Sentinel: Defense Mechanism against DDoS Flooding Attack in Software Defined Vehicular Network

Gabriel de Biasi, Luiz F. M. Vieira, Antônio A. F. Loureiro
Computer Science Department - UFMG
Belo Horizonte - Brazil
{*biasi,lfvieira,loureiro*}*@dcc.ufmg.br*

创新点？ - a new defense mechanism?

***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. This paper presents Sentinel, a new defense mechanism 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 mitigation method. The algorithm was able to mitigate the attack flow in different parameters. Sentinel reached an average mitigation rate of more than 78% in all densities scenarios.**

I. Introduction

Vehicular Ad hoc Network [1] is a category of mobile ad hoc network, which applies the same mobile routing protocols to allow inter-vehicle data communications and support the Intelligent Transportation Systems (ITS). 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 this network, the *Vehicle* and the *Road Side Unit* (RSU). The vehicles are entities able to communicate with each other, and the RSUs are antennas next to the road that sends and receives information to nearby vehicles and behaves like an exchange point. Therefore, there are two main ways of communication, being Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I). However, because of the highly dynamic topology of this network, it was necessary to look for solutions that would have the ability to manage the vehicles and the way they communicate with each other. Software-Defined Network [2] is characterized as a network which data plane and control plane are separated, allowing managing the network behavior by defining flow rules for each node belonging to the network. In order to achieve this network behavior, the OpenFlow Protocol [3] was developed allowing nodes belonging to the network to have a secure and direct communication with the SDN controller, receiving and sending forward policies packets around the network. For example, 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. 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, allowing the communication between them.

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 main benefit to apply the concepts of SDN in a vehicular environment is the possibility of optimal routing, selecting the shortest paths to forward packets between the vehicles, because of 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 high-traffic environments, the SDN controller may suggest reducing transmission power in some areas in order to reduce interference in communications, as well as increase the transmission power in low-traffic environments to reach the most distant vehicles.

However, with a new network architecture, it is necessary to check if the security requirements are still satisfied. In this paper, we propose Sentinel, a new approach to detect and mitigate Flooding Attack in SDVN. We also use a network simulator to examine the damage that this attack may cause on the network. Furthermore, we propose future improvements to mitigate flooding attack regardless of the density of the SDVN.

This paper is organized as follows. Section II describes the related work. Section III presents the software-defined vehicular networks architecture. Section IV details Sentinel, the proposed defense mechanism. Section V brings the attack simulations and defense result analysis. Finally, Section VI concludes the paper.

II. Related Work

In this section, we discuss some security approaches in the literature that handling denial of service attacks in vehicular networks and SDN.

Despite being an ad hoc-based network, VANETs also suffers from DoS attacks. In these cases, the defense mechanisms classifies 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

978-1-5386-3180-5/18/$31.00 ©2018 IEEE

Denial of Service (DDoS) Attack, using the information gathered by the RSUs [4]. 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.

In [5], they present a different approach to detecting DDoS Flooding Attack using Self-Organizing Maps (SOM). Their approach keeps the statistics about the flows in the switches in order to characterize an abnormal behavior. They send some samples of these statistics to the classifier module on the controller that runs the SOM and get a winning neuron. According to your neuron's type, they classify the behavior as an attack or not. To map the neurons on a 2D topological map, they use several traffic metrics, such as the average of packets per flow, the percentage of pair-flows, and growth of single-flows, among others. They trained the SOM using samples of no-attack and attack periods, varying a number of parameters applied to the neurons. Relying on SOM to detect attacks behavior always demands a training phase. Moreover, the network might be changing their behavior constantly, and they might update the scenario.

III. Architecture of Software Defined Vehicular Network

In this section, we present the proposed SDVN architecture used in the security attacks simulations and how we deploy the control and data plane communications. The architecture chosen for the SDVN is completely centralized, i. e., all vehicles and RSUs in the network have direct communication with a unique controller using a long-range network interface, such as LTE, in order to request and receive flow rules.

We use short-range network interface for the data communication, in this case using Wireless Access in Vehicular Environments (WAVE), defined by the IEEE 1609, which enables communication between vehicles in a dedicated range of the electromagnetic spectrum. This interface is present in all vehicles and RSUs on the network.

