loop
A Trace-based Emulator for Vehicular Ad Hoc Networks

Pedro Cirne[1], André Zúquete[2] and Susana Sargento[1]
[1]*Instituto de Telecomunicações, Universidade de Aveiro (IT-UA), Aveiro, Portugal*
[2]*DETI/IEETA, Universidade de Aveiro, Aveiro, Portugal*

Keywords: VANET Traces, Performance Analysis, Secure Routing.

Abstract: In this paper we present `loop` (loop over orderly phases), a trace-based emulator for Vehicular ad hoc networks (VANETs). This is an alternative and novel method to evaluate improvements on VANET protocols, which relies and takes advantage of real data samples collected from an existing network. Those samples are vehicles' geographical locations and radio reception events, which represent mobility and communication patterns of the VANET. From those samples, `loop` creates a synthetic environment to simulate and evaluate communication protocols on the target VANET. `loop` also includes an interactive mode to manage the emulation process and a visualization mode that shows different time and geographical-dependent aspects. The development of `loop` was motivated by the need of an non-intrusive methodology to intensively analyse and deterministically compare the impact of several strategies for communication protocols and all their possible variations in a realistic scenario, both in terms of mobility and radio communication opportunities. Namely, we created `loop` to evaluate the impact of adding security features to the routing control plane of an VANET. Since the VANET includes hundreds of vehicles, the computational performance is critical to speed-up evaluations. With `loop` we were able to perform complex, multi-variable performance evaluations of 24 hour periods in durations ranging from 6 up to 30 minutes.

1 INTRODUCTION

VANETs are a class of wireless *ad hoc* networks where the majority of the nodes are carried by vehicles and a relatively small part are placed near roads. The former is known as On Board Unit (OBU) and the latter as Road Side Unit (RSU)[1]. VANETs may have hundreds or thousands of nodes and inherit most of the characteristics of Mobile *Ad Hoc* Networks (MANETs). Nonetheless, by their very singular nature and requirements when compared with MANETs (Engoulou et al., 2014; Qu et al., 2015; Bali et al., 2014; Amit Dua, 2014), the academic research community has given an increasing amount of attention to the topic and its challenging tasks.

In wireless networks, the design of new communication protocols cannot ignore that the overall performance will be affected by external and uncontrolled conditions, such as environmental and spatial. When the existence of those conditions is not properly considered, the protocols may not behave according to the expectations when deployed in real scenarios. Moreover, some conditions may be completely irreproducible, which makes it impossible to perform a fair comparison among different strategies. Simulation tools can mitigate some of these drawbacks.

The complexity and cost involved in the deployment of a city-scale VANET led to the development of numerous tools aiming to simulate the mobility and the radio propagation characteristics of a VANET (Ros et al., 2014). Nevertheless, the results yielded by simulators, especially in large scale wireless networks and mobile networks, are highly influenced by propagation and mobility models and their discrepancies with the real world (Tan et al., 2010). Furthermore, the use of any of the existent simulators to validate communication protocols in VANETs can provide unrelated results when deploying the protocols in an existent VANET (Viriyasitavat et al., 2011; Uppoor et al., 2013; Ros et al., 2014; Lim et al., 2016).

[1]We may indistinctly refer both types of nodes as VANET Units (VUs).

Cirne, P., Zúquete, A. and Sargento, S.
loop - A Trace-based Emulator for Vehicular Ad Hoc Networks.
DOI: 10.5220/0006854503910402
In *Proceedings of 8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2018)*, pages 391-402
ISBN: 978-989-758-323-0

1.1 Motivation

The main objective of our work is to study communication protocols in VANETs. For that purpose, we consider as a very important use case the VANET operating in Porto city (FutureCities Project, 2017), currently with more than 600 VANET Units (VUs). In this VANET, the OBUs are devices carried by buses. These OBU's, together with a small set of 50 RSU's, provide an IP routing service that provides Internet connectivity to the buses' passengers.

Since we already have a deployed VANET, we could take advantage of this network and use it for real experiments, instead of modelled data. However, using a real VANET to test new ideas and functionalities would be critical, specially in the early development stages, since they could put the entire VANET operation at risk. Alternatively, limiting the tests to a small number of devices could lead to biased evaluations. In both cases, considering the uncontrolled parameters (e.g. human factors and environmental conditions), the experiments could not be completely controlled, so not repetitive nor re-analysable under the same conditions (Schiller and Knoll, 2015; Buisset et al., 2010). Finally, running tests in the VANET would make the results appear slowly, without any possibility of speedup over the wall-clock time.

However, once having a VANET up and running, an interesting a posteriori alternative is to start from mobility and radio reception samples, collected in that VANET, and use them as the starting point to study the performance of communication protocols. This allows us to improve the protocols already in place or to create new ones without having to use modelled data.

This led us to create a trace-based emulator, `loop` (loop over orderly phases), to support the evaluation of protocols in VANETs given from a set of contextual data sampled in a real VANET, instead of models (mobility, radio propagation, etc.). With `loop` we are able to simulate a variety of modifications in the protocols under test for exactly the same contextual data. Note, however, that this contextual data can also be provided by mobility and communication models, instead of being real samples.

1.2 Contributions

The main contribution of this article is the specification of `loop`, a trace-based emulator of VANET protocols. `loop` uses previously collected traces of vehicle's location and the correspondent connectivity information among VUs to simulate multiple, interdependent protocols and evaluate their overall quality. `loop` can simulate the protocols with an arbitrary speedup over wall-clock while interacting with external real system applications running at wall-clock time.

`loop` was initially created to exhaustively evaluate the performance and impact of TROPHY (Trustworthy VANET routing with group authentication keys (Cirne et al., 2018)) protocols in the routing control plane being used in the Porto's VANET. While building a tool for that specific purpose, we envisioned similar requirements in a near future for other types of protocols. Therefore, we conceived `loop` for tackling our specific and well-defined problem, but still with special attention on modularity and future extensions to support other distinct use cases.

