D4: Dynamic, Decentralized, Distributed, Delegation-Based Network Control and Its Applications to Autonomous Vehicles

Anuj Kaul, Katia Obraczka, Ramon dos Reis Fontes, Thierry Turletti

D4: Dynamic, Decentralized, Distributed, Delegation-Based Network Control and Its Applications to Autonomous Vehicles

Anuj Kaul
Katia Obraczka
anujkaul@soe.ucsc.edu
katia@soe.ucsc.edu
UCSC, Santa Cruz, CA, USA

Ramon Fontes
ramon.fontes@imd.ufrn.br
Universidade Federal do Rio Grande do Norte
Natal, RN, Brazil

Thierry Turletti
thierry.turletti@inria.fr
Inria, Université Côte d'Azur
Sophia Antipolis, France

ABSTRACT

Connected autonomous vehicles technology are expected to be an important component of Intelligent Transportation Systems (ITS). Several relevant industry standards are being created to prepare for the maturity of connected vehicles with the help of artificial intelligence, cognitive methods, software-hardware and sensor platforms. One of the significant challenges raised by connected vehicles is how the underlying communication infrastructure will be able to efficiently support them. In this paper, we propose the Dynamic, Decentralized, Distributed, Delegation-based (D4) network control plane architecture which aims at providing adequate communication support for connected vehicles. To our knowledge, D4 is the first framework to support on-demand network control decentralization. At its core, D4's flexible network control delegation framework allows network control to be dynamically distributed on demand to address quality-of-service requirements of different connected vehicle services and applications. We demonstrate the benefits of D4 through a proof-of-concept prototype running on a fully reproducible emulation platform. Experimental results using a variety of real-world scenarios demonstrate that D4 delivers lower latency with minimal additional overhead. We also showcase the benefits of D4 when compared to traditional vehicular ad-hoc network (VANET) approaches.

KEYWORDS

Self driving cars, autonomous vehicles, connected vehicles, Intelligent transport systems, software defined networking, distributed network control, next-generation vehicular network, control delegation, handover

1 INTRODUCTION

Autonomous vehicles (AVs), also known as self-driving, driverless, or robotic cars, are vehicles capable of sensing their environment and moving safely with little or no human input. According to a recent survey [41], self-driving systems can be categorized as *standalone systems* and *connected systems*. In *standalone systems*, all necessary automated driving operations are performed on a single, self-sufficient vehicle at all times. In recent years, considerable effort has been directed to making the standalone architecture viable where the AV is responsible for localization (i.e., find its location with respect to its surrounding), perception (i.e., perceive the world around it), prediction and/or navigation (i.e., generate global and local paths for end-to-end driving). An example of a standalone system is the Adaptive Cruise Control (ACC) architecture [36]. In *connected systems*, on the other hand, at least some automated driving operations are performed by other vehicles and/or nearby network infrastructure. The Co-operative Adaptive Cruise Control (CACC) architecture is an example of a connected system where autonomous vehicles connect to nearby vehicles and infrastructure (e.g., Road-Side Units, or RSUs) in order to carry out autonomous driving operations [36].

Shortcomings of standalone systems include accuracy, sensing range, blind spots, and computing resource limitations are highlighted in [41]. Additionally, there is a great deal of recent anecdotal evidence that confirms the limitations of the standalone vehicle approach. For example, the National Transportation Safety Board's (NTSB) report on Tesla's 2018 autopilot crash in California [1] attributed the accident to worn-off lane markings delineating the gore area (the area between an off-ramp and the main road) from the left-exit High-Occupancy Vehicle (HOV) lane. Tesla engineers surmised that the autosteer system likely momentarily lost its lane line prediction and/or identified a stronger lane line on the left side of the gore. Additionally, a recent article [2] points out that currently, all standalone systems rely heavily on pre-configured maps and that creating (and maintaining) maps for self-driving cars is difficult and time-consuming.

[1] https://www.ntsb.gov/investigations/AccidentReports/Reports/HAR2001.pdf
[2] https://medium.com/@ritidass29/5-key-challenges-faced-by-self-driving-cars-ed04e969301e

According to [41], connected vehicles can mitigate the limitations of standalone systems and, as a result, improve road traffic efficiency and safety. However, connected systems rely heavily on the underlying communication infrastructure to cooperate with other vehicles and carry out autonomous driving tasks efficiently and effectively. They make use of communication services such as Intelligent Transportation System (ITS)'s Road Hazard Warning (RHW) message dissemination [12, 13] to exchange messages aiming at alerting vehicles and/or their drivers in real-time of relevant events, including traffic jams and hazardous conditions. The so-called Internet of Vehicles (IoV) [17] framework is another notable effort towards the realization of connected vehicles. It uses communication infrastructure devices such as RSUs mainly as part of the data plane, i.e., for message forwarding, and follows a publish-subscribe model a la Information-Centric Networking (ICN) [2].

Due to their stringent requirements such as low latency, high reliability, availability, scalability, as well as their inherent geographic and administrative decentralization, connected systems pose significant challenges to the underlying communication infrastructure. Additionally, the network must provide connected systems a range of quality-of-service (QoS) guarantees since different autonomous driving tasks have diverse requirements, e.g., latency guarantees. For example, applications such as route planning and navigation require latency in the order of seconds to minutes, while perception tasks need latency in the order of seconds and vehicle control functions must be carried out in milliseconds. According to the 3rd Generation Partnership Project (3GPP) Technical Specification (3GPP) TS 22.185 release 17 [1], latency requirements range from 20*ms* to 100*ms* for vehicle-to-infrastructure (V2X) communication depending upon the urgency of the message. For example, pre-crash sensing messages should have maximum latency of 20ms.

In order to meet the wide range of quality-of-service guarantees needed by connected vehicles, including ultra-low latencies, we argue that the underlying communication infrastructure must provide an efficient, reliable, scalable data forwarding plane to provide adequate support for connected self-driving systems. As such, the network control plane must act as a "control continuum" where control is delegated dynamically to intelligent controllers that are decentralized and distributed across the network. For example, navigation updates (or any other road or traffic condition updates) can be offloaded to the infrastructure which can then send this information to vehicles in close to real-time via Vehicle-to-Infrastructure (V2I) communication, while driving coordination, especially under hazardous road conditions (e.g., snow or rain), can be carried out in cooperation with nearby vehicles and/or infrastructure using Vehicle-to-Vehicle (V2V) and V2I communication, respectively.

The basic premise of the network control continuum we envision involves moving network control closer to its end users in order to satisfy the diverse QoS of edge applications, in particular connected vehicles. While efforts are ongoing to determine where network controllers should be positioned [35], there is an architectural gap to go beyond controller placement and address the dynamic reconfiguration of the control fabric. This is particularly vital for upholding QoS in connected vehicles as the network should be able to adjust dynamically in response to events such as new elements (like vehicles and controllers) entering or leaving the system, changes in network connectivity, environmental conditions, etc. We propose the Dynamic, Decentralized, Distributed, Delegation-Based Network Control architecture, or D4 for short, which, to our knowledge, is the first framework to provide dynamic control delegation in support of a decentralized and distributed network control plane continuum. While D4 can find applications in a variety of edge computing scenarios, we showcase D4 as enabling technology for connected vehicles. As described in detail in Section 3, D4 can dynamically handover control of its elements (e.g., vehicles) to network controllers distributed across the network, e.g., cloud, network infrastructure (base stations, road-side units) and even vehicles themselves, depending on current road, environmental and connectivity conditions.

The contributions of this paper are many-fold and include: (1) design and implementation of D4's dynamic control delegation mechanism (2) design and implementation of D4's communication protocols to support dynamic control delegation; (3) design and implementation of two control delegation mechanisms, one aiming at providing adequate QoS to vehicles and the other aiming at balancing controller load; (4) experimental demonstration of how D4 can leverage V2V communication, (5) extensive evaluation of D4's network control delegation approach using a range of experiments emulating realistic road traffic scenarios, demonstrating that D4 provides low message delivery latency with minimal additional overhead and packet loss; (6) experimental performance evaluation of D4 compared against traditional ad-hoc vehicular networking (VANET) approaches - which, to our knowledge, is the first time a dynamically distributed decentralized network control plane architecture has been compared against traditional VANETs. Additionally, our work makes contributions to the open-source software community, namely our implementation of the IEEE 802.11p stack for the Linux kernel driver *mac80211_hwsim* [9, 10]. We are also making public [3] our D4 implementation and experiments.

The remainder of the paper is organized as follows. Section II describes the background and prior work related to

[3] https://github.com/ankaul/d4

connected systems. Section III introduces the D4 architecture and design. ETSI(European Telecommunications Standards Institute)'s Road Hazard Warning (RHW) application is presented in Section IV. D4's current implementation is presented in Section V. Sections VI and VII describe our experimental methodology and results, respectively. We present final remarks and directions for future work in Section IX.

2 BACKGROUND AND RELATED WORK

As previously discussed, there has been an ongoing paradigm shift in the autonomous driving scene from standalone- to connected systems. In this section, we highlight efforts in related areas that are relevant to our work in D4. We summarize these efforts in Table 1 and compare them to D4.

3 D4 ARCHITECTURE AND DESIGN

D4's approach to addressing the requirements of connected vehicles leverages one of the basic principles underlying the Software-Defined Networking (SDN) paradigm [24], i.e., the decoupling of the control and the data planes. From the perspective of connected vehicles, the main benefits resulting from the separation between the control- and data planes include increased agility to detect and react to the dynamics of the underlying system/network and thus provide better support for stringent quality-of-service requirements imposed by connected vehicles.

On the other hand, SDN's logically centralized control plane is not compatible with the scalability and latency requirements imposed by connected vehicle services, and their geographic and administrative decentralization. For example, control plane scalability limitations have been discussed along with approaches to partially solve them in [19]. In D4, we leverage SDN's notion of network programmability enabled by the separation between the network control and data planes to achieve flexibility and agility. However, we go a step further by proposing a logically distributed network control "continuum" which can dynamically adjust to current conditions and network characteristics in order to support connected vehicles' stringent QoS (Quality of Service) demands, including scalability, delay intolerance, and reliability. Figure 1 illustrates some of the control plane scenarios D4 can support. The "root controller" represents the root of the control hierarchy and will typically be hosted by cloud servers, while "infrastructure controllers" are distributed throughout the network and typically run on network infrastructure devices, e.g., base stations, RSUs, access points, etc.

