future internet

MDPI

Article

EmuCD: An Emulator for Content Dissemination Protocols in Vehicular Networks

Ricardo Chaves [1,2,*], Carlos Senna [1], Miguel Luís [1,3], Susana Sargento [1,2], André Moreira [4], Diogo Recharte [4] and Ricardo Matos [4]

1 Instituto de Telecomunicações, 3810-193 Aveiro, Portugal; cr.senna@av.it.pt (C.S.); nmal@av.it.pt (M.L.); susana@ua.pt (S.S.)
2 Department of Electronics, Telecommunications and Informatics (DETI), University of Aveiro, 3810-193 Aveiro, Portugal
3 ISEL-Instituto Superior de Engenharia de Lisboa, 1959-007 Lisboa, Portugal
4 Veniam, 4000-098 Porto, Portugal; amoreira@veniam.com (A.M.); drecharte@veniam.com (D.R.); rmatos@veniam.com (R.M.)
* Correspondence: ricardochaves@ua.pt

Received: 20 November 2020; Accepted: 16 December 2020; Published: 21 December 2020

Abstract: The development of protocols for mobile networks, especially for vehicular ad-hoc networks (VANETs), presents great challenges in terms of testing in real conditions. Using a production network for testing communication protocols may not be feasible, and the use of small networks does not meet the requirements for mobility and scale found in real networks. The alternative is to use simulators and emulators, but vehicular network simulators do not meet all the requirements for effective testing. Aspects closely linked to the behaviour of the network nodes (mobility, radio communication capabilities, etc.) are particularly important in mobile networks, where a delay tolerance capability is desired. This paper proposes a distributed emulator, EmuCD, where each network node is built in a container that consumes a data trace that defines the node's mobility and connectivity in a real network (but also allowing the use of data from simulated networks). The emulated nodes interact directly with the container's operating system, updating the network conditions at each step of the emulation. In this way, our emulator allows the development and testing of protocols, without any relation to the emulator, whose code is directly portable to any hardware without requiring changes or customizations. Using the facilities of our emulator, we tested InterPlanetary File System (IPFS), Sprinkler and BitTorrent content dissemination protocols with real mobility and connectivity data from a real vehicular network. The tests with a real VANET and with the emulator have shown that, under similar conditions, EmuCD performs closely to the real VANET, only lacking in the finer details that are extremely hard to emulate, such as varying loads in the hardware.

Keywords: network emulation; VANET; content dissemination; scalability

1. Introduction

Despite recent technological advances with regard to vehicular networks, it is still very difficult to develop routing and content dissemination protocols in real scenarios [1]. Due to the risk and high cost of carrying out experiments on real vehicle networks [2], the alternative solution is the use of simulators and emulators for the early stages of the development, and execute only final tests on small testbeds. However, some aspects inherent to vehicular ad-hoc networks (VANETs), such as mobility models, driver behaviour and wireless channel modelling have a considerable effect on performance results. Due to these challenges, the simulation of realistic mobility models becomes essential for research in order to achieve precisely the desired results that reflect the realistic behaviour of vehicular

traffic [3]. However, the channel modelling depends on many aspects, such as obstacles like trees, buildings, vehicles and even people on the streets, and this can only be measured in a real scenario.

Developing a content dissemination protocol for a vehicular network (VANET) is not an easy task: the protocol may be good in theory and in simulations, but with low performance in a real environment. Moreover, even if it is possible to test the protocol in a real network (which is rarely the case), not only is the network different between each test run (preventing an accurate analysis and debugging), but it is also difficult to scale. Particularly in the case of a VANET, since the nodes keep moving and the network's topology is very dynamic, repeating a test under the same conditions is not an option.

This is why simulators exist: they provide reliable, deterministic results, and whether a real or a synthetic dataset is being used, it is possible to simulate the behaviour of a VANET. However, some difficulties are presented:

- debugging—errors can be hard to debug as the simulators are very complex; most errors probably result from issues in the user's custom protocol, but lack of documentation and support makes solving them a challenge;
- custom protocols—testing a custom protocol in a simulator requires a custom implementation specifically for that simulator using its libraries or APIs; it is not a problem if the protocol is a simple one, but it may be very complex if it is an existing closed-source protocol that is not supported by the simulator;
- portability—a protocol developed or ported to work with a specific simulator may not work with a different simulator/emulator and, most importantly, is not ready to run in real hardware;
- reliability—simulators have to use models to represent the reality, encompassing mobility and radio propagation models. However, in a very challenging and mobile environment such as VANETs, these models usually fail to represent the reality as they cannot encompass the real obstacles and their effects in a real scenario.

From the issues mentioned above, the most prohibitive one that leads to the creation of a new emulator is the difficulty in testing different protocols. In a real On Board Unit (OBU) in vehicles, there is a CPU, an OS, storage, RAM and one or more wireless interfaces. Software is developed taking these aspects in consideration, and it does not need to consider the OS-related tasks, so using it in a simulation/emulation should be as effortless and transparent as running it in real hardware: making sure the required network interfaces and enough RAM are available to execute the software. Furthermore, when debugging software that is running in the simulator/emulator, the developer should only have to look at its own software, without having to know the details of the underlying simulator, its libraries or having to dive into its code.

To the best of our knowledge, there is no solution that: (1) offers easy integration with a custom protocol/software, especially if the code is unavailable, only the binaries; (2) is as resilient and invisible as possible to a developer when debugging, so the developer can focus only on the protocol/software at hand; and (3) provides results as close to reality as possible while offering customizable network metrics and tunable parameters.

In this article we propose a solution to address the previous issues. We built an emulator, denoted as EmuCD, that contains the following characteristics:

- a versatile emulator designed to host new routing and dissemination protocols for VANETs;
- a VANET emulator that allows the development of new content dissemination protocols which can be migrated to real communication box/interfaces with little to no adaptations required;
- a set of tools to facilitate the integration of mobility simulators with the VANET emulator;
- an emulation platform which allows the use of real or simulated data traces of vehicle mobility and radio connectivity between vehicles, and between vehicles and the infrastructure.

We performed tests in real vehicular interfaces with existent content dissemination protocols (without any changes required in the code/implementation versus running them with EmuCD),

in a VANET with five simultaneous nodes, and results have shown that under similar conditions, EmuCD performs closely to the real VANET, only lacking in the finer details that are extremely hard to emulate, such as varying loads in the hardware. EmuCD is also used to run the content distribution protocols in a 200-node VANET, being able to comparatively assess them with low CPU and memory usage.

The remainder of this article is organized as follows. In the following section we present a review of concepts and related works. Section 3 describes the proposed architecture. Section 4 discusses the integration facilities of our emulator to operate in conjunction with other simulators/emulators/tools. In Section 5 we discuss the results of the emulator and examples of testing protocols and scenarios. Finally, the conclusions and future work are presented in Section 6.

2. Related Work

There are several simulation tools available, and even some emulators, most of them being used to evaluate the performance of a network and/or protocols [2,4]. Some of the simulators are more specific than others, but they usually provide the most common protocols and support custom ones [5]. Following the taxonomy proposed by Mussa et al. [3] in the scope of this work, there are three types of simulators: network simulators, mobility simulators and VANET simulators.

With regard to network simulators, INET (https://inet.omnetpp.org/), Riverbed Modeler (https://www.riverbed.com/gb/products/npm/riverbed-modeler.html) (formerly known as OPNET), Network Simulator (NS2/3) (https://www.nsnam.org/) and the Opportunistic Network Environment (ONE) (https://www.netlab.tkk.fi/tutkimus/dtn/theone/) are the most used ones in the scientific community. The simulators are usually discrete-event based, where one or more state-machines coordinate the simulation. A distinguishing feature of a simulator is that it is not required to run in real time. With enough hardware resources, 1 h of network activity can be simulated in five minutes, with the results ready to be analysed at the end. However, they have to use models to simulate the real environment, which do not contain all the real impairments of the network. Moreover, it is not easy to use the same implementation in real hardware, which makes difficult a fair comparison of the algorithms in a real emulated approach.

Mobility simulators are based on mobility models employed to simulate vehicles' movement in VANETs. Mobility models can either be collected from a real VANET or synthetic simulated traces. Interesting mobility simulators are: SUMO [6], TRANSIM (https://code.google.com/archive/p/transims/), CORSIM [7], VanetMobiSim [8], GLOMOSIM [9] and MOVE [10]. All these mobility simulators are able to simulate various traffic conditions with realistic vehicular movement, and can be integrated both with simulators and emulators.

