Towards CANLay: User-centered Overlay Design for In-Vehicle Data Dissemination In a Network Virtualized Testbed

Abstract

Your abstract text goes here. Just a few facts. Whet our appetites. Not more than 200 words, if possible, and preferably closer to 150.

1 Introduction and Background

In recent years security of the Controller Area Network (CAN) has been a much talked about topic of research. CAN is a broadcast media that enables reliable and low-latency communication between in-vehicle devices, also referred to as Electronic Control Units (ECU). This broadcast nature of CAN, along with the fact that it is inherently unauthenticated, makes it susceptible to network-wide cyber threats. Security researchers have shown [2, 3, 12] that remote interfaces on modern vehicles can be used to intrude into internal CAN networks and inject messages to control and/or disrupt the operations of the vehicle. At the same time, the development of security solutions can be pursued to detect and/or prevent this scenario from occurring. To evaluate the effectiveness of their methods, researchers have typically experimented on real-vehicles or homegrown testbed setups that mimic real vehicles. While most households in the United States have at least one passenger car[1], this is not the same for medium and heavy-duty (MHD) vehicles. Moreover, creating homegrown testbeds is both logistically and economically challenging. To that end, the need for a publicly accessible testbench is imminent. This is where the concept of the Software Define Truck (SDT) [11] is critical to the in-vehicle networking community. It aims to provide a distributed network virtualized platform on which in-vehicle security experiments can be performed. Although proposed primarily for the heavy-trucks, SDT can easily be adapted for lightweight passenger vehicles. SDT's information exchange goal is shown in figure 1. CAN frames and physical signals need to be exchanged between ECUs and vehicle simulators located in different subnetworks around the globe. CANLay is the networking backbone of the SDT and aims to provide the necessary infrastructure to enable this service.

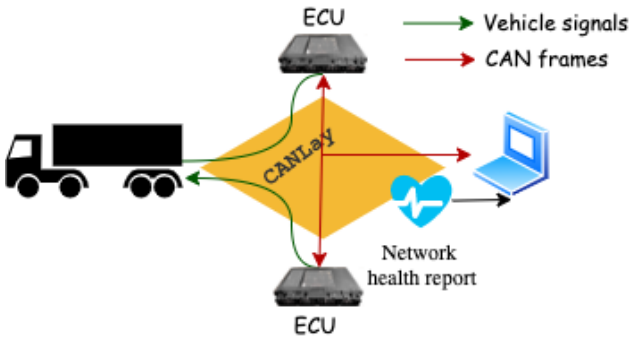

Figure 1: Scope of CANLay

Previous research [10] has established two critical criteria for quality evaluation of automotive networking testbeds: fidelity and adaptability. Fidelity is the ability to emulate a real-world in-vehicle net-

[1] https://www.statista.com/statistics/551403/number-of-vehicles-per-household-in-the-united-states/

working infrastructure. Adaptability is the ability to simulate different real-world in-vehicle networking infrastructures. As such, it may be difficult to optimize both at the same time. To make a system adaptable, the underlying components need to be virtualized so they can be reconfigured to suit user needs. Albeit, this hampers the fidelity of the system. While CANLay provides the means to configure experiment networks on-demand, it also provides a real-time health report for the underlying network. This allows the user to assess the fidelity of the overlay in terms of standard networking metrics like latency, rate of packet drop, etc.

Although, CAN is a relatively new communication technology and has a smaller application scope than TCP/IP, there has been some proposals to virtualize its operations. First, there has been the attempt to adapt the software-defined networking paradigm for CAN [4, 7, 13]. This approach is largely hardware-based and is catered for in-vehicle networking on CAN physical channels, not over long-range overlays. For range relaying of CAN frames, there has been the CAN-to-ethernet direction of research [6, 8]. The goal is not to enable ECU-to-ECU communication, rather transportation of data logged from one network to a remote endpoint. Configurability and network performance are usually not addressed. Neither is the CAN-to-ethernet paradigm designed to transport physical signals over long distances. X-in-the-loop (hardware, driver, vehicle etc.) simulation-based in-vehicle testbeds [1] have been proposed, but the signals from the simulators have been transported over physical connections, not over reconfigurable, long-range network overlays.

In summary, CANLay provides the following features:

- Transport of CAN frames and physical signals of the vehicle to a distributed network of electronic control units that can be located in different subnets
- Creation of these overlay CAN networks on demand
- Provision on runtime metrics to estimate the health of the network during the ongoing experiment.

