MITIGANDO ATAQUES DE NEGAÇÃO DE SERVIÇOS EM REDES VEICULARES DEFINIDAS POR SOFTWARE

GABRIEL DE BIASI

MITIGANDO ATAQUES DE NEGAÇÃO DE SERVIÇOS EM REDES VEICULARES DEFINIDAS POR SOFTWARE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Filipe Menezes Vieira
Coorientador: Antônio Alfredo Ferreira Loureiro

Belo Horizonte

Novembro de 2017

GABRIEL DE BIASI

MITIGATING DENIAL OF SERVICE ATTACKS IN SOFTWARE DEFINED VEHICULAR NETWORKS

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Luiz Filipe Menezes Vieira
Co-Advisor: Antônio Alfredo Ferreira Loureiro

Belo Horizonte
November 2017

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

de Biasi, Gabriel.

D286m Mitigating Denial of Service Attacks in Software Defined Vehicular Networks / Gabriel de Biasi. — Belo Horizonte, 2017.
xxii, 59 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais — Departamento de Ciência da Computação.

Orientador: Luiz Filipe Menezes Vieira
Corientador: Antônio Alfredo Ferreira Loureiro

1. Computação — Teses. 2. Redes de computadores. 3. Redes veiculares. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Mitigating Denial of Service Attacks in Software Defined Vehicular Networks

GABRIEL DE BIASI

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]
Prof. Luiz Filipe Menezes Vieira - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Antonio Alfredo Ferreira Loureiro - Coorientador
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Ítalo Fernando Scotá Cunha
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Leandro Aparecido Villas
Instituto de Computação - UNICAMP

Belo Horizonte, 1 de Novembro de 2017.

Acknowledgments

Primeiramente aos meus pais, Fernando e Egnalda, pelo incentivo e apoio imensurável durante todo este período longe de casa e dos meus amigos.

À Débora, por ter compartilhado comigo todos os momentos de felicidade e dificuldades que passei e me dado amor e carinho incondicionalmente.

Ao Prof. Luiz Filipe, pelos ensinamentos durante as aulas de Redes sem Fio, pelas orientações e sugestões passadas por nossas conversas nos fins de tarde em seu gabinete e pela imensa sabedoria e paciência que permitiram trabalharmos juntos para traçar meu caminho durante a criação desta dissertação.

Ao Prof. Antônio Loureiro, pela humildade e simpatia no início da jornada por conversarmos mesmo antes que eu chegasse em Belo Horizonte e pelas reuniões e orientações durante as aulas de Computação Ubíqua que me ajudaram a fazer a escolha, em suas próprias palavras, do meu “sorvete”.

Ao Prof. Jefersson, por ter cedido seu tempo em apresentar o PPGCC à mim e aos meus colegas da UEMS e nos guiar nos primeiros passos necessários para tornarmos pesquisadores.

Aos meus novos amigos que tive a oportunidade de fazer, em especial ao José Laerte, Francisco e Tati, por compartilhar os momentos de estudo de PAA, as comemorações e os fins de semana regados à jogos e risadas.

Aos colegas do Wisemap pela amizade e todos os professores do Departamento de Ciência da Computação da UFMG, que durante todo meu mestrado tiveram toda a dedicação de lecionar as disciplinas que me tornaram capaz de escrever esta dissertação.

Ao CNPq e ao DCC que me permitiram desenvolver esta dissertação através da concessão da bolsa de estudos.

O meu sincero: **Muito obrigado!**

Gabriel de Biasi

“When something is important enough,
you do it even if the odds are not in your favor.”
(Elon Musk)

Abstract

Software Defined Vehicular Network (SDVN) is a new network architecture inspired by the well-known Vehicular Ad Hoc Network (VANET), applying the concepts of Software Defined Network (SDN). The SDVN proposes a complete data flow management by a module that controls the routing actions. However, it is necessary to verify that the security requirements are still satisfied. We propose two SDVN architectures: (1) Centralized mode, where there is only one controller and the vehicles use LTE and WAVE as interfaces communication and (2) distributed mode, where there are several controllers installed in RSUs and it uses only WAVE communication. This work presents the Sentinel, a new defense approach to detect flooding attack by time series analysis of packet flow and mitigate the attack creating a flow tree to find out the source of spoofed packets. We divided the results between the detection rate of victim vehicles and the efficiency of the mitigation method. Sentinel was able to mitigate the attack flow in different scenarios and parameters. Sentinel reached an average mitigation rate of more than 78% in all densities scenarios. However, the speed of vehicles might decrease the efficiency due to the fast change of attack flow. Furthermore, we also propose some improvements to future approaches.

Keywords: Vehicles, Networks, SDVN.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Computer networks have advanced significantly in recent years, with new user services, entertainment, education and social networks that have been offered through the Internet. Furthermore, Internet access has reached electronic devices, smartphones, smart clocks, and even vehicles have access to these services using the mobile phone network.

Corporations and governments saw the world with autonomous and connected vehicles as a distant future. However, with the advancement of newer companies in this industry such as Tesla[1] and even the electronics seller Apple[2], make this subject widely discussed again after divulged that both companies are working in their research about autonomous and connected vehicles in order to deploy real-world testbeds. The first steps were installing embedded technologies in vehicles, e.g., cameras, proximity sensors, air condition sensor, multimedia centers, GPS, and others. Some manufacturers even include Internet connection using long-range communication, such as the 4G cellular network.

After the wave of new devices connecting to the Internet, researchers have begun to build architectures and algorithms that allow vehicles to communicate with each other, creating a range of possibilities for traffic safety, route information, on-board entertaining, among other possibilities.

In November 2004, the project IEEE 802.11 Task Group "p" was created to define enhancements to 802.11 required to support Intelligent Transportation Systems (ITS) applications, proposing the family of standards called Wireless Access in Vehicular Environments (WAVE). They developed several drafts between 2005 and 2009. In April 2010, they approved the draft 11 with 99% affirmative votes and no further

[1]https://www.tesla.com/about

[2]https://www.theverge.com/2017/8/1/16079902/apple-autonomous-systems-going-beyond-cars

comments. In May 2010, the meetings of the TGp group were closed due to all the agreements being voted on, although it still remains in standby until some issue needs to be discussed[3].

Connected vehicles that exchange information about the road ahead, weather, emergency alerts, among many possible applications. Inspired by the Mobile Ad hoc Network, Vehicular Ad hoc Network (VANET) was the first proposal and more accepted in the literature as a network architecture for connected vehicles. Using the same ad hoc-based routing algorithms and mobility models focused on the road segments, it was possible to perform network simulations, application suggestions, among other things [Mahmoud Al-Qutayri and Al-Hawi, 2010].

Because it is an ad hoc network, the flow control becomes decentralized and might become unstable depending on the current density of vehicles. In order to solve these issues, the researches proposed the deployment of Software-defined Networking (SDN) architecture in vehicle networks. The use of SDN concepts into vehicle networks tends to achieve promising results, creating a new network architecture called **Software Defined Vehicular Network** (SDVN).

The benefit of Software Defined Networking is mainly due to their centralized management. Besides that, have the possibility of optimal routing, selecting the shortest paths to forward packets between the vehicles, due to the awareness of the network topology provided by SDN controller. In addition, the dynamic topology of vehicles allows implementing multiple wireless interfaces, allowing sending and receiving packets simultaneously. In this architecture, the vehicles would have the ability to communicate with each other using dedicated short-range communication and an interface for long-range communication to exchange messages with the SDVN controller, requesting and receiving flow rules.

Although researchers have already proposed some network architectures, it is also necessary to check if the security requirements are still satisfied. In previous works, we observed that a non-secure SDVN suffers from the same security problems inherited from both previous architectures. Some attackers might create new approaches in order to cause damage to the network or gain some type of benefit. For example, generate false emergency warnings to cause accidents, pretending to be an emergency vehicle that needs a clean road to drive, not forward security packets, among other types of attacks.

We observed an opportunity looking deeply into the **DDoS Flooding Attack**. This security attack is commonly performed by a set of nodes generating false data

[3]http://grouper.ieee.org/groups/802/11/Reports/tgp_update.htm

focusing on a victim node in order to deny your network services (further discussion in section 4.6). In this work, we propose using the statistical data into analysis techniques in order to detect and mitigate a flooding attack.

Sentinel is a new defense mechanism to detect and mitigate DDoS Flooding Attack in SDVN. We use a set of simulation tools in order to examine the damage that this attack can cause on the network in some different parameters and scenarios and we test the mitigation efficiency of Sentinel in the same situations. Furthermore, we propose future improvements to the approach to mitigate regardless of the mobility pattern of the vehicles, transmission range, attack load and some other parameters.

1.1 Motivation

The SDVN architecture is a recent topic in the literature. Although several proposals for other network structures for vehicle communication and many applications suggestions already exists, a thorough study on the security issues of these architectures needs to be considered.

Once computer networks began to spread to products that were not initially designed for these uses, such as smart televisions, smart refrigerators, smart speakers and so on, a whole new set of possibilities opened up for security breaches. However, these new safety gaps in these types of networks can lead to real accidents involving physical or material damage.

This study motivated us by the importance of satisfying the security attributes of a network, especially in network architectures that directly involve the safety of human lives, such as vehicular networks. Therefore, it is necessary we discuss and compare the best security approaches for vehicle networks become a reality in the current world.

1.2 Contribution

The main contribution of this work is to propose Sentinel, which is our proposed flooding mitigation mechanism designed for highly dynamic SDVNs. It works entirely on the SDVN controller and creates a new specific flow rule to prevent forwarding of attack packets. We separated this method into two phases, which are detection and mitigation. We explain these functionalities deeply in Chapter 5.

The specific contributions of this work are:

- Present a new architectural proposal for SDVN

- Present the benefits and limitations of SDVN and its security challenges
- Introduce the “Sentinel”, an attack mitigation mechanism in SDVN
- Propose future improvements, highlighting the strengths and weaknesses of the algorithm

1.3 Dissertation Outline

We organized the remainder of this work as follows:

In Chapter 2, we present an overview of the literature on vehicular networks, their origins, application suggestions, electromagnetic spectrum allocation and PHY/MAC layers standards. We explained the concept of SDN more deeply, presenting its history of paradigm changing and we introduced the OpenFlow communication protocol. Furthermore, we present the security attributes and examples of attacks together with some work related to the subject.

In Chapter 3, we describe some works that are related to the mitigation of security attacks in computer networks. We discussed some examples of attacks on SDN and attacks on VANET. Moreover, we presented and compared four different SDVN architecture with the solution brought by this work.

In Chapter 4, we present the details of the architecture proposed by this work, showing examples of communication and the differences between centralized and distributed modes. In the end, we define the attack model applied in the network.

In Chapter 5, we discuss the details of the mitigation algorithm and explains how the detection and mitigation phases work. In addition, we do a complexity analysis of the algorithm to verify the possible overhead added to the network.

In Chapter 6, we present the software used to perform the simulations, assumptions, scenarios, and range of values used in the parameters. We divided the results between the detection capacity of the algorithm and the average mitigation rate during an attack scenario.

In Chapter 7, we discuss the overview of the results obtained in this work and presents the final determinations. In the end, we present suggestions for improvements observed after implementation.

Chapter 2

Background

To develop specific algorithms and methods for this new network architecture, it is necessary to have an overview of the operation of the previous architectures that originated the SDVN. Understanding the basic communication process is critical to building robust and efficient algorithms.

In this chapter, we present the concept of vehicle networks and their mode of operation. We discussed the data transmission standards defined by IEEE 802.11p and IEEE 1609. Furthermore, we present the concept of software-defined networking and its history of updates and improvements in academia. All these network architectures are affected by security attacks, and then it is necessary to study how these attacks work and the behavior of the entities that execute it.

2.1 Vehicular Ad hoc Network

Vehicular Ad hoc Network (VANET) is a new category of mobile ad hoc network, which applies basically the same mobile routing protocols in order to allow inter-vehicle data communications and support the Intelligent Transportation Systems (ITS) [Mahmoud Al-Qutayri and Al-Hawi, 2010]. This type of network provides many advantages for communications between vehicles, for example, wide area coverage, low-latency communication, and it does not require a sophisticated power management.

There are two important entities present on the network: the *Vehicle*, also referred as *On-board Unit* (OBU) and the *Road Side Unit* (RSU). The vehicles are entities able to communicate with each other and the RSUs are towers next to the road that sends and receives information to nearby vehicles. Therefore, the two main types of communications in VANETs are:

Vehicle-to-Vehicle (V2V) Communication between vehicles, using mainly for safety messages and information about the road, for example.

Vehicle-to-Infrastructure (V2I) Communication between vehicles and the RSUs, allowing forward messages over long distances for another type of messages.

However, the network is not limited only to these types of communications. For example, there are types of networks which the vehicles connects directly to the Internet and V2X, which means *Vehicle-to-Everything*, possibly the next generation of VANETs.

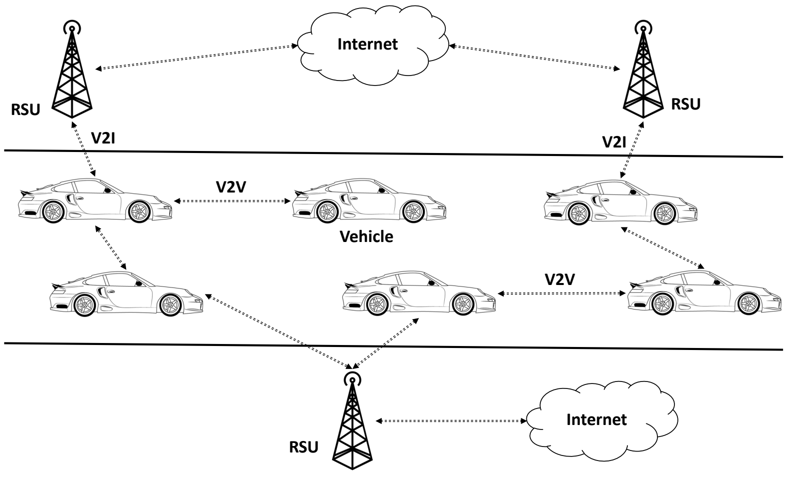


Figure 2.1. Example of a Vehicular Ad hoc Network, introducing the communication Vehicle-to-Vehicle and Vehicle-to-Infrastructure.

The range of applications for the vehicular environment is huge, with potential for security applications, entertainment, real-time news, among others. We have the specification of four great examples of applications with potential in the context of vehicular networking:

Traffic Control Using statistical data captured from vehicles on the network to get insights about better on-demand routes. Changing current routes followed by vehicles according to real-time events, such as unplanned traffic jams, weather changes or accidents, even in areas without an Internet connection.