Regarding environments for simulating VANETs, it should be noted that the key point is the realism of vehicle mobility [11]. Outstanding solutions in this category are: GrooveNet, Traffic and Network Simulation Environment (TraNS), National Chiao Tung University Network Simulator (NCTUns), Vehicles in Network Simulation (Veins), Malinverno et al.'s simulation framework, Loop and mOVERS. GrooveNet [12] is a hybrid simulator that allows communication between simulated vehicles, real vehicles and both. It uses a real topography based on a street map to model communication between vehicles, and it supports multiple network interfaces, GPS and events triggered from the vehicle's on-board computer. Traffic and Network Simulation Environment (TraNS) [13] provides a vehicular environment by linking two open-source simulators: a traffic simulator, SUMO [6], and a network simulator, NS2. Thus, the network simulator can use realistic mobility models and influence the behaviour of the traffic simulator based on the communication between vehicles. National Chiao Tung University Network Simulator (NCTUns) [14] is a hybrid simulator and emulator which allows integrated traffic and network simulation. One of its distinguishable features is the direct use of the TCP/IP protocol stack in the Linux kernel, enabling real-life application programs to be run directly. Vehicles in Network Simulation (Veins) [15] is an hybrid simulation framework composed of the general purpose simulator OMNeT++ and the road traffic simulator SUMO. It grants the network

simulation capabilities to directly control the road traffic, and thus to simulate the influence of VANET communications on road traffic. Similarly, the road traffic simulation provides information to the network simulation. Malinverno et al. [16] developed a simulation framework based on NS-3 and SUMO, where they support the use of 802.11p Wireless Access in Vehicular Environments (WAVE), Long Term Evolution (LTE) and Cellular vehicle-to-everything (C-V2X) as communication technologies and allow for applications/protocols to use Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM) messages. Custom applications can be implemented using C++. Loop (loop over orderly phases) [17] is a trace-based simulator for VANETs which takes vehicles' geographical locations and radio reception events, and creates a synthetic environment to simulate communication protocols on the target VANET. It is an interesting solution developed specifically for VANETs, but its focus is security and it does not easily simulate other types of protocols. Finally, there is mOVERS [18], an emulator that is able to recreate scalable vehicular scenarios of data gathering and content dissemination by replicating the same software of the OBUs and integrating with real vehicular mobility and connectivity. This emulator has been previously used to test different routing and content dissemination strategies [18–20], but its main drawback, as most simulators/emulators, is the need for a custom implementation of the protocol within the code (rendering the testing of a closed-source protocol impossible).

The existing solutions require the protocol to be integrated with the simulator/emulator, according to their rules and APIs, in a predefined programming language, usually C++. This results in the need for the developer to first become familiar with the simulator/emulator, and only then start working on the development of the protocol. Besides, the integration of real traces with the most common simulators is also a task with limitations and challenges, caused by different approaches (simulation versus emulation) and their influence on the software's architecture [21].

Considering these aspects, we designed an emulator where each node of the network is built in a container where we install a simple engine (daemon) that consumes a data trace with the mobility and connectivity of the node. Our daemon works by configuring the communication interfaces of the container's operating system, making our solution completely independent of the protocol being developed. This gives freedom to the protocol's developer(s), without API or programming language restrictions, and allows portability on any hardware, as long as it has a compatible operating system.

3. EmuCD Architecture

Testing a new protocol for content dissemination in networks using a traditional simulator requires, at least, two steps: first, it is necessary to configure the network with all its details (communication technology, node characteristics, interfaces, etc.); after that, it is necessary to deploy the new protocol, that is, the customization of the protocol inside of the simulator following all the rules defined by this software. In order to do these tasks, a deep knowledge of the simulator is required, starting with an initial stage of studies to absorb the entire process for the network configuration, and then the implementation process of the new protocol for the simulator, where a knowledge of its libraries is required.

On the other hand, what is expected from a virtual environment where it is possible to develop and test networks and protocols is for it to be as realistic as possible, and offer conditions that allow transferring what was developed virtually to real physical networks, with a minimum effort to deploy and prepare for the real use. However, this is not offered by traditional simulators, as after the development period a customization period is still necessary for the protocols to be properly adapted to the conditions of the real physical network. A simple example is the configuration of the network and its nodes: each simulator has its own process for these definitions, usually stored in tables/files of the simulator, which do not correspond to the settings of the operating systems.

Another aspect closely linked to the realism of the simulation is what is expected from the behaviour of the network nodes. This aspect is particularly important in mobile networks where a delay tolerance capability is desired. Taking into account a VANET where the characteristics of each

city, the habits of the inhabitants and the weather conditions have a great influence on the mobility of the vehicles, it is very important to use real traces taken directly from the vehicular networks installed in the cities. Such traces provide a degree of realism that simulators like SUMO cannot achieve.

With all these aspects in mind, we propose an emulator where each node of the network consists of a Docker container where we install a simple engine (daemon) that consumes a data trace with the node's mobility and connectivity. That is, at each step (timestamp), the node's daemon updates the configuration of the container's operating system (IPs, routing tables, etc.) according to the neighbours table indicated by the information coming from the data trace. In this way, we emulate all the networking aspects directly in the OS of the container and the new protocol can be developed and tested as if it were running in real conditions such as a notebook, a vehicle communication box, a Raspberry Pi, etc. We adopt a service choreography strategy, where nodes are fully independent and our emulated network has the same conditions of the real network. Besides, our emulator enables the testing of the new protocols with many types of communication technologies without any modifications, as the protocol is not aggregated inside the emulator or bound to it.

In our emulator, each container acts as a real Ubuntu node with a configurable number of network interfaces, and follows the mobility provided by a dataset from a VANET. Because the nodes are not really moving, and to support everything being run locally in a single machine, the mobility aspect of the emulation is created by manipulating the routing table in the OS of each container. Each node knows who its neighbours are at every instant, and the routing table is updated so that the node can only reach its neighbours, as it would if it were in a real VANET.

It should be noted that, due to the aimed abstraction between the emulator and the software/protocol being tested, the emulator only supports standalone software, whether it is a Python script or a standalone binary. It is the software's responsibility to perform tasks such as neighbour discovery (using UDP broadcasts, network pings, etc.) or data management, as the emulator does not provide any support. As a rule of thumb: if the software/protocol can run in real hardware, it is ready to run in the emulator. This is the first time that a protocol can be tested in an emulated VANET without any modifications to the protocol itself: all that is required is a data trace, the software to be tested and the edition of some configuration files in the emulator.

An overall view of the architecture and its components is available in Figure 1. The nodes are Docker containers, with each one running a daemon and being controlled by a single manager. Ubuntu 16.04 is the base of the image, so each node can be considered to be a full OS with its own routing tables, network interfaces and storage.

The manager is decoupled from the nodes in a way that allows for multiple managers to exist in an emulation. For example, each manager coordinating a different set of containers in separate hardware, in order to better emulate realistic conditions. The coordination between all nodes is done by this component, as it is responsible for:

- Loading the emulation's parameters from a JSON file;
- Starting the emulation;
- Waiting for each container to be "booted" and ready;
- Synchronizing every container;
- Building the emulation log file;
- Providing data for the dashboard (nodes' positions, content distribution progress, content distribution metrics);
- Polling the current status from each container's daemon;
- Collecting and storing content distribution metrics.

Each node is running a daemon process. The main task of the daemon is to update the node's IP routing table according to the neighbours in a given instant. In order to do this, two important pieces of information are required: the current node's neighbours and their IP addresses. The neighbours are read from the dataset (stored in a file), for the relevant node at the current timestamp. These node

IDs are then converted into the respective container's IP. If for example one of the neighbours is the node with ID 332, its container's IP address is 172.20.3.32, so an entry in the routing table is added by running the command "ip route add 172.20.3.32 dev $IF".