Each control plane scenario in Figure 1 is described below.
Scenario 1: When vehicles do not have stable connectivity to the network infrastructure, one of them can be chosen as the controller and is responsible for controlling the communication flow amongst vehicles in its cohort. This is known as "vehicle platooning".
Scenario 2: In some cases, one of the vehicles in the platoon, e.g., the one that has better/more stable connectivity to the infrastructure, can act as the controller for the platoon while that vehicle, itself, is controlled by a controller hosted by the network infrastructure (e.g., a nearby RSU).
Scenario 3: Vehicles can also be controlled by a controller hosted by the network infrastructure, e.g., an RSU in the area. This would typically be the case when vehicles are enjoying robust connectivity to the local network infrastructure. This can also occur when none of the vehicles is able or willing to perform controller functions.
Scenario 4: When no infrastructure is available (e.g., in sparse infrastructure deployments and/or they are under heavy load), vehicles can be directly controlled by a "root controller", i.e., a controller hosted at a cloud server. This is assuming vehicles will have stable connectivity to the cloud.

D4's control plane "continuum" is flexible and dynamic and is based on a control hierarchy as exemplified in Figure 2. It can adjust the control plane hierarchy's "depth" depending on the QoS requirements of the driving application as well as current network conditions. The control hierarchy illustrated in Figure 2 has up to 3 levels. Under some conditions, vehicles connect directly to controllers hosted by RSUs which in turn are connected through a backhaul network to a controller hosted in a base station. Depending on network connectivity, vehicles can be controlled by a vehicle controller, which in turn can be controlled by an RSU controller. D4 gets its name from the fact that D4 distributes the network control among different controllers, and the administration of the controllers can be decentralized among different entities, e.g., car manufacturers, cellular communication providers, transportation authorities, etc. Note that throughout the paper, we have used the terms decentralized and distributed interchangeably.

In the remainder of this section, we describe the D4 architecture in detail. We start by introducing D4's components and then describe D4's data- and control planes.

3.1 D4 Components

The different logical components of the D4 architecture and their descriptions are listed below:

- **D4-switch:** In D4, D4-switch work as sources and/or destinations of network traffic; they can also serve as relays and forward data plane as well as control plane traffic. Throughout this paper, we will use the D4-switch and switch interchangeably.
- **D4-controller:** The D4-controller manages D4's control plane. It decides how data plane flows will be carried by the D4-switches. To this end, it configures forwarding rules based on the QoS requirements of

Table 1: Related Work: Comparison between Prior Art and D4

Area	Prior Art	D4
Standalone versus Connected Vehicles	(1) SAE's and NHTSA's focus on standardizing standalone systems; (2) [41] highlight downside of standalone systems and point to connected vehicles as solution; (3) The US DoT[4] and ETSI[5] in Europe working towards building connected systems for ITS.	D4 promotes connected systems for enhanced coordination between vehicles and infrastructure.
Vehicular Cloud Model	[17] proposes a vehicular cloud for enabling autonomous driving. Assumes availability of nearby infrastructure.	(1) D4 introduces a dynamic "network control plane continuum" accommodating changing road and network conditions as well as application requirements. In D4, controllers can be spawned on demand as the network environment and requirements change; (2) The work reported in [43] supports D4's hierarchical control plane approach.
Control Delegation	(1) [3, 8, 22, 28, 31] discuss the extensions and optimization made in layers 2 and 3 to improve the handover performance; (2) [42] uses RSU virtualization to guarantee seamless delegation; (3) In the context of IEEE 802.11 networks, [42] proposes two handover decision algorithms, namely PRAHA (PRoactive Ap load driven Handover Algorithm) and ADNA (proactive AP/score Driven haNdover), which consider multiple metrics including RSSI (Received Signal Strength Indicator) and traffic load. One of the main shortcomings of this work is that it uses a centralized approach, which is not compatible with VANET's scalability, mobility, geographic distribution, administrative decentralization, and diverse quality-of-service requirements	D4 proposes a flexible approach to control delegation that adapts to changing network conditions and application needs. It introduces the concept of dynamically spawned controllers and localized decision-making to improve scalability, mobility support, and QoS adaptability
Edge Computing in Vehicular Networks	(1) The work reported in [27] aims to minimize service interruption during handovers, which is a critical issue for vehicular networks due to their high mobility and the need for uninterrupted connectivity. (2) Both [34] and [33] propose machine learning-based resource management and network slicing approaches to meet the quality of service requirements of vehicular networks.	D4 extends the concept of control delegation to edge computing based on a hierarchical control plane architecture and on-demand controller spawning and delegation to adapt to diverse scenarios.

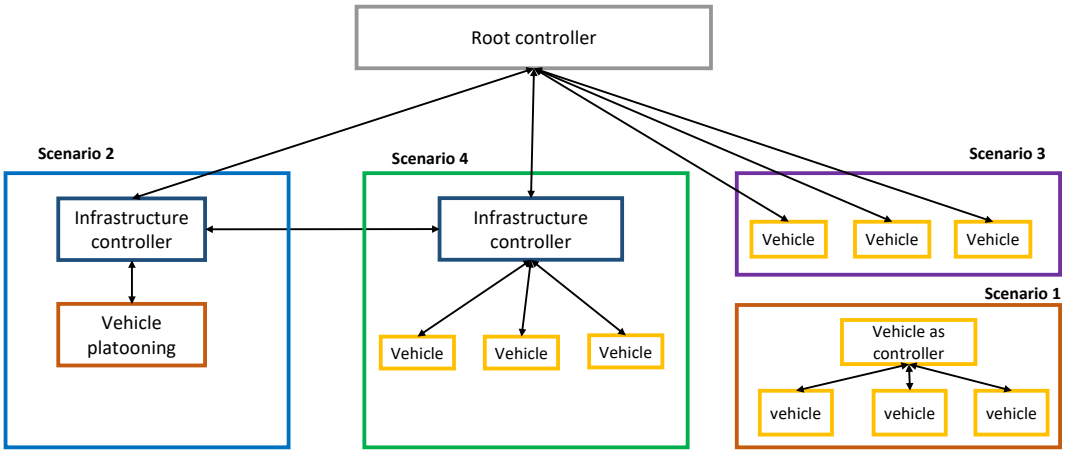


Figure 1: D4 Control Plane Scenarios

D4's data plane messages. The D4-controller also acts as a "bridge" between the driving application and the D4-switches. In the case of Connected Vehicles, the application generates driving-related information for the vehicles and the D4-controllers decide to which vehicles driving information should be sent and how to send/route them. Consider for example an application that provides real-time road conditions and traffic information. An instance of such an application would display a local map showing road conditions and traffic details. Vehicles could then periodically request road and traffic condition updates. However, this approach would not scale and could strain the map service and cause congestion in the underlying network due to the volume of requests. A more effective approach would be to have the map application proactively send updates to relevant vehicles whenever conditions change in their vicinity. For this, the map service needs to know every vehicle's location in order to dispatch relevant information. D4's control hierarchy greatly facilitates acquiring that knowledge by having maps communicate with the root controller. In this scenario, instead of vehicles directly communicating with the application, D4 controllers can consolidate information and transmit only pertinent data required by vehicles, considering factors such as their respective locations.

D4's logical components can be hosted by a variety of devices. Below, we list some examples relevant to Connected Vehicles:

- **Vehicles** typically act as mobile D4-switch. As such, they can generate as well as relay D4 data plane messages. When acting as a D4-switch, vehicles host a network switch that is controlled and configured by a controller. Vehicles may also perform controller functionality. In this case, vehicles can generate and forward data plane as well perform control plane functions. While most vehicles are capable of carrying out control plane functionality, we envision that they will assume the role of a controller in special circumstances, such as when controllers hosted by network infrastructure devices are not available (e.g., because of sporadic connectivity, sparse infrastructure deployment, etc.).
- **Road Side Units (RSUs)** [6] are network infrastructure devices mounted along a road or pedestrian passageway. RSUs may act as D4-switch or D4-controllers. Like vehicles, when acting as D4-switches, RSUs perform network switch functions. As controllers, RSUs perform both control- and switch functionality.
- **Base Stations** managed by cellular providers can also perform D4 controller functions. Following D4's hierarchical architecture described in detail below, we envision that a controller running on a base station will typically oversee controllers running on RSUs in the Base Station's range.
- **Cloud Servers:** As illustrated in Figure 1, root controllers can be hosted by cloud servers and may communicate with a number of infrastructure controllers deployed throughout the network. Note that, as shown in Figure 1 Scenario 3, vehicles could be controlled directly by the root controller. However, longer communication delays and/or disruptions when communicating with the cloud are likely to be more frequent compared to when the controller is "nearby". This is especially important for delay-intolerant edge applications, such as Connected Vehicles. Note that D4's control hierarchy is flexible and does not require root controllers which can be instantiated on-demand per

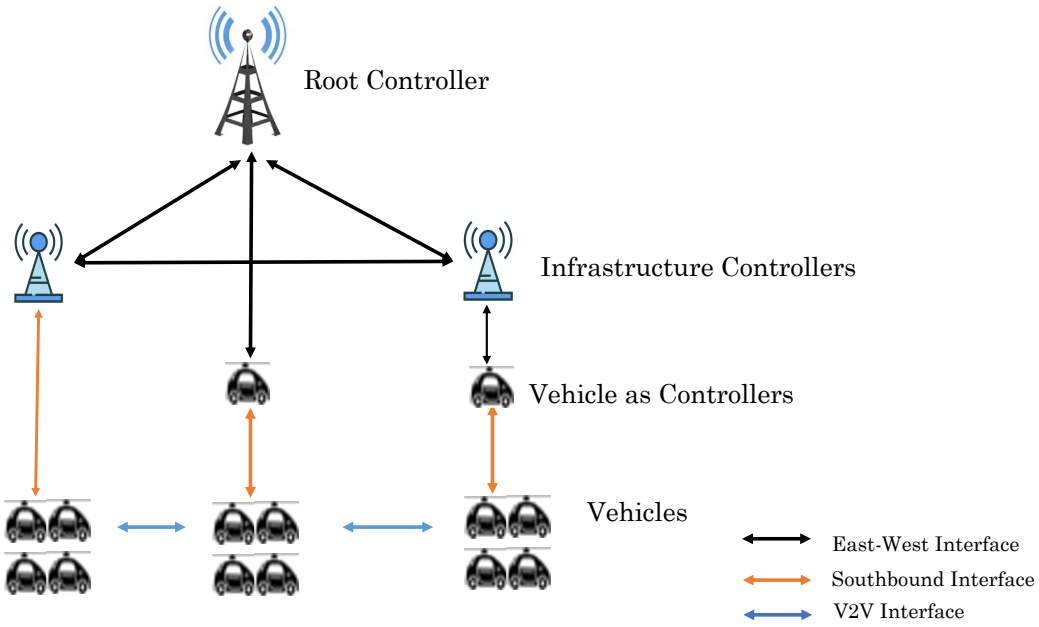


Figure 2: Example D4 Control Plane Hierarchy

the needs of the application and/or conditions of the underlying communication infrastructure. For connected systems such as Connected Vehicles, we believe that the presence of a root controller could be beneficial. For example, a navigation service running as an application on a cloud server can send updates to the root controller via the northbound interface. The root controller can then route these notifications to the appropriate infrastructure controllers or vehicles based on their location, avoiding the need to update all vehicles and reducing the number of messages sent in the network. Additionally, driving coordination, particularly under hazardous road conditions (e.g., snow or rain), can be carried out through V2V and V2I communication with nearby vehicles and infrastructure, respectively. However, applications running on the infrastructure would need to be aware of the RSUs or secondary controllers present in the network. The root controller can provide such information, given its high availability, reliability, as well as the fact that it is a trusted component of the system.

