Improved Rule Installation for Real-time Query Service in Software-Defined Internet of Vehicles

Xin Wang, Cheng Wang, Junqi Zhang, *Senior Member, IEEE,* Mengchu Zhou, *Fellow, IEEE,* and Changjun Jiang

***Abstract*—Internet of Vehicles (IoV) has recently gained considerable attentions from both industry and research communities due to the development of communication technology and smart city. However, a proprietary and closed way of operating hardware in network equipment slows down the progress of new service deployment and extension in IoV. Moreover, the tightly coupled control and data planes in traditional networks significantly increase the complexity and cost of network management. By proposing a novel architecture, called Software-Defined Internet of Vehicles (SDIV), we adopt a software-defined network (SDN) architecture to address these problems by leveraging its separation of the control plane from the data one and a uniform way to configure heterogeneous switches. However, the characteristics of IoV introduce some great challenges in rule installation due to the limited size of flow tables at OpenFlow-enabled switches that are the main component of SDN. It is necessary to build compact flow tables for the scalability of IoV. Accordingly, we develop a novel rule installation mechanism to reduce the number of rules for real-time query services in SDIV. Specifically, we separate the wired data plane from wireless one and use multicast addresses in the latter. Furthermore, we introduce a destination-driven model in the wired data plane to reduce the number of rules at switches. Experiments with a real data trace show that the developed approach significantly reduces the number of rules without degrading the performance of data transmissions for real-time query services in IoV.**

***Index Terms*—Internet of Vehicles (IoV), Software Defined Network (SDN), real-time query service.**

Nomenclature

SDN	Software-Defined Network
IoV	Internet of Vehicles
SDIV	Software-Defined Internet of Vehicles
AP	Access Point
V	Vehicle
TCAM	Ternary Content-Addressable Memory
T	Timeout for rules at switches
D(n, d)	Euclidean distance between n and d
F_i	Data flow i
(S, D)	Packet with the source address S and the destination address D

I. Introduction

Internet of Vehicles (IoV) is attracting considerable attention from both academia and industry. The vigorous development of communication technology and smart city makes various services possible in IoV, which significantly improves the quality and safety of human driving. Research and industry communities are carrying out several projects [1], [2] for the development of IoV. For example, EU's CVIS [1] is committed to the development of the technologies needed to allow cars to communicate with each other and with the nearby roadside infrastructure. Smartway [2] focuses on integrating all intelligent transportation system functions to a uniform platform and provisioning services by two-way communications.

Though there is a promising future in IoV, the proprietary and closed way of operating devices in networks slows down the deployment and extension of new services in IoV. Network devices such as switches and routers are developed by different manufacturers. Their changes require substantial manual configuration by trained operators, which makes network management expensive and error-prone. The lack of an open and unified interface for flexible and dynamically customizable networks makes it difficult to deploy and extend new services in large-scale IoV. A new network architecture is expected for the development of IoV.

There are several projects for a new network architecture and next-generation Internet. Named Data Networking [3] aims to develop a new Internet architecture that concentrates on getting 'what' services rather than 'where' to get services. MobilityFirst [4] supports seamless and smooth mobility. It takes the mobility of nodes as a common case rather than a special case in traditional networks. NEBULA [5] develops a new network architecture based on cloud computing and data centers. Software-Defined Network (SDN) provides a unified interface to configure network equipment and separates a control plane from a data one. A unified interface of configuring network equipment makes a large-scale customizable network possible, and accelerates new service deployment in IoV.

We adopt SDN to support IoV for its open and unified interface, and propose Software-Defined Internet of Vehicles (SDIV), a new architecture for the development of IoV. SDIV has several advantages in supporting IoV besides the open and unified interface: (1) it essentially has high scalability by separating a data plane from a control one, (2) it achieves desired network manageability by using a logically centralized controller, and (3) its controller is able to choose the best path for data transmission and plays as a coordinator for roadside electronic devices according to the current network state. However, IoV characteristics introduce challenges in rule (match and corresponding action) installation due to the limited size of flow tables in its OpenFlow switches. It is thus necessary to build compact flow tables for scalability of IoV.