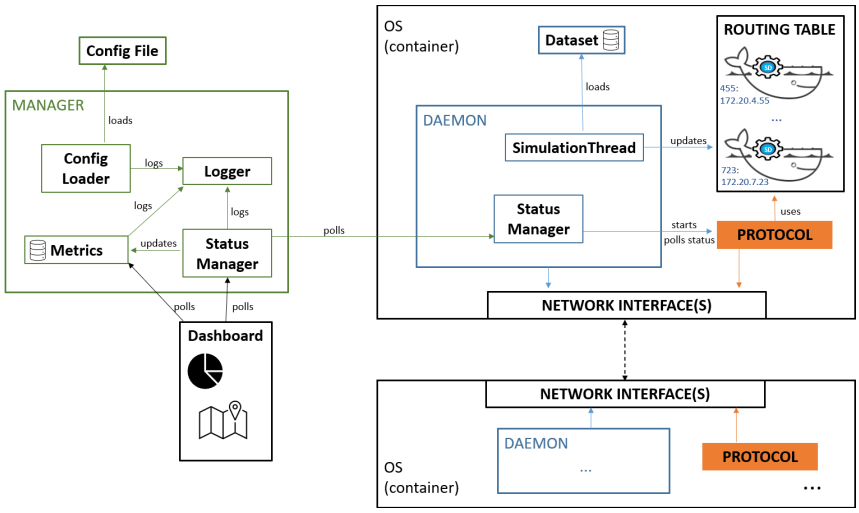

Figure 1. Architecture of the EmuCD Emulator.

The dataset must be available prior to the emulation start, as it is included in the Docker image when being built. It consists of several files, one per node, with each file containing several entries. An entry is represented by a timestamp, latitude, longitude and a list of neighbours in the format of (nodeID, Received Signal Strength Indication (RSSI)) tuples. A node is only considered to be a neighbour if its RSSI meets a configurable threshold.

Although the manager and daemon are the core of the EmuCD emulator, a web dashboard was created to assist with the visualization of the emulator's status and progress. It periodically polls the `/status` endpoint of the manager to display the positions of every node in a map, as well as the progress per node of the content's distribution. The dashboard can also show an overview of the metrics collected, like the cumulative download progress, the number of OBUs that reached 100% in each timestamp, or the average download progress across all OBUs.

4. EmuCD Integration Facilities

To evaluate a protocol regarding its performance, its efficiency in the use of the communication infrastructure, and its correct functioning for content dissemination, having a real communications network is ideal. However, when it comes to vehicular networks, this objective is practically impossible, notably in the initial tests due to the preparation and configuration required of hardware and software. The most suitable alternative is to use data traces harvested in production networks, such as the one of Oporto. However, yet again there are difficulties, as saving detailed information about the behaviour of all nodes in the network requires a huge effort in terms of time and resources, as this data collection can easily generate terabytes of data that must circulate through the network to the cloud. With these conditions in mind, it is easy to note that the use of mobility simulators, such as SUMO, are of great importance in the development of new protocols for the dissemination of content in VANETs.

Considering all options necessary to be able to perform exhaustive tests with the new protocols, we built a tool to integrate the mobility traits generated by SUMO, preparing them for use in our emulator. That was how the Processor of Information to Perform an Emulation (PIPE) was born. It should be reminded that, when using real mobility data, SUMO and PIPE are not required.

SUMO can be used to generate the mobility of the vehicles according to the region of the city while providing a mean velocity or a range of velocities-among other conditions, as it is a traffic simulator. Finally, since SUMO does not generate information regarding the connectivity between

vehicles, the PIPE tool determines the set of neighbours of each vehicle at each time instant, through a connectivity model that estimates the signal strength (RSSI) between nodes based on the distance between them without considering the terrain profile, i.e., the absence or presence of obstacles [22,23]. PIPE also does the compatibility between SUMO and the EmuCD emulator, so that we can use not only real data traces, but also these SUMO-based ones. These components form our emulation platform, shown in Figure 2.

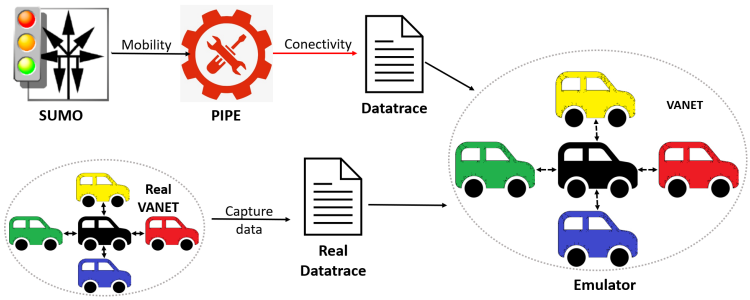

Figure 2. Tools of the emulation platform.

Once the dataset is ready, the process of integrating a new protocol requires the editing of a few files in the emulator (the protocol's software itself is untouched):

1. New branch—optional, but the creation of a new branch for the protocol is recommended, maintaining the repository's structure;
2. Docker—a new Docker image must be created: in `docker/build.sh` the image name must be edited, alongside any required files and changes that must be set up, prior to building the Dockerfile; in `docker/Dockerfile` any software dependencies should be added, as well as the files that will be present when the container starts; the `docker/docker-startup.sh` script could also need a small customization as it is an initial script that runs when the container boots, used to start relevant processes;
3. Daemon—the emulation's daemon will likely need changes in the `thread_simulation` function, as each protocol has different requirements. The status' update function also needs to be adapted for each protocol because of the download progress (some softwares update the progress to stdout, others provide an API, etc.): it is not mandatory, but metrics will not be available unless it is done.

This section highlighted the simplicity and efficiency when integrating a protocol and datatrace to be used in the EmuCD emulator. The following section will present how EmuCD was used to test and adapt three P2P protocols to the VANET environment, as well as some relevant results.

5. EmuCD in Action

EmuCD is evaluated in its performance at hardware level (Section 5.1), in its ability to emulate real scenarios (Section 5.2), and in its ability to compare content distribution protocols (Section 5.3).

5.1. Baremetal Performance Results

Regarding hardware, there are two main factors that have an impact on the number of nodes and protocols running in EmuCD: RAM and CPU.

RAM limits how many containers/nodes can be launched, largely independent of their activity (whether they have constant network traffic or not). For example 16 GB of memory may be enough to run 200 nodes with protocol A, but only 100 nodes when using protocol B. This can happen for several reasons, two of them being that A's binary is smaller than B's, or protocol B has a larger memory consumption than protocol A.

CPU, on the other hand, can usually limit the emulation in three ways: (1) usage must be under 100% (ideally under 50%) to assure a valid parallelism between containers (e.g., if there are events and position updates every second, when the CPU is under full load, there is no guarantee that every packet transfer and operation that should be performed within each second, really happened as intended or if the CPU did not have time to do everything in that time span); (2) high network activity causes high CPU usage (a network with 50 nodes transferring data simultaneously between each other may have the same CPU usage as 300 nodes in a sparse scenario); (3) the amount of processing done by the protocol/software before/after sending each chunk/block influences CPU usage (a protocol with a simple logic or algorithm allows for EmuCD to run more nodes than a very complex protocol).

Across every test and scenario presented in this document, EmuCD was running in an Ubuntu 18.04 virtual machine (VM), with seven dedicated CPU cores at 4.6 GHz and 27 GB of RAM. This section presents the CPU and RAM usage of the VM while running the emulator (disk usage is not discussed, maximum observed read and write speeds during the stress tests were under 100 MB/s, so this aspect only has an impact if it is an old hard disk drive). Profiling shows the emulator booting the containers and the beginning/end of the tests. The CPU and RAM graphs are not restricted to the emulator and its processes, they are for the entire VM: idling and without an emulation running, CPU averaged 0% across all cores and 1 GB of RAM was in use, so the CPU graphs can be considered to be representative of the emulator and its processes, and in the RAM graphs 1 GB must be subtracted from the memory in use.

Starting by analyzing just the emulator, without any protocol running on the containers, Figures 3 and 4 show the CPU and RAM usage of 100 nodes, respectively. Regarding CPU, there was a spike starting at 1:19:50 p.m. until 1:20:20 p.m., which corresponded to the required Docker containers being created and booted (hence the corresponding spike in RAM); the small spike 30 s later was due to the emulation start, also noticed by a slight increase in the RAM used. Because the containers were running "empty", there was not any CPU usage until 1:24:20 p.m., when the containers were stopped and removed. Memory usage during this test remained steady at 4 GB, which means that 3 GB were used by the emulator's containers and processes, resulting in an average 30 MB per container. The same test with 200 nodes showed 6 GB used by the emulator's containers and processes.

Adding the RAM footprint of the tested protocol/software to these 30 MB yielded the RAM usage per container, which could then be multiplied by the number of nodes to find the required amount of RAM.