Collision Avoidance Using the Wave Short Message Protocol (WSMP) to exchange messages between vehicles without having to create a Basic Service Set (BSS), decreasing the delay between messages. Therefore, WSMP enables application deployment against collision between nearby vehicles by sending the location and current speed between them.

Inter-vehicle Entertainment The mobility model designed by the vehicles allows passengers to use applications that use the neighboring vehicles for entertainment. Passengers can start a multi-player game and play while they are close enough. Communication via voice and video would also be possible if applicable in the context of the application.

Context-aware Advertisements Given current passenger context, such as current vehicle position, current time, combining passenger personal preferences to generate context-aware advertisements. Food service advertisements near lunchtime, ice cream on summer days, suggest going into vehicle support service if vehicle reported problems, etc.

2.1.1 IEEE 802.11p

The IEEE 802.11p is a standard originated from the allocation of Dedicated Short Range Communications (DSRC) spectrum band, specifying the physical and MAC upper layers required of a device into a vehicular environment [Jiang and Delgrossi, 2008].

As can be seen in Figure 2.2, the reserved spectrum is in the range of 5,855 GHz to 5,925 GHz. DSRC uses Orthogonal Frequency-Division Multiplexing (OFDM) to multiplex data. This technique works by dividing the signal into small sub-carries, allowing multiple transmission of information in just one signal.

They divided the spectrum into seven channels, one control channel, and six service channels. Furthermore, they implemented other improvements, such as Random MAC address, 10MHz channels (being half of the previous 802.11 standards), 16QAM modulation, frame priority control, power control, among others.

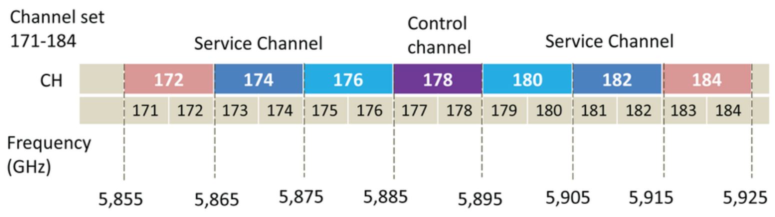


Figure 2.2. Frequency spectrum reserved to WAVE divided into separated channels. Classifying the channel 178 as exclusive for control messages, channel 172 as Collision Avoidance Safety and channel 184 as Public Safety. Adapted from Sahoo et al. [2014].

The default communication channel in this standard is the control channel, located in the middle of the spectrum that allows for rapid exchange of messages without network participants being part of a set of services. However, the standard also characterizes the remaining six channels as possible service providers for the creation of WAVE Basic Service Sets (WBSS), allowing vehicles belonging to a WBSS to exchange information on a unique channel and that do not necessarily control data or safety data.

2.1.2 IEEE 1609.x Standards

The IEEE 1609 is a family of standards called Wireless Access in Vehicular Environments (WAVE), which mainly defines an area of the exclusive electromagnetic spectrum for inter-vehicle communications, that enable secure vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) wireless communications, also proposes a set of services and architectures [Uzcategui et al., 2009].

In order to manage the priority service arriving the MAC layer, the Enhanced Distributed Channel Access (EDCA), a technique inspired from 802.11e [IEEE, 2005], arranges the frames in queues so they can wait for the correct moment to be transmitted. The MAC layer also has a packet-detection service that checks its type (IP or WSMP) according to its header, assigning correctly to your transmission channel.

The following items below show the specifications for each sub-document of the IEEE 1609 standard:

IEEE 1609.4 Provides enhancements to the IEEE 802.11p MAC to support multichannel operation. This standard defines the EDCA implementation, channel synchronization operations, error corrections, and multiple channel transmission queues. The effective bandwidth of communications in standard is normally 18 $Mbps$.

IEEE 1609.3 Provides addressing and routing services within a WAVE system. Implementing the Logical Link Control (LLC), Wave Short Message Protocol (WSMP), IPv6, TCP, and UDP, according to their existing standards. In addition, this document requests the deployment of WBSS management, IPv6 automatic configuration, receive channel power indicator (RCPI), channel usage monitoring, etc.

IEEE 1609.2 Covers the format of secure messages and security attributes needed in order to deploy a safety WAVE system, such as confidentiality, authenticity, data integrity, among others. We discussed these security attributes more deeply in subsection 2.3.1.

IEEE 1609.1 Describes an application that allows the interaction of vehicles with limited computing resources and complex processes running outside the vehicles in order to give the impression that processes are running in the vehicles, known as offload processes.

In Figure 2.3, we have the allocation of the IEEE standards compared with the OSI model for communications. They placed some standards in parallel to multiple OSI layers, because it does not fit directly with the OSI layers.

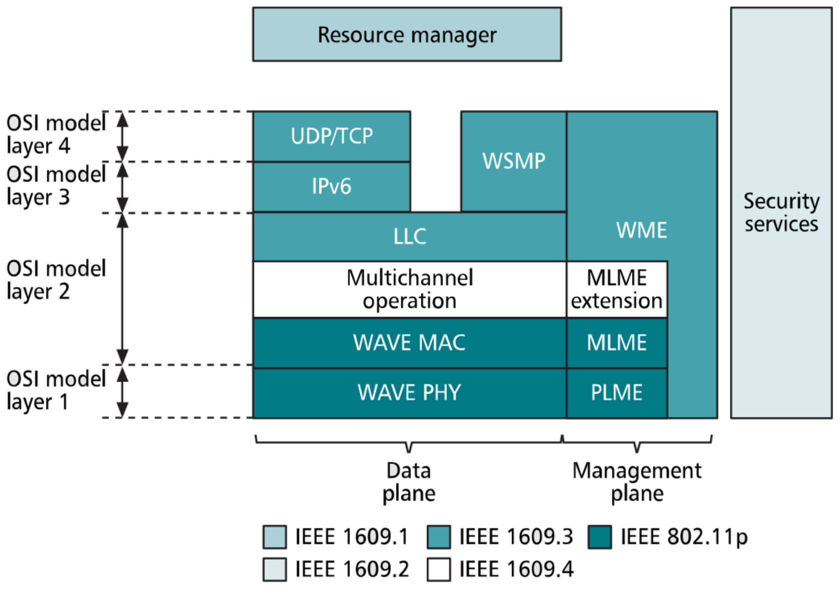


Figure 2.3. Wave communication stack is indicating the IEEE standard that covers each set of OSI layers. Adapted from Uzcategui et al. [2009].

2.2 Software-Defined Networking and OpenFlow Protocol

Software-Defined Networking (SDN) consists of a new paradigm for developing research in computer networks. The initiative became successful on defining the OpenFlow protocol [McKeown et al., 2008], where message forwarding nodes offer a simple programming interface allowing access and control of the flow table to determine the next routing point. In this way, the routing of the messages becomes more efficient locally, since the decision of routing of messages is reduced in only a query in a table and the processing of this decision is transferred to a centralized module, allowing the whole network to be controlled in a sustainable way, expressed via software rules.

The basic principle of SDN is the possibility of programming the network elements (Figure 2.4). The programming of these elements is restricted to the manipulation of flows, that is, the sequence of nodes that a packet must pass in order to reach its destination node. In order to perform the necessary calculations to create a message flow, the network controller must be aware of all connections between the message forwarding nodes, making it possible to use minimum path algorithms and obtain statistical data in order to check the state of each link in use on the network.

The control plane uses simple rules associated with each input of a switch's flow table. By default, these rules can be (1) forward the packet to a specific port, (2) change part of the message header, (3) drop message or (4) forward message to the controller for inspection [Guedes et al., 2012].

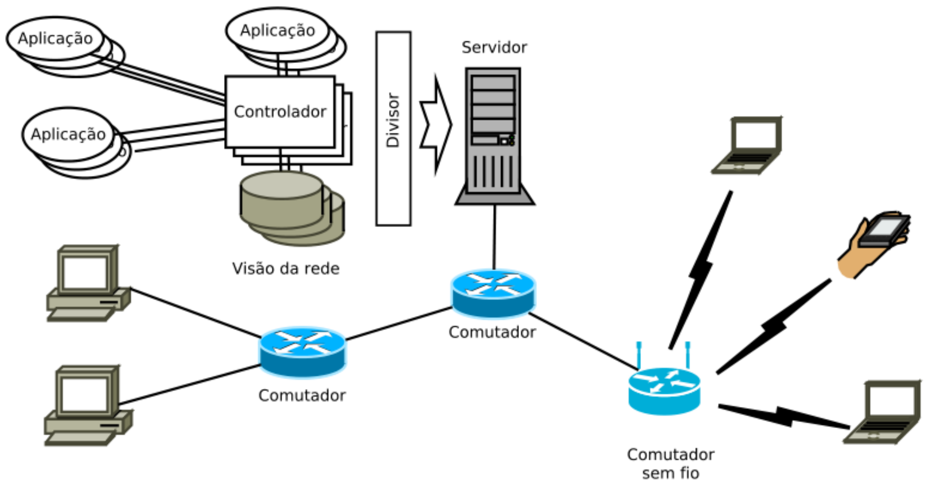


Figure 2.4. Overview of a structure of Software-Defined Network. The hosts on the network communicate with each other using a central module that instructs the correct path by the data and the flow rules in this network. Adapted from Guedes et al. [2012].

Many works propose their implementations of SDN without the concern of large demands scalability, opting for centralized structures. However, there are efforts to deploy distributed controllers to ensure scalability and system availability. An example of open-source SDN controller is Floodlight[1].

In order to achieve this network behavior, the OpenFlow Protocol [McKeown et al., 2008] is applied, allowing nodes belonging to the network can have a secure and direct communication with the SDN controller, receiving and sending forward policies around the network.

[1]http://www.projectfloodlight.org/floodlight/

A node in an OpenFlow-based SDN sends a *PACKET_IN* message to the controller whenever there are no references on the packet in its flow table, requesting an action to be taken. Whenever the controller calculates the best action for a particular packet type, it sends a *FLOW_MOD* message to one or more nodes on the network to insert a new rule into its flow tables.

On an OpenFlow managed network, whenever a host needs to send a message, the following steps below occurs [Macedo et al., 2015]:

1. Whenever the switch receives a message, it checks if there is any flow rule that matches this message, if it exists, the switch executes the stipulated action, otherwise the following steps are performed below.
2. It extracts a copy of the message header and sends this information to the OpenFlow controller while storing the entire message into a local buffer. If there is no space left, the message is dropped.
3. The controller receives the message header and analyzes all routing data (source node and destination node), communication protocol (TDP, UCP), the port used, and other information to create an appropriate flow rule for this type of message.
4. The controller sends this flow rule to the switch. Upon receiving the flow rule, the switch installs the rule and begins to apply the rule action until the time stipulated by the controller for its use to end.

The OpenFlow controller can take advantage of this communication interface to obtain statistical data on the network, for example, how many times the flow rule was used, packages processed by the switch per second, total flows generated, among other values.

2.3 Security in Vehicular Networks

In this section, we introduce the security attributes to deploy a safety vehicular network, the characteristics and behaviors of network attackers, and some examples of security attacks in VANETs and countermeasures.

2.3.1 Security Attributes

Vehicular networks have important requirements in order to provide security for the users. Raya and Hubaux [2005] presents the main security attributes for a safety and reliable network:

2.3.1.1 Authentication

A node inside the network can only exchange messages with another legitimate node. The authentication of the nodes plays an important role in the security of the network. In some applications, there is a key management infrastructure to ensure the authenticity.

2.3.1.2 Non-Repudiation

It is a security mechanism that whenever a message request is required, the sender and the receiver cannot deny them. All nodes present in the network must follow this attribute.

2.3.1.3 Availability

The channels on the bandwidth must be available all the time, even if it is under attack. Some new vehicles may arrive with new security packets and the communication channel must be prepared to receive them. In this case, the network needs to have an operation policy, for example.

2.3.1.4 Privacy

This attribute is similar to the data verification. However, the problem is if other nodes accessed the packet. The network must prevent unauthorized access to the content. Using some encryption approaches like Public-Key cryptography, Message Authentication Code, etc.

2.3.1.5 Data Integrity

Data integrity ensures that any node on the network do not modify the data during their trajectory to the destination, neither by attackers nor by a network communication problem.

2.3.2 Attackers Classification

There are several entities that are potential attackers of a vehicular network, such as coordinated groups, an adversary companies, government agencies or any individual who has a specific interest. The reasons for the attack may be monetary gains, political reasons, and even intellectual challenges.

We considered the entities that generate the security attacks as attackers. According to their behavior and the type of attack they want to perform, we classified them in a different way. We organized the classification as follows:

2.3.2.1 Insider Vs. Outsiders Attackers

We define the authenticated attackers present on the network as insiders. We considered these security attacks made by these nodes very dangerous because they are already present on the network. However, outsiders are not able to conduct communication within the network because they are not connected and their attacks are potentially less dangerous.

2.3.2.2 Active Vs. Passive Attackers

Active attackers perform actions within the network, sending false safety messages or not forwarding messages. However, passive attackers are limited to just listening to the communication channel or generating noise.

2.3.2.3 Malicious Vs. Rational Attackers

Malicious attackers aim to disrupt network connectivity without any personal benefit. On the other hand, rational attackers have their goals defined by using their security attacks.

2.3.3 Availability Attacks

Although there are several types of availability attacks on computer networks, the Denial-of-Service (DoS) attack is one of the main security attacks that break the availability property of a particular node or network.

However, we consider several distinct security attacks as DoS attacks. These attacks have different behaviors within the network to prevent their normal operation, such as dropping all incoming messages, creating a large amount of data and sending them to a victim, sending false routing messages, among others. In the following sections, we presented how some of these attacks perform.

2.3.3.1 Black Hole & Selective Forwarding Attack

We characterized the Black Hole Attack by an attacker node that pretends to be the best node to forward packets according to the routing algorithm, but whenever this node receives a packet, it immediately dropped. This attack can be made through three strategies:

Black Hole Attack A single node performs the attack and can be easily detected by packet loss analysis on the network.

Cooperative Black Hole Attack Multiple nodes perform the attack in the same time, making packet loss analysis more difficult.

Selective Forwarding Attack the nodes analyze the packet data (destination, content, packet type) to make the decision to drop the packet. The rules depend on the purpose of the attack.