`loop` was developed in C++ for performance and modularity. It includes an almost lock-free logging system to collect data for a posteriori analysis with minimum interference, a visualization system to show any time-dependent state of the VANET and an interactive mode in a Command-Line Interface (CLI) style with the possibility to run pre-scheduled commands in batch. Such commands can be used to trigger events in the simulation in specific circumstances (e.g. time).

While developing `loop`, we focused on performance and scalability, decoupling and parallelizing processes as much as possible, since those would be important factors for the immediate and future success of the tool. Each time a compromise between performance and non-technical usability emerged, we have chosen performance. We made considerable efforts to make `loop` specially developer and technical-user friendly. We did not sacrifice performance neither increased code complexity for non-technical user facilities, e.g. a visually rich command GUI (Graphical User Interface).

In our specific use case (Cirne et al., 2018), running in a high-end desktop computer[2], `loop` emulated 24 hours of VANET interactions on our most demanding scenario, with a 1 millisecond time granularity and devices exchanging messages in periods of 100 milliseconds, in less than 30 minutes. This includes the simulation of all the required protocols among 446 devices, namely, a protocol for the vanilla routing and another distinct set of protocols for the secure routing. In that time frame, we simulated both protocols, comparing and logging the relevant differences for a posteriori analysis.

When compared with scenarios where traffic and network simulators are used, our strategy has the advantage of having a high level of detail and accuracy since both mobility and network conditions are realistic. Besides that, this gives an extreme flexibility

[2]Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz

to evaluate scenarios on deployed VANETs without the need for valid mobility or network models. On the other hand, when compared with tests performed in deployed VANETs, it has the benefits of being non-intrusive and non-destrutive, as well as free of logistic and business constraints.

2 RELATED WORK

Simulators have been considered the most important tools for the design and evaluation in works related with VANETs (Ros et al., 2014). Under the VANETs context, we have two distinct categories of simulators to be considered: microscopic traffic simulators and network simulators.

The microscopic traffic simulators generate traces of vehicles and possible other entities (Krajzewicz, 2010; Conceição et al., 2008; Park and Qi, 2006; Cameron et al., 1994; Choffnes and Bustamante, 2005; Owen et al., 2000; Härri et al., 2006), while network simulators use those traces to simulate the communications between entities (Ikeda et al., 2011; Varga and Hornig, 2008; Barr et al., 2005; Pal, 2012).

With a few exceptions that implement both types of simulators side-by-side (Wang et al., 2007; Mangharam et al., 2006), the two types are decoupled and independently developed from each other. Other simulators that integrate one network simulator and one microscopic traffic simulator provide a bridge between them and give some extra facilities to deal with VANET' specific scenarios (Sommer et al., 2008; Piórkowski et al., 2008).

The use of simulators is subject to the problem of the assumptions and the correspondent performance of the models in use. It is widely accepted that simplistic wireless-related models lead to biased results (Ros et al., 2014; Lim et al., 2016) and mobility patterns have a huge influence in the results yielded by mobility dependent simulators (Viriyasitavat et al., 2011; Uppoor et al., 2013). Moreover, increasing the accuracy in VANET models, as well as the number of devices being simulated, is intrinsically associated to an increase of mathematical complexity (Uppoor et al., 2013) and subsequent performance issues.

By using real traces with mobility and network characteristics embedded, we remove a huge stack of complexity and possible sources of biased results. However, the use of collected data to emulate VANETs is not a common approach found in literature, because it requires a deployed VANET in the first place.

The emulation of mobile networks based on collected traces was firstly proposed by Noble et al. (Noble et al., 1997). Although in distinct conditions, the authors highlighted a question for which, in the VANETs context, neither real tests nor simulations are the answer: *"How does one subject a mobile computing system to realistic yet reproducible wireless networking conditions?"*. The proposed answer given by the authors excludes real tests due to the lack of reproducibility and relies on collected traces to create a "*synthetic networking environment rather than a synthetic workload*" as done by simulators. We followed the idea of creating a synthetic networking environment (`loop`, the VANET emulator) but with clear differences in terms of implementation, since on top of the emulator we still have synthetic workloads (the protocols being emulated).

In (Noble et al., 1997) the authors developed their work based on traces of end-to-end connections and the emulation of unmodified applications, running at wall-clock time. We, by contrast, used traces of point-to-point connections to simulate different protocols based on a model of the protocol, possibly interacting with external entities. Since we are modeling protocols, we still have synthetic workloads. However, compared with mobility and propagation models, modeling network protocols is much less critical and prone to biased results. Besides, we have the advantage of running simulations faster then wall-clock time, even when interacting with external applications.

The concept of a simulated network interacting with a real system was introduced by Fall (Fall, 1999) as *network emulation*. Along with it, the author highlights the problem of coupling two entities with different notions of time: real system and a simulator running faster than wall-clock time. Fall solves the problem by introducing a delay in the simulation, running the coupled system at wall-clock time.

(Weingärtner et al., 2011) tackled the same problem by using an independent synchronization component and execute both the simulated network and the real system in a virtualized environment with virtualized time controlled by the synchronization component. This solution was proposed for cases where the simulation runs slower than wall-clock time. Each entity is allowed to run for a certain amount of time and then blocks until all others entities reach the same virtual point in time.

We solved this problem using self-synchronised interactions between `loop` and coupled real systems, i.e. on each contact made by `loop` it is sent the time-stamp of the simulation and (if needed) the coupled system should use this temporal information. By doing this we allow `loop` to run with an arbitrary speedup over wall-clock time. Internally, `loop` shares

with the work presented by Weingärtner et al. a similar notion of multiple slices of time and synchronization points between slices.

Buisset et al. (Buisset et al., 2010) presented a work focused in vehicular network emulation. Although being the very same topic, the requirements and directions of their work do not match any of ours. The emulation of a small number of devices implies the use of multiple operative systems executing multiple processes. Our requirements do not allow such architecture, mainly due to the lack of scalability.