We consider every vehicle present in the network as an OpenFlow switch, capable of sending, receiving and forwarding packets. To achieve this behavior, the vehicles have two network interfaces, LTE to exchange control packets and 802.11p to exchange data packets between the other vehicles and RSUs. We implemented a flow table to store the flow rules received from the controller that is fresh enough to reuse. 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. However, it is possible that the vehicle is isolated

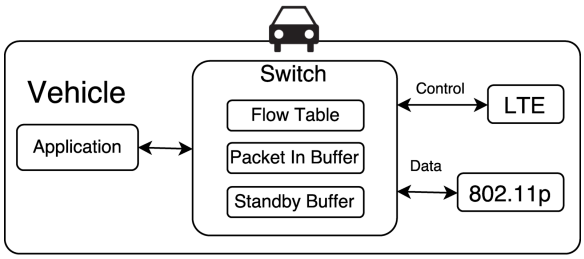


Fig. 1. Centralized Vehicle Architecture.

from the others and RSUs, so a new flow rule response called *STANDBY* has been created, which tells the vehicle to store the message in the "Standby Buffer" and resend the *PACKET_IN* after a certain time. This architecture concept can be seen in Figure 1.

To calculate an efficient routing of network packets, the SDVN controller needs to be aware of the active vehicles and their direct neighbors. Thus, every vehicle in the network must send beacon messages to their neighbors by informing its presence and storing the beacons received. The vehicles must send these beacon messages in a short interval, somewhere about 500 milliseconds. Furthermore, they must send the neighborhood messages, which are messages sent to the SDVN controller informing the current neighbors. Upon receiving the message, the controller keeps refreshing the adjacency matrix between the vehicles in order to create the flow rules using efficient routes. However, the interval between the neighborhood messages is inversely proportional to the maximum speed of the vehicles on the roads, considering that the speed of vehicles makes it difficult to characterize the current network topology. The illustration of the architecture defined for the simulations can be seen in Figure 2.

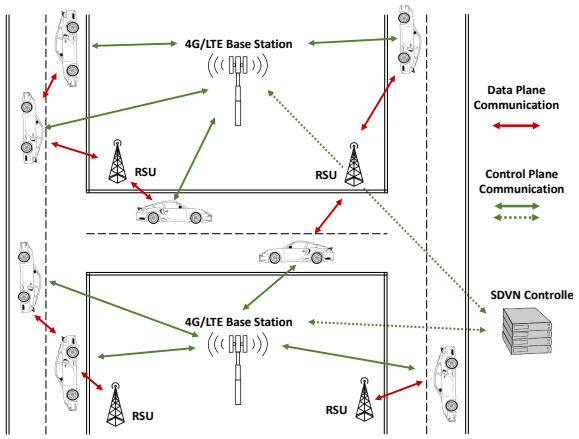


Fig. 2. Software Defined Vehicular Network.

IV. Proposed Approach: Sentinel

Sentinel is our proposed flooding mitigation mechanism designed for highly dynamic SDVNs. It works entirely within the SDVN controller and creates a new specific flow rule to

prevent forwarding of attack packets. We separated the mechanism into two phases, which are detection and mitigation. We explain these functionalities in the next subsections.

A. Phase 1: Detecting the Attack

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 integer constant N, which means the last N values of packets processed and flow rules generated are stored in order to characterize your traffic load. Every time that a vehicle sends its neighborhood message to the controller informing its current neighborhood, it also sends the number of packets processed P since the last message. At the same time, the controller maintains the number of flow rules generated F to this vehicle. To determine if a given vehicle on the network is under attack, the controller performs as follows: Let the functions $g(x)$ be the arithmetic average of a given set x, $h(x)$ be the variance of a given set x and $f(x, c)$ be a function defined according to (1).

$$f(x, c) = c \times \left(g(x) + \sqrt{h(x)} \right) \tag{1}$$

The constant c is applied to refine the detection behavior of the algorithm according to the attack dimensions. As the values of P and F are usually different and unrelated, it is referred by two different constants, α and β.

If the vehicle already sent N values to the controller, every time that report comes with new values of packets processed and flows rules generated the controller calculates $f(x, c)$ for the P and F sets, according to (2).

$$\begin{aligned} newValuePacket &> f(P, \alpha) \\ newValueFlow &> f(F, \beta) \end{aligned} \tag{2}$$

The function $f(x, c)$ represents a threshold to detect an abnormal behavior of the vehicle based on the previous values of P and F. If the vehicle sends the report and the new values exceed the threshold for both sets, Sentinel classifies the vehicle as a possible victim. Sentinel manages these possible victims differently based on the current situation. The next values of P and F will not be stored anymore until the 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.

Using the number of packets processed and the number of flow rules generated together to detect the flooding attack helps to avoid false positives detections. For instance, using only α might detect vehicles located out of an RSU range and processing large amounts of packets when recovering its connectivity. On the other hand, using only β might detect service provider vehicles as victims by the flow rules generated but without generating a large traffic load.