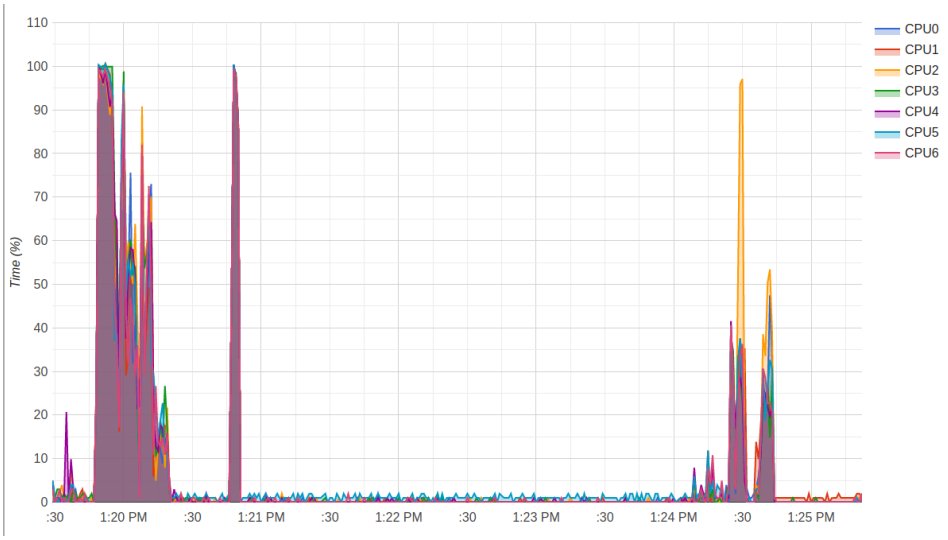

Figure 3. CPU usage of EmuCD without any protocol, 100 nodes.

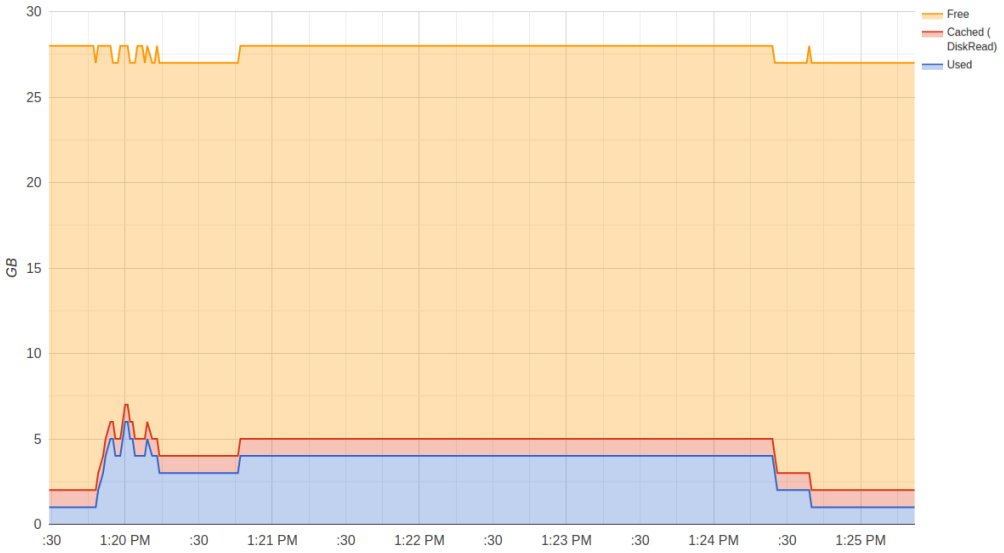

Figure 4. RAM usage of EmuCD without any protocol, 100 nodes.

Figures 5 and 6 show the CPU and RAM usage, respectively, of running 100 nodes with BitTorrent disseminating a 50 MB file during an hour of real VANET data mobility. Starting with the CPU analysis, the first spike at 11:40 a.m. corresponded to the nodes being booted and the emulation starting, whereas the last spike at 12:40 p.m. was the emulation ending and the containers shutting down. The spikes in between were due to an increased network activity between containers, with the average CPU usage hovering around 5% during the process, naturally higher than in the previous case when the emulator was not running any protocol. Regarding RAM, BitTorrent can be a lightweight protocol, with an average 35 MB used per container (meaning around 5 MB were used by the protocol).

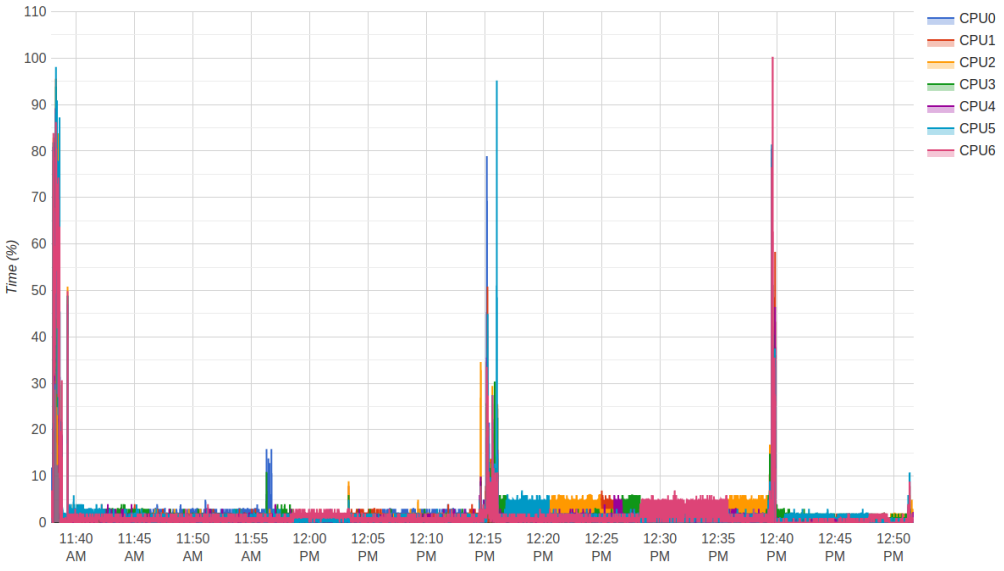

Figure 5. CPU usage of EmuCD running BitTorrent on 100 nodes, 50 MB file.

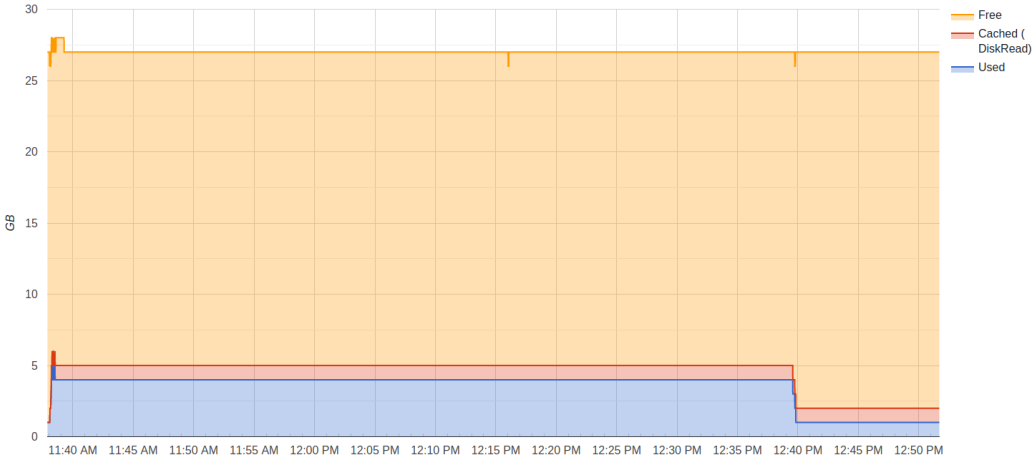

Figure 6. RAM usage of EmuCD running BitTorrent on 100 nodes, 50 MB file.

The worst case scenario was an edge case where every node was at full activity. It is very rare to have this behaviour in a VANET, so a stress test was developed, consisting of 95 OBUs and 5 RSUs (a 50 MB file starts only in the RSUs), where each OBU was connected to one of the RSUs and to every other OBU (each OBU has 95 neighbours). This was an extremely dense scenario, built to push the limits of the emulator (in fact it was so demanding that the hardware could only handle 100 nodes). Each RSU was sending the BitTorrent files to the OBUs, which then propagated to other OBUs. Figures 7 and 8 show the CPU and RAM usage during this stress test, respectively. There was an increased CPU usage in the beginning, from 11:40 p.m. until 11:41 p.m., due to the containers being booted. The test started at 11:41 p.m. and an increasing graph was observed, matching the progress of the file distribution: initially only the RSUs had the file's chunks but as they spread, more and more simultaneous transfers between OBUs occurred. At 11:45 p.m. every container was already shutdown and the test was finished. Total RAM usage hovered around 6 GB.