Schiller et al. (Schiller and Knoll, 2015) presented a work related with VANETs' emulation focused on the evaluation of automotive embedded systems. Their goal is clearly distinct from ours, since it is focused on the evaluation of unmodified applications running on embedded system, rather than network protocols.

Pessoa et al. (Pessoa et al., 2017) proposed the emulation of a VANET focused on content distribution and Delay-Tolerant Networks (DTNs). The authors propose the use of real traces of vehicles and real network information along with the emulation of one VU per operative system process, sockets for inter-process communication and a complex architecture to deliver time-dependent information, made out of a distributed messaging framework and an Apache server with a PHP module to connect to a MySQL database in order to provide JSON data over HTTP.

Comparing with this last approach, we model the protocols instead of having them being implemented by independent processes, one per VU. Each VU is represented by its own internal state, and each phase is responsible for the modification of a well-defined part of it. The interaction between the different emulated VUs is implemented by lock-free memory accesses, and the time-dependent information is preloaded and made available to the emulation without causing delays. All those differences allow `loop` to run with a high speedup over wall-clock, not possible when using numerous operating system processes. Furthermore, in terms of resource consumption `loop` it is extremely lighter, since it uses a single process with a small number of threads. The results presented in this article were obtained with a single thread. Still, we are not limited to a single thread and can adapt this number to the target machine and the tasks running in the emulator.

Finally, in general the integration of real traces with the most common simulators used nowadays is a task with remarkable limitations and challenges caused by the different approaches (simulation versus emulation) and their influence in the software architecture (Fontes et al., 2017). On the contrary, `loop` can be easily integrated with mobility and network simulators, as these can provide the data (modeled data, in this case, instead of sampled data) it uses for emulating scenarios where protocols are to be tested, as shown in Figure 1.

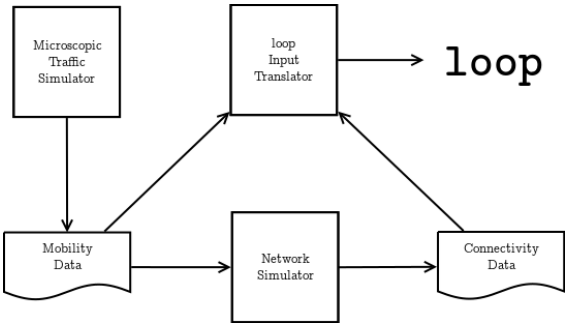

Figure 1: `loop` relying on mobility and network simulators to generate the input data.

3 loop

`loop` is a trace-based VANET emulator that creates a synthetic environment to simulate VANET-related protocols that may need to interact with real applications. It was conceived tacking into consideration the following assumptions:

1. Simulate mobility and network status of a VANET is hard and not accurate.
2. Network protocols are the easier and less critical component to model and simulate.
3. Applications external with the VANET are likely to be complex and so, not likely to be easily simulated nor emulated.

Based on (1), we decided to emulate both mobility and network status from input data in order to capture all the real details. Based on (2), we model protocols, on a needed basis, according to strict rules that benefit speedup over wall-clock. Finally, based on (3) we guarantee that we can interconnect external applications with very limit modifications, even if such requirement becomes reflected in the models of the protocols being simulated.

`loop` has multiple threads of execution. The visualization and the interaction features run in two separate threads. Since they only offer extra output and input features, and may not be needed in all use cases, they can be activated and deactivated on a needed basis. The only thread always running is the one associated to the evolution on the emulation state.

The core of the emulation is a loop over different phases, repetitively over the time, where each of

the phases is responsible for a well-defined part of the task being simulated.

We prepared `loop` to emulate discrete time slices. The total emulated time is divided into two fixed periods: macro periods and micro periods, each composed by a set of homogeneous time splices. The macro period has to be equal or greater than the micro period, so one macro period always has one or more micro periods. All these time intervals can be customized. For emulations in (Cirne et al., 2018), each time slice was set to 1 millisecond, macro periods to 2 seconds and micro periods to 100 milliseconds.

The macro period is associated to the periodicity of the changes on the emulator internal state. By 'emulator internal state' we mean all the state that only exists for convenience of the emulation (e.g. update the traces' information available and the graphical visualization). In other words, macro periods reflect intervals where the status of the VANET, in terms of location of VUs and their connectivity, remains constant. Naturally, it should be equal to the time interval between consecutive real VANET samples.

The micro period is associated to the periodicity of the changes on the VUs internal state. This state is solely associated to modelled protocols, which may or not be able to exchange messages between VUs given the current connectivity status within by the macro period.

3.1 Phases of the Emulation

`loop` separates its internal state from the state of the VUs. Different protocols are modeled based on the division into multiple tasks. Generically, each task can have up to 4 different execution phases:

1. once on macro period start;
2. once on micro period start;
3. on macro period;
4. on micro period.

The former two execution phases are associated to the state of the emulator; thus, the state of the VUs is only accessible for reading operations and not allowed to be modified. The latter two phases of execution are VU's specific and executed for each VU with permission to modify its state (as well as permission to read all the other VUs' public state).

VUs are associated to one specific time slice of the micro period and keep that same position in the multiple micro periods until the end of the emulation. `loop` jumps from time slice to time slice and for each task, on a needed basis, it executes the code associated to each phase. The different phases of execution are shown in Figure 2.