Xin Wang, Cheng Wang, Junqi Zhang, and Changjun Jiang are with the Department of Computer Science and Engineering, Tongji University, and with the Key Laboratory of Embedded System and Service Computing, Ministry of Education, China. (E-mail: 13xinwang@tongji.edu.cn, cwang@tongji.edu.cn, cjjiang@tongji.edu.cn.)

Mengchu Zhou is with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA. (E-mail: mengchu.zhou@njit.edu)

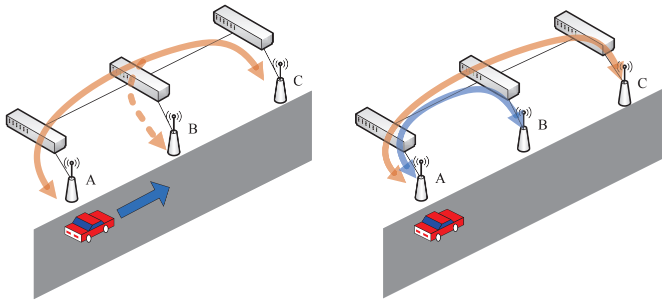


Fig. 1. (a): When a vehicle moves to another place (from A to B), it needs to establish a new connection. The dash line represents the connection which is not established yet; (b): When a vehicle is connecting to multiple cameras simultaneously, it needs to build a path for each connection which is not efficient since all paths have the same destination (i.e., all data flow's destination is A).

In this work, we consider real-time query services, that are typical in IoV, to show the design details for improved rule installation in SDIV. Real-time query services provide drivers the real-time road and other information and drivers decide which way to go according to such information. Road information coming from the surveillance cameras (or other kind of roadside electronic devices) is transmitted to every vehicle with demanding. In this scenario, the specific issues are: (1) the mobility of vehicles increases the number of rules since it needs to establish new connections between vehicles and surveillance cameras, and (2) when a vehicle is connecting to multiple cameras, it is not efficient to install rules for every path since it has the same destination. Fig. 1 illustrates the issues of rule installation in IoV. If the controller simply installs rules upon the request of drivers, the table size at switches may become the performance bottleneck of a real-time query service in IoV and even fail the service.

To address the rule installation and table size growth problems for real-time query services, we develop several techniques by leveraging the centralized and fine granularity of data flow control in SDN for improved rule installation: (1) To reduce the number of requests sent by vehicles, we use multicast addresses instead of general destination address as the last address when vehicles receive data from cameras; (2) To keep uninterrupted connection between vehicles and roadside electronic devices, we install rules in switches in advance based on the conditions of vehicles; and (3) To decrease the number of rules and maintain the correctness of data transmission, we modify the headers when packets come to branching nodes.

In this paper, we first describe the SDIV architecture. We then analyze the problem of flow table size growth in OpenFlow-supported switches. We design and develop an optimization approach to reduce the number of rules in switches. We validate the feasibility of improved rule installation by considering four situations in a real-time query service and analyze the details of data transmission in each case. Finally we use Floodlight [6] as a controller and Mininet [7] as a testbed to evaluate the performance of our improved rule installation method with a real data trace. Experimental results show that when using improved rule installation, the flow table in switches is more compact than simply installing rules without compromising the performance of data communication for real-time query services.

The rest of the paper is organized as follows. In Section II, we review related works. In Section III, we introduce the SDIV architecture. In Section IV, we discuss the improved rule installation issues and the proposed solution. We conduct extensive experiments and report results in Section V. We conclude the paper with future work in Section VI.