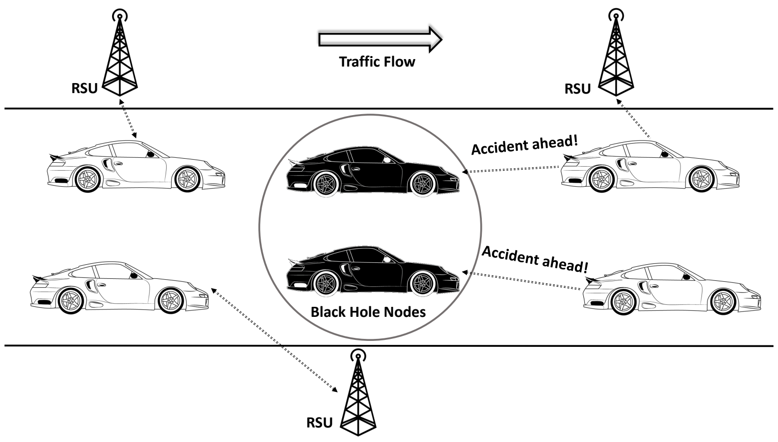


Figure 2.5. A cooperative black hole attack being executed by two vehicles, they are dropping security messages about the road condition denying this information to later vehicles.

Another approach to avoid this security attack is focused on low-density networks and scenarios where vehicles move on straight lines. Almutairi et al. [2014] suggest using a trust table on every node on the network, in order to evaluate the reliability of neighboring nodes on each step of the simulation. To evaluate their approach, they perform the simulations using OMNeT++ and *Vanet Car Mobility Manager* (VaCaMobil), applying a geographic routing protocol. However, their approach works only for single black hole attack and its efficiency is reduced in high-speed scenarios or over realistic mobility models, different from Manhattan mobility model.

2.3.3.2 Link Spoofing Attack

In wireless ad hoc networks with OLSR routing algorithm, the communication between the nodes is given by the HELLO and TC messages, which allow divulging to their near nodes their neighborhood state information. However, an attacker can generate HELLO and TC messages with false data, creating a different connection scenario and disrupting current communications, creating a link spoofing attack.

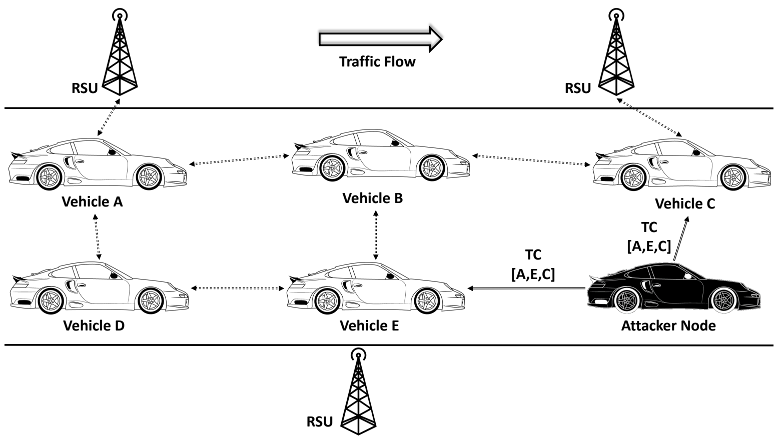


Figure 2.6. The attacker node F informs by a TC message to the nodes D, E and G that the node A is an active neighbor, but it is an incorrect information and aims to disrupt the communications destined to the node A.

An approach to mitigating link spoofing attack proposed by Jeon et al. [2012] is to perform some modifications to the OLSR algorithm. The first modification is the range of HELLO messages, being sent to the 2-hops neighbors as well. All nodes have a trust table with their 2-hop neighbors, in order to manage the "trust flag" between these nodes for possible new MPRs. With these mechanisms, attacks such as adding non-existent nodes, deleting node neighbor information are completely mitigated.

However, there is another approach that does not imply making modifications to the OLSR algorithm, working based only on logs provided by the nodes according to the behavior of the network. Alattar et al. [2012] worked by creating a signature for a link spoofing attack so that it could be detected by the generated logs. A detection algorithm is presented with a series of rules for checking, using a cooperative trust system and confidence levels. The results were validated using network simulations where attackers are starting the link spoofing attack. At the end of the rounds, the attackers obtained very low confidence values and were detected.

2.3.3.3 Flooding Attack

In a communication network, whether by wire or wireless, it has bandwidth limits and a maximum saturation point where the network can no longer function, being forced to decrease the message traffic. Attackers can take advantage of this behavior in order to saturate the communication of a victim node by denying communication with legitimate nodes. In Figure 2.7 a flooding attack scenario is presented in a vehicular environment.

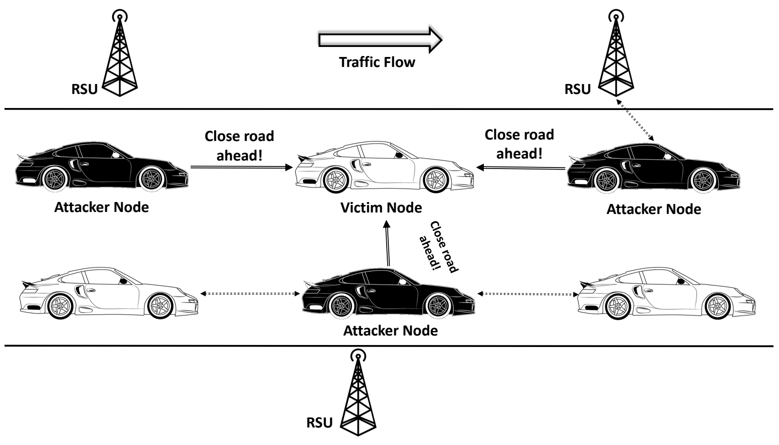


Figure 2.7. Flooding attack being executed by three attackers next to the victim vehicle, which is receiving a large amount of false data, denying its communication to other legitimate vehicles.

Chapter 3

Related Work

The mitigation of security attacks on computer networks is a subject widely discussed in academia. Since then, security attacks already existing in conventional networks have begun to be adapted for wireless ad hoc networks and for this reason, new mitigation methods need to be developed.

We will present in this chapter several works that proposed solutions for similar security attacks in different network architectures, realigning directly with we have done in this work. Moreover, we discussed and compared other SDVN architectures in relation to their characteristics.

3.1 Denial of Service Attacks on VANET

Despite being an ad hoc-based network, VANETs also suffers from DoS attacks. In these cases, the defense mechanisms classify some vehicles as attackers whenever they generate large amounts of false information directed to a victim vehicle. An approach proposes an identification method for Distributed Denial of Service (DDoS) Attack, using the information gathered by the RSUs [Pathre et al., 2013].

The process works as follows: The RSUs checks the network communications and identify vehicles that are generating false information. Whenever a legitimate vehicle receives some false information, they report with a safety message to the nearest RSU. When receiving a safety message from other nearby vehicles, RSU infers which vehicle is the attacker. After that, all communications from the attacker node are blocked. To evaluate their method, they used a network simulator with random way-point mobility and realized that the RSUs were able to detect network congestion and infer the attackers to block their communications. As future work, they suggested using a mobility model closer to the reality of the vehicles.

A well-known security attack on VANETs is the black hole attack. One or more vehicles that drop any received packet, not responding or forwarding them, characterize this attack. One variation of this attack is the gray hole, which analyzes the packet context and selectively drops packets.

An identification method of this attack suggests analyzing the behavior of vehicles from a trace file, separating them into normal and abnormal [Alheeti et al., 2015]. They performed the security attacks using an adaptation of the AODV ad hoc routing algorithm. To generate the trace file, they used SUMO for mobility model and NS2 to simulate the network. After that, they used several records of different vehicle behaviors to train an Artificial Neural Network (ANN). In its experimental results, IDS achieved an accuracy very close to 99% with a false positive rate less than 1%.

3.2 Denial of Service Attacks on SDN

Using the SDN controller to create network statistics is a good practice, however, it is possible to infer a Denial of Service (DoS) attack on a node by analyzing the lack of randomness of a network. An approach suggests performing the entropy calculation of the packets in the network in order to find out if someone is performing a DoS attack [Mousavi and St-Hilaire, 2015].

Usually in a common network, the entropy value of the packets will always be high, in other words, there will always be the flow of a normal packet between the nodes. However, when there are several packets destined for a specific node, the entropy value of the network drops and by definition of a threshold, it is possible to detect the DoS attack.

To evaluate their method, they used an SDN network simulator and three python programs to generate the amount of packets needed for the attacks. First, they tested several entropy thresholds to decide which value was most appropriate to use. They decided to detect a DoS attack whenever the entropy value of the packets in the network falls below 25%. On the test cases, they obtained a success rate of 96% for an attack of 25% traffic rate and a complete detection for 50% and 75% traffic rates.

A common attack on OpenFlow-based networks is flooding the flow table in some nodes, in order to force the nodes to ignore new rules sent by the SDN controller resulting in a brute drop on delivery packet rate [Qian et al., 2016]. The first countermeasure that they used in these cases is to apply timeout policies on the flow table, in other words, setting a lifetime in seconds for every flow rule. The switch removed the flow rule automatically after time runs out. Another countermeasure is to define a limit rate of incoming flow rules on the nodes, acting as an early flood protection.

However, only the policies are not enough, so they created a reactive method to perform a filter on the flow rules that the switches receive in the node. Separated into two modules, the Learning Switch module learns the network mapping, forward the data packets according to the MAC address, while the Flow-Checking module performs the validation of MAC addresses, IP addresses, and mitigates overflow.

In their simulations, they limited the storage space of the nodes' flow table, initiated the attacks using packets with forged MAC addresses, packets with source and destination IP addresses exchanged and packets with source and destination UDP ports also exchanged. It was possible to observe in the evaluations that the FlowChecking module can perform the cleaning of the useless flow rules on the nodes and restore the normal bandwidth values.

3.3 SDVN Architectures Proposals

We defined the SDVN architecture used in this work after a study of several existing architectures, trying to find the best approaches and trying to correct the errors found. In this section, we will present four proposals of architectures for SDVN present in the academy. In the end, we discuss the differences found in each proposal.

3.3.1 Multiple Modes SDVN

An approach proposed by Ku et al. [2014] is an SDVN architecture based on multiple operation modes. The operation works as follows: (1) Centralized Mode, where the SDVN controller has direct communication with every vehicle on the network and controls all the actions and flow rules, (2) Distributed Mode, where the RSUs has the local control and the vehicles may forward flow rules with each other and (3) Hybrid Mode where both previous modes are used simultaneously and the vehicles receive flow rules directly from the SDVN controller and whenever the controller is not available, the flow rules may be received by another vehicle.

To evaluate their network architecture, they executed the simulation observing the packet delivery rate of the SDVN compared to the common ad hoc routing algorithms, such as DSDV, AODV, OSLR, and GPSR. They also provide a solution in case of loss of connection with the SDVN controller, forcing the vehicles to use ad hoc routing until the controller has your connection established again.

3.3.2 Cost-efficient Heterogeneous SDVN

An approach presented by He et al. [2015] is an SDVN architecture with multiple network interfaces to forward data packets, but using different frequencies and cost connections. They present a trade-off between delivery ratio and costs. Transformed into an optimization problem, they use a network availability matrix created from predictions made in the simulation, using three network interfaces with different transmission costs and use an algorithm to find out the best scheduling of interfaces usage according to the time.

They proposed two algorithms, an exhaustive search that always found an optimal solution and a greedy algorithm, with a much smaller complexity runtime, but did not guarantee the optimal solution. They evaluated their algorithms using Simulation of Urban Mobility (SUMO), which is a traffic simulator and Network Simulator 3 (NS3), where they could realize that SDVN architecture gain in packet transmission rate. However, the relative speed of the vehicles drops the delivery rate in all the simulation cases.

3.3.3 QoE-Based Flow Management SDVN

Bozkaya and Canberk [2015] presents a new flow and power management model for SDVN. The method classified the vehicles according to a specific metric called QoE, checking vehicles that need to change the amount of transmission power in order to reach more vehicles while reducing network noise. In addition, they modified the OpenFlow protocol so the controller could apply new actions to the RSUs and forward to the vehicles.

If the QoE of a vehicle is below the threshold, the method classified as an unsatisfactory vehicle and vehicles that communicate normally with high bandwidth classified as vehicles satisfied. Using this classification, the flow management uses an algorithm that finds an RSU that allows better communication. In addition, by calculating the Signal-to-Interference-and-Noise-Ratio (SINR) and the ordinary Kriging method, power management returns the best transmission power value so the QoE of the vehicle grows above the threshold.

3.3.4 Delay Efficient SDVN

Sudheera et al. [2016] considers three requirements to implement an SDVN: (1) Short setup time, where whenever a vehicle wants to communicate with another, the route setup time should be short enough to compensate for the trade-off between ad hoc and SDN, (2) use only the spectrum dedicated to vehicular communication (DSRC), avoiding the extra cost of deploying a new communication model and (3) ensuring flexibility to the network, allowing its behavior to be fully programmable.

They model various delay estimations, such as queue delay, contention delay, and end-to-end delay over the Internet. The two architecture proposals for SDVN are (1) Internet-based SDVN, where they separated the controller into a single module and the nodes need to access the Internet to obtain flow rules and (2) RSU based SDVN, where they deployed multiple controllers to the RSUs in order to do local network control. They performed the tests using only functions of mathematical delay and setup time, varying the parameters in several execution attempts.

3.3.5 Discussion

In most proposals presented, there is concern about the use of heterogeneous communication. They also must analyze the delay added in the messages when implementing the SDN concepts, based on the delay of ad hoc algorithms such as AODV. Another change cited in several proposals is adding a cost related to the transmission, according to the communication model used. As the objective of this work is to present a mitigation mechanism that is able to work in most SDVN architectures, we defined two architectures that do not have distinct characteristics, but we can implement in future work.

The questions answered in Table 3.1 are as follows: (1) if the network has acceptable transmission delay values, (2) if the network has heterogeneous communication, (3) if the network has cost-based transmission and (4) if the network has transmission power management.

Table 3.1. Comparative between SDVN proposals.

SDVN Proposals	**Delay**	**Heterogeneous**	**Cost**	**Power**
Ku et al. [2014]	X	O	X	×
He et al. [2015]	X	X	X	×
Bozkaya and Canberk [2015]	X	O	X	X
Sudheera et al. [2016]	X	O	×	×
This Work (Centralized)	X	X	×	×
This Work (Distributed)	X	O	×	×

Caption: X: Present; O: Not Present; ×: Might be added.

Chapter 4

Architecture of Software Defined Vehicular Network

As discussed in the previous chapter, there are already several proposals for SDVN architectures. However, in order to design a mitigation approach that works for most proposals, this work defines its own architecture, simplifying the specific cases created by the proposals presented, such as cost-based transmission or transmission power control. We defined two sub-architectures: centralized and distributed. The centralized architecture focuses on using a single controller to manage the network; on the other hand, the distributed architecture uses local controllers installed in RSUs allowing sectional network management.