D4 components are agnostic of the communication technology used by the underlying network infrastructure and can use whichever communication technologies are available, including Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), Infrastructure-to-Vehicle (I2V), Infrastructure-to-Infrastructure (I2I) protocols, or a combination thereof.

3.2 D4 Data Plane

Since it is decoupled from the control plane, D4 data plane focuses solely on forwarding application-generated messages using "rules" installed in forwarding devices by controllers. In the case of Connected Vehicles, data plane messages are exchanged between vehicles and controllers, between controllers, and between vehicles (see Figure 2). They may carry self-driving information, as well as road and traffic conditions (e.g., accidents and road congestion), and navigation information. D4 data plane can also carry user traffic (e.g., video, audio, text) generated by or destined for users inside vehicles, etc. This could happen when, for some reason, user cellular connectivity is disrupted. The D4 data plane will forward application messages based on the current rules established by the control plane.

3.3 D4 Control Plane

Control plane messages carry signaling information, including controller-to-controller and vehicle-to-controller connection establishment, topology discovery, and vehicle and controller health checks. Control plane messages are also exchanged for security purposes, more specifically for controller authentication. As illustrated in Figure 2, control plane communication happens through the *East-West Interface*. Controllers use the equivalent to SDN's *Southbound Interface* to communicate with forwarding devices (e.g., vehicles, RSUs). D4 control plane messages are described below.

Through the experiments we conduct to evaluate D4, we exemplify how D4 control plane messages are used in Section 6, where we also discuss the overhead they incur.

3.3.1. Control Plane Messages

- **East-West Interface** messages are exchanged by controllers and include:
  - **Status** messages are either generated periodically or triggered by messages generated by the driving application after the controller receives and processes them. For example, a controller can periodically send information regarding its current load (e.g., the number of vehicles it controls, its CPU load) or performance/health (e.g., network errors, packet losses, battery status, etc). Status messages act as beacons or “keep-alives" among neighboring controllers. They can also include information such as controller location, speed, signal strength, etc. Depending upon the driving application(s), Status messages can also be triggered by application-generated messages. The "RoadWork Warning" message described in ETSI’s specification [14] is an example of application-generated message. When this message is received by a controller, depending upon the location where the road work is being done, the controller will forward the information to impacted neighboring controllers via a Status message.
  - **Authentication** Controller authentication is part of East-West communication. In D4’s current implementation, controller authentication uses a private-public key mechanism in which each controller possesses a unique key pair, with the public keys registered with the root controller as the trusted authority. Authentication messages are encrypted using the AES-128 (Advanced Encryption Standard) algorithm [11]. This ensures that only trusted controllers with the correct private key, who can decrypt the Authentication message, are added to the network. It is important to note that the authentication process leverages an enhanced version of the methodology suggested in [7], enabling controllers to authenticate using the P4 protocol. Even though not implemented currently, a similar approach can be used to authenticate vehicles by the controllers.
- **Southbound Interface** messages are exchanged between controllers and forwarding devices and include:
  - **Beacon** controllers broadcast Beacon messages periodically. Beacons contain information about the controller, including IP address, port number, current load, etc. Once a forwarding device receives a Beacon from a controller, it calculates the corresponding RSSI value.
  - **Keep Alive** messages are periodic control plane messages sent by forwarding devices to their controller and carry information like node health, network conditions, location, mobility, RSSI, and load.
  - **Authorize Delegation** message is sent by the forwarding device to the controller when the forwarding device decides to do the delegation to another controller.
  - **Control Delegation** messages are generated by a controller and sent to forwarding devices or neighboring controllers when the controller decides to delegate the control to a neighboring controller. It contains information such as the target controller ID (e.g., IP address, port) required by the vehicles or controllers to identify the target controller and the port on which connection should be established.

3.3.2. Control Delegation

Control delegation is at the core of the D4 architecture to ensure that the network provides continuous service while satisfying application quality-of-service requirements when subject to conditions such as unpredictable mobility patterns of fast-moving vehicles, communication channel impairments, current network infrastructure conditions (e.g., CPU load, available storage, battery levels, etc.).

In D4, the control delegation process can be divided into 3 phases:

- **Controller discovery:** As described above, controllers broadcast Beacon messages periodically. Beacons contain information about the controller, including IP address, port number, current load, etc. Once a forwarding device receives a Beacon message from a controller, it calculates the corresponding RSSI value and propagates it to the controller with which it is currently associated through a Keep Alive message. The controller or the forwarding device can use the information in the Keep Alive message to decide whether a delegation should be initiated. We discuss controller-initiated and forwarding device-initiated delegation below.
- **Controller selection:** Using the information in Keep Alive messages, different criteria (e.g., RSSI value,

controller load, etc.) can be used to select the delegation's target controller. Once the associated controller selects the new target controller, either the controller or the forwarding device can initiate the delegation as explained below. In D4's current implementation, controller selection is performed by the current controller without consulting with the target controller. In future D4 implementations, we plan to add support for mutual agreement between the involved controllers in the selection phase before the association.

- **Association:** In D4, once the target controller for the delegation is selected, the forwarding device proceeds with the association. Additionally, the current controller can send current information it has about the forwarding device to the next controller. This helps reduce or eliminate the overhead associated with the authentication process between the forwarding device and the target controller.

3.3.3. Types of Delegation

The delegation currently supported by D4 is based on who initiates the delegation process. Figure 3 illustrates the message flow for controller-initiated and forwarding device-initiated delegations.

Controller-initiated delegation: As explained in Section 3.3, during the controller discovery phase, the current controller gathers information received from the forwarding device through Keep Alive messages and from neighboring controllers through Status messages. Based on this information, the current controller may select a new controller for its currently associated device(s). For example, depending on its current load and the load information received in Status messages (see Section 3.3.1 for more details) from neighboring controllers, the current controller can decide to delegate one or more forwarding devices that it currently controls (e.g., it can choose vehicles currently generating high traffic load) to a neighboring controller, which is carried out in the controller selection phase. The association to the selected target controller is done by the forwarding devices selected by the associated controller during the association phase. In its current implementation, D4 supports control delegation based on two criteria, namely: (1) the device's quality of service as reported by the measured RSSI and (2) the load currently experienced by the controller as measured by the number of associated devices. In future work, we plan to consider additional delegation criteria that can be adjusted dynamically based on the environment and application requirements driven by machine learning algorithms as demonstrated in [40].

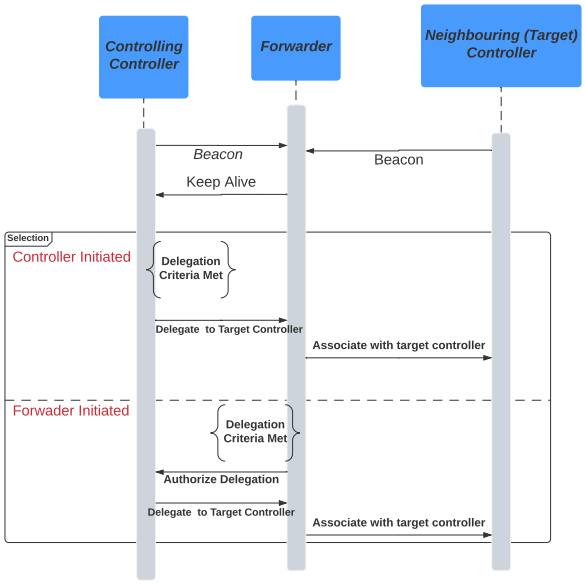


Figure 3: Delegation message exchange

Forwarding device-initiated delegation: Based on the information received by the forwarding device in the controller discovery phase in the broadcast Beacon messages from the nearby controllers, the forwarding device itself decides to initiate the delegation to a neighboring controller. For example, depending upon the signal strength or transmission errors between the device and the current controller, the device may decide to associate itself with a neighboring controller with better signal quality. However, the forwarding devices cannot initiate the associated until the associated controller approves the request from the forwarding device. This is done to make sure the delegation requests are only sent by the authorized forwarding device in the network. Any request from the unauthorized forwarding device is ignored by the controller.

The D4 architecture supports both types of delegation since, depending on the scenarios and applications, controller-based delegation may perform better than vehicle-based delegation and vice-versa. For example, controllers get information about neighboring controllers, e.g., their CPU load, number of connected vehicles, network queuing statistics, etc., from Status messages (refer to Section 3.3) exchanged between controllers. Note that the best candidate controller from the vehicle's perspective based on the RSSI may not be ideal if the controller is overloaded, which can be measured by its CPU and network load, the number of vehicles currently associated, etc. In MEC environments, the decision on the best compute location is typically based on several factors, such as the user's location, network conditions, application requirements, and available computing resources. For

example, applications that require high computation power may need to be processed on a more powerful computing resource. Similarly, applications that require real-time processing may need to be processed on a computing resource with low latency. To support low latency, network and edge computes need to be aware of the respective conditions. A D4 controller has the capability to facilitate MEC use cases by enabling the exchange of information with an edge computing system. This exchange of data allows the controller to make informed decisions regarding the delegation or handover of vehicles based on network conditions and the availability and performance of the computing resources. In other words, the D4 controller can assess the current state of the network and computing capabilities and determine whether it is necessary to delegate control of a vehicle to the edge computing system for optimal performance and efficiency. This functionality underscores the importance of a reliable and efficient D4 controller in enabling advanced vehicle control and management in modern transportation systems. Alternatively, edge computation capacity can be sent to vehicles directly and let vehicles make the handover decision, but this could lead to more network traffic as vehicles are likely to be more numerous than controllers.

4 USING D4 TO ENABLE ETSI'S ROAD HAZARD WARNING

The European Telecommunications Standards Institute (ETSI) has defined the standard for Intelligent transport system (ITS), which aims to provide services relating to different modes of transport and traffic management, enable users to be better informed, and make safer, more coordinated, and 'smarter' use of transport networks. To demonstrate the advantages of using the D4 architecture for self-driving vehicular networks, we use ETSI's Road Hazard Warning (RHW) [12, 13] as the driving application in our current D4 prototype and experiments. RHW, which is an event-based road message dissemination service specified by ETSI, defines a variety of events, including traffic jams, hazardous conditions, and messages that will be generated to alert road users (e.g., cars, trucks) of such events.[6]

4.1 RHW Messages

RHW's Decentralized Environmental Notification Messages (DENMs) [13] are mainly used to provide the necessary alerts in the case of emergency situations (e.g., imminent risk of collision) or warnings, e.g., in events like road congestion. As such, DENMs should be conveyed by the underlying communication infrastructure and delivered to road users in the geographic area affected by the event, also known as "relevance area" [13].