B. Phase 2: Mitigating the Attack

Whenever the controller detects a possible flooding attack by detection phase, the mitigation process starts to build a flow tree to localize the bots generating the traffic load. We assumed that the botnet is broadcasting spoofed packets, i. e., there is no direct reference of whom 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 bots. In this approach, we will refer to them as gateway vehicles.

Once Sentinel identifies the vehicles receiving the attack flow, it is possible to perform the mitigation process at different points in the network to deny the attack forwarding to the victim. The flow tree building process works as follows: first, Sentinel sets the victim vehicle as tree root and selects its direct neighbors based on which one has flow rule with final 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 remains, generating a flow tree. Algorithm 1 details this functionality.

Algorithm 1: Flow Tree Builder

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

To mitigate the attack flow, some specific gateway vehicles are selected on the flow tree 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 the packet to be forward. The idle and hard timeouts for this

flow rule are 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 network changing topology characteristic constantly by the vehicles movement. Only the detection process can start or stop a mitigation process because only from the traffic load time series analysis it is possible to verify if the mitigation actions work properly. To verify if the attack is finished, the SDVN controller continues to perform the detection check with the new values received from all victim vehicles, but these values are no longer stored.

To define an attack flow as terminated, Sentinel expected a consecutive number of neighborhood messages with values below the threshold defined in the mitigation phase. After that, the target vehicle loses its victim status, and the values of flows generated and packets processed from neighborhood messages are stored again.

C. Complexity Analysis

It is important to be aware of the computational cost and overhead added to the network when deploying the method. Here, we analyze the computational and network complexity of Sentinel according to its phases. First, we analyze 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 2 is calculated. Separating the complexities of $g(x)$ and $h(x)$, we have the arithmetic average of a set, defined as $g(x)$:

$$\begin{aligned} g(x) &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ &= O(n) \end{aligned} \tag{3}$$

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:

$$\begin{aligned} h(x) &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \\ &= O(n) \end{aligned} \tag{4}$$

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:

$$\begin{aligned} v \times 2f(x, c) &= v \times 2\Big(O(n) + \sqrt{O(n)}\Big) \\ &= v \times 2 \times O(2n) \\ &= O(vn) \end{aligned} \tag{5}$$

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 to the victim vehicle. 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:

$$\begin{aligned} Build\_Flow\_Tree &= O(|E| + |V|) \times O(v) \\ &= O(2v) \times O(v) \\ &= O(v^2) \end{aligned} \tag{6}$$

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). The vehicles need only to keep a processed packet counter in order to send them along with the neighborhood messages.

V. Performance Evaluation

In this section, we present the assumptions, the scenario used in the simulations, the applied attack model and the specific parameters of density, evaluation metrics, mobility model, propagation model, among others.

A. Scenario

We generated the scenario 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[1] with approximately 12 km of medium length. The scenario overview can be seen in Figure 3.

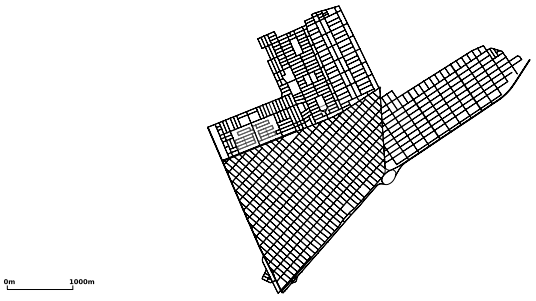


Fig. 3. Simulation scenario.

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

B. Assumptions

Every legitimate vehicle present in the simulation executes an Internet Control Message Protocol (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.

We established the routes of every vehicle randomly. The vehicles will choose the shortest path between the starting point and the destination. They can store up to 100 data packets in the "Packet In Buffer" and more 100 packets 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 density set on simulation parameters.

C. 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 and so on. On average, the simulation selects 20% of vehicles as 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 vehicles belonging to the botnet send this request to the closest 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.

D. Parameters and Evaluation Metrics

To execute the simulations, we used OMNeT++ 5.0, which is a well-known network simulator in the literature. Furthermore, we used the Veins framework 4.4 [6] to connect movement of vehicles and drive commands from SUMO to the network simulator and it also provides the implementation of the IEEE 1609.4 upper MAC layer and IEEE 802.11p physical layer. We applied the SimuLTE framework [7] to represent the LTE connectivity between the vehicles, RSUs and the SDVN controller. Table I presents the parameters description used in the simulations.