In the rest of this paper we describe the design of CANLay (section 2), provide a usability analysis (3), and finish with conclusive remarks and future works.

2 Design and Current Development

Figure 2 shows the proposed system design of CANLay. The system serves three functions: offline configuration of the network overlay and CAN frame and vehicle signal exchange at runtime. A description of the components and their roles in the system is provided next. Following that, a description of the behavioral aspects of the system is provided. Together, these aspects combine to accomplish the functional objectives.

2.1 Component Descriptions

2.1.1 Smart Sensor Simulator and Forwarder (SSSF)

The Smart Sensor Simulator and Forwarder acts as a gateway enabling the ECU to access and, more importantly, to be accessed by the CANLay system. In an active experiment, it acts as a forwarder between the Controller and the ECU through User datagram protocol (UDP) channels and CAN interfaces. SSSFs can forward two types of messages. The first type carries signals from the vehicle simulator. Eventually, these signals may have to be transmitted on the analog wiring that is shown using a dashed line figure 2. The second type is CAN data carriers from the ECUs as well from other SSSFs in the current experiment. Through the SSSFs, multiple ECUs can actively communicate with each other to create a rich testing environment.

The SSSF is developed a built on a Teesny 3.6 .... a paragraph describing the SSS2's. Talk about SD cards. PLease include a blcok diagram etc. CAN Forwarding The real time clock on the SSF

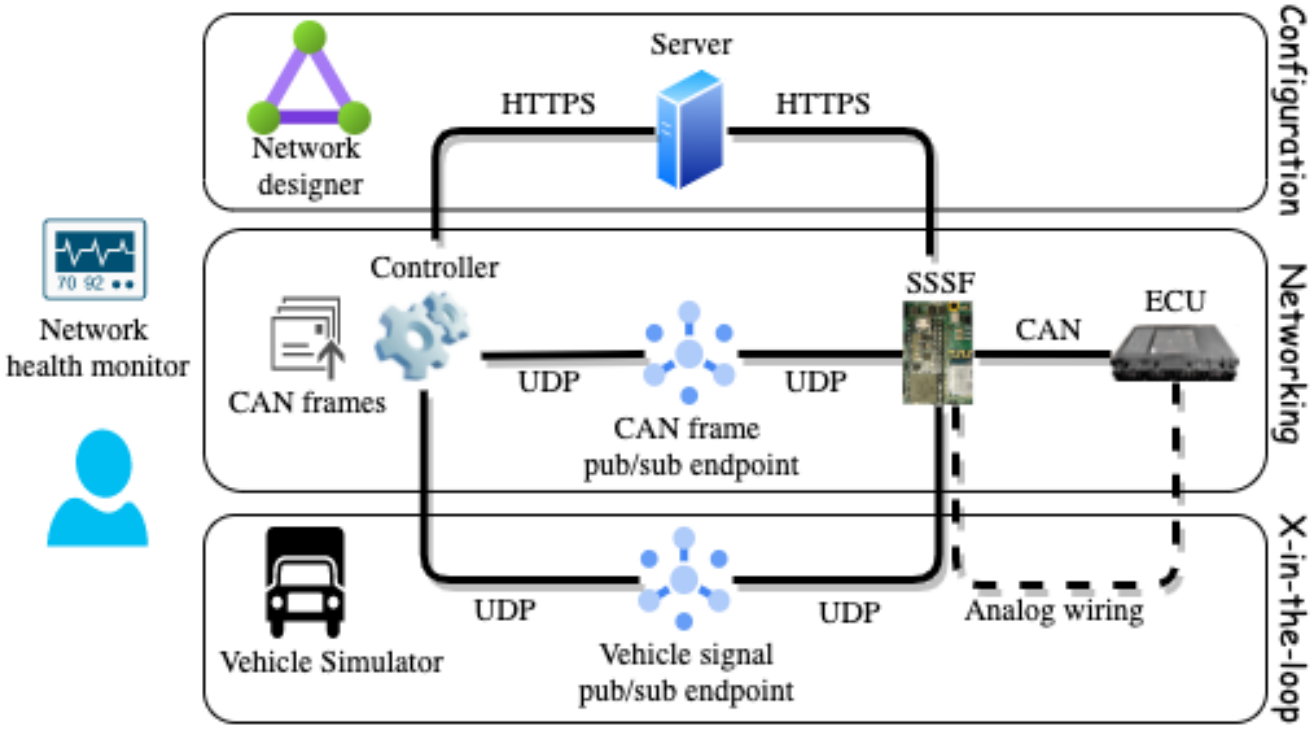

Figure 2: Proposed System

is synchronized through the network time protocol (NTP).