II. Related Work

Most applications and services in IoV, e.g., road security, fleet management, navigation, billing, and multimedia, rely on data transfers between vehicles and roadside infrastructure (V2I) and among vehicles (V2V) [8]. Several researches [9], [10] have demonstrated the feasibility of providing connectivity via road-side APs and the ubiquity of WiFi.

The studies [11], [12] deal with the efficient communication between APs and vehicles. For routing and data forwarding, [13]–[16] study the routing protocol and fast forwarding. Routing in IoV in different categories is well reviewed in [17]. To manage vehicular networks, a centralized policy framework is introduced and called Virtuoso in [18]. It manages spectrum resources while ensuring users to have suitable access to meet their communication needs. In [19], traffic flow prediction for traffic forecasting and congestion management is made. The work [20] estimates the attitude of a vehicle for low-cost inertial navigation system/GPS. Ahn et al. [21] present the Road Information Sharing Architecture (RISA), representing the first distributed approach to road condition detection and dissemination for vehicular networks. Calafate et al. [22] study reliable data delivery through broadcasting by computing the optimal packet size and determining the best Forward Error Correction (FEC) scheme.

To establish a new architecture in vehicular networks, Grassi et al. [23] apply Named Data Networking to networking vehicles and enable networking among all computing devices independent of whether they are connected through wired infrastructure, ad hoc, or intermittent Delay Tolerant Network. Yap et al. [24] present an OpenFlow [25] wireless network to achieve a free travel between any wireless infrastructures by separating the network service from the underlying physical infrastructure.

Software-defined network (SDN) is applied to improve network management in [26]. A distribution framework is proposed for decomposing large switch tables into small ones to handle limited-size switch tables [27], which reduces the flow table size. Voellmy et al. [28] introduce Maple to discover reusable forwarding decisions and reduce the number of rules by a trace tree structure that records the access to a specific packet.

From the above discussions, we can find that though there are many approaches for IoV development as well as various applications of the SDN concept, there is no or little work of the new service deployment and centralized management in IoV. Thus, we provide an SDIV architecture to do so.

III. Network Architecture: Software-Defined IoV

In this section, we propose an SDIV architecture and discuss its utility through a real-time query service in IoV. Before introducing it, we describe the basic SDN model and its main features.

A. Software Defined Networks (SDN)

The principal endeavors of SDN aim to separate the control plane from the data one and centralize a network's intelligence and state to a single device. SDN consists of two parts: (1) switches process flows' packets based on actions of flow entries (i.e. rules) in their flow tables, and (2) the controller generally runs on a remote commodity server and communicates over a secure connection with the switches by using a southbound interface.

Controller: A logical centralized controller connects to all the switches directly or indirectly in the network. It can have a global view (e.g. topology) of the whole network by collecting the state of switches. The controller uses a southbound interface to config the switch for adding or deleting rules.

Southbound Interface: The southbound interface is used by the controller to communicate with the switch. By this interface, the controller can query the state of switches (e.g. bandwidth of links) and also set rules to forward packets. OpenFlow, a protocol maintained by Open Networking Foundation [29], is viewed as a promising implementation of such an interaction. In this work, we use OpenFlow as the southbound interface between the controller and switches.

Rule: A rule is an entry in the switch's flow table. It consists of match fields, a counter, and a set of actions. When a packet matches a rule's match fields, it applies the corresponding actions and updates the counter. The actions include: (1) Forwarding the packet to particular ports; (2) Dropping the packet; (3) Forwarding the packet to the controller; and (4) Modifying the value of the packet header.

B. Software-Defined IoV (SDIV)

Our proposed SDIV network has a three-tier architecture. From bottom to top, it has physical, control and application layers, as illustrated in Fig. 2.