In this chapter, we present the characteristics of centralized and distributed approaches and the differences between them. Moreover, the details of DoS attacks are presented and your implementation.

Firstly, there are two special types of messages that exist in the SDVN that are fundamental to its operation and for both sub-architectures: the beacons and the neighborhood messages. We present the details in the following subsections.

4.1 Beacon Messages

The vehicles and RSUs broadcasts short frames containing your identifier, current speed, and geographical position, called as beacon messages. All reachable neighbors receive these messages in order to be aware of your presence. The vehicles must broadcast these messages with a short window of time, ensuring the freshness of the information. The vehicles must store all information received from the beacon messages in order to send to the SDVN controller afterward.

4.2 Neighborhood Messages

In order to calculate an efficient routing of network packets, the SDVN controller needs to be aware of the active vehicles and their direct neighbors. Thus, the vehicles and RSUs must send a special control message called Neighborhood Message.

This message has the following fields: vehicle identifier, a timestamp of the message and updated list of neighbor nodes since the last message sent. The node drops all information previously whenever a newer message arrives. The SDVN controller will use this information to calculate the flow rules necessary for communication between vehicles.

4.3 Vehicle Architecture

We modified the communication stack for both vehicles and RSUs including a flow table in order to store the flow rules received from the controller, which is fresh enough to reuse afterwards. The data packets that still do not have flow rules are stored in the “Packet In Buffer” waiting for the response of the *PACKET_IN* request.

The controller module also monitors the average time between the neighbor messages sent by the vehicles and RSUs, in order to verify which node is not available anymore. For instance, if a vehicle does not send a neighbor message in a window of 10 seconds, it is considered unavailable vehicle and any *PACKET_IN* request sent to the SDVN controller with destination to that vehicle will reply with a *DROP* rule.

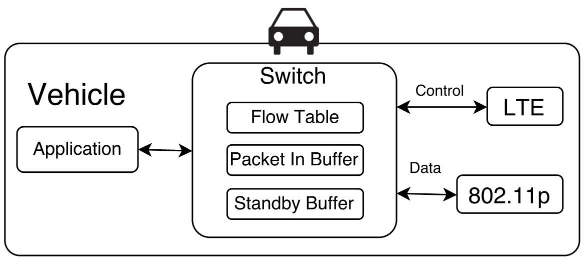


Figure 4.1. Vehicle architecture using heterogeneous communication. The control plane data is sent through LTE interface while data packets use 802.11p

4.4 Centralized SDVN

We consider every vehicle present in the centralized architecture an OpenFlow switch, capable of sending, receiving and forwarding packets. In order to achieve this behavior, the vehicles have two network interfaces, LTE to exchange messages directly with the controller and WAVE to exchange data packets between the other vehicles and RSUs. The beacon messages are sent over the WAVE control channel using WSMP, ensuring that all nodes nearby are capable of receiving it. We use LTE interface to exchange OpenFlow requests and neighborhood messages between the controller and vehicles. We illustrated the centralized architecture in Figure 4.2.

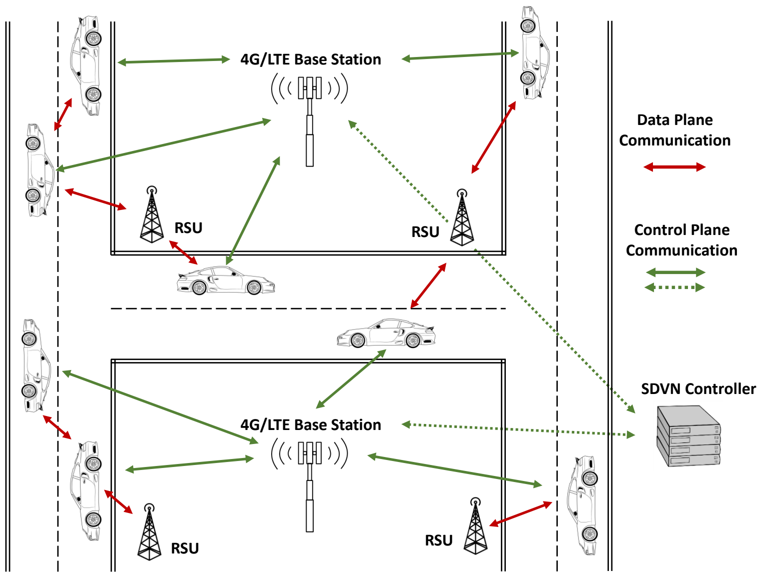


Figure 4.2. Software Defined Vehicular Network in Centralized Mode. The vehicles communicate with each other and the RSUs using 802.11p, while using LTE to communicate to the controller.

Besides the common OpenFlow requests and actions, the *STANDBY* is an exclusive action rule for centralized SDVN, created in order to allow temporarily disconnected vehicles. For instance, if a vehicle has connectivity to LTE and send your neighbor messages normally, but does not have any neighbor vehicle or RSU to forward messages, the controller sends a *STANDBY* rule, informing to store all packets arriving from the application layer and wait for a given time in order to resend the *PACKET_IN*.

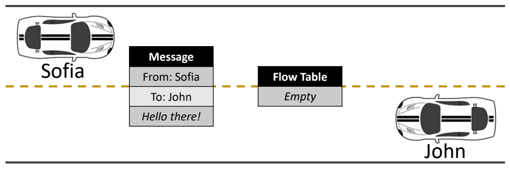
The following characteristics are the overview to deploy a centralized SDVN:

- Vehicles and RSUs have OpenFlow switches
- Unique controller for entire network
- Higher operational cost compared against distributed mode
- It is possible to have uncovered areas by RSUs

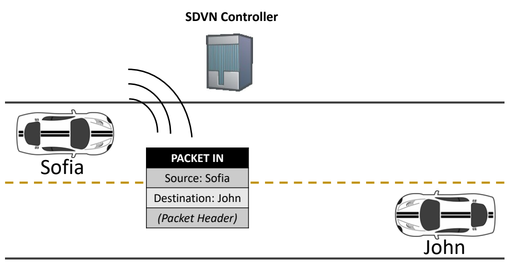
- The vehicles must have extra storage space for standby buffer
- Heterogeneous communication

In Figure 4.3, we have a use case where Sophia would like to send a message to John. First, (1) Sofia checks in her flow table if there is an active flow for John. When confirming that there is no rule, (2) Sofia stores the message in a buffer and creates a message called *PACKET_IN* to controller, informing the source and final destination of the message and its header.

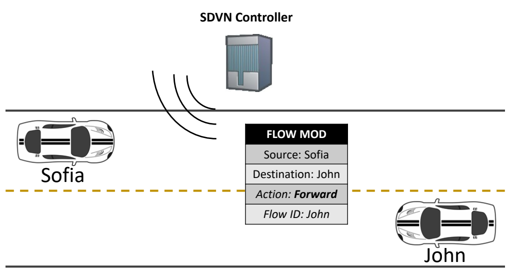
Once the controller receives the message, it calculates whether there is a valid path between the source and the destination at the moment. Upon finding a valid path, (3) it creates a message called *FLOW_MOD* containing the data needed to create a new flow rule and sends it back to Sofia. Upon receiving the controller's message, (4) Sofia now has a valid flow rule for John. It then retrieves the message stored in the buffer and finally sends the message to its final destination.



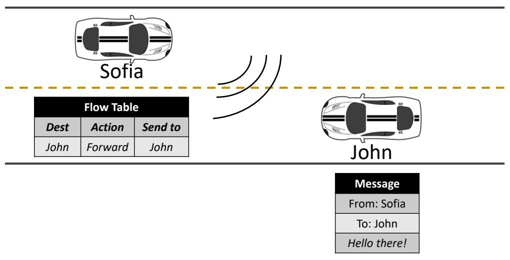
(a) Step 1



(b) Step 2



(c) Step 3



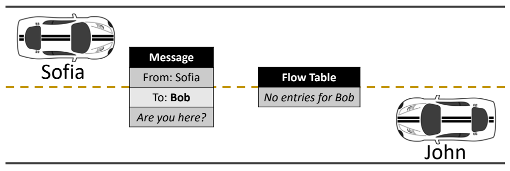
(d) Step 4

Figure 4.3. Example of a communication in SDVN.

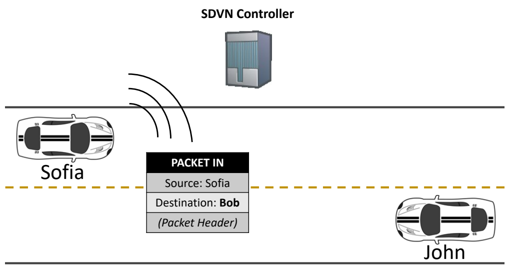
Following the same example from the previous case, Sofia would like to send a message to Bob, however Bob is not reachable in the current network topology. In

Figure 4.4, (1) Sofia checks in her flow table if there is an active flow for Bob. When confirming that there is no rule, (2) Sofia stores the message in a buffer and creates a message called *PACKET_IN* to controller, informing the source and final destination of the message and its header.

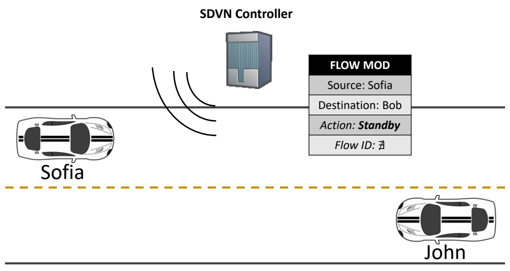
Once the controller receives the message, (3) the controller checks that there is no valid path between Sofia and Bob. So it creates the *FLOW_MOD* message with the *STANDBY* action, telling Sofia to store this message and wait for Bob to recover his connectivity to the network.



(a) Step 1



(b) Step 2



(c) Step 3

Figure 4.4. Scenario of communication in SDVN when the source and the destination are not reachable in the current network topology.

4.5 Distributed SDVN

Deploying a vehicle network with heterogeneous communication can result in increased maintenance-related costs and can also be a limiter for areas without long-range communication. In order to solve this issue, we proposed a distributed architecture using only WAVE communication.

In the distributed architecture, the vehicles do not communicate with the unique controller and do not have an LTE interface anymore. Besides that, every RSU on the network behaves as a local controller and receives and send packets over the control plane for the reachable vehicles, the schematic of the distributed architecture of SDVN can be seen in Figure 4.5.

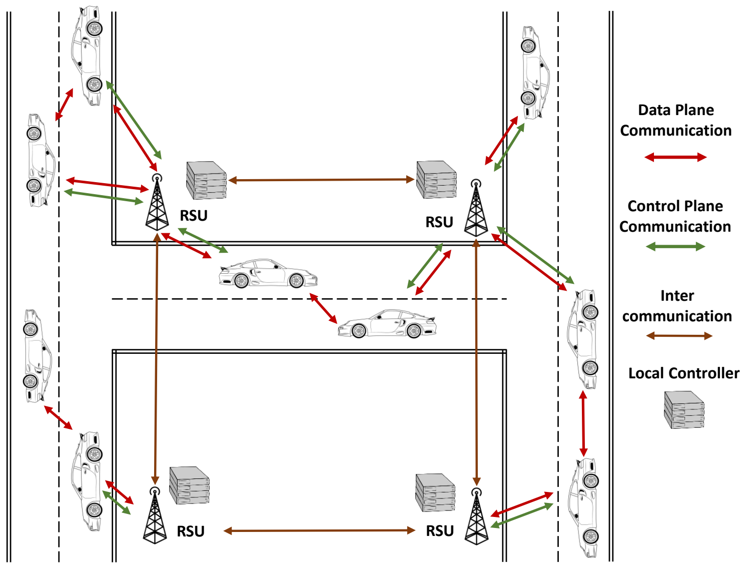


Figure 4.5. Software Defined Vehicular Network in Distributed Mode. The vehicles communicate with each other and the RSUs using 802.11p, while RSU itself manages the control plane of the area.

In this mode, the switch sends both beacon and neighborhood messages though the WAVE control channel using WSMP while the data packets use the remaining service channels. We implemented the vehicle exchange management between the RSUs. If a vehicle is under control of a particular RSU, it will exchange the control messages with it and vice versa. Once it moves and enters into the transmission range of another RSU, both RSUs will make a handshake to perform the management change, warning the vehicle about the process afterwards.

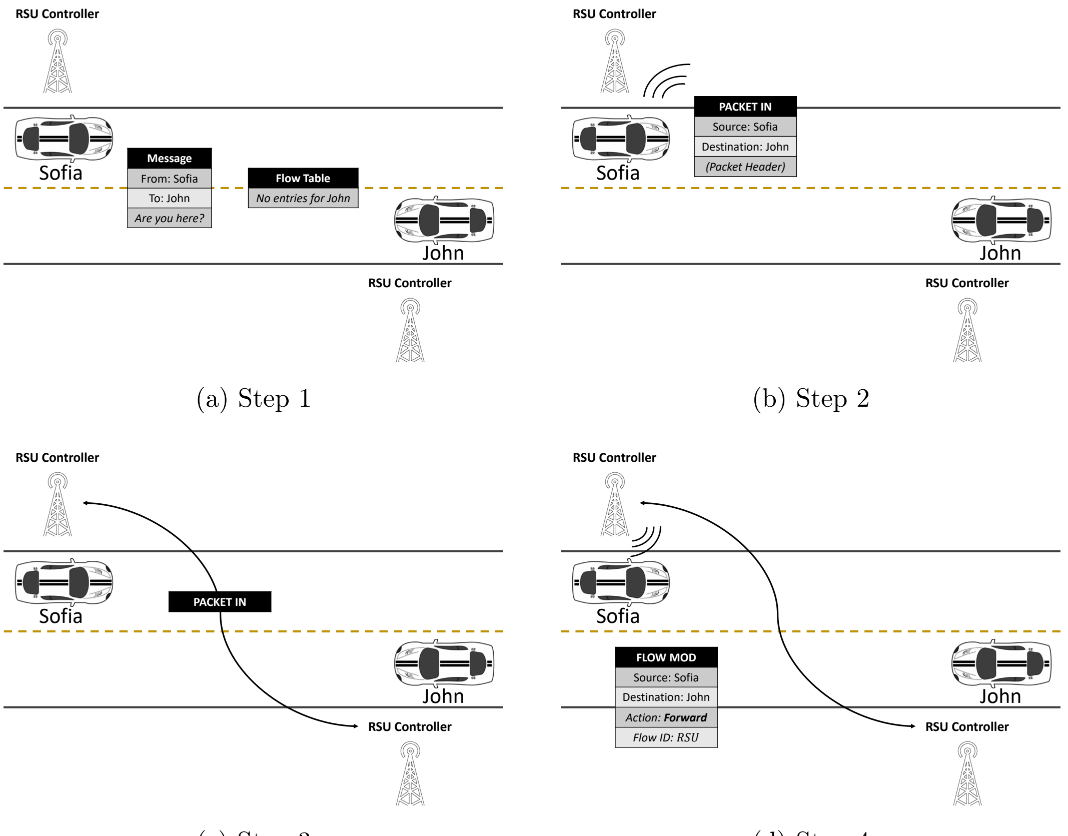
We defined the following characteristics to deploy a distributed SDVN:

- Homogeneous communication
- Low Latency
- Lower operational cost and deploy cost compared against centralized mode
- RSUs shares their neighborhood data with each other

- RSUs must cover all possible areas of the network

Using the same case used in centralized mode, we will apply this same scenario now to distributed mode. In Figure 4.6, Sophia would like to send a message to John. First, (1) Sofia checks in her flow table if there is an active flow for John. When confirming that there is no rule, (2) Sofia stores the message in a buffer and creates a message called *PACKET_IN* to its RSU manager, informing the source and final destination of the message and its header.