RHW's Cooperative Awareness Messages (CAMs) [12], on the other hand, carry signaling information. CAMs' contents vary according to the type of RHW service. They may carry information about vehicle location, vehicle type, as well as time of day, vehicle speed, etc. The types of messages supported in our current D4 implementation follow the ETSI's ITS specifications [12, 13] for RHW services and include both DENMs and CAMs.

- **Decentralized Environmental Notification Messages (DENMs)** are forwarded by the D4 data plane and can be of the following types:
  - **Emergency** messages are time-sensitive data plane messages sent by vehicles or road infrastructure devices (e.g., RSUs) to convey information about nearby accidents, imminent risks, etc. As such, Emergency messages have the highest delivery priority and are flooded to vehicles in the affected region. In D4's current implementation, we opted to follow ETSI's "awareness range" criteria, i.e., Emergency messages are delivered to vehicles in the neighborhood using a time-to-live approach. This approach is similar to 3GPP's broadcast-based V2V communication [1]. However, in D4, broadcast messages can travel multiple hops to support, for instance, the vehicle-platooning scenario described in Figure 1. Note that D4 can be adjusted to account for different message delivery criteria.
  - **Warning** messages are also treated as D4 data plane messages. They are sent by vehicles to report abnormal situations on the road such as road congestion or traffic jams. Like Emergency messages, Warning messages are also disseminated within the affected area. Depending upon the type of warning and the extension of the area it affects, Warning messages can also be delivered more broadly. This message delivery mechanism is similar to the approach employed by 3GPP [1] that uses infrastructure devices such as RSUs to augment the delivery of messages by V2V where the destination vehicle is out of communication range of the initiating vehicle.
- **Cooperative Awareness Messages (CAMs)** are D4 data plane messages that are periodically sent by vehicles. They typically carry information such as the vehicle's current location, speed, etc. Controllers use the information in CAM messages to compute relevance areas, compute and install routes in vehicles, etc. CAM messages can trigger controllers to

[6] More information on ETSI Intelligent Transport Systems standards is available at http://www.etsi.org/technologies-clusters/technologies/intelligent-transport.

generate D4 control plane messages like the Status message discussed in subsection 3.3.1

5 D4 IMPLEMENTATION

The basic building blocks for our D4 implementation are the SDN controller and the software switch (e.g., *Open vSwitch (OVS)* [29] and P4 [5]). As we describe in detail in this section, some of our contributions include the design and implementation of the East-West interface for controller-to-controller communication, as well as the extension of the Southbound interface used for controller-to-switch communication. The latter includes new message types required by RHW services and to support delegation of control.

As our experimental platform, we used Mininet-WiFi [16], a fork of the Mininet SDN network emulator [25], and supports both SDN controllers and software switches. To simulate vehicle mobility, we used the SUMO urban mobility simulator [23], which allows us to use real road maps. We built our maps using Open Street Map (OSM)[7], a free street-level map of the world created by an ever-growing community of mappers. Although SUMO does not support network infrastructure components such as Base Stations, RSUs, and controllers, we were able to add them to our experiments using Mininet-WiFi. In our experiments, SUMO was used to simulate mobility on maps generated by OSM, and Mininet-WiFi was used to emulate network communication.

Figure 4 illustrates the components of the current D4 implementation and are described below.

- **D4-Controllers** run our modified version of the SDN controller. We developed a module that constantly monitors the status of the network, checks the number of controlled vehicles, and updates D4-switch rules using the Southbound interface. In order to collect information about current network conditions to make control delegation decisions, controllers use the Mininet-WiFi API to query the underlying LTE and WiFi networks as well as vehicles to gather connectivity, channel, and link latency information. Controllers store information about vehicles controlled by them and neighboring controllers in a hashtable. Currently, we use Python's default implementation of hashtables [8]. In our current implementation, the keys to these hashtables are the vehicles' and controllers' MAC addresses. In our future work, we plan to use distributed and highly scalable in-memory databases such as Redis [9].
- **D4-Switches** work as sources and/or destinations of network traffic as explained in Section 3.1; they can also serve as relays and forward data plane- as well as control plane traffic. Vehicles that do not have controller capabilities only act as switches. RSUs that do not host a controller also act as plain switches. Mininet-WiFi supports P4's implementation of the OpenfFlow Switch [10]. P4 (Programming Protocol-Independent Packet Processors) is a programming language for defining the packet processing behavior of network devices, including switches and routers. P4 switches are often used in conjunction with software-defined networking (SDN) controllers, which provide centralized management and control for the underlying communication network. Detailed implementation details can be found in our open-source repository.[3]

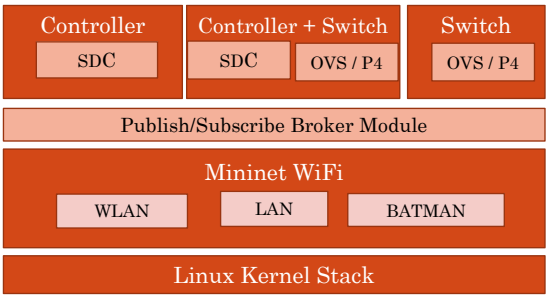


Figure 4: The D4 prototype.

- **D4-Controller + D4-Switch** is a node that supports both switch and controller functionality. These can be hosted by RSUs and vehicles.
- **Publish/Subscribe (Pub/Sub) Broker** As shown in Figure 4, in our current D4 implementation, participating devices (switches and controllers) use the Pub/Sub Broker to publish and receive both data and control plane messages. We use a pub/sub-based communication approach for scalability, flexibility, and to facilitate group communication. For instance, instead of maintaining individual connections between nodes, they communicate through the Pub/Sub Broker. Currently, for simplicity and ease of implementation, we assume that the Pub/Sub Broker is administered by the same authority as the other network entities, which may not be the case in real-world scenarios. And, depending on the experimental platform, this assumption can be relaxed as it is not a limitation of the D4 architecture. For our experiments, we use Mosquitto[11], which is an open-source implementation of the message broker that implements the

[7] https://www.openstreetmap.org
[8] https://docs.python.org/3/tutorial/datastructures.html#dictionaries
[9] https://redis.io/
[10] https://github.com/p4lang/switch/blob/master/p4src/switch.p4
[11] https://mosquitto.org/

MQTT protocol. MQTT is one of the oldest M2M communication protocols, which was introduced in 1999. It is a publish/subscribe messaging protocol in which the MQTT client publishes messages to an MQTT broker subscribed by other clients. Every message is published to an address known as a "topic". Clients can subscribe to multiple topics and receive every message published to each topic. MQTT uses TCP as its transport protocol and TLS/SSL for security. Thus, communication between clients and the broker is connection-oriented. Another salient feature of MQTT is its three levels of Quality of Service (QoS) for reliable delivery of messages [4].

- **Mininet-WiFi** D4 is designed to be agnostic of the communication technologies used by the underlying network infrastructure. Since our experiments ran on the Mininet-WiFi emulation platform, they used Mininet-WiFi supported network protocol stacks e.g., WiFi and IEEE 802.11p. For our experiments, we extended Mininet-WiFi by adding virtualized WiFi stations and access points, which act as RSUs in our experimental scenarios. Mininet-WiFi also allowed us to emulate the backhaul network as well as use the BATMAN (Better Approach To Mobile Ad-hoc Networking) advanced protocol (often referred to as batman-adv) [26], an open-source proactive (or on-demand) L2 routing protocol. In addition to enabling communication at the data link layer, batman-adv also provides native routing support. The current implementation of D4 relies on BATMAN's data link layer communication functionality to enable V2V communication. These features of Mininet-WiFi allowed us to demonstrate vehicle-to-vehicle (V2V) communication capabilities in D4 and to compare D4 against traditional vehicular networks or VANETs (see Section 6.2.3). Additionally, to allow experiments to be performed in more realistic scenarios, we implemented the IEEE 802.11p protocol stack in the Linux kernel of Mininet-Wifi.

6 D4 EVALUATION

We conducted a wide range of experiments that showcase D4's decentralized network control plane to support ITS applications for connected vehicles. In particular, our experiments aim at demonstrating the benefits of using a decentralized control plane approach to support ETSI's Road Hazard Warning services (described in Section 4).

We created two different sets of experiments to validate D4. The goal of the first set of experiments (see Section 6.2) is to evaluate the performance of D4's control delegation while the second set demonstrates how D4 can support real-world scenarios, in particular, decentralized smart road traffic re-routing (see Section 6.3). The focus of the latter set of experiments is to showcase, from the point of view of real-world applications, the benefits of using D4's decentralized network control architecture when compared to traditional centralized approaches. Specifically, we use a real-world scenario where D4 outperforms centralized architectures leveraging its superior latency performance as demonstrated in our prior work [20, 21].

As discussed in Section 5, our experiments were performed using the Mininet-WiFi network emulator [16] and the SUMO urban mobility simulator [23] running on a Linux Ubuntu machine. Our experimental setup, which is summarized in Table 2, was used to demonstrate four different scenarios, namely: (a) RSSI-based delegation; (b) Load-balancing-based delegation; (c) Comparing D4 with VANETs by leveraging vehicle-to-vehicle (V2V) communication in the first set of experiments, and (d) Road traffic re-routing in the second set of experiments. In our experiments, we emulated the area near 23rd Street and Madison Square Park, in New York City, USA, using SUMO. Vehicles travel along the same trajectory at ±40km/h until T = 250s. However, they can cover different distances since traffic lights may slow some of them down. The trajectory of the vehicles was created with *netedit*, a visual road network editor for SUMO's road traffic simulation. The wireless channel propagation model used in our experimentation was log-distance path loss, which is available in the current Mininet-WiFi distribution. In the V2V communication experiments, we used BATMAN (`batman-adv`) [26] as the V2V layer-2 protocol. More details about our experiments can be found in our open source github repository [3].

Table 2: Experimental Setup

Network Emulator	Mininet-WiFi 2.4.3
Vehicular Simulator	SUMO 1.6
Wireless Technologies	IEEE 802.11ac
MANET Routing Protocol	batman-adv
RSU	OVS/P4 Switches
Propagation Model	Log-distance path loss
Packet crafting	Scapy
Authentication	WPA2
MQTT Broker	Mosquitto
MQTT Python client library	Eclipse Paho
Operating System	Ubuntu 20.04
Linux Kernel version	5.8.0-050800-generic

The scenario used in delegation experiments is shown in Figure 5. It consists of ten vehicles and three RSUs, each of which hosts a D4 controller and a switch. Vehicles travel along the solid (yellow) line from point A to point D at ±40km/h.