1) Physical Layer: Physical layer includes vehicles, access points (APs), roadside electronic devices, switches, and servers (different from an SDN controller). Vehicles act as mobile nodes and communicate with the server through road-side APs. When sensors in a vehicle collect conditions of the vehicle, e.g. speed, direction and location, the data should be transmitted to the server as soon as possible. Vehicles receive responses from the server through nearby APs. Note that we assume that the number of road-side APs is sufficient to cover every road. Road-side APs are static WiFi APs reachable from the road that offers the capability of data transmission to vehicles. Roadside electronic devices like surveillance cameras gather road conditions and also send data to servers or vehicles. Switches connect APs, surveillance cameras and servers. Servers here provide information service to vehicles, such as road conditions, based on the requests from vehicles and data collected from roadside electronic devices. One vehicle can connect to multiple servers at the same time.

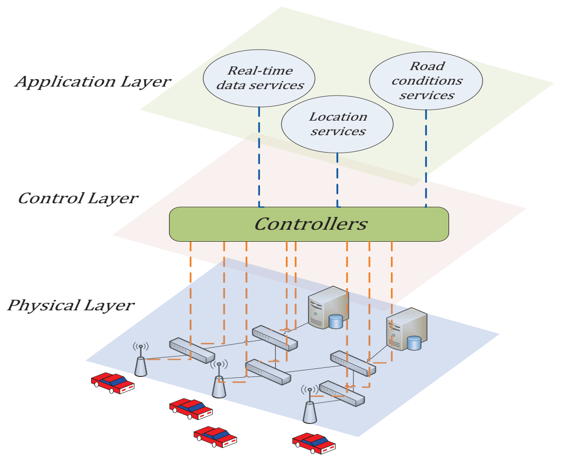


Fig. 2. Three-tier SDIV architecture with physical layer, control layer and application layer from bottom to top.

2) Control Layer: In the control layer, the controller connecting to every switch (including APs) acts as a mediator between upper layer applications and underlayer network through SDN. By using OpenFlow, it is able to control all data flows in IoV. Also when the network state changes, the switches (including APs) notify it. In SDIV, they act as a connector among vehicles, servers and other roadside electronic devices by forwarding data flows based on rules which are installed by the controller. Then the controller can convert the strategy of application, e.g., the path selection or access control, into the OpenFlow rules in particular switches, since it has an up-to-date, global view of the network topology and traffic flows. The global network view makes the controller available to provide a customized network state to applications, which is not possible in the traditional network. By this means, different applications have different views which benefit both applications and network. The former can have a much simpler view of the network than the detailed, but unnecessary view of the whole network, which can simplify their tasks, e.g., traffic scheduling. For the later, privacy and security benefit from the customized network state which guarantees the former to access their needed data only instead of all data.

3) Application Layer: The strategy of each application is defined in the application layer. Applications provide services for vehicles in IoV, such as a real-time query service, location service and road information service. Each application gets the customized network state from the control layer and makes decisions according to its strategy. The strategy here instructs how to provide services to vehicles. A real-time query service example is to provide drivers the real-time conditions of roads. The network topology is required to compute the path for improving performance, which we will discuss later in Section IV.

After introducing the SDIV components, we describe how SDIV works with two scenarios as depicted in Fig. 3.

Fig. 3(a) describes a typical scenario where a vehicle wants to receive the information about road conditions (maybe three blocks away from its current location) via surveillance cameras.

In Step 1, vehicle V_1 sends a request to a nearby road-side AP as shown in Fig. 3(a); in Step 2, since there is no rule

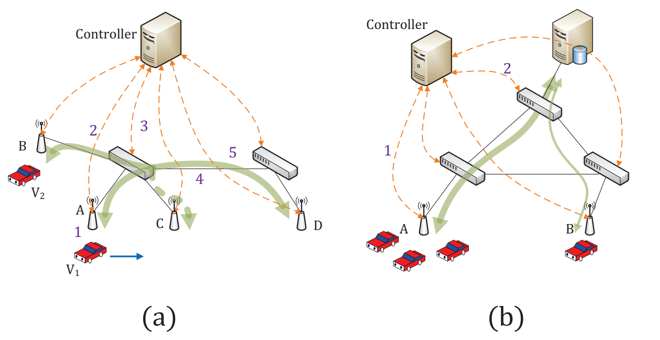


Fig. 3. (a): A scenario shows, by analyzing vehicle conditions, the controller can install rules (dash line) in advance to avoid extra queries; (b): A scenario of achieving intelligent bandwidth allocation based on a global view in the controller.