Upon verifying that the destination vehicle is not on its management, (3) the RSU sends the message *PACKET_IN* to the other RSUs to find out which one is managing the target. (4) The RSU that controls the destination sends *FLOW_MOD* directly to the RSU that controls the source, forwarding the new flow rule to Sofia.



(a) Step 1

(b) Step 2

(c) Step 3

(d) Step 4

Figure 4.6. Scenario of communication in Distributed SDVN.

When using RSUs to route messages to distant vehicles, this involves multiple routes being routed only by RSUs. By observing this fact, we have implemented a service load threshold for every RSU, causing the controller to be forced to find an alternative route for the route, avoiding passing RSUs. Once a particular RSU passes the threshold for messages per second, its internal controller does not create any more routing through it.

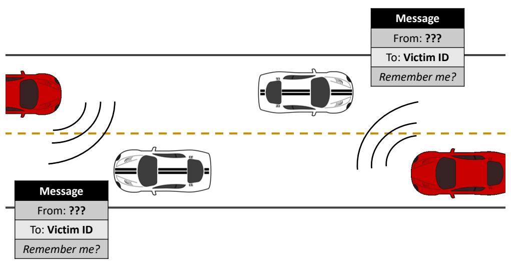
4.6 Model of Distributed Flooding Attack on SDVN

The distributed flooding attack happens when an ordered group of valid nodes within the network, which an attacker invaded previously, waits for the master to start a large date flow to a victim node. This behavior is not limited only to wireless networks and the attack is able to perform on various types of computer networks.

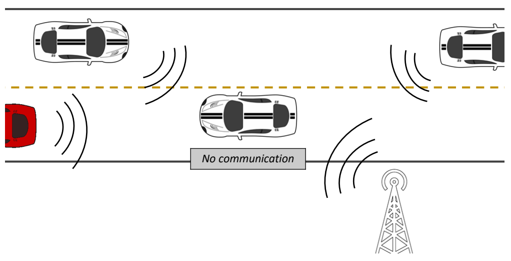
Zargar et al. [2013] classified the DDoS flooding attacks into two categories: flooding in the transport/network layer or flooding in the application layer. The *botnets* are the mechanisms that facilitate DDoS flooding attacks on wireless networks or applications.

The attack node control (a.k.a. Master) can be managed in three modes: (1) IRC-based, using the instant messaging software, allows communication with the zombies nodes in an easy way, (2) Web-based, using small servers where vehicles can receive commands using the HTTP protocol and (3) P2P-based, where the message exchange between the members of the botnet occurs without the use of a specific protocol [Zargar et al., 2013].

They defined the defense mechanisms based on four types: (1) source-based mechanisms, which are deployed near the sources of the attack to prevent network customers from generating DDoS flooding attacks, (2) destination-based mechanisms, such as packet marking, link testing, hop-counting filtering, (3) network-based mechanisms, such as packet filtering, detecting malicious routes and (4) hybrid mechanisms, such as throttling, congestion control, attack diagnosis, among others [Zargar et al., 2013].



(a) The messages generated by the zombies uses spoofed source addresses.



(b) The attack flow reaches the victims saturating its communication interface.

Figure 4.7. Flooding attack process.

This work assumes that the vehicle that initiates the flooding attack sends a simple data message to the zombie vehicles. This message has a list of the victim's IDs and the exact time to start the data flow. As showed in Figure 4.7, the attack flow is spoofed, i.e., there is no reference to the zombie vehicle into source address. Once the entire attack flow reaches the victim, its network interface becomes saturated by the amount of data to process and can no longer communicate with legitimate vehicles.

In the example of Figure 4.8, to initiate a flood attack, the vehicle A sends a data message to the zombie vehicles B, C and D informing the vehicle which will be the target of the attack (solid green arrows). When initiating the attack, each zombie vehicle randomly chooses a neighbor and begins sending the attack flow (single dot blue arrows). Upon receiving the attack flow, legitimate vehicles and RSUs treat the attack as a normal flow and begin the routing process until reaching the victim vehicle V (double dot red arrows).

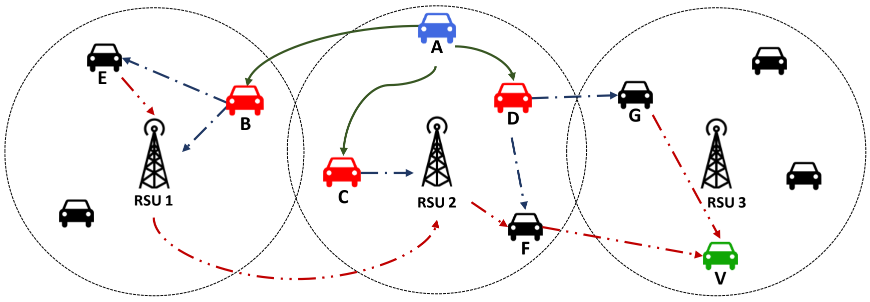


Figure 4.8. Operation scheme of a flooding attack in SDVN. Vehicle A is the attacker, vehicles B, C and D are zombies and vehicles E, F and G are receiving the attack flow.

Chapter 5

Defense Mechanism: Sentinel

Sentinel is our proposed flooding mitigation mechanism made for highly dynamic SDVNs. It works entirely on the SDVN controller and creates a new specific flow rule to prevent forwarding of attack packets. We separated into three phases, which are data collection, detection and mitigation. The source code of this mechanism and the SDVN architecture is available on-line[1] over the *MIT* license.

5.1 Collecting the Data

We characterized this phase on monitoring the statistics using time series analysis of the network, storing the number of packets processed and a number of flow rules generated with destination to a given vehicle.

These values are managed according to a constant N, which means the last N values of packets processed and flow rules generated are stored in order to characterize your traffic load. We start our tests with N equal to 25 and we decrement this value over the same simulation scenario until reach the lowest possible value with still satisfactory results, ending up with 10 as the final value.

Every time that a vehicle sends your neighborhood message to controller informing your current neighborhood, it also sends a number of packets processed P since the last message. At the same time, the controller already maintains the number of flow rules generated F to this vehicle.

[1]https://github.com/gabrielbiasi/sdvn

5.2 Detecting the Attack

To determine if a vehicle is on the attack, we need to use the data retrieved in the data collection phase and arrange them in a way that allows us to detect an anomaly of the values. We created three equations, using function $g(x)$ as the arithmetic average of a given set x, $h(x)$ as the variance of a given set x and c a calibration constant. The equations tested are in the Table 5.1.

Table 5.1. Equations tested for the detection phase.

Name	Equation
E1	$g(x) + \sqrt{c \times h(x)}$
E2	$g(x) + c \times \sqrt{h(x)}$
E3	$c \times \left(g(x) + \sqrt{h(x)}\right)$

We realized in the tests with E1 the need to use constants with large values due to the constant c being inside the square root calculation, making a low value impossible to fine tune the algorithm. In the tests with E2, we also saw the need to use large numbers because the constant c influences only the value of the standard deviation. In order to use small values to perform the calibration of the algorithm, E3 proved to be more efficient in the detection tests, multiplying the whole equation by the constant c.

Thus, we defined the final equation utilized into detection phase according to (5.1). The constant c is applied in order to refine the detection behavior of the algorithm according to the dimensions of attack. As the values of P and F are normally different and unrelated, it is referred by two different constants, α and β.

$$f(x, c) = c \times \left(g(x) + \sqrt{h(x)}\right) \tag{5.1}$$

If the vehicle already sent N neighborhood messages to the controller, every time that a new message comes with new values of packets processed, the controller gets the number of flows rules generated and calculates $f(x, c)$ for the P and F sets, according to (5.2).

$$\begin{aligned} newValuePacket &> f(P, \alpha) \\ newValueFlow &> f(F, \beta) \end{aligned} \tag{5.2}$$

The function $f(x, c)$ represents a threshold in order to detect an abnormal behavior of the vehicle based on the previous values of P and F. We classified as a possible victim if the vehicle sends the neighborhood message and the new values exceed the threshold for both sets.

We treated the vehicles classified as a possible victim differently. The next values to P and F will not be stored anymore until these values return to below the thresholds. If the vehicle continues to receive up to t consecutive abnormal values, it will be treated as a confirmed victim.

We use the number of packets processed and the number of flow rules generated together in order to detect the flooding attack avoiding false positives detections and low detection rates at the same time. For instance, using only the constant α might detect vehicles located out of an RSU range and process large amounts of packets when recovering its connectivity. On the other hand, using only the constant β might detect service provider vehicles as victims by the flow rules generated but without generating a large traffic load.

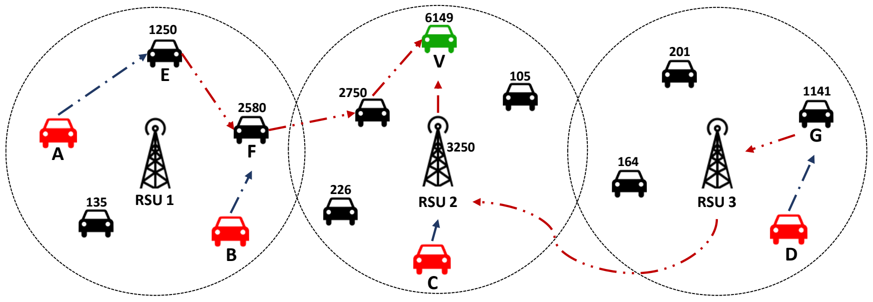


Figure 5.1. An SDVN scenario under attack. Vehicle V is getting all the attack flow generated by vehicles A, B, C and D, which are zombie vehicles. Meanwhile, the vehicles outside the flow tree are with normal values of P.

In Figure 5.1, vehicle V is under attack. Vehicles A, B, C and D are zombie vehicles and generating the attack flow. The number indicated above the vehicles are the current P values of each vehicle. Looking at the P values of the vehicles out of attack flow area, it is notable that the vehicle V is processing a large number of packets, setting abnormal behavior.

5.3 Mitigating the Attack

Whenever the controller detects a possible flooding attack by detection phase, the mitigation process starts to build a flow tree in order to localize the *botnet* generating the traffic load. We assumed that the botnet is broadcasting spoofed packets, i. e., there is no direct reference to who is creating these packets and they have no flow rules with destination to the victim.

Due to this fact, the leaves of the flow tree are either vehicles that are trying to have a legitimate communication with the victim or vehicles that are receiving spoofed packets from the *botnet*. In this approach, we will refer to them as gateway vehicles (Figure 5.2). Once Sentinel identifies the vehicles receiving the attack flow, it is possible to perform the mitigation process at different points in the network in order to deny the attack forwarding to the victim.

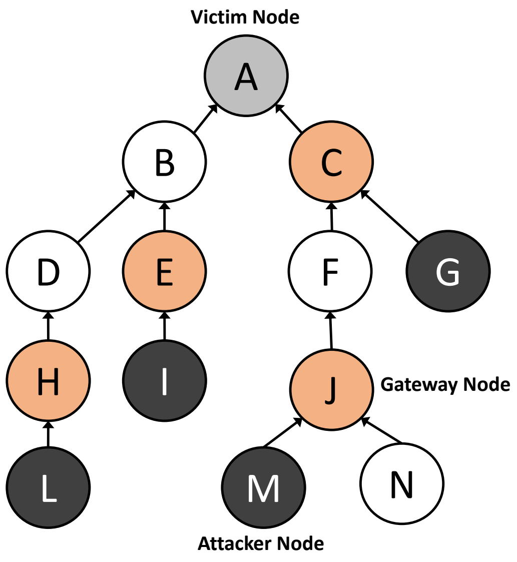


Figure 5.2. Flow tree generated after detecting an attack flow on node A. The nodes C, E, H and J have flow rules with destination to A, but their neighbors do not. This means that your neighbors are possible *botnet* members.

The flow tree building process works as follows: Firstly, Sentinel puts the victim vehicle in the root of the tree and after that selects your direct neighbors, based on which one has flow rule with destination to the victim. The result becomes the first layer of the tree. After that, the direct neighbors of the first layer perform the process similar to breadth-first search until no neighbor remaining, forming a flow tree. We detailed this functionality on algorithm 1.

In order to mitigate the attack flow, some specific gateway vehicles are selected on the flow tree in order to receive a specific flow rule based on their last P value and the $f(P, \alpha)$ value of victim. If a given gateway vehicle exceeds the threshold, it receives the S_DROP flow rule.

The S_DROP tells them to drop any received packet addressed to the victim from another vehicle. If the vehicle created the packet by itself, the rule allows being

Algorithm 1: Flow Tree Builder

```
1  Function Build_Flow_Tree($T, victim$)
2      Let $T$ be an empty map and $T_x$ an empty list with key $x$
3      Let $S$ be an empty stack
4      $i \leftarrow 0$
5      $counter \leftarrow 0$
6      $T_i \leftarrow T_i \cup \{victim\}$
7      $S.push(victim)$
8      while not $S.empty()$ do
9          $vehicle \leftarrow S.pop()$
10         if $counter = 0$ then
11             $counter \leftarrow |T_i|$
12             $i \leftarrow i + 1$
13         for $neighbor \in vehicle.neighborhood$ do
14             if neighbor has a forward flow rule to victim via vehicle then
15                 $T_i \leftarrow T_i \cup \{neighbor\}$
16                 $S.push(neighbor)$
17         $counter \leftarrow counter - 1$
18     return T
```

forward. The idle and hard timeouts for this flow rule is based on the scale of the attack, increasing the time based on the difference between $f(P, \alpha)$ value and the abnormal value received.