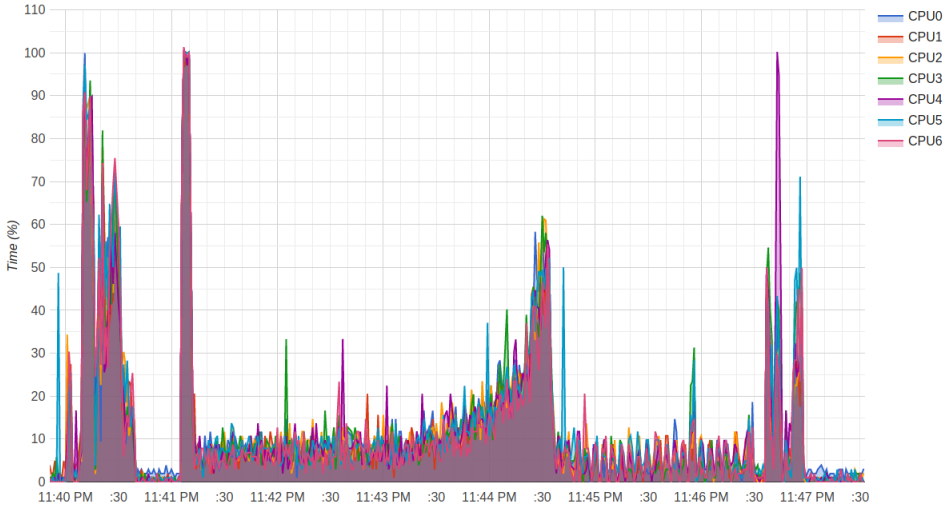

Figure 7. CPU usage of BitTorrent during a stress test, 100 nodes, 50 MB file.

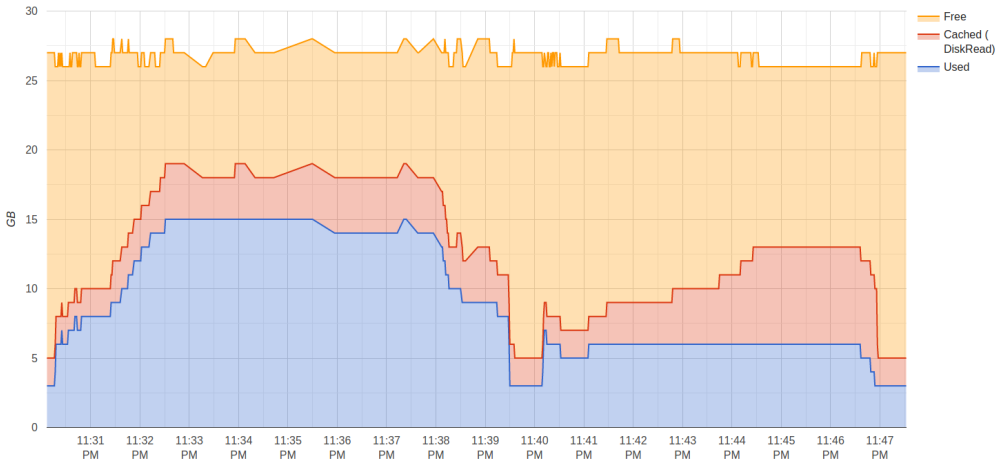

Figure 8. RAM usage of BitTorrent during a stress test, 100 nodes, 50 MB file.

These results show that, with enough hardware resources, every scenario was possible. Excluding the above mentioned edge case and focusing on the BitTorrent example, in a "normal" VANET, at 35 MB per container and with the observed CPU usage, the deciding factor was RAM; therefore, if 35 GB were available to be used by the emulator, 10× more nodes could be launched and emulated. Note that, with other more complex protocols, the limiting factor may not be RAM, but CPU instead. The hardware requirements must be adapted according to the protocol and scale of the emulation.

5.2. EmuCD Validation

This section presents the validation of EmuCD against a real vehicular network, when running a content distribution protocol. This validation is performed with the content distribution protocol InterPlanetary File System (IPFS) [24]. IPFS is a decentralized, peer-to-peer distributed file system [24]; it can be seen as a single BitTorrent swarm exchanging objects within one Git repository, providing a content-addressed block storage model. As far as we know, IPFS has never been tested in a VANET before, so it was tested not only in our emulator, but also in a small subset of a real VANET. Despite the OBUs running a custom version of OpenWRT and EmuCD's Docker containers Ubuntu, it is exactly the same code/software running on both: what runs in the real boards is what runs in the emulator. This VANET is located in Oporto city [25] from which data traces were collected to build the dataset used in the emulator. This dataset considers both mobility traces and connectivity traces between VANET nodes. A small 5-node subset of this VANET was made available for testing in real conditions with Veniam's (Veniam, The Internet of Moving Things-https://veniam.com/) support.

In this section we consider the validation metric the progress rate (average of the content's download progress across all OBUs) over time, when running IPFS in a real VANET and in the EmuCD. We consider three different use cases for the validation with one seeder and four OBUs: Scenario (1) a fast content dissemination, where the nodes are close neighbours (not only spatially but also temporally) with contact times long enough for data transfers to occur; Scenario (2) a medium-speed content dissemination, where the nodes are occasionally in each other's neighbourhood but, contact times are not enough for the file to be fully transferred between two OBUs, requiring multiple contacts and taking potentially hours until the data are distributed across every node; Scenario (3) long content dissemination, where some OBUs are very far away from nodes with useful data (e.g., non-overlapping bus routes), and it can take a full day or longer for the content to reach every node.

Figure 9 compares the progress rate of a IPFS file of 50 MB in Scenario 1.

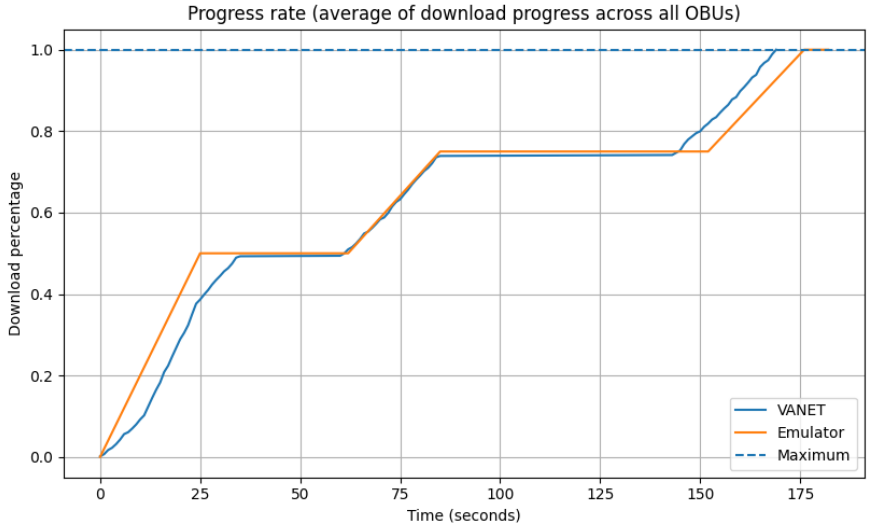

Figure 9. Scenario 1: Comparison of download times of a 50 MB file dissemination in a five node vehicular ad-hoc network (VANET) using InterPlanetary File System (IPFS) vs. EmuCD using IPFS.

The graph shows that the results are similar, with only a few seconds of difference in some parts and fully distributing the content to every node within 175 s, but there are some important observations to be made:

- the emulator graph only shows straight lines during data transfers, whereas the real VANET's graph has more granularity and variance-in the real VANET there is more than connectivity information, such as interference, leading to an inconsistent throughput, which ranged from under 0.5 MB/s up to 2.8 MB/s; EmuCD sets a 2 MB/s upload and download fixed limit on the wireless interfaces, which was observed to be the average in initial lab tests with the hardware;
- the emulator graph reached 50% progress 10 s earlier than the real VANET, it accompanied the VANET in the second dissemination period and fell 8 s behind the real VANET in the final dissemination period—this can happen because of several factors, but the most probable one is the connection establishment time, which is not negligible in ad-hoc networks. The emulated scenario did not replicate this, it only added/removed IP addresses from the OS's routing table.