matching the header of the flow's first packet, the switch (AP A) encapsulates and forwards the packet to the controller; in Step 3, the controller recognizes the header and installs rules based on the vehicle's requests and information (e.g., location, speed and direction) and also the current network state; in Step 4, switches along the path towards the destination forward the matched packet to given ports based on the rules installed; in Step 5, the data flow from the surveillance camera (D) to the vehicle, goes through the same procedure as described in Steps 2-4. When the number of vehicles increases (V_2 at B appears), there needs a scalable approach to installing rules for data transmission. The conditions (e.g., directions) of vehicles should be considered for installing rules, e.g., installing rules at C in advance.

Fig. 3(b) describes a scenario where vehicles need to build connections with the server through nearby APs for data uploading such as their current speed and location. The upper layer application (different from the servers in the network) can benefit from the data uploading scenario for analyzing road conditions. If the controller receives many packets from a particular AP, it can tell that there are many vehicles in this area compared with other places receiving few packets only. In Step 1, the controller gathers the information from APs A and B. When switches receive packets with no rule matching, they send the packets to the controller. In Step 2, the controller installs rules in switches according to the destination addresses. If the upper layer application finds more vehicles in A than in B, it can allocate more bandwidth to A.

In addition, SDIV is more attractive when considering large-scale multicast and mobility of vehicles. As the number of vehicles connecting to one camera increases, it is easy to think of multicast for the efficiency of data transmission. Although dense-mode multicast (reasonable for cameras) goes well in small networks, as the range of data transmission increases, the growing number of (S, G) (S: Source IP, G: Group IP) entries kept in APs (or switches) and broadcasting messages periodically for establishing shortest path tree (SPT) limit the scalability of data transmission. The mixed model of control and data planes makes it necessary to keep (S, G) entries even not used currently at every router for a fast graft. In Fig. 3(a), it is necessary to reserve (S, G) entries at AP B for a new coming vehicle V_2. The mobility of vehicles also brings difficulty to traditional network technologies. Vehicle V_1 moving

TABLE I
PROS AND CONS OF TRADITIONAL NETWORK TECHNOLOGIES (MULTICAST) AND SDIV

Traditional Technologies	SDIV
Con: Broadcasting messages periodically	Pro: Reactive mode
Con: Keeping (S, G) entries at routers	Pro: Installing rules when needed
Con: SPT cannot match the driving path	Pro: Finding the path according to the direction of vehicles
Pro: Do not need a controller	Con: Need a controller

from A to C as described in Fig. 3(a) needs to send another request for retrieving data, which results in interruption of data stream and extra workload of the surveillance camera (D). In our design, we address the mobility problem by predicting the most possible path that the vehicles would choose and set the rules in flow tables in advance to multicast the packets from nearby APs along the path. Multicast along the data transfer path in SPT cannot address the mobility issue since the shortest path for data transmission seldom matches the driving path.

Table 1 summarizes the benefits of SDIV against traditional network technologies (multicast) besides its open interface of network equipment. Compared to the traditional multicast method that needs to broadcast messages periodically for establishing SPT, SDIV leverages the benefits of OpenFlow that makes switches forward packets with no matching rule in a flow table to the controller and then the controller installs rules for the packets. It is a reactive mode and rids the need to broadcast messages periodically. This reactive mode also makes it possible that the switches only need to keep the least number of rules in their flow tables instead of keeping (S, G) entries at routers in a traditional multicast method. With the support of an up-to-date, global view of the network topology and traffic states, the application in the controller is able to compute the most possible path that a vehicle would choose and installs rules in advance for providing persistent connection between vehicles and devices.