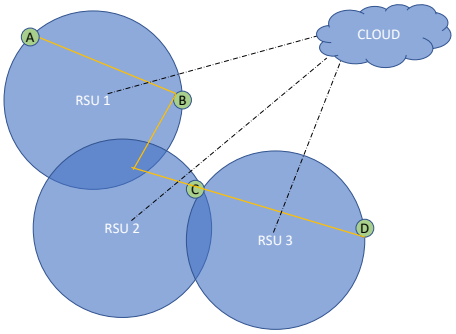


Figure 5: Experimental scenario and mobility pattern for the delegation experiments

In our experiments, we made the following assumptions:

- All network nodes including vehicles, RSUs, Base Stations, controllers, switches, and the Publish/Subscribe broker are under a single administrative authority. We should point out that the question of who administers the network control plane is orthogonal to D4 and needs to be addressed by stakeholders including transportation authorities, communication providers, car manufacturers, and government regulatory agencies.
- Since controllers are trusted entities, i.e. they are authenticated using Authentication messages (refer to Section 3.3), we assume that they cannot be compromised.
- For our network latency requirements, we use the ones set by ETSI [1].
- We assume that hosting controllers in vehicles will not have a significant computation impact and will not interfere with driving tasks.
- We assume that all D4 components remain functional for the duration of the experiments.

All the messages exchanged in the following experiments are described in Section 3.3)

6.1 Evaluation metrics

In our experiments we used the following performance metrics: **End-to-end latency:** End-to-end latency is measured as the round trip time it takes for data messages to be delivered from the source node to the cloud. **Packet loss:** In addition to end-to-end latency, for the delegation experiments, we also report the percentage of packet loss observed during the experiment. **Time to destination:** For the decentralized smart road traffic re-routing experiments, we measure the time taken by each vehicle to reach the destination when D4 is used and under a centralized network control approach.

6.2 Delegation Experiments

The primary goal of the delegation experiments is to evaluate D4's control delegation mechanism to respond to vehicle mobility and show that it yields adequate performance in terms of packet loss and latency. To illustrate that, in each scenario, vehicles move according to the mobility pattern (solid line from point A to D, passing through B and C) shown in Figure 5. Note that when vehicles are near points B, C and D, they are only "covered" by one controller, and thus do not have delegation choices. In each experiment, vehicles are generating RHW information or user-generated messages. The different types of delegation experiments we conducted as well as their results are described below in Sections 6.2.1, 6.2.2, and 6.2.3.

6.2.1 *RSSI-based delegation.*

In this set of experiments, we demonstrate controller-initiated delegation based on RSSI as described in Section 3-3.2.
Experimental setup: We use Mininet-WiFi's RSSI values which is calculated based on the log-distance path loss. Mininet-WiFi's RSSI calculations can be found in the Mininet-WiFi open-source github repository [12]. We consider that all vehicles are sending user-generated traffic (in this case ICMP ping messages) to the cloud and show the performance experienced by `veh1`. Figure 6 illustrates the scenario used in the experiment which follows the steps below:

(1) `Controller1` hosted by `RSU1` receives Keep Alive messages (described in Section 3-3.1) sent periodically by vehicles. `veh1` also sends "user-generated" traffic (in the form of ICMP ping messages) every 10 seconds to the cloud;
(2) At around $t \approx 105$s into the experiment, `veh1` reaches the edge of `Controller1`'s range and gets closer to `Controller2` which is hosted by `RSU2`. Around this time, `veh1`'s *Keep Alive* messages report lower RSSI values;
(3) Based on a pre-configured RSSI threshold (-80dBm in our experiments), the current controller, in this case, `Controller1`, selects a target controller for `veh1`. `Controller1` then sends the control delegation message to `veh1`;
(4) Upon receiving the delegation message, `veh1` initiates the delegation procedure with the target controller, in this case `Controller2`. Note that, since this is a controller-initiated delegation, the delegation process is initiated by the controller but connection establishment is performed by the vehicle. Another delegation event happens around $t \approx 170$s when `veh1` reaches

[12] https://github.com/intrig-unicamp/mininet-wifi/blob/master/mn_wifi/propagationModels.py#L100

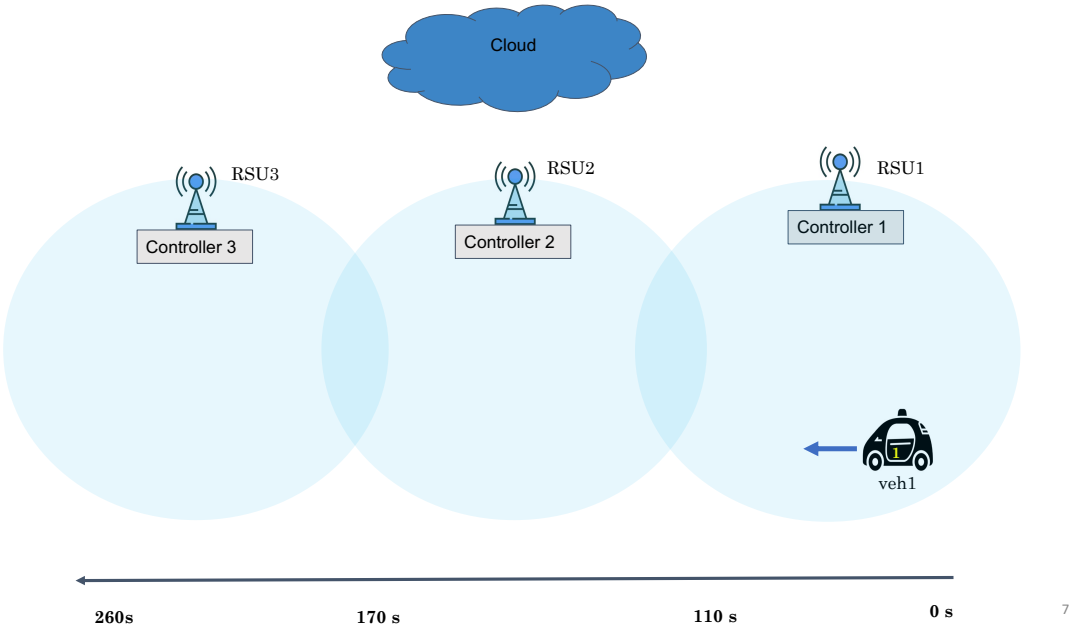


Figure 6: RSSI-based delegation scenario

the edge of `Controller2`'s range and gets closer to `Controller3`.

if *distributed_control* **then**
  if $(RSSI_{Vehicle} < RSSI_{Threshold})$ **then**
    do_handover(*TargetController*);

Algorithm 1: RSSI-based delegation algorithm.

The message exchange that takes place as part of the RSSI-based delegation decision is summarized in Figure 3 and the RSSI-based delegation algorithm is described in Algorithm 1.

In our current experiments, we opt for a −80dBm RSSI threshold, because our empirical observations indicate that this threshold value would still be an acceptable received signal strength. Lowering the threshold limit causes packet loss. Moreover, lowering the signal strength threshold has a negative impact on the data rate. This means that the user may want to consider increasing the threshold in order to have less impact on applications that require higher bandwidth.

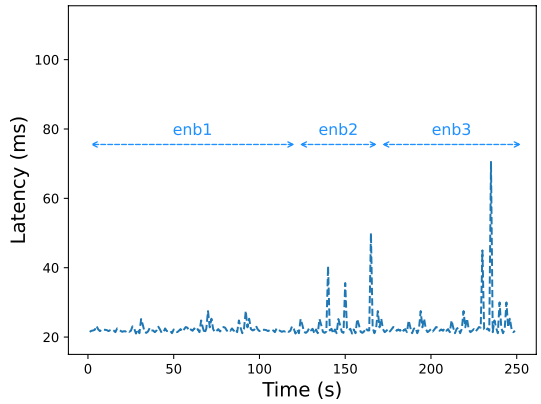
Results:

As described above `veh1` periodically sends user-generated traffic in the form of ICMP ping messages to a server in the cloud. While we only show latency results for `veh1`, we observe that the other vehicles in the experiment experience similar latency performance.

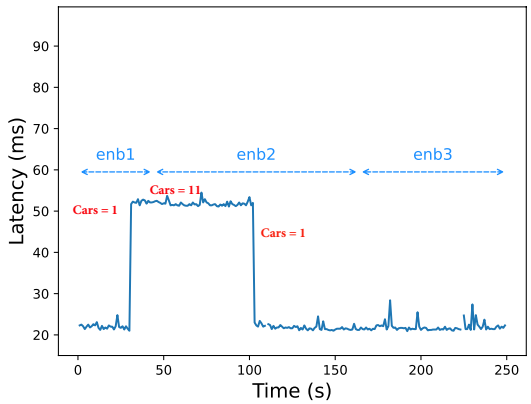
Figure 7a shows the latency experienced by `veh1` (`y-axis`) over the duration of the experiment (`x-axis`) demonstrating that D4's distributed network control yields low latency ($20 - 25ms$) with *zero* packet loss.[13] Note that, at around $t = 100s$, D4 is able to detect when `veh1` gets closer to the edge of `Controller1`'s coverage and it proactively initiates control delegation in order to maintain adequate quality-of-service. Since the delegation between `Controller1` and `Controller2` is done proactively, `veh1` does not experience a significant latency increase while it is being delegated from one controller to the other. This is an improvement over a centralized control architecture which, as demonstrated in our prior work ([20, 21]), reported considerably higher latency (of around $115-120ms$). As discussed in Section 1, these higher latencies do not meet the 3GPP requirements for high-priority V2X messages (e.g., pre-crash sensing) [1]. Note that, in our prior work the delegation process had to be initiated and conducted manually at fixed time intervals as the controller did not include automatic delegation mechanisms. In our current approach, delegation happens automatically in these experiments based on the RSSI.

According to the latency results in Figure 7a, at times $t = 70s, 150s$, and later around $220s$, we observe short-lived latency spikes. Those can be explained by the trajectory

[13] We did not observe packet losses in these experiments. As such, we discuss packet loss performance in more detail in the context of the experiments comparing D4's decentralized network control approach against traditional VANETs.



(a) End-to-end latency for D4's RSSI-based delegation



(b) End-to-end latency for D4's load-balancing based delegation

Figure 7: Results of delegation experiments

taken by veh1 as shown in Figure 5. At these times, veh1 is at the edge of the coverage area of the current controller, i.e., points B, and D in Figure 5, where there is no other neighboring controller to which the vehicle can be delegated. Note that in C, while there is RSU3/Controller3 in the neighborhood, it does not immediately meet the established minimum RSSI threshold criterion.

6.2.2 *Load-balancing based delegation.*

These experiments showcase D4's delegation in order to achieve controller load balancing.