2.1.2 Controller

The Controller is the user's interface to the CANLay system and enables vehicle simulators to communicate with the CANLay network. The Controller's user interface is used to assist the user in building their virtual testbed. It does so by communicating with the central Server over hypertext transfer protocol secure (HTTPS). Once the experiment setup is completed the Controller transitions to acting as a gateway for a graphical vehicle simulator to communicate bi-directionally with the CANLay system. It forwards simulator outputs to the publish/subscribe (pub/sub) endpoint and listens for CAN messages from the same.

Add some implementation details - like python, thread etc. The real-time clock on the Controller is synchronized through the network time protocol (NTP).

2.1.3 Server

The Server helps in setting up the publishers and subscribers for an experiment. Each device opens and must maintain a persistent transmission control protocol (TCP) connection with the Server while they participate in the CANLay system. Once the TCP connection is established the devices communicate with the Server through HTTP application programming interfaces (API). The Server can monitor the health of the devices and take actions if a device is malfunctioning or goes offline. This also allows the Server to keep track of free devices and free pub/sub endpoints, so it can validate new experiment requests and allocate the requested devices and endpoints without running into race conditions or double use issues that may arise if each Controller was in charge of allocating its own experiment. Finally, the Server keeps track of ongoing experiments and ensures the proper closure of a experiment in the event a device is experiencing issues.

Add some implementation details - like python, thread etc. Sockets. The Server accepts HTTP API calls. API calls were chosen because they clearly define the object to invoke and the manner in which to invoke it.

2.1.4 Publish/Subscribe Endpoints

UDP is used to connect the publishers and subscribers in the CANLay system. The pub/sub model was chosen because it can easily emulate the broadcast nature CAN [9] in that an ECU is subscribed to all other ECUs on the same CAN bus and all other ECUs on that CAN bus are subscribed to that ECU.

To find a suitable pub/sub mechanism that closely resembles a CAN network, we used a few criteria. The first is that the transport mechanism must support some form of message broadcasting that enables a sender to send one message that can be received by many receivers without significant duplication and delay-related overheads [9]. The next requirement is that the transport mechanism must enable the devices to receive messages from one or more devices while having to maintain only one connection.

At this time we have chosen UDP multicasting as a suitable pub/sub mechanism as it does not require a message broker with high-performance requirements. We realize that multicasting outside a local network may lead to increased cost for the implementers, but the current goal was to test its usability and make future decisions based on the observed performance. At this time, we are also exploring other potential pub/sub implementations such as MQTT.

2.1.5 Front-End Components

CANLay provides two front-end components: network designer and network health monitor. Figures – and – show the snapshots of each ofn these components.

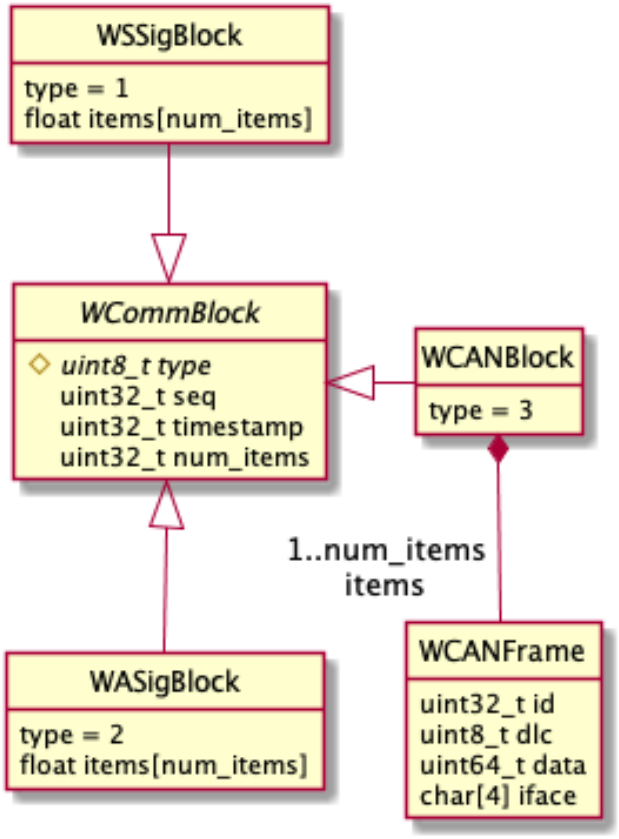

Figure 3: Transport Data Structures

2.2 Behavior Descriptions

2.3 Network Setup

2.4 Data Transport

2.5 Network Health Measurement

3 Current Implementation

Figure 4 shows CANLay at work. The windows in the figure display CAN frames on left, the vehicle simulator on the bottom right and CANLay's network health monitoring on the top right. Each of these components were already introduced in figure 2. For the current purpose we have been using the CARLA graphical vehicle simulator [5]. Although the Carla project mainly focuses on autonomous driving research it exposes its in-game signals through an easy-to-use python API and pays