IV. Improved Rule Installation

Having described the SDIV architecture and how SDIV works, we give a solution for the rule installation problem and explain that it could bring significant complexity and overhead if naively installing rules are adopted for packet forwarding. We consider a real-time query service as the context to explain the necessity of its optimal solution.

In a real-time query service, if the controller simply installs rules for the requests from drivers, the size of flow tables will incur the performance bottleneck due to the limited size of ternary content-addressable memory (TCAM) in switches. Thus, it is important to produce compact tables to fit more cached policy decisions in the switches. For each flow established, two rules are needed at the selected switches for sending the request packets and receiving the packets from the destination. Even if we only consider the second data flow, since the request flow is only useful at the beginning and can be deleted via timeout mechanism in OpenFlow switches, we still need at least one rule for each flow. It sounds that one rule for each flow is compact enough for a flow table, but in reality drivers attempt to connect multiple surveillance cameras at the same time. As a result, we need to install rules for every such data flow at every selected switch even though these flows all have the same destination. To summarize, we have to reduce the number of packets forwarded to the controller and establish a compact flow table due to the limited size of TCAM at switches.

Fig. 4 depicts the basic scenario that how a real-time query service works. Suppose that V_1 wants to obtain the conditions of road (or crossroads) E. Here, the conditions can be a video or any other kinds of data stored in the roadside devices. At the first step, V_1 sends a request to AP A and then the data flow is forwarded to E. After confirming the request from V_1, E sends data to AP A. Finally, AP A transmits the data to V_1 and completes the data transmission. The entire process is simple, but the simplicity may introduce significant performance penalty. As shown in Fig. 4, consider the situation that there is another vehicle V_2 that wants to get the information of E. And also the V_2 is ahead of V_1. When V_2 follows the same procedure as V_1 does, there is another path from E to B established, which makes switch D install two rules following the same path. As the result, as the number of vehicles connecting to E increases, the flow table size at switches along the path may reach the maximum that in turn reduces the performance of the real-time query service. Moreover, when V_1 moves to another place, it needs to send a request again.

Next, we consider another example to show the importance of optimal rule installation. V_1 wants to get the data from E and F at the same time. Switches B and D need to install more rules to make data flow transmitted to A, even though both flows ($E \to A$ and $F \to A$) have the same destination. Such situations become more complex if V_3 requests the same service. To enable V_3 at C to receive data, there is a need of building a new path (i.e., the same data flow will split at the attaching switch of the three vehicles, then the switch becomes a branching node), which again increases the flow table size. The mobility of vehicles increases the complexity and degrades performance of the real-time query service as well.

To solve the rule installation problem and mitigate the flow table size growth, we propose to separate the wireless data plane (for communication between vehicles and APs) from the wired data plane (for communication among switches) and develop a destination-driven model (given later in Section IV-C) for the wired data plane. When vehicles receive data from nearby APs, they do not care how data transmitted in the wired data plane. Instead, they just want a persistent connection even their locations change over time. To make the separation efficiently, we introduce three techniques: (1) using a multicast address as the last hop address from cameras to vehicles, (2) installing rules in the most possible path in advance according to the conditions of vehicles, and (3) modifying the headers when packets come to branching nodes to be defined.