Experimental setup: We consider that all vehicles are sending user-generated ICMP ping messages to the cloud and show the performance experienced by veh1. This vehicle, like the others, is traveling along the solid (yellow) line shown in Figure 5 from point A to point D. In this scenario, veh1 departs, then veh2, which departs before veh3, and so on. For this reason and due to the traffic lights along the way, veh1 distances itself from the other vehicles and will reach the area covered by Controller2 sooner than the other vehicles. Hence, once Controller1, which is hosted by RSU1, detects that the number of vehicles connected to it is higher than the set threshold (10 vehicles for our experiments), it checks if the neighboring controller, i.e., Controller2 has lower load, which is the case. Then, Controller1 initiates the delegation of veh1 to Controller2.

Figure 3 illustrates the message exchanged among the D4 entities participating in the delegation process. As previously noted, in our load balancing experiments, controller load is measured by the number of vehicles currently managed by a controller. As such, controllers periodically send the number of vehicles that they currently control to neighboring controllers using *Status* messages (see Section 3.3). Hence, when the current controller detects that a neighboring controller has fewer vehicles connected to it (compared to the current controller's load), it sends a delegation message through the Publish/Subscribe Broker to veh1.

The load-balancing based delegation algorithm is described in Algorithm 2. Note that in addition to load, controllers also consider vehicle perceived signal strength (RSSI) when making load balancing delegation decisions. This means that the candidate controller must also provide at least minimum RSSI levels. In our experiments, the minimum RSSI threshold $RSSI_{Min}$ is set to -80dBm. Other threshold values can be considered, e.g., in the case of applications that require higher-bandwidth connections.

if *distributed_control* **then**
  if $((Load_{Controller} > Load_{Threshold})$ & $(RSSI_{VehicleToTargetController} > RSSI_{Min}))$ **then**
    do_handover(*TargetController*);

Algorithm 2: Load-balancing based delegation algorithm.

Results

As described above, veh1 departs first at $t = 0$ followed by veh2, which in turn departs before veh3, and so on. At $t = 0$, veh1 is connected to Controller1. At around $t = 40s$, we spawn a total of 11 vehicles, all connected to RSU1. Since the controller load capacity threshold is set to 10 *vehicles*, Controller1 periodically checks if any vehicle is reporting RSSI measurements from neighboring controllers so that it can try to delegate the vehicle to shed some of its load. Around $t = 95s$, veh1 finds itself in the area covered by both RSU1 and RSU2. At this point, Controller1 requests veh1 to transition to Controller2. Figure 7b reports latency

measurements (y-axis) for veh1 over the duration of the experiment (x-axis) and shows that veh1 experiences high latency when its controller, in this case, RSU1, is under high load which in this experiment happens between ($t \approx$ 35s-110s). Then, once veh1 is delegated to RSU2, which does not currently control many vehicles, veh1's perceived latency drops down to the $20 - 25ms$ range again. In all, our results show that D4's distributed network control yields low latency ($20 - 25ms$) with *zero* packet loss.

6.2.3 *Comparing D4 with VANETs by leveraging V2V communication:*

The main goal of these experiments is to showcase that D4's control plane approach is able to mitigate connectivity disruptions by proactively assigning vehicles to relays before they lose connectivity to the infrastructure.

Data is then forwarded through relays using V2V links (e.g., using DSRC communication). As baseline, we compare D4 against a VANET-based control plane where vehicles, when loosing connectivity to the infrastructure, employ an ad-hoc routing protocol to discover routes and forward data through V2V links (e.g., using DSRC).

Experimental setup: Our experiments use the scenario depicted in Figure 8 and consist of a set of vehicles moving from one point to another, all following an identical trajectory one vehicle after another, i.e., veh1, veh2, and so on. In order to illustrate how vehicles can benefit from V2V communication in scenarios with sparsely deployed infrastructure, we use a single RSU (RSU1) which hosts a controller (Controller1). In all these experiments, veh1 sends user-generated traffic (ICMP packets) to a cloud.

In the D4 V2V-based delegation experiments, all vehicles were initially connected to (RSU1) where the local controller (Controller1) is hosted. After approximately 70 seconds into the experiment, one of the vehicles (veh1) departed from the RSU's coverage area. Instead of relying on the vehicle to autonomously detect the deteriorating RSSI signal as it exited the RSU's range, the D4 controller proactively intervened. It determined the most appropriate relay for veh1 and promptly dispatched a handover message to the vehicle, triggering the establishment of a V2V link with the selected relay (veh2). The D4 delegation algorithm is described in Algorithm 3 and works as follows: when the controller detects that the RSSI of a vehicle is below the minimum RSSI threshold, it tries to find another controller (as previously described in the RSSI-based delegation experiments). But since another controller meeting the minimum RSSI threshold cannot be found, the current controller sends a delegation message to veh1 to initiate a connection with a candidate relay, in this case veh2. Note that the signal strength of the target relay also needs to meet the minimum RSSI threshold requirement.

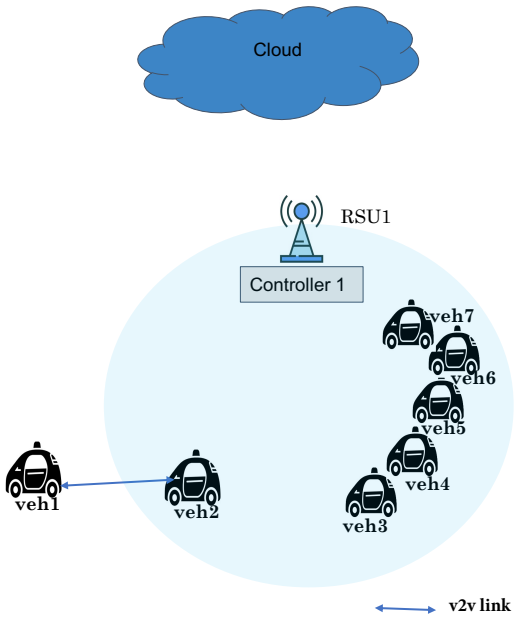


Figure 8: D4's V2V scenario.

The D4 V2V-based delegation experiments were conducted as follows:

(1) Initially, from $t \approx 0s - 70s$, Controller1 receives Keep Alive messages (refer Section 3.3) from all vehicles including veh1. veh1 is also sending passenger data traffic to the cloud (in the form of periodic ICMP ping messages) every 10s.
(2) At around $t \approx 62s$, veh1 reaches closer to the edge of Controller1's range. Controller1 detects the low RSSI signal for veh1 and Controller1 is aware that there are no neighboring controllers to handover, it then adds rules in veh2's switch such that it can serve as a relay for traffic from veh1. Note that Controller1 picks veh2 as a relay for veh1 because of its close proximity to veh1. The controller is aware of the proximity between veh1 and veh2 based on the information it receives from the Keep Alive messages from veh1 and veh2.
(3) From $t \approx 65s$ on, veh1's data traffic is relayed through veh2.

For the VANET experiments we used a similar experimental setup to the one shown in Figure 8, except that: (1) There is no controller co-located with RSU1; (2) Vehicles communicate directly with RSU1 as long as they are in the RSU's range; (3) All vehicles communicate with each other using V2V; and (4) Vehicles use the BATMAN ad-hoc routing [26] protocol to decide how to route traffic when they are not connected to RSU1.

```
if distributed_control then
    if ((RSSI_Vehicle < RSSI_Threshold) &
     (RSSI_Threshold > RSSI_Min)) then
        do_handover(TargetController);
    else if (RSSI_Vehicle < RSSI_Min) then
        if (RSSI_VehicleFromNeighController <
         RSSI_VehicleThroughAdhoc) then
            do_adhoc_handover(TargetVehicle);
```

Algorithm 3: V2V-based delegation algorithm.

The VANET experiments were conducted as follows:

(1) Similar to the D4 experiments, from $t \approx 0s - 70s$, veh1 sends periodic data traffic (using ICMP ping messages) to the cloud every 10s through RSU1;
(2) At $t \approx 70s$, veh1 reaches the edge of RSU1's range. There is a network disruption which causes veh1 to use the BATMAN routing protocol to find a suitable relay, in this case veh2, with which veh1 establishes a V2V connection.
(3) Then, from $t \approx 71s$ on, veh1 uses its V2V interface and sends messages to the cloud through veh2. Messages from *veh1* to veh2 are sent using the route established by the BATMAN ad-hoc routing protocol. veh2 then relays veh1 traffic to RSU1;

Results comparing the performance of the VANET scenario against D4 are reported below. To the best of our knowledge, our work is the first to compare the performance of a decentralized network control plane (D4) against a traditional ad-hoc network approach.

Results

Figure 9 reports latency measurements (y-axis) for veh1 over the duration of the experiment (x-axis) under D4 and VANET control planes. It shows that, D4 allows veh1 to keep communicating most of the time with some packet drops between $t = 62$ and $t = 65$. D4 is able to maintain veh1's connectivity mostly uninterrupted because the D4 controller quickly and proactively detects degradation in veh1's signal quality; and since there were no neighboring controllers to delegate veh1 to as it got closer to the edge of Controller1's coverage region, Controller1 instructs veh2 to act as a relay for veh1's traffic. Note that veh1's latency increases at around $t = 100s$ due to the fact that veh2 also gets near the edge of Controller1's range but still meets the minimum threshold value of $-80dBm$.

The setup for the VANET experiments is similar to the one explained above except that no controllers are used. In the VANET experiment, vehicles are also connected to the Internet through the RSU. When a vehicle looses connectivity with RSU, it activates the ad-hoc interface. Relays are selected by the BATMAN ad-hoc routing protocol based on its internal single hop list of all single hop neighbors it detects. In this experiment veh2 is selected as the relay since it is closest to the veh1.

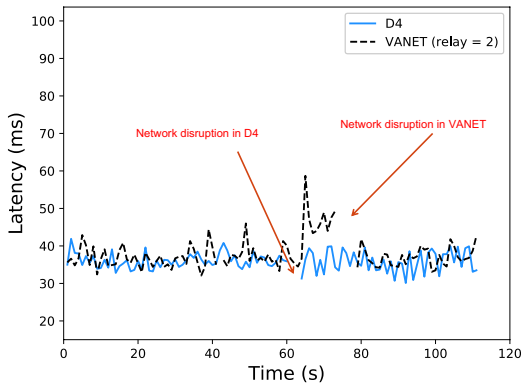


Figure 9: End-to-end message latency comparison between D4 and traditional VANET latency.