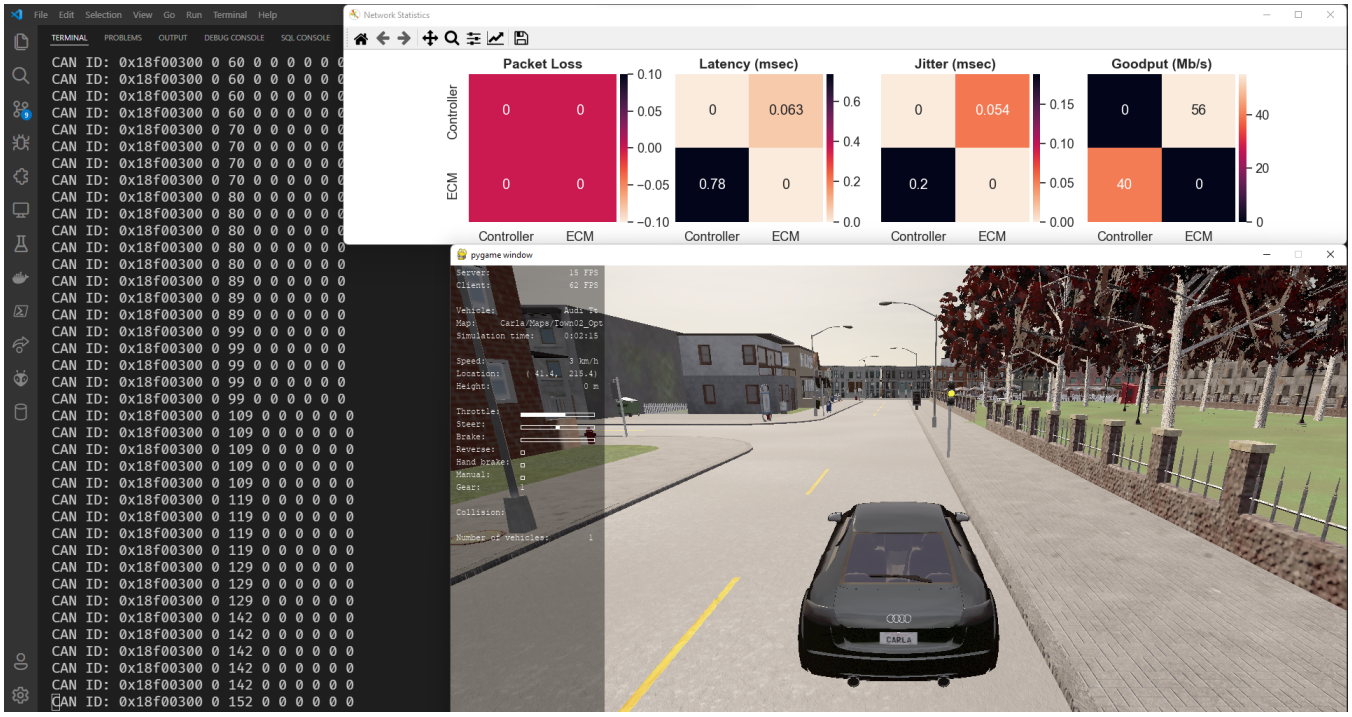

Figure 4: CANLay at work in the Software Defined Truck

close attention to the scientific details represented in its simulator. While this is not required, the more realistic and accurate the signals are, the easier it will be to transform them into CAN messages. In this case, a specific CAN frame is printed as they are broadcasted on the overlay for this particular experiment. The ID of this frame is defined by the SAE-J1939 standards [14] and identifies engine parameters transmitted by an engine control module (ECM). The data bytes carried in the CAN frame are shown next. Of these, the third byte is shown to be changing. This particular byte carries the percentage throttle demanded by the driver. The value is also non-zero on the simulator frontend provided by the CARLA simulator.