A. Multicast Address for the Last Hop

We utilize multicast addresses for the purpose to deal with the mobility and enhance the scalability of SDIV. Every

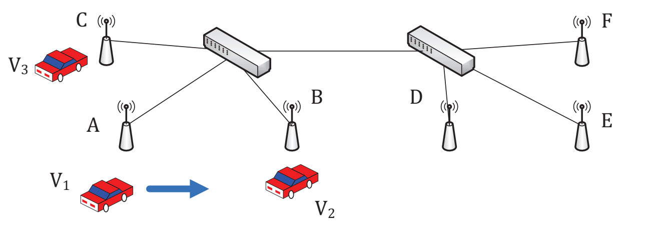


Fig. 4. V_1 is connecting to E and F simultaneously, and V_2 at the path of V_1 to E asks the data from E. V_3 also requires the data from E but at the different path.

surveillance camera has a unique multicast address which can be generated from a MAC address by the same method in traditional networks. Multicast addresses can be used in the last hop from cameras to vehicles for data transmission in the wireless data plane. Vehicle V_1 receives packets with the destination address of E's multicast address in Fig. 4. The advantage of using multicast addresses is apparent, i.e., every vehicle, within the range of an AP that has been equipped with the rule, can get the real-time data without sending new requests. Furthermore, in a real-time query service, the first packet of user requests has to be forwarded to the controller for installing rules. If every request needs to contact the controller for rule installation, it leads to a serious bottleneck at the controller since both computational capacity and bandwidth are limited. A multicast address can address this problem efficiently. After the first vehicle connected the camera with the intervention of the controller, the data flow starts to multicasts (along the path to the destination), and every new vehicle asking for the same camera receives the data without sending the request, which reduces the frequency of contacting the controller. For example in Fig. 4, if there is another vehicle that wants to see the conditions of E, it can receive the data without sending a request after V_1 sent the request.

Due to the limited size of TCAM in OpenFlow-supported switches, we need to remove the useless rules from flow tables. In our design, we use timeout in OpenFlow to delete rules for saving flow table resources. A switch maintains a per-flow-entry variable that indicates if there has ever been a period of T seconds in which no packet arrived for the flow. In practice, in OpenFlow, this is maintained by adding an "idle timeout" value to a flow entry. Instead of setting equal values of timeout, we choose different values of timeout in switches depending on the distance away from the destination. In reality, the number of vehicles interested in the destination gets larger as they are closer to the destination. Then we set larger values in timeout for the closer switches to the destination. As an example in Fig. 4, the timeout value T_i in switch i meets the constraint: $T_A < T_B < T_D < T_E$. The closer to the information source, the less changes on data flows, since these flows would merge to a stable singe flow. The stable data flow would require a long timeout than unstable flows.

B. Predicting a Path and Installing Rules in Advance

To deal with the mobility of vehicles in SDIV, it is necessary to predict the most possible path that a driver chooses and

Algorithm 1 PathFind(s, d, v)

```
1:  Algorithm PathFind($s, d, v$)
2:    put $s$ into set $N$, $R$;
3:    for each child node $c$ of $s$
4:        if the angle between $v$ and $A(s,c)$ is less than 90 degree then
5:          put $c$ into set $O$;
6:    Find($s, O, d$);
7:    return $R$;
8:  Procedure Find($p, O, d$)
9:    put $p$ into set $N$;
10:   for each $n$ in $O$
11:     if $n$ is $d$ then
12:       put $n$ into set $R$;
13:       finish;
14:     put $n$ into set $N$;
15:     remove $n$ from set $O$;
16:     calculate $D(n,d)$, $D(p,n)$;
17:     if $D(p,n) + D(n,d) < max$ then
18:       $max = D(p,n) + D(n,d)$;
19:       $r = n$;
20:   put $r$ into set $R$;
21:   for each child node $c$ of $r$
22:     if $c$ not in set $N$ then
23:       put $c$ into set $O$;
24:   Find($r, O, d$);
25:   return;
```

then install rules in advance along the path in order to keep data transmission uninterrupted while the vehicle is moving. A simple method to predict such a path is computing the shortest path, but it is not always a correct path in reality. We propose and describe $PathFind(s, d, v)$, in Algorithm 1 that finds the most likely path in a topology. Here s is the current location, d is the destination and v is the direction of the vehicle. $A(s, c)$ denotes the angle of s and c, $D(n, d)$ means Euclidean distance between n and d. Note that Manhattan distance may also be used to cope with an urban environment.