Figure 10 shows the comparison between IPFS distributing a 50 MB file in a five node real VANET and in an emulated five node VANET in Scenario 2. The emulator had very close results when compared with the VANET, albeit with lower accuracy, namely in the first 300 s where in the real test the transfer slowed down but in the emulator it kept going, arriving at 50% content dissemination earlier than it should. In both cases, the content was fully distributed after 3500 s, taking 20 times longer than Scenario 1 due to the long periods without communication and also because of the existence of periods where two nodes did not "see" each other for too long.

Figure 11 shows the results of Scenario 3, with IPFS disseminating a 50 MB file in a five node VANET, real and emulated. This type of dissemination was characterized by extremely long periods without communication (more than 1 h), as shown by the fact that, after more than 3 h only 40% of the content had been distributed. As in the previous cases, EmuCD showed the same results as the real VANET, except this time in the middle transfer with the OBUs in the real scenario, probably due to an interference only observed in the real scenario, the download progress did not occur as a quick continuous event.

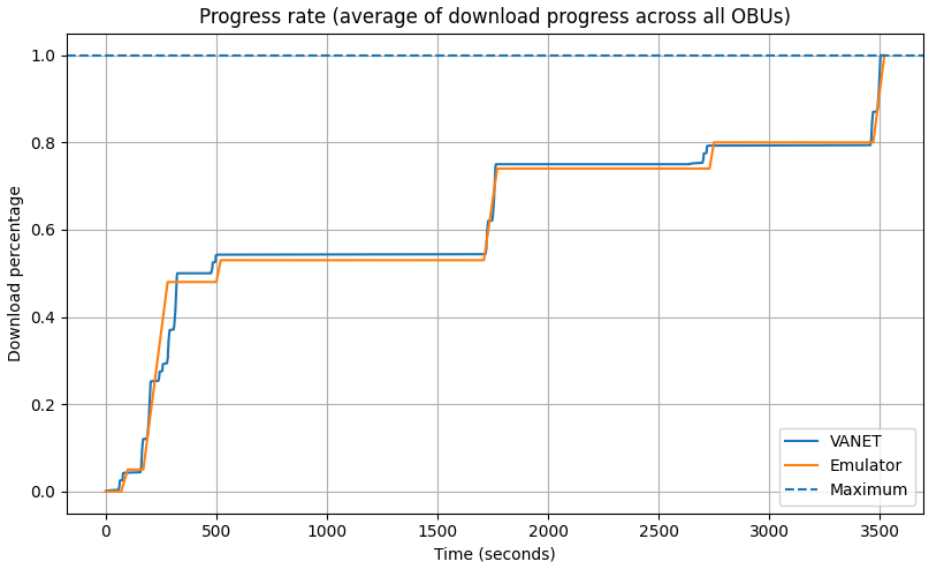

Figure 10. Scenario 2: Comparison of download times of a 50 MB file dissemination in a 5 node VANET using IPFS vs. EmuCD using IPFS.

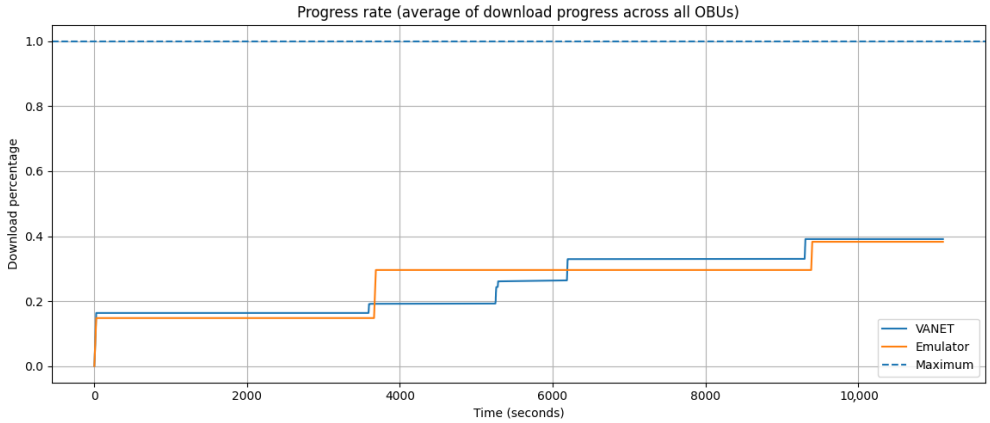

Figure 11. Scenario 3: Comparison of download times of a 50 MB file dissemination in a 5 node VANET using IPFS vs. EmuCD using IPFS

5.3. Content Distribution Results

We used the facilities of our emulator to test and adapt three P2P content distribution protocols for content dissemination in a VANET environment: IPFS, Sprinkler and BitTorrent. This section describes that development process, and the obtained results. IPFS was compared with Sprinkler [26], Veniam's P2P protocol being used for V2V content dissemination. Additionally BitTorrent was also evaluated, given its immense popularity as a P2P protocol. From these three, only Sprinkler was specifically developed for use in a VANET.

In EmuCD, results can be analyzed through content distribution metrics (which are easily customizable and new ones can be added), with 3 being default: cumulative download progress (sum of bytes that each OBU has downloaded), number of OBUs that reached 100% in each instant and progress rate (average of the content's download progress across all OBUs). Next, the results of a 50 MB file being distributed through 200 nodes from 15 h to 16 h are shown in Figures 12 and 13; it can be observed that 20% of the content is distributed in the first 5 min, but the progress then slows down, as it takes 20 min for another 20% to be disseminated. After an hour, 60% of the content has been distributed across the network. As Figure 12 shows, this does not mean that every node has 60% of the content: dozens of OBUs receive everything in the first 5 min and after an hour, there may be

OBUs that have not downloaded a single byte due to constraints in the network's topology as they may be far away from the other nodes.

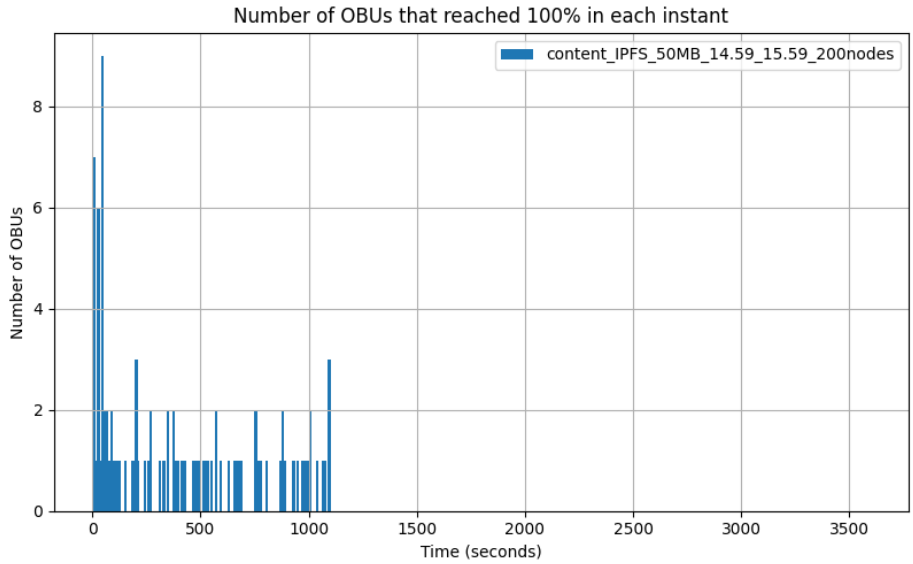

Figure 12. OBUs that downloaded 100% of a 50 MB file distributed across 200 nodes from 15 h to 16 h in EmuCD running IPFS.

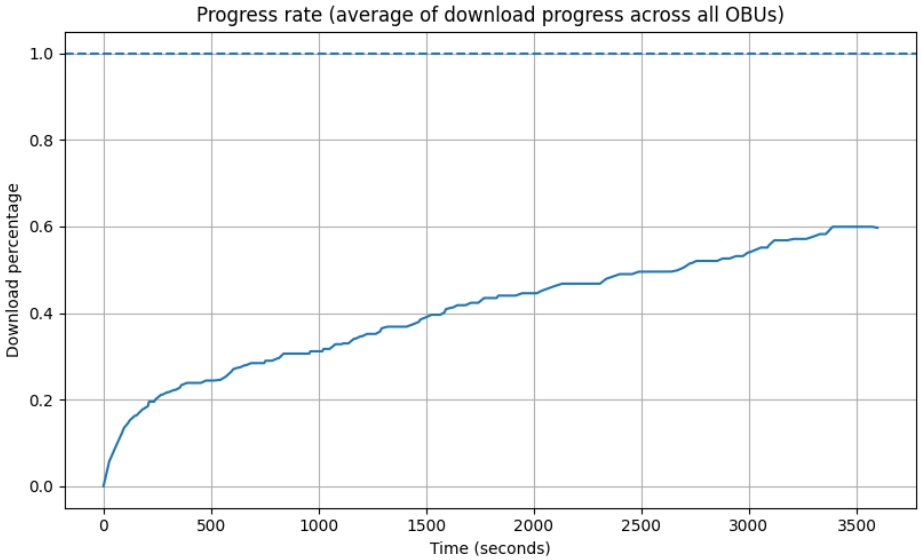

Figure 13. Progress rate of a 50 MB file distributed across 200 nodes from 15 h to 16 h in EmuCD running IPFS.