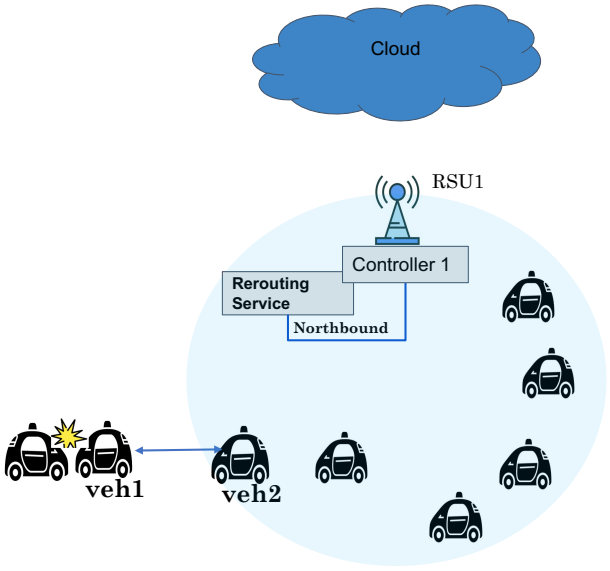
From Figure 9, we observe that between $\approx 73s$ and $80s$, we noticed a disruption in veh1's connectivity. This connectivity interruption is caused by the time it takes veh1 to realize that it lost its network connection and establish the link with veh2 through BATMAN using the V2V interface. While with D4 the controller proactively delegates the vehicle based on the RSSI, in the VANET experiments, the vehicle only takes action to find an alternate relay when it detects that it is no longer connected to the RSU. The latency spike at $t \approx 80s$ represents the moment when veh1 starts sending data traffic through the relay vehicle. When using D4, we observed a total of **3** ICMP packets dropped out of 111 packets sent, i.e., 2.7% drop rate, compared to the VANET scenario where there were **7** ICMP packets dropped, i.e., 6.3% drop rate with veh2 as relay.

6.3 Road traffic re-routing

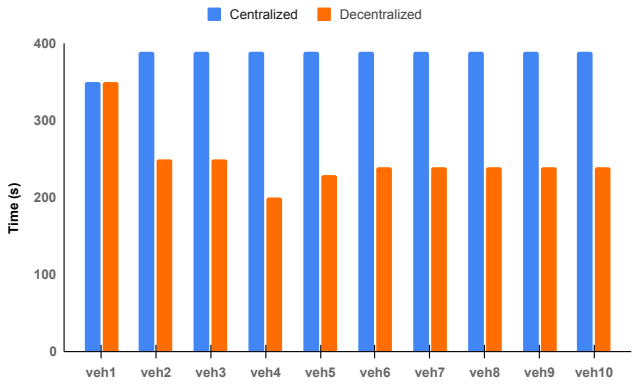
Experimental setup:

In this experiment, we demonstrate how D4 enables vehicle rerouting, for example, in the case of for road emergency situations such as accidents. In this scenario, as illustrated in Figure 10a, vehicle veh1 is involved in an accident and blocks the highway causing a major traffic jam. D4's decentralization enables timely detection and reaction to emergency situations, and, in this scenario, allows vehicles to use an alternate route to avoid traffic jams. The alternate route is pushed by a re-routing application through the D4 controllers' Northbound interface and, in turn, D4 controllers push it to vehicles. In our experiments, the re-routing application is hosted by RSU1 which hosts Controller1. As

a consequence, vehicles that use the alternate route are expected to reach their destination in less time when compared to vehicles that did not re-route. We ran the same experiment in a centralized environment as well, where, instead of having a local controller running the re-routing service, the traffic re-routing application runs on a centralized controller located in the cloud which connects to vehicles through an LTE eNodeB. In our experiments, the latency between eNodeB and vehicles was set to approximately 100*ms* which corresponds to the upper bound delay expected for LTE mobile backhaul [30]. We should point out that Mininet-WiFi does not yet support the LTE protocol, so, we artificially added an additional latency in the WiFi setup of Mininet-WiFi similar to what we did in our prior work [20].



(a) Smart traffic re-routing scenario.



(b) Time to reach the destination using D4 compared against traditional centralized control.

Figure 10: Road traffic re-routing scenario.

Figure 10a illustrates the re-routing experiment, which follows the steps below:

(1) `veh1` gets involved in an accident and wants to send out an Emergency message. However, it is only connected to another vehicle (`veh2`) via IEEE 802.11p through its ad-hoc interface. No controller is connected to `veh1` due to controller unavailability at that location. Thus, the Emergency message is sent to `veh2` so that it can forward the message to `Controller1` hosted by `RSU1`;
(2) `Controller1` receives the message and broadcasts it to connected vehicles using the switch hosted by `RSU1`. It also sends the message to the re-routing application connected through the controller's Northbound interface;
(3) The re-routing application calculates an alternate route and sends the information back to `Controller1`;
(4) `Controller1` receives the alternate route information and forwards it to vehicles under its control;
(5) The alternate route is finally received by vehicles who manage to avoid the congested route.

Results

Figure 10b compares the average time to destination (`y-axis`) over 10 different runs[14] for each vehicle (`x-axis`) under D4's decentralized control compared to traditional centralized control. We can note that under D4, the rerouting service allowed all the vehicles except `veh1`, which was the one involved in the accident, to reach the destination point earlier compared to the centralized control approach. Using D4, the re-routing service, which has access to a list of alternate routes, chooses the most appropriate one based on the current position of the vehicles, and is able to send re-routing information to vehicles in time for them to take an alternate route. However, due to higher end-to-end latency in the centralized experiment, alternate routes were not received by the vehicles in time, i.e., they had already passed the point where they could take the proposed new route. These results show the advantage of using control decentralization as it is able to reduce latency and response time which are critical for autonomous driving delay-sensitive applications and services.

In addition to end-to-end latency, for the delegation experiments we also report the percentage of packet loss observed during the experiment.

6.4 Complexity analysis

The table in Figure 11 summarizes D4's communication overhead as well as its time and space complexity. For each D4 component, namely the controller and the switch, we break

[14]Note that variance across different runs was negligible.

down the communication overhead and time-space complexity by tasks and events that trigger these tasks.

Communication Overhead: As detailed in Section 3.3, D4 controllers periodically communicate with switches to collect information about current conditions of the underlying communication infrastructure (e.g., RSSI) and location data for each vehicle via Keep Alive messages. Controllers also exchange information among themselves about their own conditions (e.g., their current load) via Status messages. As mentioned in Section 3.3, in our current implementation, Keep Alive and Status messages are sent periodically. Status messages are also triggered by events.

Time-Space Complexity: The D4 Controller has two primary tasks: switch rule creation and control delegation. Rule creation is driven by events like new switch association request, switch flow table miss and new switch delegation. The time complexity of adding rules is given by $O(R) \times O(1)$, where: R represents the number of rules to be created and $O(1)$ corresponds to the controller database lookup operation, as detailed in Section 5. Rule creation's space complexity is $O(V+C)$, where $(V+C)$ is the number of entries in the controller database. Each entry corresponds to either a vehicle (V) connected to the controller or a neighboring controller (C). In our current D4 implementation, control delegation is triggered when some specified condition is met, e.g., the RSSI or controller load reaches a certain threshold. The time complexity for control delegation is expressed as the product of the number of vehicles being delegated (V) and the constant time complexity associated with the controller database lookup operation ($O(1)$). The space complexity of control delegation is identical to switch rule creation as explained above. The D4 switch executes two main tasks: packet forwarding and rule installation. The switch forwards data packets according to its rule table. Packet forwarding's time complexity is thus given by O(H) * O(1) where H is the number of packet header fields being matched and O(1) is the constant time for flow table look up. The space complexity O(R) is bounded by the total number of rules across all of the switch's flow tables. The time complexity for rule installation is denoted as $O(R) \times O(1)$, where R is the number of rules installed (R) and ($O(1)$) is the time complexity associated with flow table insertion. The space complexity is O(R) similar to the space complexity of packet forwarding.

Discussion: Regarding D4's communication overhead, while control plane message exchanges incur additional overhead as shown in Figure 11, this aligns with the expectations outlined in the ETSI specification [15] which describes the exchange of status messages using Cooperative Awareness Message (CAMs) and recommends that CAMs be generated with a frequency between 1 and 10Hz. Note that there is a trade-off when setting the frequency of these messages - although sending them at lower frequencies reduces overhead, it may result in using outdated knowledge to make informed control delegation as well as data forwarding decisions. In future work, we plan to explore different techniques aiming at reducing D4's communication overhead. A notable example is to piggyback status information onto other messages, e.g., IEEE Beacons similar to work by Sascha et al. [32]. Another approach we are considering is to develop an adaptive mechanism that will allow D4 to dynamically adjust the periodicity of its Keep Alive and Status messages based on factors such as underlying network conditions, mobility patterns, etc. As far as D4's time-space complexity, we note that typically, D4 controllers as well as D4 switches will be hosted by network infrastructure devices such as base stations, access points and road-side units (see Figure 1 and Section 3.1) which are not usually resource constrained. D4 switches will also reside in vehicles and, as illustrated in Figure 1-Scenario 1, vehicles might also need to host D4 controllers. While current-, and even more so, future vehicles are usually equipped with enough computing and energy resources which will enable them to host and execute this additional functionality, we argue that by enabling vehicles to be connected to neighboring infrastructure and other vehicles, D4 will help improve driver and road safety which will outweigh the additional overhead and cost. Furthermore, D4-enabled connected vehicles will be able to save resources by offloading costly computation to neighboring infrastructure and vehicles.

Component	Task	Time Complexity	Space Complexity	Communication Overhead	Event Triggers
D4 controller	Switch rule creation	O(R) * O(1) R= Number of rules to be created. O(1) = Controller database look up (see Section 5).	O(V+C) V+C = Number of entries in the controller database. V = No of vehicles connected to the controller. C = Number of neighboring controllers connected to controller.		• New switch association request. • Switch flow table miss. • New switch delegation.
D4 controller	Control delegation	O(V) * O(1) V = Number of vehicles being delegated. O(1) = Controller database look up (see Section 5)	O(V+C) V+C = Number of entries in the controller database. V = No of vehicles connected to the controller. C = Number of neighboring controllers connected to controller.	• Status • Keep Alive • Control Delegation(see Section 3.3.1)	• Delegation threshold match (RSSI or controller load)
D4 switch	Packet Forwarding	O(H) * O(1) H = Number of packet header fields being matched. O(1) = Flow table lookup.	O(R) R = Total number of rules across all flow tables.		• Data plane packet
D4 switch	Rule Installation	O(R) * O(1) R = Number of rules installed. O(1) = Flow table insertion.	O(R) R = Total number of flow across all flow tables.		• New rule from controller.

Figure 11: Worse case time and space complexity and communication overhead.

7 CONCLUSION

In this paper, we propose the D4 decentralized and dynamically distributed network control plane framework which

aims at providing adequate communication support for connected autonomous vehicles. We describe the D4 decentralized control plane architecture, design, and prototype implementation which supports efficient dynamic control delegation. Our proof-of-concept experiments using a network emulation platform show that D4 yields lower message delivery latency with minimal additional overhead when compared to traditional centralized network control as well as ad-hoc VANET network architectures.