It is important to emphasize that the detection and mitigation phases are separate processes, i. e., the detection process must continue to work even during an attack mitigation. This is due to the characteristic of the network of changing its topology constantly by the movement of the vehicles. Only the detection process can initiate or stop a mitigation because only from time series analysis of traffic load it is possible to verify if the actions of mitigation work properly.

In Figure 5.3, we have the same example of attack scenario of the previous section. However, the detection phase of Sentinel has already classified V as a victim vehicle. The mitigation process generates the flow tree and selects the gateway vehicles in order to send the S_DROP rules to them, mitigating the attack flow and allowing the vehicle V to retrieve his communication interface and being able to exchange messages again.

In order to verify if the attack is finished, the SDVN controller continues to perform the detection verification with the new values received from all victim vehicles, but these values are no longer stored. In order to define an attack flow as terminated, Sentinel expected a consecutive number of neighborhood messages with values below

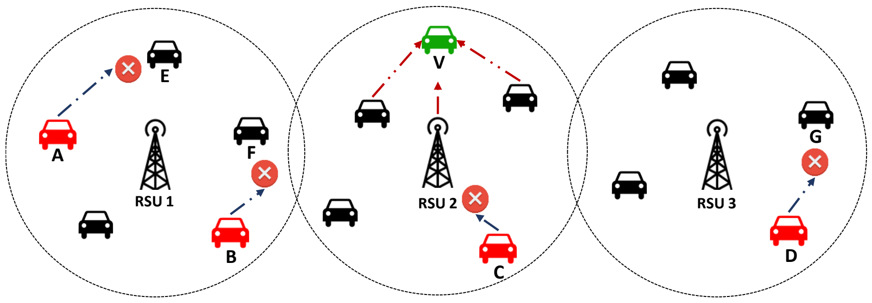


Figure 5.3. An SDVN scenario under attack, but Sentinel has already detected V as a victim vehicle. After flow tree generation, Sentinel selected the vehicles E, F, G and the RSU 2 as gateway nodes and send them the S_DROP flow rule, allowing V to retrieve his communication capability.

the threshold defined in the mitigation phase. After that, the target vehicle loses your victim status, and the values of flows generated and packets processed from neighborhood messages are stored again.

In order to adapt the algorithm to the distributed architecture, the following modifications are required: (1) every RSU manages the P and F values only of the vehicles that are on its command, (2) the RSUs are able to request from the other RSUs of the network a list of generated flows destined for the victim vehicle in the last time interval and (3) the RSU that manages the victim vehicle is the only responsible for distributing the S_DROP rules to all gateway vehicles on the network.

5.4 Complexity Analysis

When designing an algorithm that uses a mathematical approach to get its results, it is important to be aware of the computational cost and overhead added to the network when deploying the method. Now, we will analyze the computational and network complexity of Sentinel according to its phases. First, analyzing the calculations required for the attack detection.

It is necessary to define the value of n, i.e., amount of samples required of packets processed (P) and flows generated (F) that will be stored in the SDVN controller for every vehicle in order to perform the detection calculations. Whenever the SDVN controller receives a neighborhood message, the Equation 5.2 is calculated. Separating the complexities of $g(x)$ and $h(x)$, we have the arithmetic average of a set, defined as

$g(x)$:

$$\begin{aligned} g(x) &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ &= O(n) \end{aligned} \tag{5.3}$$

Now, for the complexity of $h(x)$ which is defined as the variance of a set, we already got the result of μ from $g(x)$, avoiding recalculation. Thus:

$$\begin{aligned} h(x) &= \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2 \\ &= O(n) \end{aligned} \tag{5.4}$$

For every vehicle present in the network, the controller will perform the calculation for P and F set, resulting in two executions whenever it receives the neighborhood message. Defining that v is the number of vehicles present in the network at the time of calculation, we finally have:

$$\begin{aligned} v \times 2f(x,c) &= v \times 2\Big(O(n) + \sqrt{O(n)}\Big) \\ &= v \times 2 \times O(2n) \\ &= O(vn) \end{aligned} \tag{5.5}$$

For the mitigation phase, we have the algorithm 1, which is a modified breadth-first search algorithm. The modified condition in line 14 of the algorithm verifies if the current vehicle has a forward flow rule for the victim vehicle. In order to do this, the SDVN controller needs to perform a linear search on the list of vehicle's flow rule.

During an attack scenario, there will be several vehicles in the network with flow rules for the victim vehicle due to the attackers' action. Therefore, the number of flow rules stored in the controller can reach up to $O(v)$ and its cost is tied to every iteration of the search.

Since the asymptotic complexity of a breadth-first search is $O(|V| + |E|)$, the number of edges $|E|$ of the graph formed from the network connectivity can be considered a linear function from $|V|$ due to transmission range restrictions and geographic characteristics of the network. As V is the set of vehicles in the network, we have that $v = |V|$. Finally:

$$\begin{aligned} Build\_Flow\_Tree &= O(|E| + |V|) \times O(v) \\ &= O(2v) \times O(v) \\ &= O(v^2) \end{aligned} \tag{5.6}$$

The overhead added to the network is mainly into the SDVN controller since it needs to perform the attack detection calculation, where its complexity cost is directly related to the density of the network and the number of samples held per vehicle (Equation 5.5). The vehicles need only keep a processed packet counter in order to send them along with the neighborhood messages.

Chapter 6

Simulation Environment and Result Analysis

Although a real-world vehicular network environment is far from today, we can perform computer simulations to help us to validate this network architecture and mitigation algorithm. Currently, we have well-known software in academia that have the ability to simulate the behavior of vehicles in a virtual road network and simulate wireless communication, including accurate calculation of RSSI between nodes according to the environment.

In this chapter, we will present the software used for the simulations, the assumptions, scenarios, parameters, evaluation metrics, attack model and the results obtained for discussion later.

6.1 Environment Setup and Definitions

In this section, we present the software required to perform the simulations. We detailed how the software communicates between themselves and the versions used. In addition, we present the assumptions and restrictions like transmission power, storage capacity, routes used by vehicles, among other values.

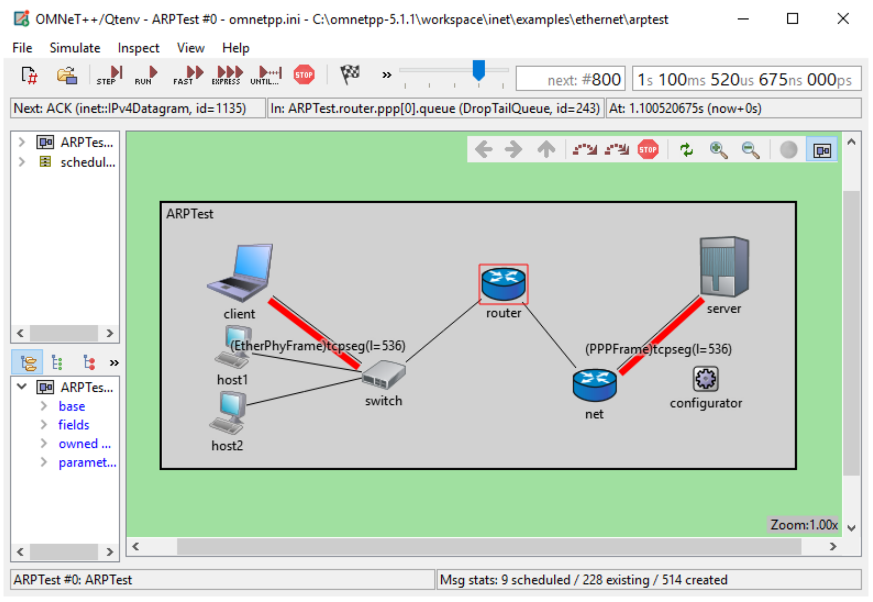
6.1.1 OMNeT++ and INET Framework

The OMNeT++ is a discrete event simulator based on C++ components, used primarily to build and perform network simulations. In order to allow the developers and researchers to build their own simulations, they provide graphical environment Eclipse-based IDE as such as other tools to build the binaries.

The modules are defined using NED files, which define the sub modules, name and default values of the parameters and the message exchange connections between the sub modules, called gates. Furthermore, using the INI file to define a range of the values for the parameters of all the modules present in the simulation, allowing creating an ordered sequence of simulations according to the parameter change.

Compiling the implemented simulations and exported to another computer, as well as allowing the execution of multiple simulation instances in parallel, taking advantage of all available processing cores in the system. One can find detailed information on their website[1]. We show a screenshot of OMNeT++ graphical interface in Figure 6.1.

The INET Framework is a complete set of tools made to work along with OMNeT++, in order to build network simulations, such as wired networks, wireless networks, ad hoc networks and even satellite communications. This open-source library contains the common layers and protocols for the OSI model, such as TCP, UDP, IPv4, IPv6, wired and wireless link layers protocols, such as Ethernet, PPP, IEEE 802.11, and support for mobility models and many other protocols and components.



```
next: #800   1s 100ms 520us 675ns 000ps
Next: ACK (inet::IPv4Datagram, id=1135) | In: ARPTest.router.ppp[0].queue (DropTailQueue, id=243) | At: 1.100520675s (now+0s)
(EtherPhyFrame)tcpseg(l=536)
(PPPFrame)tcpseg(l=536)
ARPTest #0: ARPTest | Msg stats: 9 scheduled / 228 existing / 514 created
```

Figure 6.1. OMNeT++ performing a simulation from INET framework called ARPTest, containing one client and multiple hosts exchanging ARP messages to be aware of the neighborhood address.

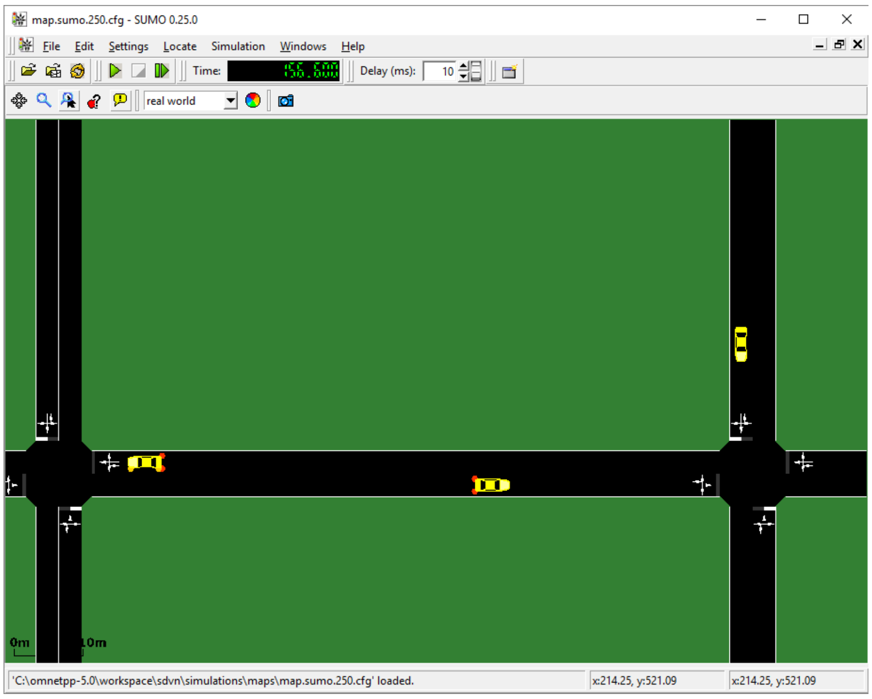
[1]https://omnetpp.org/

6.1.2 SUMO

Simulation of Urban MObility (SUMO) is a set of tools for creating e executing vehicular simulations. It is possible to generate synthetic road networks using the random generator, grid pattern (Manhattan model) and web pattern. It is also possible to import a real scenario from OpenStreetMap[2] to ensure the accuracy of the simulations. Using the buildings and constructions obtained from the real scenarios as input to an obstacle model of signal propagation in vehicular networks.

Beyond to the scenario generation, it is possible to generate trips and routes. Given the road network as input, it is possible to generate a desired number of trips, specifying the minimum distance between start and destination, types of vehicles such as cars, motorcycles, buses, trucks, pedestrians, trains, with different acceleration patterns, maximum speed, etc.

One can find detailed information on their website[3]. We show a screenshot of the graphical interface of SUMO in Figure 6.2.



'C:\omnetpp-5.0\workspace\sdvn\simulations\maps\map.sumo.250.cfg' loaded.

Figure 6.2. SUMO executing simulation on a grid network. The vehicles are aware of corners and traffic lights, accelerating and braking, making the simulation more realistic.

[2]http://www.openstreetmap.org
[3]http://sumo.dlr.de/

6.1.3 Veins and SimuLTE Framework

Vehicles in Network Simulation (Veins) is an open source framework implemented in C++ and presented by Sommer et al. [2011] that allows the simulation of vehicular networks. It offers complete implementation of the IEEE 802.11p standard, signal propagation models such as Two-Ray Interference Model, Obstacle Shadowing, Vehicle Obstacle Shadowing, Antenna Patterns, among other modules. They offer the IEEE 1609.4 DSRC/WAVE network layer with the implementation of multiple channels operation, priority control, and QoS channel access.

Every simulation using Veins works by executing two simulators in parallel: OMNeT++ (for network simulation) and SUMO (for road traffic simulation). Connecting both simulators via a TCP socket. The protocol for this communication has been standardized as the Traffic Control Interface (TraCI). This allows bidirectionally-coupled simulation of road traffic and network traffic. Reflecting movement of vehicles in SUMO as the movement of nodes in an OMNeT++ simulation [Sommer, 2016]. We show the structure of this architecture in Figure 6.3.

SimuLTE is a simulation tool presented by Virdis et al. [2015] that allows the use of long-range communication, defined as Long Term Evolution (LTE). They implemented the tool in C++ and built on the OMNeT++ and the INET framework. Veins can also be integrated allowing vehicles to have heterogeneous communication.

