, pp. 2792–2801, Dec. 2019. doi:10.1109/TVLSI.2019.2929420

[107] H. Liu *et al.*, “Blockchain Empowered Cooperative Authentication with Data Traceability in Vehicular Edge Computing,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 4221–4232, Apr. 2020. doi:10.1109/tvt.2020.2969722

[108] R. Jabbar *et al.*, “Blockchain Technology for Intelligent Transportation Systems: A Systematic Literature Review,” *IEEE Access*, vol. 10, pp. 20995–21031, Feb. 2022. doi:10.1109/access.2022.3149958

[109] M. B. Mollah *et al.*, “Blockchain for the Internet of Vehicles Towards Intelligent Transportation Systems: A Survey,” *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4157–4185, Mar. 2021. doi:10.1109/jiot.2020.3028368

[110] P. Bagga, A. K. Das, M. Wazid, J. J. P. C. Rodrigues, and Y. Park, “Authentication protocols in internet of vehicles: Taxonomy, analysis, and challenges,” *IEEE Access*, vol. 8, pp. 54314–54344, Mar. 2020. doi:10.1109/access.2020.2981397

[111] A. A. Monrat, O. S. Schelén, and K. Andersson, “Performance Evaluation of Permissioned Blockchain Platforms,” *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Dec. 2020. doi:10.1109/CSDE50874.2020.9411380

[112] P. M. Abhishek, D. G. Narayan, H. Altaf, and P. Somashekar, "Performance Evaluation of Ethereum and Hyperledger Fabric Blockchain Platforms," *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Oct. 2022. doi:10.1109/icccnt54827.2022.9984288

[113] A. Antonopoulos and G. Wood, Mastering Ethereum, https://github.com/ethereumbook/ethereumbook (accessed Nov. 5, 2023).

[114] J. Cui, X. Zhang, H. Zhong, Z. Ying, and L. Liu, "RSMA: Reputation System-Based Lightweight Message Authentication Framework and Protocol for 5G-Enabled Vehicular Networks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6417–6428, Aug. 2019. doi:10.1109/jiot.2019.2895136

[115] "Truffle Suite Documentation," Truffle Suite – ConsenSys Software Inc., https://trufflesuite.com/docs/ (accessed Nov. 5, 2023).