Motivated by our promising experimental results and by the fact that, as discussed in Section 4, some of D4's architectural features are also part of the 3GPP standard's release 17 (Rel-17) [1], in future work, we plan to deploy D4 in real-world testbeds such as the one used in [39], the Anthem testbed [15], or the California Connected Vehicle Testbed [16]. Other directions of future work include exploring different publish/subscribe approaches, for example, the work done by Wolf et al. [37] on a scalable event notification service. We also plan to investigate how D4 could benefit from programmable data planes. In particular, we will evaluate how D4 can leverage the work done by Jepsen et al. [18] on programmable data planes with support for the P4 protocol [5]. We will also explore how D4 can be extended to support offloading of autonomous driving tasks - we envision that tasks such as perception, path planning, and trajectory generation can be offloaded/shared with the infrastructure and/or neighboring vehicles. While the D4 architecture is agnostic of the communication technology employed by the underlying network, we plan to explore D4's performance when different network protocol stacks are used. We plan to compare our current results with other V2V protocols like C-V2X [38], which is designed to support both safety and non-safety applications.

REFERENCES

[1] 3GPP. 2022. *Service requirements for V2X services.* Technical Specification (TS) 22.185. 3rd Generation Partnership Project (3GPP). Version 17.0.0.

[2] M. Awais and M. A. Shah. 2017. Information-centric networking: A review on futuristic networks. In *2017 23rd International Conference on Automation and Computing (ICAC)*. 1–5. https://doi.org/10.23919/IConAC.2017.8082033

[3] Laurence Banda, Mjumo Mzyece, and Guillaume Nóel. 2013. Fast handover management in IP-based vehicular networks. In *2013 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 1279–1284.

[4] Soma Bandyopadhyay and Abhijan Bhattacharyya. 2013. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. In *2013 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 334–340.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[6] James Chang et al. 2017. An overview of USDOT connected vehicle roadside unit research activities. *JPO* (2017).

[7] Xiaoqi Chen. 2020. Implementing AES encryption on programmable switches via scrambled lookup tables. In *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 8–14.

[8] Sanjib Debnath and Abhishek Majumder. 2015. Enhanced MMIP6 for vehicular ad-hoc networks. In *2015 International Symposium on Advanced Computing and Communication (ISACC)*. IEEE, 303–309.

[9] Ramon dos Reis Fontes. 2019. mac80211_hwsim: add more 5GHz channels, 5/10 MHz support. https://github.com/torvalds/linux/commit/b5764696ac409523414f70421c13b7e7a9309454

[10] Ramon dos Reis Fontes. 2019. mac80211_hwsim: add support for OCB. OCB (Outside the Context of a BSS) interfaces are the implementation of 802.11p, support that. https://github.com/torvalds/linux/commit/7dfd8ac327301f302b03072066c66eb32578e940

[11] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. 2001. Advanced Encryption Standard (AES). https://doi.org/10.6028/NIST.FIPS.197

[12] ETSI. 2014. *Intelligent Transport Systems; Vehicular Communications; Basic Set of Applications; Specification of Cooperative Awareness Basic Service.* Technical Report EN 302 637-2 V1.3.1. ETSI.

[13] ETSI. 2014. *Intelligent Transport Systems; Vehicular Communications; Basic Set of Applications; Specifications of Decentralized Environmental Notification Basic Service.* Technical Report EN 302 637-3 V1.2.1. ETSI.

[14] ETSI. 2021. *Intelligent Transport Systems (ITS); Security; ITS communications security architecture and security management; Release 2.* Technical Report ETSI TS 102 940 V2.1.1 (2021 07). ETSI.

[15] ETSI. 2023. *Vehicular Communications; Basic Set of Applications; Cooperative Awareness Service; Release 2.* Technical Report ETSI TS 103 900 V2.1.1 (2023-11)). ETSI.

[16] Ramon R Fontes, Samira Afzal, Samuel HB Brito, Mateus AS Santos, and Christian Esteve Rothenberg. 2015. Mininet-WiFi: Emulating software-defined wireless networks. In *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, 384–389.

[17] M. Gerla, E. Lee, G. Pau, and U. Lee. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*. 241–246. https://doi.org/10.1109/WF-IoT.2014.6803166

[18] Theo Jepsen, Ali Fattaholmanan, Masoud Moshref, Nate Foster, Antonio Carzaniga, and Robert Soulé. 2020. Forwarding and Routing with Packet Subscriptions. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies* (Barcelona, Spain) *(CoNEXT '20)*. Association for Computing Machinery, New York, NY, USA, 282–294. https://doi.org/10.1145/3386367.3431315

[19] Murat Karakus and Arjan Durresi. 2016. A Survey: Control Plane Scalability Issues and Approaches in Software-Defined Networking (SDN). *Computer Networks* (2016).

[20] A. Kaul, K. Obraczka, M. A. S. Santos, C. E. Rothenberg, and T. Turletti. 2017. Dynamically distributed network control for message dissemination in ITS. In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. 1–9.

[21] Anuj Kaul, Li Xue, Katia Obraczka, Mateus AS Santos, and Thierry Turletti. 2018. Handover and load balancing for distributed network control: applications in ITS message dissemination. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–8.

[15] https://www.wsp.com/en-US/projects/anthem-connected-vehicle-testbed

[16] https://path.berkeley.edu/research/connected-and-automated-vehicles/california-connected-vehicle-testbed

[22] Mun-Suk Kim, SuKyoung Lee, and Nada Golmie. 2012. Enhanced fast handover for proxy mobile IPv6 in vehicular networks. *Wireless Networks* 18, 4 (2012), 401–411.

[23] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. 2012. Recent development and applications of SUMO-Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements* 5, 3&4 (2012).

[24] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2014), 14–76.

[25] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (Monterey, California) *(Hotnets-IX)*. Association for Computing Machinery, New York, NY, USA, Article 19, 6 pages. https://doi.org/10.1145/1868447.1868466

[26] S. Wunderlich M. Lindner, S. Eckelmann. 2020. The B.A.T.M.A.N. Project. http://www.open-mesh.org/

[27] Nirmin Monir, Maha M. Toraya, Andrei Vladyko, Ammar Muthanna, Mohamed A. Torad, Fathi E. Abd El-Samie, and Abdelhamied A. Ateya. 2022. Seamless Handover Scheme for MEC/SDN-Based Vehicular Networks. *Journal of Sensor and Actuator Networks* 11, 1 (2022). https://doi.org/10.3390/jsan11010009

[28] Qazi Bouland Mussabbir, Wenbing Yao, Zeyun Niu, and Xiaoming Fu. 2007. Optimized FMIPv6 using IEEE 802.21 MIH services in vehicular networks. *IEEE Transactions on Vehicular Technology* 56, 6 (2007), 3397–3407.

[29] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J. Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Jonathan Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Oakland, CA) *(NSDI'15)*. USENIX Association, Berkeley, CA, USA, 117–130.

[30] E. Piri and J. Pinola. 2016. Performance of LTE uplink for IoT backhaul. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*. 6–11. https://doi.org/10.1109/CCNC.2016.7444723

[31] Victor Sandonis, Maria Calderon, Ignacio Soto, and Carlos J Bernardos. 2013. Design and performance evaluation of a PMIPv6 solution for geonetworking-based VANETs. *Ad Hoc Networks* 11, 7 (2013), 2069–2082.

[32] Sascha Schnaufer, Stephan Kopf, Hendrik Lemelson, and Wolfgang Effelsberg. 2010. Beacon-based short message exchange in an inner city environment. In *2010 The 9th IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. 1–8. https://doi.org/10.1109/MEDHOCNET.2010.5546868

[33] Guolin Sun, Gordon Owusu Boateng, Daniel Ayepah-Mensah, Guisong Liu, and Jiang Wei. 2020. Autonomous Resource Slicing for Virtualized Vehicular Networks With D2D Communications Based on Deep Reinforcement Learning. *IEEE Systems Journal* 14, 4 (2020), 4694–4705. https://doi.org/10.1109/JSYST.2020.2982857

[34] Guolin Sun, Kai Liu, Gordon Owusu Boateng, Guisong Liu, and Wei Jiang. 2022. Intelligent Cruise Guidance and Vehicle Resource Management With Deep Reinforcement Learning. *IEEE Internet of Things Journal* 9, 5 (2022), 3574–3585. https://doi.org/10.1109/JIOT.2021.3098779

[35] Guodong Wang, Yanxiao Zhao, Jun Huang, and Wei Wang. 2017. The Controller Placement Problem in Software Defined Networking: A Survey. *IEEE Network* 31, 5 (2017), 21–27. https://doi.org/10.1109/MNET.2017.1600182

[36] Ziran Wang, Guoyuan Wu, and Matthew J. Barth. 2018. A Review on Cooperative Adaptive Cruise Control (CACC) Systems: Architectures, Controls, and Applications. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2884–2891. https://doi.org/10.1109/ITSC.2018.8569947

[37] Alexander Wolf, August Carzaniga, David Rosenblum, and Er Wolf. 1998. *Design of a Scalable Event Notification Service: Interface and Architecture*. Technical Report. Colorado Univ at Boulder Dept of Computer Science.

[38] Qiang Xu, Wei Liu, Ziyi Su, and Weibin Zhang. 2022. Review on the evolution process and key technologies of V2X-based protocol. In *2022 34th Chinese Control and Decision Conference (CCDC)*. 979–983. https://doi.org/10.1109/CCDC55256.2022.10033808

[39] Zhigang Xu, Xiaochi Li, Xiangmo Zhao, Michael Zhang, and Zhongren Wang. 2017. DSRC versus 4G-LTE for Connected Vehicle Applications: A Study on Field Experiments of Vehicular Communication Performance. *Journal of advanced transportation* 435 (08 2017). https://doi.org/10.1155/2017/2750452

[40] Tingting Yuan, Wilson da Rocha Neto, Christian Esteve Rothenberg, Katia Obraczka, Chadi Barakat, and Thierry Turletti. 2021. Dynamic Controller Assignment in Software Defined Internet of Vehicles Through Multi-Agent Deep Reinforcement Learning. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 585–596. https://doi.org/10.1109/TNSM.2020.3047765

[41] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. 2020. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* 8 (2020), 58443–58469. https://doi.org/10.1109/ACCESS.2020.2983149

[42] Ensar Zeljković, Johann M Marquez-Barja, Andreas Kassler, Roberto Riggio, and Steven Latré. 2018. Proactive access point driven handovers in IEEE 802.11 networks. In *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 261–267.

[43] Weihua Zhuang, Qiang Ye, Feng Lyu, Nan Cheng, and Ju Ren. 2020. SDN/NFV-Empowered Future IoV With Enhanced Communication, Computing, and Caching. *Proc. IEEE* 108, 2 (2020), 274–291. https://doi.org/10.1109/JPROC.2019.2951169