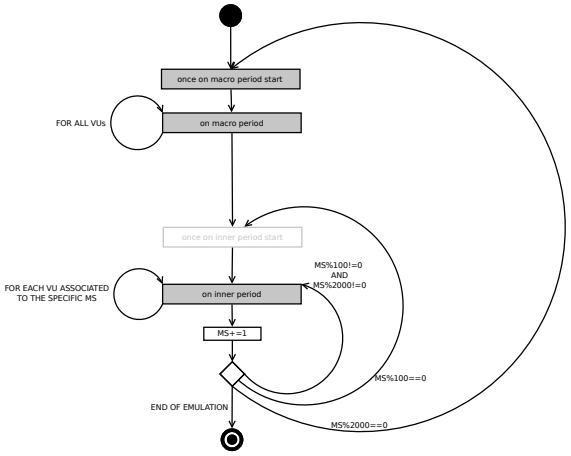

Figure 2: Execution phases of `loop` (with `"%"` representing the modulo operation and `"=="` and `"!="` boolean tests).

To guarantee the determinism of the execution of the tasks, if two or more VUs are associated to the same millisecond, running each of the task on that millisecond should yield the same result independently of the order. This is obtained enforcing a strict rule for all the phases: the visible state of each VU is immutable. With this guarantee, we can concurrently apply the same task in all the VUs associated to the same time slice.

`loop` was built with C++ and the generic task was implemented as an abstract stateless C++ class. Well defined tasks should be implemented by stateless classes with a relation of inheritance to the abstract class. After the initial setup, where VUs are associated to a millisecond of the micro period and a list of tasks (a list of C++ object pointers) is created, `loop` runs based on successive calls to the members of the task objects that represent the phases of execution (see Figure 3).

```
1  for ms in emulation; do
2
3    for t in tasks; do
4
5      if starting a macro period
6        if t->hasWorkOnStartOfMacroPeriod()
7          t->OnceOnMacroPeriodStart(&emulation_data, ms,  &all_VUs, ...)
8
9      if starting a inner period
10       if t->hasWorkOnStartOfInnerPeriod()
11         t->OnceOnInnerPeriodStart(&emulation_data, ms,  &all_VUs, ...)
12
13
14     for v in VUs associated to ms; do
15
16       if first time on macro period
17         if t->hasWorkOnMacroPeriod()
18           t->onMacroPeriod(&emulation_data, ms, &v,  &all_VUs, ...)
19
20       if t->hasWorkOnInnerPeriod()
21         t->onInnerPeriod(&emulation_data, ms, &v,  &all_VUs, ...)
22     done
23
24   done
25
26 done
```

Figure 3: Pseudo code of `loop`.

3.1.1 Communication with External Applications

Since `loop` was built to run protocol models faster than wall-clock time and possibly interact with real applications, time synchronization is an issue. We solve this problem while being conservative, i.e., by creating well-defined points of synchronization to avoid inconsistent states. This is done by imposing all of the following three conditions:

1. Interactions are always initiated by `loop`;
2. The time of the real application is synchronized with `loop` time in all the interactions;
3. `loop` blocks while the real system takes actions relatively to `loop` time.

As consequence, the real applications interacting with `loop` can only react to contacts made by `loop`. Taking in consideration that we want a fully reproducible environment, truly sporadic events from external applications to `loop` are not desirable. Effects of predicted events from applications can be supported as part of the implemented protocol model, by using a polling strategy. Although this clearly increases the complexity of the protocol being modeled, it is essential to let the coupled system run faster than wall-clock. The alternative would be bidirectional synchronization with an external application running at wall-clock time, which would force the coupled system to run all together at wall-clock time.

Time synchronization is currently achieved by sending the simulation time in all the contacts made by `loop` with external applications, which need to be modified to receive it.

For a unidirectional flow of information, from `loop` to an application, during the exchange of information `loop` blocks and the application takes actions relatively to the emulation time.

For a non-instantaneous bidirectional flow of information, both round-trip delay and the time taken to process the request is considered in the protocol model used within `loop`. Namely, the (blocking) connection should be delayed and the effect will only be visible at a well-defined time.

3.2 Simulation of Protocols

We simulate the 2 versions of the routing protocol side by side: a protocol for the vanilla routing already in use (an insecure one), and another distinct set of protocols for the secure routing being tested. We identified 3 distinct tasks and implemented the 3 correspondent C++ classes with members (representing phases) to be executed in loop during the emulation time. Those tasks are, in order:

1. Apply the effect of time (Proc)
2. Transmit information (Tx)
3. Receive information (Rx)

The different tasks influence each other, as described in Figure 4. In the figure, an arrow between tasks A $\rightarrow$ B, represents a relation of influence of A on B. The labels are used to identify the cases in the following explanations. Left to right arrows ($\rightarrow$) represent an immediate influence visible for the next phase, right to left ones ($\leftarrow$) represent delayed influence, only visible one micro period or one macro period later. Black arrows represent internal influence (i.e., the same VU), red ones represent influence across multiple VUs, and green ones represent the optional external interaction (IN) and the optional visualization (OUT). Since tasks are implemented by stateless objects, the influence is not in the tasks themselves but in the VUs and `loop` global state used by the tasks.

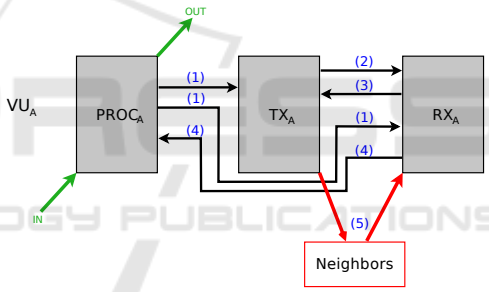

Figure 4: Influence of tasks between each other.

We built a Key Distribution Center (KDC) as a TCP/IP server that receives and handles the requests made by the emulator as they were made by real RSUs. The KDC was built to run in a real environment but prepared to receive the emulation time along with each contact made by `loop`. Since the KDC has to interact with `loop`, some facilities were included for the start and the end of each emulation.

Since all the `loop` requirements will not be needed when running the KDC in real environment, we kept the logic of the KDC decoupled from them. For the start and the end of each emulation this is transparent, since in real environment such functionalities can be simply deactivated. The time synchronization mechanism is implemented at an early stage of each request. Although not completely transparent, the required changes are limited in scope. All the other specific emulation requirements are part of the protocol models, and so, the KDC used with `loop` can be used with real RSUs.