On the top right is the network health monitoring window. It shows four matrices showing four different metrics to estimate network health: packet loss, latency, jitter, and goodput. The significance of each of these metrics and their calculation methods were already described in the previous section. In this case, the experiment is performed over a gigabit local area network with a layer 3 switch in between an SSSF and a Controller. The figure shows no packets were lost while the latency in the last cycle of health report collection was about 4 milliseconds between the endpoints. Although CANLay does not explicitly perform any latency reducing functions, the general latency of 4 milliseconds is considered to be sustainable for seamless CARLA emulation at standard frame rates. In this particular example, the CARLA emulation frame rate was chosen to be 60 frames per second. The jitter is also fairly low in comparison to the latency. The goodput, i.e. the application data rate is understandably higher for the Controller as it sends WSignalBlock frames that are slightly larger than the WCANBlock frames.

4 Conclusion and Future Work

In this paper, we described the concepts behind the design of CANLay, the networking backbone for the Software Defined Truck. SDT is a virtualization based experimentation framework for CAN-based security experiments and CANLay is the carrier of physical control and CAN data over long dis-

tance networks. Essentially CANLay enables network virtualization for SDT. CAN is a reliable and low-latency network. CANLay does not explicitly ensure reliability and low latency, but provides a health monitoring service that provides real-time measures of network parameters to the user. This allows the user to make critical decisions about the state of the experiment they are in.

We believe more than one additional works can still be done on CANLay. Need response Dynamic buffer adjustment

References

[1] Matthew Appel, Pradeep Sharma Oruganti, Qadeer Ahmed, Jaxon Wilkerson, and Rubanraj Sekar. A Safety and Security Testbed for Assured Autonomy in Vehicles. In *Proceedings of the WCX SAE World Congress Experience*, page 8, 2020.

[2] Yelizaveta Burakova, Bill Hass, Leif Millar, and Andre Weimerskirch. Truck Hacking: An Experimental Analysis of the SAE J1939 Standard. In *Proceedings of the 10th USENIX Conference on Offensive Technologies*, pages 211–220, Austin, TX, USA, 2016. USENIX Association.

[3] Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, Stefan Savage, Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *USENIX Security Symposium*, volume 4, pages 447–462, San Francisco, CA, USA, 2011. USENIX Association.

[4] Michael Doering and Marco Wagner. Retrofitting SDN to classical in-vehicle networks:. Technical report, Universit{\"a}t T{\"u}binge.

[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[6] Florian Polzlbauer and Allan Tengg. Experience Report: Lightweight Implementation of a Controller Area Network to Ethernet Gateway. In *Proceedings of the Brief Presentation Track of the RTAS'19 Conference*, MONTREAL, CANADA, 2019. IEEE.

[7] Dennis Grewe, Naresh Nayak, Deeban Babu, Wenwen Chen, Sebastian Schildt, and Clemens Schroff. BloomyCAN: Probabilistic Data Structures for Software-defined Controller Area Networks. In *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, pages 1–6, 2021.

[8] Mathias Johanson, Lennart Karlsson, and Tore Risch. Relaying Controller Area Network Frames over Wireless Internetworks for Automotive Testing Applications. In *2009 Fourth International Conference on Systems and Networks Communications*, pages 1–5, 2009.

[9] J. Kaiser and M. Mock. Implementing the real-time publisher/subscriber model on the controller area network (CAN). In *Proceedings 2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99) (Cat. No.99-61702)*, pages 172–181, 1999.

[10] Shahid Mahmood, Hoang Nga Nguyen, and Siraj A. Shaikh. Automotive Cybersecurity Testing: Survey of Testbeds and Methods. In *Digital Transformation, Cyber Security and Resilience of Modern Societies*, volume 84, pages 219–243. Springer International Publishing, Cham, 2021.

[11] Subhojeet Mukherjee and Jeremy Daily. Towards a Software Defined Truck. In *Proceedings of the 31st Annual INCOSE International Symposium*, page 16, Online, 2021. INCOSE.

[12] Subhojeet Mukherjee, Hossein Shirazi, Indrakshi Ray, Jeremy Daily, and Rose Gamble. Practical DoS Attacks on Embedded Networks in Commercial Vehicles. In *International Conference on Information Systems Security*, pages 23–42, Jaipur, Rajasthan, India, 2016. Springer.

[13] Randolf Rotermund, Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt. Requirements Analysis and Performance Evaluation of SDN Controllers for Automotive Use Cases. In *2020 IEEE Vehicular Networking Conference (VNC)*, pages 1–8, 2020.

[14] Society of Automotive Engineers. SAE J1939 Standards Collection.