The graphs previously shown can be generated in EmuCD's frontend dashboard, Figure 14, which also displays the nodes in a map and their individual download progress (by default On-Board Units (OBUs) as blue dots and Road-Side-Units (RSUs) as orange dots), presents the emulation's status and is also used to start and stop the emulation.

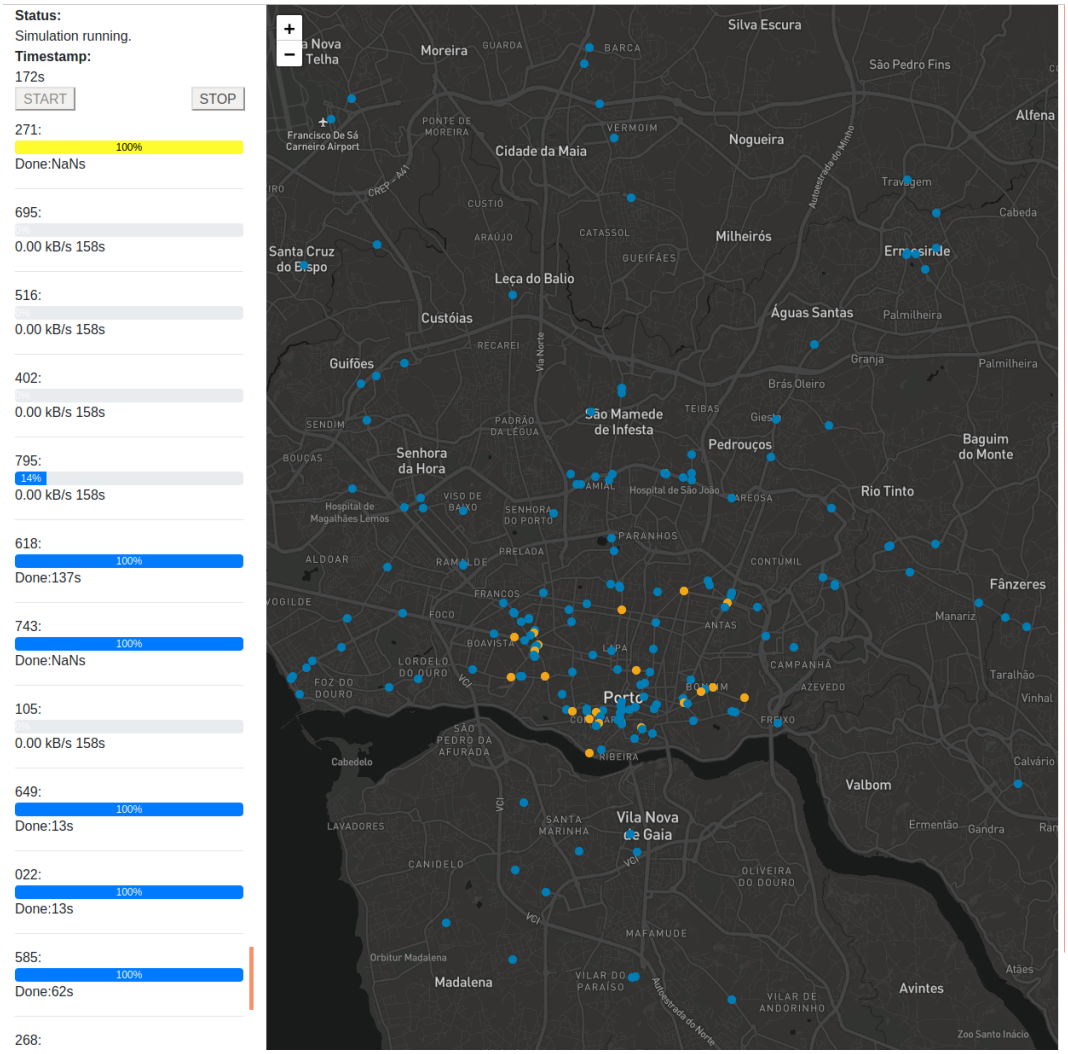

Figure 14. Frontend of the network emulator running a scenario with 200 nodes in Oporto.

Figure 15 shows the results of running three protocols-IPFS, BitTorrent and Sprinkler-in a real VANET datatrace of Oporto for 20 min, where a 50 MB file was distributed across 200 nodes (RSUs and OBUs). The number of OBUs that reached 100% and average download progress across all OBUs, show how both IPFS and Sprinkler can distribute on average 30% of the total content to the network in 20 min, whereas BitTorrent was poorly suited to the high-mobility characteristic of a VANET, as demonstrated by the 5% average progress rate.

These results show how EmuCD can be very fruitful to the evaluation of content dissemination protocols: it was able to run a large number of vehicles, the emulated results were very similar to the real ones, and it enabled a comparison between different protocols with a low programming overhead.

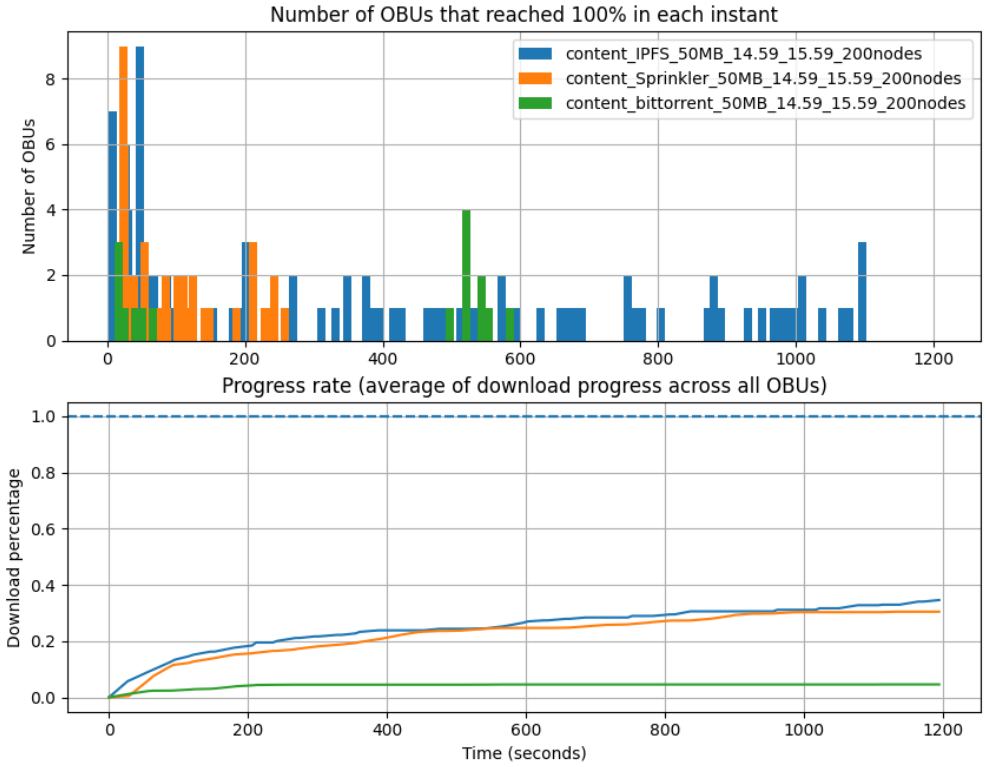

Figure 15. Comparison between IPFS (blue), BitTorrent (green) and Sprinkler (orange) with a 50 MB file being distributed through 200 nodes in Oporto from 15 h to 15:20 h.

6. Conclusions

In this article we presented EmuCD, an emulator for supporting the development and testing of content dissemination protocols for VANETs. EmuCD models the VANET mobile nodes in containers, configuring the network by customizing the interfaces of the container's operating system.