The interactions between vehicular elements and KDC are supported by the CLI of `loop`.

3.2.1 Task `PROC`: Apply the Effect of Time

This task has two phases of execution, both associated with macro periods: one to update the state of VUs and another one to update the state of the emulator. Updating the state of VUs consists on set them as online or offline (1). When the VUs are online, the position and the list of neighbors are also updated on a needed basis. Updating the state of the emulator consists on handling all the counters in use for logging purposes, as well as to prepare a new visualization for the specific macro period. The visualization is dependent of the previous messages received (4).

3.2.2 Task `TX`: Transmit Information

This task has two phases of execution for VUs, for both macro and micro periods. In the micro periods (of 100 milliseconds) we model the transmission of routing messages by all VUs. The other type of transmissions, the much less frequent requests for new information made by RSUs to the KDC, are modeled on a needed basis in the macro periods.

The transmission of beacons is modeled by making information available to all other VUs, so, when needed, such information can be read (5). All the transmissions of a VU are influenced by the previous data received (3). When the requests made by RSUs to the KDC have a reply, such reply is received and stored, and all the effects of the received data on the simulation are handled by Rx (2).

3.2.3 Task `RX`: Receive Information

This task has one only phase of execution for VUs: in the micro period. In this phase we model the reception of routing beacons and also, in case of RSUs, the reception of messages from the KDC (2). The reception of beacons is modeled by reading the messages on the correspondent neighbors at that time (5). The beacons read from neighbors represent the beacons received in the last 100 milliseconds, on a FIFO order.

The received information handled in Rx are beacons from others (5) and KDC replies in the case of RSUs (2). The former may or may not contain signed messages, that need to be validated with the KDC public key before being used. Since such validation is a time consuming operation, we limited its number in the micro periods (corresponding to 100 milliseconds) and delayed to the next micro period the missing validations. We limited the number of validations based on tests performed in VUs' hardware, where each signature validation has a well defined cost.

The messages are used at this point by VUs to update their internal state: the received messages are stored and will influence the future transmissions (3), and the update of the internal state in VUs will influence the next visualizations (4).

Based on the received messages we build, update and compare two routing tables: one based on all the beacons and another one based on the authenticated messages.

3.3 Input Data

`loop` uses as input sets of real traces collected from each of the VUs in operation. We pre-process the collected information associated to each VU. For the active periods of each device, the information present in the traces is a list of entries containing:

- The timestamp;
- The geographic coordinates;
- The number of neighbors listened;
- The number of neighbors that listen the VU;
- A list of pairs of values from listened neighbors: identification, signal strength;
- A list of identifications of neighbors that listen to the VU.

3.4 Visualization

The visualization let us follow the process and easily understand what is happening at which moment. Relying on the OpenCV library, a video-like view is created based on successive images during the emulation, based on the state of VUs. There are two different windows: one with a map showing the state of VUs, and another one with some temporal parameters plotted.

The map is associated to the area used by all VUs. A base image to the visualization area can be initially loaded, to help the identification of the different areas.

At the beginning, the data of all VUs are loaded and the base image is populated with the coordinates of the VUs along the time. These coordinates will end up showing all the paths used by the VUs (see Figure 5).

After the loading of the data, the simulation starts, and a second window appears (see Figure 6). This window shows the time of the simulation, and the network state regarding to 2 different metrics: the percentage of devices that are able to participate in the

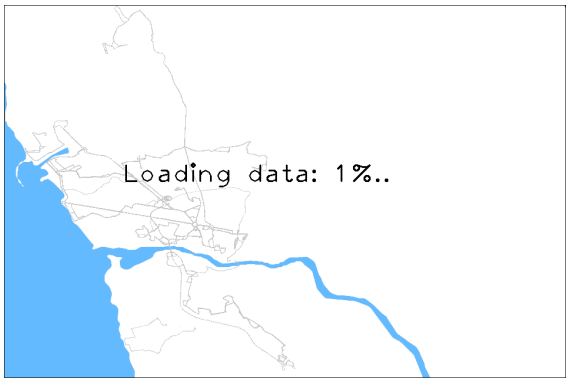

Figure 5: Loading the data for the emulation. Gray paths are formed by coordinates used by OBUs.

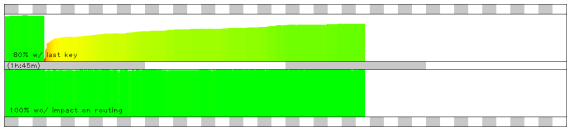

Figure 6: Parameters plotted during the emulation. The reddest parts are closer to 0, while the green ones are closer to 100. The smaller gray bars represent 1 minute of the simulation time, while the bigger ones represent 10 minutes.

secure routing; and the percentage of impact relating the 2 routing strategies being simulated side by side (secure and insecure one). Those are the useful metrics for the analysed scenario and exemplify the generic capabilities of `loop`.

During the emulation the map may have multiple views available, selected with the `map` command (follow by an identifier of the map) available in `loop` CLI. Each view can show a different aspect of the internal state of devices. In the analysed scenario, RSUs are always represented by a violet square while the OBUs are shown as squares of different colors according to their state. The 2 map views created were:

The big brother view: Shows the active OBUs in 3 possible colors, green, orange and red, according to the state of their cryptographic material when compared with the KDC (see Figure 7). In this view it is possible to analyse the epidemic propagation of the context produced by the KDC and the impact that RSUs have to the closest OBUs. The OBUs that have received the last message produced by the KDC are shown in green. The other OBUs are shown in orange (if they miss a "few" messages and can be updated by an OBU) or red (if they miss a "great amount" of messages and can only be updated by RSUs). The outdated OBUs are highlighted: they are bigger and present their identification. This helps to diagnose why a particular OBU is not updated.