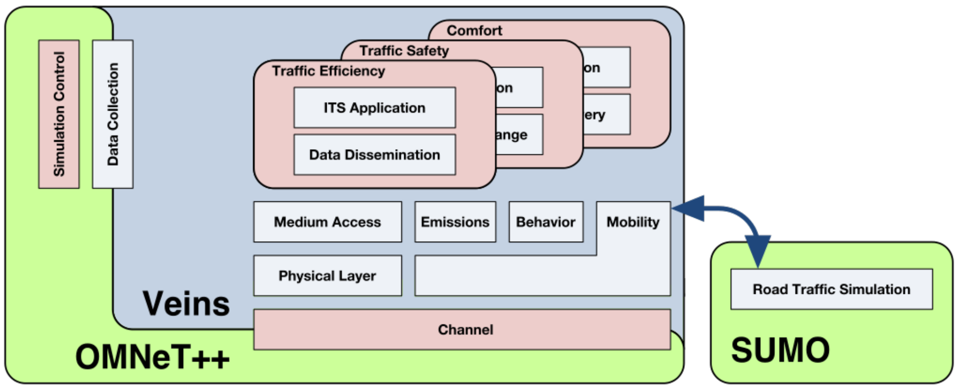
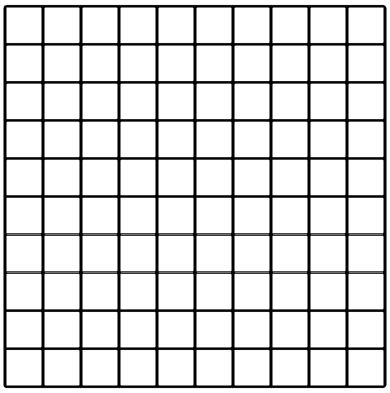


Figure 6.3. A vehicle simulation is performed at a low level via OMNeT ++, performing message exchanges through the basic modules, while SUMO provides the information about the movement of vehicles. Veins allows intercommunication between the two simulations. Adapted from Sommer [2016].

6.1.4 Scenarios

Two scenarios were used to perform the simulations, the synthetic was used to evaluate the first results of the α and β parameters and the density of the RSUs in the environment uniformly placed. This scenario was generated using *netgenerate*, which is a road generation tool included in SUMO package, resulting in a grid map with 10 x 10 road segments where each one is 100 meters long, as can be seen in Figure 6.4a.

Using the realistic scenario to evaluate the behavior of the algorithm according to the most realistic model of mobility for the vehicles. This scenario was generated using *netconvert*, which is also included in the SUMO package, allowing to import public road information data from OpenStreetMap. The scenario used in the simulations is based on a real location[4] with approximately 12 km of medium length. We show the scenario overview in Figure 6.4b.


(a) Synthetic Scenario.

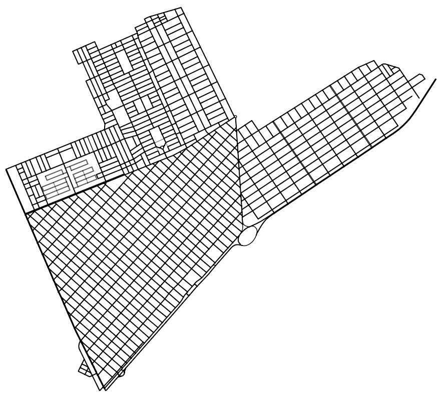

(b) Realistic Scenario.

Figure 6.4. Scenarios utilized to perform the simulations. Both scenarios were generated using the tools provided by SUMO package.

6.1.5 Assumptions

Every legitimate vehicle present on the simulation executes an ICMP request that generates five requests per second to a valid random destination; broadcasts two beacon packets per second reporting to nearby vehicles their presence; also sends one neighborhood report per second to the SDVN controller, reporting their current neighbors in order to upgrade the global topology of network.

The routes of every vehicle were established randomly, but the vehicles will choose the shortest path between the starting point and the destination. They can store up

[4]Coordinates: (-20.389893, -54.560195)

to 100 data packets in the "Packet In Buffer" and more 100 in the "Standby Buffer", with a maximum of 50 active flow rules simultaneously.

The RSUs have the same communication interfaces as the vehicles, allowing 802.11p communication with the vehicles and LTE with the SDVN controller, as well as obtaining a direct communication interface between all the RSUs present in the network. However, RSUs only forward received messages and they cannot be a final destination for a message. We positioned the RSUs uniformly in the scenario according to the set density on simulation parameters.

6.1.6 Attack Model

We implemented the attack model in the simulations as follows: For the detection tests, the simulation chooses an average of 25% of the vehicles to belong to the *botnet* and attack the victim vehicle for a short period of time.

After that, the simulation chooses another victim vehicle victim and so on. On average, the simulation selects 20% of vehicles as a victim at each run. For mitigation and complexity tests, the *botnet* selects only one victim vehicle and attacks it for 60 seconds. We varied the attack load in each run.

The attack load is a large amount of ICMP requests with spoofed source addresses. The zombie vehicles send this message to a near vehicle to be forward as a normal packet, without requesting a flow rule to the SDVN controller in order to hide their behavior. If an attacker moves enough to stay out of range of the chosen car, it will search for another legitimate vehicle in order to forward the attack flow.

6.1.7 Parameters and Evaluation Metrics

In order to execute the simulations, we use OMNeT++ 5.0, which is a well-known network simulator in the literature. Furthermore, the Veins framework 4.4 [Sommer et al., 2011] is used to connect movement of vehicles and drive commands from SUMO to the network simulator and also provides the implementation of the IEEE 1609.4 upper MAC layer and IEEE 802.11p physical layer. We applied the SimuLTE framework [Virdis et al., 2015] in order to represent the LTE connectivity between the vehicles, RSUs and the SDVN controller. We presented the description of the parameters used to execute the simulation in Table 6.1.

Table 6.1. Parameters of the simulation

Parameter	**Value**
Vehicles Density	100 up to 500 vehicles
Vehicles Max Speed	15 up to 40 m/s
RSUs Density	4 $units/km^2$
Transmission Power	5 $mW \approx 250$ m
Bit Rate	18 $Mbps$
Propagation Model	$Free\ Space\ Path$

To evaluate the proposed approach, we analyzed the simulation results with the following metrics: The detection method uses the detection rate and false positive rate and we evaluated the mitigation method according to the attack packets dropped directly by the flow rule created by Sentinel. We used the following metric equations:

Detection Rate (DR) Percentage value of vehicles correctly detected as victims. TP means *True Positive* and FN means *False Negative.*

$$DR = \frac{TP}{TP + FN} \tag{6.1}$$

False Positive Rate (FPR) Percentage value of vehicles incorrectly detected as victims. FP means *False Positive* and TN means *True Negative.*

$$FPR = \frac{FP}{FP + TN} \tag{6.2}$$

Average Mitigation Rate (AMR) Percentage value of the attack flow totally mitigated by Sentinel's actions. Ps is a number of packets dropped by Sentinel and Pa is the total amount of packets generated by attackers.

$$AMR = \frac{Ps}{Pa} \tag{6.3}$$

6.2 Results

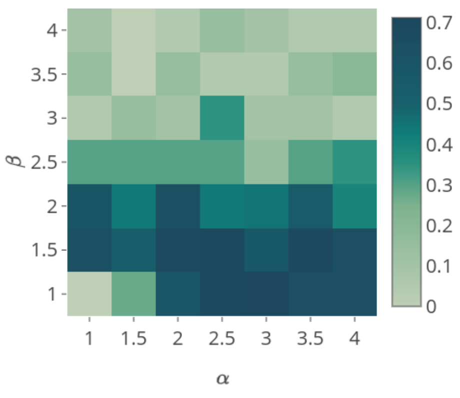
The results were obtained with the average of 10 executions for each scenario, traffic load, α and β values, vehicles density and maximum speed.

6.2.1 Victims Detection

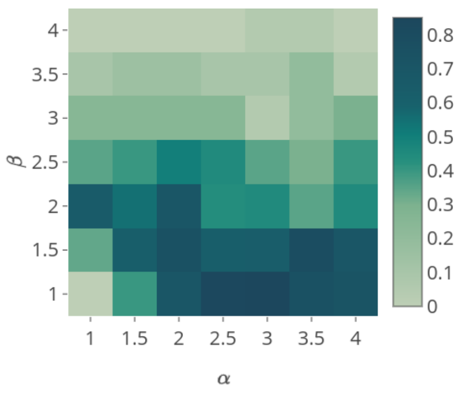
In order for the Sentinel detection phase to work correctly, we tested the values of the constants presented in section 5.2 repetitively during the same simulation scenario until a tuple of values is found that is reliable enough to perform a fine-tuning in later simulations.

In order to find the best combination of α and β values for the algorithm, the same simulation settings were used by varying the values from 1 to 4 for both constants and the attack load on victim vehicles.

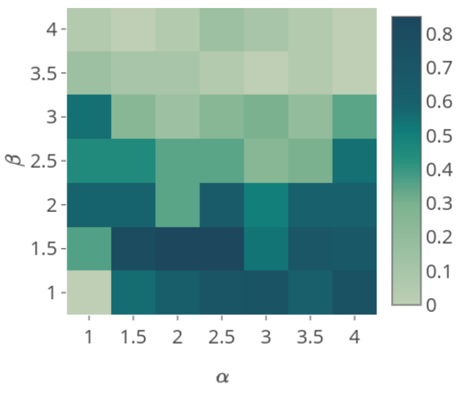
The results of detection rate and false positive rate were combined with the function $h = max\{0, dr - 5fpr\}$ in following heat maps below to visualize the best calibration of parameters. In Figure 6.5, we have the result of this set of executions.



(a) 5x Load



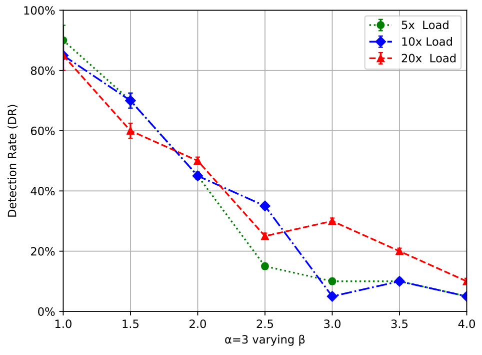
(b) 10x Load



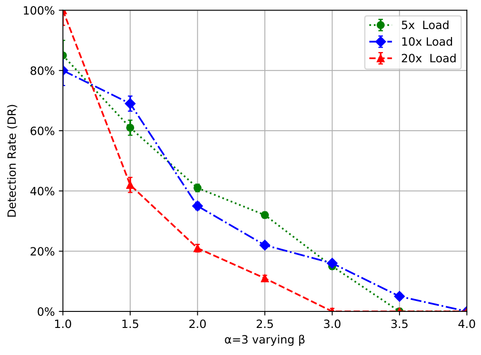
(c) 20x Load

Figure 6.5. Heatmaps using function h varying the values of α and β and attack load applied.

The values closest to satisfying the relationship between DR and FPR were $\alpha = 3$ and $\beta = 1$. Using these values, in Figure 6.6 we have the impact on DR of simulations while increasing the traffic load generated by the bots, the α value set as 3 and varying the β value for both scenarios.



(a) Synthetic Scenario

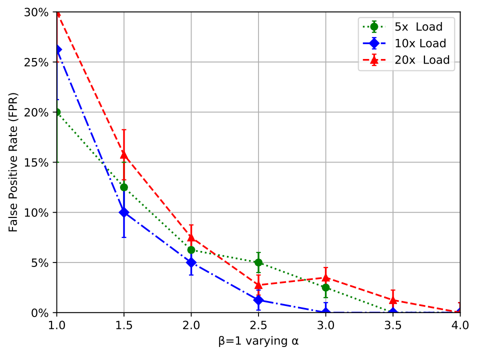


(b) Realistic Scenario

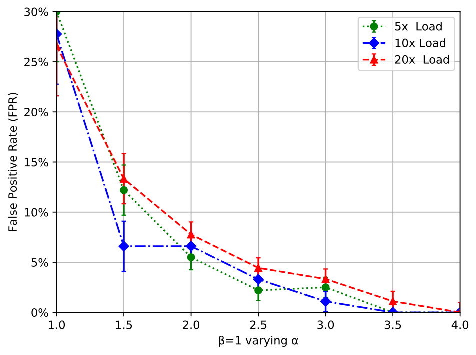
Figure 6.6. Impact of β on the detection rate between the synthetic and realistic scenarios using different attack loads.

As seen in the results, the β values affect directly on DR. These detection rates showed that the size of the botnet used in attack traffic is dependent of β, due to the relation between the growth amount of flows generated and members of the botnet.

Also using these values above, in Figure 6.7 we have the impact on FPR of simulations while increasing the traffic load generated by the bots, the β value set as 1 and varying the α value for both scenarios.



(a) Synthetic Scenario



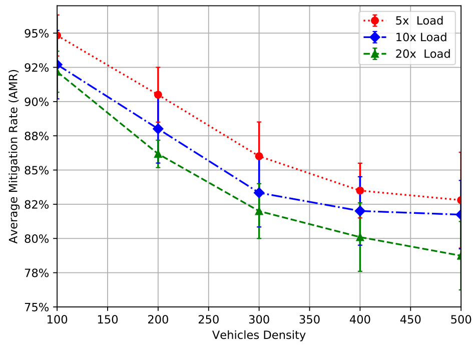
(b) Realistic Scenario

Figure 6.7. Impact of α on the false positive rate between the synthetic and realistic scenarios using different attack loads.

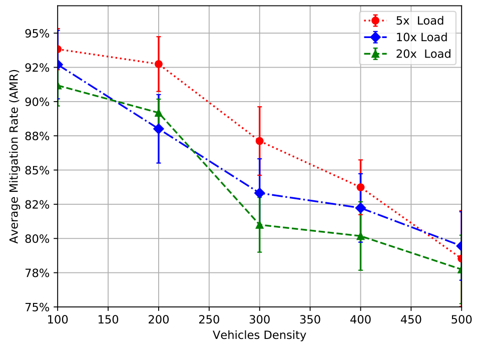
The α values influence directly on FPR, due to some moments of high packet rate of legitimate vehicles are incorrectly detected as attacks flow due to the low α value. In order to classify as an attack flow, the network manager must configure this parameter properly.

6.2.2 Attack Mitigation

In order to validate the attack mitigation results, we used the relationship between all attack packets created and the attack packets successfully deleted by Sentinel. In Figure 6.8, we have the value of AMR according to the density of the network and executed over different attack loads.



(a) Synthetic Scenario.

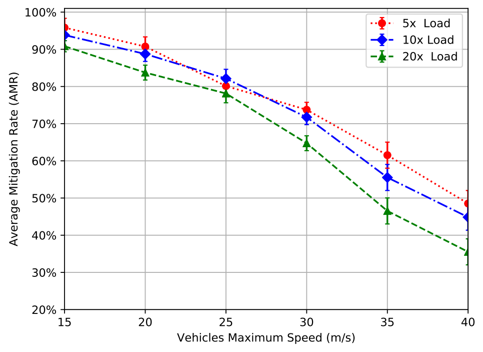


(b) Realistic Scenario.