This strategy to emulate a VANET guarantees independence between the protocol and the communication layer, allowing the development of new protocols that can be tested on the emulated network and used in real equipment (hardware for vehicular communication) without any change in its code. Besides, EmuCD is versatile and can emulate VANETs using real datatraces harvested directly in a production network such as Oporto's or generated through simulators such as SUMO. Finally, we demonstrate the facilities of EmuCD through the evaluation of three content dissemination protocols, IPFS, BitTorrent and Sprinkler.

When comparing EmuCD with a real VANET, results have shown that if connectivity/mobility data is accurate, EmuCD can successfully emulate content dissemination transfers in a VANET, being able to work as a platform for the development of content dissemination protocols in VANETs.

As future work, we aim to expand PIPE's facilities by adding a procedure to automate the creation of containers and manage the execution of simulations. Finally, we will evaluate the inclusion of a real time dashboard to show all information about the execution of the simulation, and integrate new content distribution protocols.

Author Contributions: All authors designed the solution, analyzed the results and wrote the paper. R.C. implemented the solution, prepared the evaluation scenarios and the results. C.S., M.L. and S.S. supervised the entire research process. A.M., D.R. and R.M. supervised this work on a company environment. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the European Regional Development Fund (FEDER), through the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework, and Public and National Financial Support (FCT)(OE) through project MobiWise (POCI-01-0145-FEDER-016426).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Conde, J.; Senna, C.; Sargento, S. Content Distribution Optimization Algorithms in Vehicular Networks. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 871–877. [CrossRef]
2. Chengetanai, G.; O'Reilly, G.B. Survey on simulation tools for wireless mobile ad hoc networks. In Proceedings of the 2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), Coimbatore, India, 5–7 March 2015; pp. 1–7. [CrossRef]
3. Ben Mussa, S.A.; Manaf, M.; Ghafoor, K.Z.; Doukha, Z. Simulation tools for vehicular ad hoc networks: A comparison study and future perspectives. In Proceedings of the 2015 International Conference on Wireless Networks and Mobile Communications (WINCOM), Marrakech, Morocco, 20–23 October 2015; pp. 1–8. [CrossRef]
4. Mittal, N.M.; Choudhary, S. Comparative Study of Simulators for Vehicular Ad-hoc Networks (VANETs). *IEEE Int. J. Emerg. Technol. Adv. Eng.* **2014**, *4*, 528–537.
5. Hajlaoui, R.; Moulahi, T.; Guyennet, H. Vehicular ad hoc networks: From simulations to real-life scenarios. *J. Fundam. Appl. Sci.* **2018**, *10*, 632–637.
6. Lopez, P.A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; Wießner, E. Microscopic Traffic Simulation using SUMO. In Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 4–7 November 2018.
7. Marfia, G.; Pau, G.; Giordano, E.; De Sena, E.; Gerla, M. VANET: On Mobility Scenarios and Urban Infrastructure. A Case Study. In Proceedings of the 2007 Mobile Networking for Vehicular Environments, Anchorage, AK, USA, 11 May 2007; pp. 31–36. [CrossRef]
8. Harri, J.; Fiore, M.; Filali, F.; Bonnet, C. Vehicular mobility simulation with VanetMobiSim. *Simulation* **2011**, *87*, 275–300. [CrossRef]
9. Bagrodia, R.; Gerla, M. A Modular and Scalable Simulation Tool for Large Wireless Networks. In *Computer Performance Evaluation*; Puigjaner, R., Savino, N.N., Serra, B., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; pp. 1–14.
10. Khairnar, V.D.; Pradhan, S.N. Mobility models for Vehicular Ad-hoc Network simulation. In Proceedings of the 2011 IEEE Symposium on Computers Informatics, Kuala Lumpur, Malaysia, 20–23 March 2011; pp. 460–465. [CrossRef]
11. Hafeez, K.A.; Zhao, L.; Liao, Z.; Ma, B.N. Impact of Mobility on VANETs' Safety Applications. In Proceedings of the 2010 IEEE Global Telecommunications Conference GLOBECOM 2010, Miami, FL, USA, 6–10 December 2010; pp. 1–5. [CrossRef]
12. Mangharam, R.; Weller, D.; Rajkumar, R.; Mudalige, P.; Bai, F. GrooveNet: A Hybrid Simulator for Vehicle-to-Vehicle Networks. In Proceedings of the 2006 Third Annual International Conference on Mobile and Ubiquitous Systems: Networking Services, San Jose, CA, USA, 17–21 July 2006; pp. 1–8. [CrossRef]
13. Piórkowski, M.; Raya, M.; Lugo, A.L.; Papadimitratos, P.; Grossglauser, M.; Hubaux, J.P. TraNS: Realistic Joint Traffic and Network Simulator for VANETs. *Sigmobile Mob. Comput. Commun. Rev.* **2008**, *12*, 31–33. [CrossRef]
14. Wang, S.Y.; Chou, C.L. NCTUns 5.0 Network Simulator for Advanced Wireless Vehicular Network Researches. In Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, Taipei, Taiwan, 18–20 May 2009; pp. 375–376. [CrossRef]
15. Sommer, C.; German, R.; Dressler, F. Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis. *IEEE Trans. Mob. Comput.* **2011**, *10*, 3–15. [CrossRef]
16. Malinverno, M.; Raviglione, F.; Casetti, C.; Chiasserini, C.F.; Mangues-Bafalluy, J.; Requena-Esteso, M. A Multi-Stack Simulation Framework for Vehicular Applications Testing. In Proceedings of the 10th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications DIVANet '20, Alicante, Spain, 16 November 2020; pp. 17–24. [CrossRef]

17. Cirne, P.; Zúquete, A.; Sargento, S. Loop—A Trace-based Emulator for Vehicular Ad Hoc Networks. In *8th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2018)*; SCITEPRESS-Science and Technology Publications, Lda: Setubal, Portugal, 2018; pp. 391–402. [CrossRef]
18. Pessoa, G.; Dias, R.; Condeixa, T.; Azevedo, J.; Guardalben, L.; Sargento, S. Content distribution emulation for vehicular networks. In Proceedings of the 2017 Wireless Days, Porto, Portugal, 29–31 March 2017; pp. 208–211. [CrossRef]
19. Pessoa, G.; Luis, M.; Guardalben, L.; Sargento, S. On the Analysis of Content Dissemination through Real Vehicular Boards. In Proceedings of the 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), Porto, Portugal, 3–6 June 2018; pp. 1–7. [CrossRef]
20. Pessoa, G.; Guardalben, L.; Luis, M.; Senna, C.; Sargento, S. Evaluation of Content Dissemination Strategies in Urban Vehicular Networks. *Information* **2020**, *11*, 163. [CrossRef]
21. Fontes, H.; Campos, R.; Ricardo, M. A Trace-Based Ns-3 Simulation Approach for Perpetuating Real-World Experiments. In Proceedings of the Workshop on Ns-3 WNS3 '17, Porto, Portugal, 13 June 2017; pp. 118–124. [CrossRef]
22. Yokoyama, R.S.; Kimura, B.Y.L.; Villas, L.A.; Moreira, E.D.S. Measuring Distances with RSSI from Vehicular Short-Range Communications. In Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool, UK, 26–28 October 2015; pp. 100–107. [CrossRef]
23. Branquinho, J.; Senna, C.; Zúquete, A. An Efficient and Secure Alert System for VANETs to Improve Crosswalks' Security in Smart Cities. *Sensors* **2020**, *20*, 2473. [CrossRef] [PubMed]
24. Benet, J. IPFS-Content Addressed, Versioned, P2P File System. 2014. Available online: https://arxiv.org/abs/1407.3561 (accessed on 20 December 2020).
25. Santos, P.M.; Rodrigues, J.G.P.; Cruz, S.B.; Lourenço, T.; d'Orey, P.M.; Luis, Y.; Rocha, C.; Sousa, S.; Crisóstomo, S.; Queirós, C.; et al. PortoLivingLab: An IoT-Based Sensing Platform for Smart Cities. *IEEE Internet Things J.* **2018**, *5*, 523–532. [CrossRef]
26. Recharte, D.; Aguiar, A.; Cabral, H. Cooperative Content Dissemination on Vehicular Networks. In Proceedings of the 2018 IEEE Vehicular Networking Conference (VNC), Taipei, Taiwan, 5–7 December 2018; pp. 1–8. [CrossRef]