The naive view: Shows the active OBUs in 4 possible colors, black, green, orange and red, according to the state of their cryptographic material when compared with their neighbors (see Figure 8). In this view it is possible to analyze the evolution of the relations created between neighbors. The isolated OBUs, the ones that are not listened nor listen any other, are shown in black. The OBUs that listen one or more neighbors all with the cryptographic material in the same state (so they can produce and validate messages among each other) are shown in green. The OBUs that are listening neighbors with cryptographic material in a different state are shown in orange or red, if they are the more or the less updated ones, respectively. Along with the red OBUs, it is shown a red line connecting them to any neighbor that has a more updated cryptographic material. When listening or listened by neighbors with cryptographic material in a different state, the OBUs are bigger and showing their identification. This map highlights the situation where routing beacons may be discarded

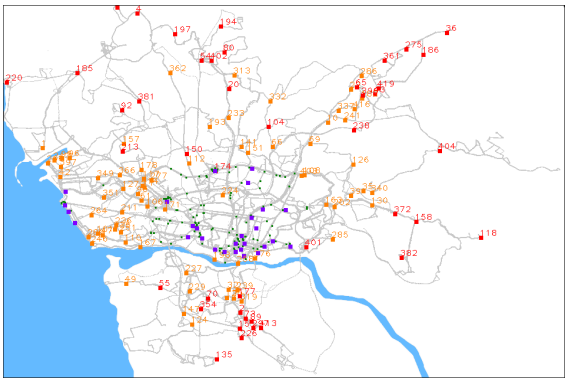

Figure 7: Map showing the big brother view. RSUs are shown in violet. Active OBUs are shown in 3 colors (green, orange and red) according to the state of their cryptographic material when compared with KDC.

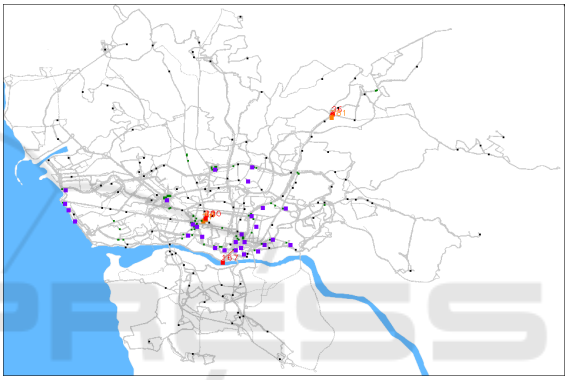

Figure 8: Map showing the naive view. RSUs are shown in violet. Active OBUs are shown in 4 colors (black, green, orange and red) according to the state of their cryptographic material when compared with their neighbors.

due to discrepancies on the keys being used by neighbor OBUs.

4 RESULTS

We run `loop` with 24 hours of traces collected from the VANET, containing 396 OBUs and 50 RSUs. All VUs send periodic routing messages (beacons) in intervals of 100 milliseconds, informing all neighbors about their current routing state. The traces have the logged state of all nodes in periods of 2 seconds, including their geographic position and a list of neighbors listened during the 2 seconds, along with the correspondent beacon signal strength. The area used by OBUs was about 21.1 by 18.5 kilometers.

Based on the lists of the neighbors per VU, we simulated the transmission and reception of beacons in periods of 100 milliseconds. We simulate 2 different scenarios side by side: the insecure routing implemented by the base routing protocol, SB2RP (Ameixieira, 2016), and the secure version relying on Trustworthy VANET Routing with group authentication keys (TROPHY) (Cirne et al., 2018). In the secure version we consider the limitations imposed by the cryptographic operations in the VUs' hardware, and also the connections made by the RSUs to the KDC and respective content exchanged.

4.1 Test Scenarios

As previously stated, `loop` was built, at start, to guide the modification of a routing protocol that was supporting a service in a VANET. To compare the impact of TROPHY with the original insecure routing strategy, we tested multiple parameters associated to TROPHY and analyse the differences between them and between insecure routing.

With the base routing all the messages received are considered valid messages; this is our reference since it is the best possible case. With TROPHY VUs may or may not be able to validate the received routing beacons from neighbors. One of the purposes of the evaluation is to analyse the percentage of beacons that can be validated and its real impact in the routes. Since TROPHY needs to transmit extra information, another aspect to analyse is the overhead in terms of number of larger messages. Besides that, it is important to evaluate different parameters that can influence the distribution of the messages produced by the KDC across the VANET. Finally, the VANET covers Porto city and its periphery and RSUs are placed in the center of the city. With TROPHY, we rely on the epidemic propagation across the VANET of messages produced by KDC (which only communicate with RSUs) to update the cryptographic material of the OBUs moving around the periphery, without forcing them to visit the city center. This epidemic propagation is also an important aspect to analyse.

With `loop` we evaluated 8 distinct parameters:

1. Percentage of secure messages exchanged considered valid (according to TROPHY);
2. Percentage of lost routes due to security (assuming routes between OBUs and RSUs with a maximum of 3 hops, according to the base routing protocol);
3. Percentage of equivalent routes (same route to same destination with TROPHY and base routing);
4. Percentage of secure routes with more hops;
5. Percentage of secure routes with the same number of hops but using a path with lower quality (based on Received Signal Strength Indication (RSSI));
6. Total number of messages stored in the active VUs;
7. Percentage of beacons transmitted with 2 different types of overhead.

Due to the large extension of the VANET and the distinct situations that may occur, we also instructed `loop` to produce maps where geographical-dependent parameters are shown. Considering our case, it was relevant to detect any area of the city where the impact of the secure strategy would be worst than in the overall city. When the parameters associated to TROPHY were not properly configured, it was possible to see the different magnitudes of the negative impact associated to different areas of the city.