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:

TABLE I
SIMULATIONS PARAMETERS

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

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

$$DR = \frac{TP}{TP + FN} \tag{7}$$

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

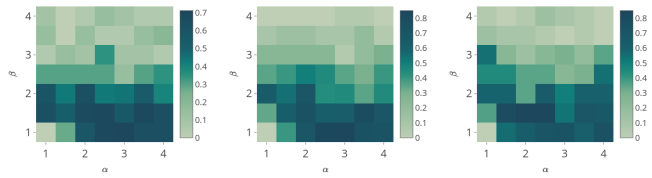
$$FPR = \frac{FP}{FP + TN} \tag{8}$$

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

$$AMR = \frac{Ps}{Pa} \tag{9}$$

E. Results

1) Victims Detection: To find the best combination of α and β values for the algorithm, we used the same simulation settings varying the values from 1 to 4 for both constants and the attack load on victim vehicles. We combined the results of DR and FPR with the function $h = max\{0, dr - 5fpr\}$ in three heat maps to visualize the best parameters calibration. In Figure 4, we have the result of this set of executions.



(a) 5x Load (b) 10x Load (c) 20x Load

Fig. 4. Heatmap varying the α and β and attack load.

The values closest to satisfying the relationship between DR and FPR were $\alpha = 3$ and $\beta = 1$. Using these values, in Figure 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. Also using these values above, in Figure 5 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.

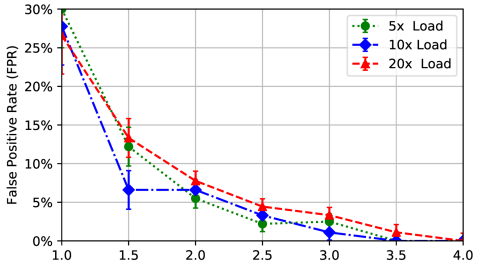


Fig. 5. α on FPR.

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. We must configure this parameter according to the desired attack load to Sentinel classifies as attack flow.

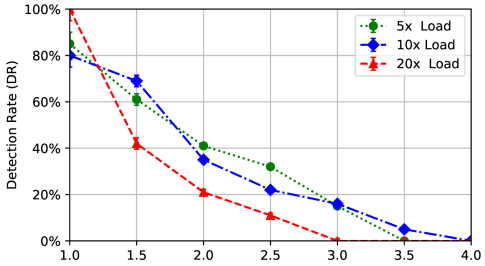


Fig. 6. β on DR.

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.

2) Attack Mitigation: In Figure 7, we have the value of AMR according to the density of the network and executed over different attack loads with the speed of vehicles fixed at 20 m/s. In Figure 8, we have the value of AMR according to the maximum speed of vehicles with the density fixed in 250 vehicles.

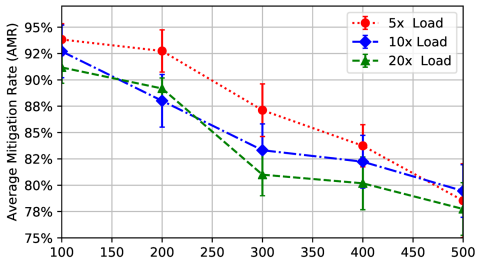


Fig. 7. AMR over vehicle density.

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.
vehicle and most of the attack flow might reach the victim vehicle.

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

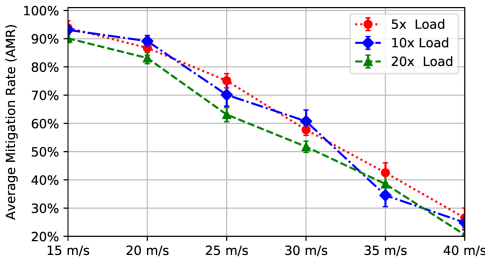


Fig. 8. AMR over vehicle maximum speed (m/s).

VI. Conclusion

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

The results obtained by the simulations were promising. It showed that Sentinel was able to drop most of attack packets before they reach the victim vehicle despite the attack load used. 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.

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

Acknowledgment

The authors would like to thank the following research agencies for their financial supported: CNPq-Brazil under the Grant No. 132866/2016-1, CAPES and FAPEMIG.

References

[1] C. Y. Mahmoud Al-Qutayri and F. Al-Hawi, "Security and privacy of intelligent vanets," in *Computational Intelligence and Modern Heuristics*. InTech, 2010.

[2] D. F. Macedo, D. Guedes, L. F. Vieira, M. A. Vieira, and M. Nogueira, "Programmable networks—from software-defined radio to software-defined networking," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 1102–1125.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, March 2008.

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

[5] R. Braga, Braga, E. Mota, Mota, and A. Passito, Passito, "Lightweight ddos flooding attack detection using nox/openflow," in *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, ser. LCN '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 408–415.

[6] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, January 2011.

[7] A. Virdis, G. Stea, and G. Nardini, *Simulating LTE/LTE-Advanced Networks with SimuLTE*. Springer International Publishing, 2015, pp. 83–105.