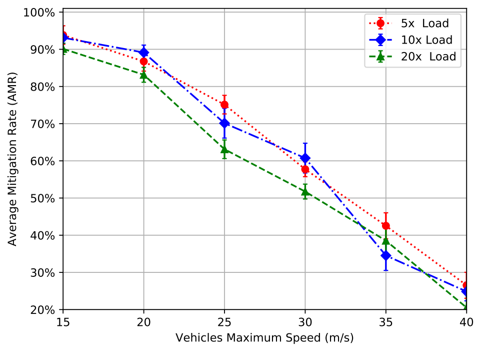
Figure 6.8. Impact of vehicle density on average mitigation rate between the synthetic and realistic scenarios using different attack loads.

As can be seen, the AMR falls slightly with increasing vehicle density due to the constant changes of gateway vehicle made by the attackers. The process of flow tree building must continue to work in order to find the new gateway vehicles until the values fall below the threshold. In the realistic scenario, the decrease in the AMR is more pronounced.

In Figure 6.9, we have the value of AMR according to the maximum speed of vehicles with the density fixed in 250 vehicles.



(a) Synthetic Scenario.

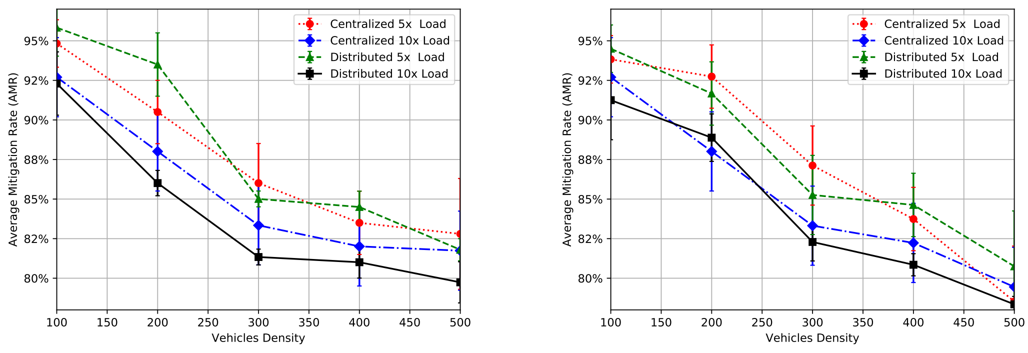


(b) Realistic Scenario.

Figure 6.9. Impact of maximum vehicle speed on average mitigation rate between the synthetic and realistic scenarios using different attack loads.

The results show that the maximum vehicle speed heavily affects the efficiency of the mitigation, mainly because the timeouts used in the S_DROP rule are fixed. With the vehicles moving faster, the attackers can quickly switch the gateway vehicle and most of the attack flow might reach the victim vehicle.

The results in Figure 6.10 shows the behavior of the AMR using the same parameter conditions and vehicles density simulations, but in different SDVN architectures in order to verify some behavior change.



(a) Synthetic Scenario. (b) Realistic Scenario.

Figure 6.10. The impact between centralized and distributed SDVN architectures on average mitigation rate using the synthetic and realistic scenarios over different attack loads.

The results presented a very similar AMR behavior in both architectures. Indicating that the paradigm change to multiple controllers does not affect the operation of the algorithm. Suggesting the distribution of your processing load between the local controllers without performance loss or DR and AMR drops.

6.2.3 Messages Overhead

During the mitigation phase, Sentinel must continuously generate the flow tree in order to obtain the gateway vehicles and send them a S_DROP rule. So, we check in the tests below if these generated rules are not increasing the data flow in the network to the point of saturating the network interface of other vehicles.

In Figure 6.11, we have the number of S_DROP rules generated in the network during an attack scenario, applying moving average in the results in order to visualize the trend of the number of rules generated by the Sentinel. We executed the flooding attack during 60 seconds on a single victim (after a warm-up period) while the amount of S_DROP rules was observed.

However, we expected that as the attack occurs during the middle of the simulation where half of the vehicle density is active. Therefore, the simulation selects 25% of active vehicles as attackers and thus we expected that there would be at least one gateway vehicle for every attacker in action. We marked this expected trend in a solid horizontal line on Figure 6.11.

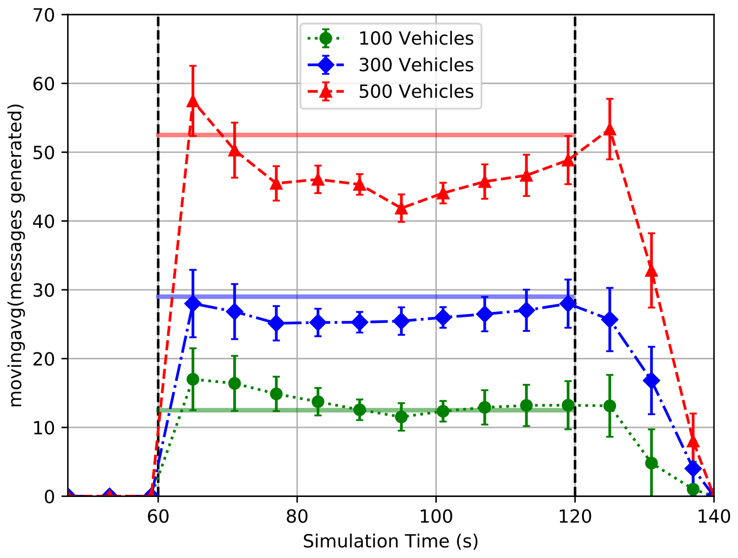


Figure 6.11. Moving average of S_DROP flow rule generation for the gateway vehicles during an attack scenario. The values remain approximately constant due to gateway vehicle exchanges performed by Sentinel.

As seen in the above results, the number of S_DROP rules created remains apparently constant to the number of possible attackers present on the network, showing that Sentinel was able to approximately select one gateway vehicle for each attacker in order to mitigate the attack flow generated.

In the next chapter, we will give an overview of the results obtained in this section and we will discuss the next steps to be taken in this work about the problems of speed of the vehicles and the possible change in the detection approach.

Chapter 7

Conclusion and Future Work

In this work, we presented two proposals for SDVN architecture and a new defense mechanism to mitigate flooding attack. Although many research works already proposes new security mechanisms of VANETs and SDN, we observe the lack of requirements for this new architecture and we expect more proposals of defense approaches and security mechanisms in the future.

As seen in chapter 3, there are already SDVN architecture proposals in the academy, where each work has its specialty (cost-based transmission, power control, etc). To make sure that Sentinel could work on any SDVN, it was necessary to define the architecture itself in this work in order to reduce its complexity. Although even after simplifying, both architectures worked very well obtaining high package delivery rates.

The results obtained by the simulations were promising, showing that despite the attack load used, Sentinel is able to drop most of the attack packets before reaching the victim vehicle. Simulations using scenarios with high-speed vehicles have shown a loss of efficiency in mitigation rate, although it does not affect the detection of an attack scenario.

The increased density of vehicles in the network adds fluctuations in the results of DR and RPF, being necessary the variation of the alpha and beta constants values to obtain satisfactory results. However, the attack load does not seem to directly influence to the point that it needs to change the values of the constants.

We also observed that the results between the centralized and distributed SDVN architectures had no practical effect for Sentinel execution. We considered a significant reduction of the algorithm complexity since executing simultaneously several instances of the algorithm within each controller module installed in the RSUs and with a reduced number of vehicles.

It was possible to observe various other security issues inheriting from the previous network architectures, but we needed to treat them differently for the context of vehicular networks. Although most mitigation approaches are specific to one specific security attack, the concept of applying authentication schemes seems to be promising for this area of research.

This proposal for a defense mechanism opens the path for new approaches to detection and mitigation of availability attacks in networks based on statistical analysis of data traffic. New mechanisms of data classification using machine learning may even replace classical classification algorithms based on probabilities.

7.1 Future Work

The results show that the mitigation method does not work optimally in all cases, especially in high-speed scenarios. However, it has great potential for improvement. There are suggestions in the subsections below for possible improvements in the algorithm:

7.1.1 Dynamic timeouts for flow rules

The results obtained by changing the speed of the vehicles in the network showed that the flow rules created by Sentinel are not effective enough. A suggested improvement is to change the flow rule timeout values to ensure that rules are cleared quickly in high-speed scenarios and that rules are maintained for longer time in low-speed scenarios.

7.1.2 Predicting flow tree modifications

The flow tree is a fundamental component of the mitigation phase of the algorithm. However, the results obtained in the high-speed scenarios show that the flow tree generation might not mitigate the attack flow due to the rapid exchange of gateway vehicles by the attackers. In order to improve these results, a suggestion is to use prior knowledge of the road network where SDVN is implanted and try to predict which vehicles will be next to receive the attack flow.

7.1.3 Identifying botnet members

The algorithm traces the source of attack packets at the current time when generating the flow tree. We might use the data obtained from the generation of this tree to identify the vehicles that are members of the botnet. By identifying the vehicles that generate the attack packets, it would be possible to create a new flow rule for all legitimate vehicles on the network to drop any messages coming from those vehicles.

7.1.4 Other approaches for detection analysis

Even using time series analysis, the detection algorithm proved to be efficient in successfully detecting attack scenarios. However, a significant improvement would be to remove the use of calibration constants and use other classification approaches known in the literature. We have below a list of applicable classification approaches:

Naive Bayes classifier Based on Bayes' theorem, Naive Bayes classifier is a supervised learning method and a statistical method for data classification. Creating a tuple from the values of P and F, use this tuple as input to classify the data into classes of normal and abnormal behavior.

Support Vector Machine Proposed by Vapnik and Chervonenk in 1963, Support Vector Machine (SVM) is a discriminative classifier formally defined by using a hyperplane to separate the data, i.e., given a training data (supervised learning), the algorithm must outputs an optimal hyperplane which classifies the data in different categories.

Local outlier factor Proposed Breunig et at. in 2000, Local outlier factor (LOF) is an algorithm used to identify anomalies based on how isolated the object is with respect to the surrounding neighborhood. Also creating a tuple from the values of P and F, use this tuple as training set and a given threshold for the outlier factor, it will be possible to detect an anomaly within the SDVN without generating major changes in the detection and mitigation phases. It is by far the anomaly detection method that most fits into the Sentinel implementation.

Bibliography

Alattar, M., Sailhan, F., and Bourgeois, J. (2012). Trust-enabled link spoofing detection in manet. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 237–244. ISSN 1545-0678.

Alheeti, K. M. A., Gruebler, A., and McDonald-Maier, K. D. (2015). On the detection of grey hole and rushing attacks in self-driving vehicular networks. In *Computer Science and Electronic Engineering Conference (CEEC), 2015 7th*.

Almutairi, H., Chelloug, S., Alqarni, H., Aljaber, R., Alshehri, A., and Alotaish, D. (2014). A new black hole detection scheme for vanets. In *Proceedings of the 6th International Conference on Management of Emergent Digital EcoSystems*.

Bozkaya, E. and Canberk, B. (2015). Qoe-based flow management in software defined vehicular networks. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6.

Guedes, D., Vieira, L., Vieira, M., Rodrigues, H., and Nunes, R. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. In *Minicursos do Simpósio Brasileiro de Redes de Computadores SBRC*, pages 160–210.

He, Z., Zhang, D., and Liang, J. (2015). Cost-efficient heterogeneous data transmission in software defined vehicular networks. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 666–671.

IEEE (2005). Ieee standard for information technology - local and metropolitan area networks.

Jeon, Y., Kim, T. H., Kim, Y., and Kim, J. (2012). Lt-olsr: Attack-tolerant olsr against link spoofing. In *37th Annual IEEE Conference on Local Computer Networks*, pages 216–219. ISSN 0742-1303.

Jiang, D. and Delgrossi, L. (2008). Ieee 802.11p: Towards an international standard for wireless access in vehicular environments. In *VTC Spring 2008 - IEEE Vehicular Technology Conference*, pages 2036–2040. ISSN 1550-2252.

Ku, I., Lu, Y., Gerla, M., Gomes, R. L., Ongaro, F., and Cerqueira, E. (2014). Towards software-defined vanet: Architecture and services. In *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, pages 103--110. IEEE.

Macedo, D. F., Guedes, D., Vieira, L. F. M., Vieira, M. A. M., and Nogueira, M. (2015). Programmable networks - from software-defined radio to software-defined networking. *IEEE Communications Surveys Tutorials*, 17(2):1102–1125.

Mahmoud Al-Qutayri, C. Y. and Al-Hawi, F. (2010). Security and privacy of intelligent vanets. In *Computational Intelligence and Modern Heuristics*. InTech.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69--74. ISSN 0146-4833.

Mousavi, S. M. and St-Hilaire, M. (2015). Early detection of ddos attacks against sdn controllers. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 77–81.

Pathre, A., Agrawal, C., and Jain, A. (2013). A novel defense scheme against ddos attack in vanet. In *2013 Tenth International Conference on Wireless and Optical Communications Networks (WOCN)*.

Qian, Y., You, W., and Qian, K. (2016). Openflow flow table overflow attacks and countermeasures. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 205–209.

Raya, M. and Hubaux, J.-P. (2005). The security of vehicular ad hoc networks. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks*.

Sahoo, P. K., Chiang, M.-J., and Wu, S.-L. (2014). Svanet: A smart vehicular ad hoc network for efficient data transmission with wireless sensors. *Sensors*, 14(12):22230--22260. ISSN 1424-8220.

Sommer, C. (2016). Documentation - vehicles in network simulation - veins. `http://veins.car2x.org/documentation/`.

Sommer, C., German, R., and Dressler, F. (2011). Bidirectionally coupled network and road traffic simulation for improved ivc analysis. *IEEE Transactions on Mobile Computing*, 10(1):3–15.

Sudheera, K. L. K., Ma, M., Ali, G. G. M. N., and Chong, P. H. J. (2016). Delay efficient software defined networking based architecture for vehicular networks. In *2016 IEEE International Conference on Communication Systems (ICCS)*, pages 1–6.

Uzcategui, R. A., Sucre, A. J. D., and Acosta-Marum, G. (2009). Wave: A tutorial. *IEEE Communications Magazine*, 47(5):126–133. ISSN 0163-6804.

Virdis, A., Stea, G., and Nardini, G. (2015). *Simulating LTE/LTE-Advanced Networks with SimuLTE*, pages 83--105. Springer International Publishing.

Zargar, S. T., Joshi, J., and Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Communications Surveys Tutorials*, 15(4):2046–2069. ISSN 1553-877X.