One of the important parameters associated to TROPHY is the period of the cryptographic material

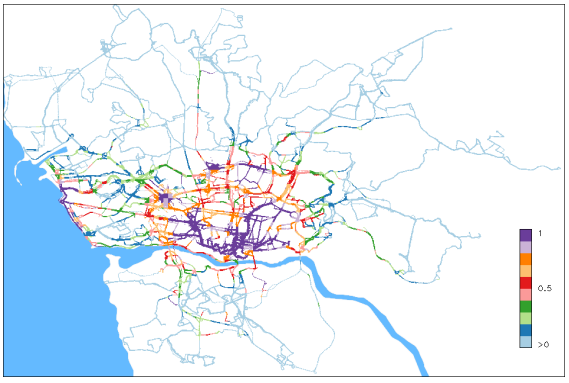

Figure 9: Percentage of updated OBUs per location while requesting the update of the cryptographic material in periods of 10 minutes.

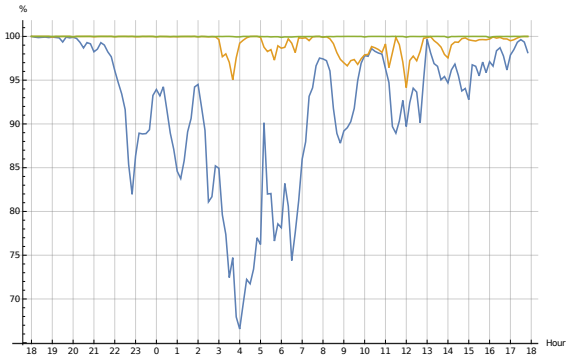

Figure 10: Percentage of routing beacons received and considered authentic while requesting the update of the cryptographic material in periods of 10 minutes (blue), 1 hour (orange) and 2 hours (green).

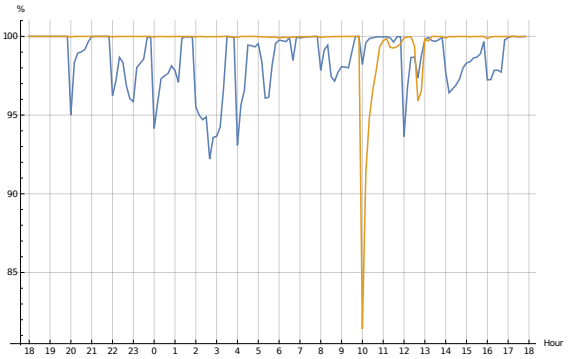

Figure 11: Percentage of routing beacons received and considered authentic while performing the exclusion of 1 VU each 2 hours (blue) and the exclusion of 16 VUs at once at 10:00 (orange).

update. To manage the secure routing and to prevent the cryptanalysis of some of the cryptographic keys, the cryptographic material used by TROPHY has to be strategically updated. The updates are initiated by the KDC and distributed across the VANET based on an epidemic propagation.

Figure 9 shows the case where the KDC continuously demands the update of the cryptographic material of all VUs in periods of 10 minutes. Since 10 minutes is not enough to deliver the information to the periphery of the city, the cryptographic material the VANET is not synchronised. As a consequence, the negative impact in the secure routing is visible in the periphery of the city. With a periodicity of 2 hours, such impact is not relevant in any part of the city.

Besides the impact in different areas, updating the cryptographic material in different periodicity also has a different impact according to the time of the day. This impact is shown in Figure 10 for 3 different periods: 10 minutes, 1 hour and 2 hours. With the video-like visualization, we are able to go a bit further, and see the iterations of VUs at a specific time and the behavior associated to those iterations.

We evaluated different associations of parameters related to distinct actions with the management of the cryptographic material in the VANET. Since we were exploring a strategy to secure a routing protocol, TROPHY was developed taking in consideration the need to exclude multiple VUs (detected as compromised). The usefulness of such actions is associated to their consequences when they are applied to the VANET, i.e. it would be useless to be able to exclude VUs if such action was breaking the connectivity for long periods of time. With `loop` we were able to simulate and analyse such actions along the 24 hours period. Figure 11 shows the temporal behavior of the VANET in 2 hypothetical scenarios: the exclusion of 1 VU each 2 hours, and the exclusion of 16 VUs at once.

5 CONCLUSION

In this article we presented `loop`, a trace-based emulator of VANET environments, that uses real data containing the mobility and connectivity patterns of VUs operating in a VANET. Based on the collected data, `loop` emulates the VANET and allows the simulation of protocol models much faster than wall-clock.

`loop` takes advantage of an existent VANET and emulates a reproducible environment where communication protocols can be simulated in the same conditions without putting in risk the original VANET.

Currently most of the VANET communication protocols are evaluated based on real environments or based on mobility and network simulators, with the latter being the preferable solution.

`loop` introduces an alternative with considerable benefits to both cases: it is easy to setup and manage compared to the tools relying in mobility and network models, and at the same time it uses realistic information in the way of real test-beds.

ACKNOWLEDGEMENTS

This work is funded by FCT/MEC through national funds under the project, S2MovingCity CMUP-ERI/TIC/0010/2014.

REFERENCES

Ameixieira, C. (2016). Method and system for operating a vehicular data network based on a layer-2 periodic frame broadcast, in particular a routing protocol. WO2016135711A1.

Amit Dua, N. K. (2014). A systematic review on routing protocols for Vehicular Ad Hoc Networks. *Vehicular Communications*, 1(1):33–52.

Bali, R. S., Kumar, N., and Rodrigues, J. J. P. C. (2014). Clustering in vehicular ad hoc networks: Taxonomy, challenges and solutions. *Vehicular Communications*, 1(3):134–152.

Barr, R., Haas, Z. J., and van Renesse, R. (2005). JiST: An Efficient Approach to Simulation Using Virtual Machines: Research Articles. *Softw. Pract. Exper.*, 35(6):539–576.

Buisset, A., Ducourthial, B., Ali, F. E., and Khalfallah, S. (2010). Vehicular Networks Emulation. In *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pages 1–7.

Cameron, G., Wylie, B. J. N., and McArthur, D. (1994). PARAMICS-moving vehicles on the connection machine. In *Proceedings of Supercomputing '94*, pages 291–300.

Choffnes, D. R. and Bustamante, F. E. (2005). An Integrated Mobility and Traffic Model for Vehicular Wireless Networks. In *Proceedings of the 2Nd ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '05, pages 69–78, New York, NY, USA. ACM.

Cirne, P., Zúquete, A., and Sargento, S. (2018). TROPHY: Trustworthy VANET routing with group authentication keys. *Ad Hoc Networks*, 71:45–67.

Conceição, H., Damas, L., Ferreira, M., and Barros, J. (2008). Large-scale Simulation of V2v Environments. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 28–33, New York, NY, USA. ACM.

Engoulou, R. G., Bellaïche, M., Pierre, S., and Quintero, A. (2014). VANET security surveys. *Computer Communications*, 44:1–13.

Fall, K. (1999). Network Emulation in the Vint/NS Simulator. In *Proceedings of the The Fourth IEEE Symposium on Computers and Communications*, ISCC '99, pages 244–, Washington, DC, USA. IEEE Computer Society.

Fontes, H., Campos, R., and Ricardo, M. (2017). A Trace-based Ns-3 Simulation Approach for Perpetuating Real-World Experiments. In *Proceedings of the Workshop on Ns-3*, WNS3 '17, pages 118–124, New York, NY, USA. ACM.

FutureCities Project (2017). *Living lab : Vehicular adhoc networking, url : http://futurecities.up.pt/site/vehicular-ad-hoc-networking-testbed*.

Härri, J., Filali, F., Bonnet, C., and Fiore, M. (2006). VanetMobiSim: Generating Realistic Mobility Patterns for VANETs. In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, VANET '06, pages 96–97, New York, NY, USA. ACM.

Ikeda, M., Kulla, E., Barolli, L., and Takizawa, M. (2011). Wireless Ad-hoc Networks Performance Evaluation Using NS-2 and NS-3 Network Simulators. In *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 40–45.

Krajzewicz, D. (2010). Traffic Simulation with SUMO – Simulation of Urban Mobility. pages 269–293.

Lim, K. G., Lee, C. H., Chin, R. K. Y., Yeo, K. B., and Teo, K. T. K. (2016). Simulators for vehicular ad hoc network (VANET) development. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–4.

Mangharam, R., Weller, D., Rajkumar, R., Mudalige, P., and Bai, F. (2006). GrooveNet: A Hybrid Simulator for Vehicle-to-Vehicle Networks. pages 1–8. IEEE.

Noble, B. D., Satyanarayanan, M., Nguyen, G. T., and Katz, R. H. (1997). Trace-based Mobile Network Emulation. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, pages 51–61, New York, NY, USA. ACM.

Owen, L. E., Zhang, Y., Rao, L., and McHale, G. (2000). Traffic flow simulation using CORSIM. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165)*, volume 2, pages 1143–1147 vol.2.

Pal, D. (2012). A Comparative Analysis of Modern Day Network Simulators. In Wyld, D. C., Zizka, J., and Nagamalai, D., editors, *Advances in Computer Science, Engineering & Applications*, volume 167, pages 489–498. Springer Berlin Heidelberg, Berlin, Heidelberg.

Park, B. and Qi, H. (2006). Microscopic simulation model calibration and validation for freeway work zone network - a case study of VISSIM. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1471–1476.

Pessoa, G., Dias, R., Condeixa, T., Azevedo, J., Guardalben, L., and Sargento, S. (2017). New Insights on Content Distribution Emulation for Vehicular Networks.

Piórkowski, M., Raya, M., Lugo, A. L., Papadimitratos, P., Grossglauser, M., and Hubaux, J.-P. (2008). TraNS: Realistic Joint Traffic and Network Simulator for VANETs. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12(1):31–33.

Qu, F., Wu, Z., Wang, F. Y., and Cho, W. (2015). A Security and Privacy Review of VANETs. *IEEE Transactions on Intelligent Transportation Systems*, 16(6):2985–2996.

Ros, F. J., Martinez, J. A., and Ruiz, P. M. (2014). A survey on modeling and simulation of vehicular networks: Communications, mobility, and tools. *Computer Communications*, 43:1–15.

Schiller, M. and Knoll, A. (2015). Emulating Vehicular Ad Hoc Networks for Evaluation and Testing of Automotive Embedded Systems. In *Proceedings of the 8th International Conference on Simulation Tools and Techniques*, SIMUTools '15, pages 183–190, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Sommer, C., Yao, Z., German, R., and Dressler, F. (2008). On the Need for Bidirectional Coupling of Road Traf-

fic Microsimulation and Network Simulation. In *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models*, MobilityModels '08, pages 41–48, New York, NY, USA. ACM.

Tan, K., Wu, D., Chan, A., and Mohapatra, P. (2010). Comparing simulation tools and experimental testbeds for wireless mesh networks. In *2010 IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 1–9.

Uppoor, S., Fiore, M., and Härri, J. (2013). Synthetic Mobility Traces for Vehicular Networking. In *Vehicular Networks*, pages 209–245. John Wiley & Sons, Inc.

Varga, A. and Hornig, R. (2008). An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Viriyasitavat, W., Bai, F., and Tonguz, O. K. (2011). Dynamics of Network Connectivity in Urban Vehicular Networks. *IEEE Journal on Selected Areas in Communications*, 29(3):515–533.

Wang, S., Chou, C., and Lin, C. (2007). The design and implementation of the NCTUns network simulation engine. *Simulation Modelling Practice and Theory*, 15(1):57–81.

Weingärtner, E., Schmidt, F., Lehn, H. V., Heer, T., and Wehrle, K. (2011). SliceTime: A Platform for Scalable and Accurate Network Emulation. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 253–266, Berkeley, CA, USA. USENIX Association.