As the preparatory work (Lines 3-5), $PathFind$ filters the possible first node of a resulting path according to the condition (e.g., direction) of the vehicle, since the driver seldom turns back in a street. We apply $Find$ recursively to calculate the resulting path by choosing the node that has the minimum value of $D(p, n) + D(n, d)$, e.g., the distance between the chosen location and the destination as shown in Lines 17-20. In the current design, $D(n, d)$ denotes Euclidean distance between n and d, but we can change it to a more intelligent calculation. Finally, $Find$ identifies the destination as the next node and returns the result (Lines 12-14).

We apply $PathFind$ in Fig. 5. Consider a vehicle in location A plans to find a path to F. First, it puts B into set O according to the direction of the vehicle at the current location. It then finds D has a shorter path to F compared with C. It chooses D to start again, and finds D that directly connects to F and finishes. The result $A \rightarrow B \rightarrow D \rightarrow F$ is more reasonable than the shortest path $A \rightarrow E \rightarrow F$ since reversing a vehicle is rarely seen in reality (unless the distance difference

Algorithm 2 ModifyAddress($s, d, path$)

```
1:  Algorithm ModifyAddress($s, d, path$)
2:    for each node $n$ in $path$ do
3:      $nx$ = the next node of $n$;
4:      $setRule$ = false;
5:      for each rule $r$ in $n$ do
6:        $no$ = the node connecting the output port of $r$;
7:        if $r$ matching $s$ and $no$ != $nx$ then
8:            $act$ = forward to $nx$ and modify the destination address to $d$;
9:          $emitRule(matchFor(d), act)$;
10:         $setRule$ = true;
11:       else if $r$ matching $s$ and $no$ = $nx$ then
12:         $setRule$ = true;
13:     if $setRule$ == false then
14:       $act$ = forward to $nx$;
15:       $emitRule(matchFor(d), act)$;
16:   return;
```

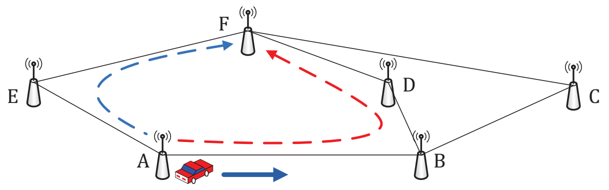


Fig. 5. The blue dash line is the shortest path between current location and destination, while the red dash line is a more likely path that the vehicle may choose according to the direction.

is quite large). Though the delay of path $A \rightarrow B \rightarrow D \rightarrow F$ is prolonged comparing to path $A \rightarrow E \rightarrow F$, the vehicle sends more requests when it finds no data from its nearby AP. In this case, the vehicle more likely chooses B as its next location, and thus the resulting path is better than the shortest path since the vehicle can receive data without sending another request. In Fig. 5, after installing rules at the switches along the predicted path ($A \rightarrow B \rightarrow D \rightarrow E$), the multicast address also makes V_2 receive data without sending a new request.

C. Modifying Address in Branching Nodes

In this subsection, we will show how to install rules at switches. The main idea is that we add a packet header modifying action at the branching nodes (Line 8-9) by leveraging the properties of OpenFlow as shown in Algorithm 2. The input s denotes the source of data, d denotes the current location of the vehicle and $path$ is computed by $PathFind$. When a packet matches a rule to modify the match field, it will first apply the modification before forwarding to a particular port, and this process is guaranteed by the OpenFlow protocol. The modified address makes the packet forwarded to the switch directly connecting to the vehicle. We clarify that the rule matching s means that the controller records the source address of every request, not the rule in the switch matches s. By using the address modification algorithm, we let packets with different source addresses but the same destination address match the same rule and hence reduce the number of rules and reduce the flow table size. We call this approach a destination-driven model, since it needs to match the destination address only.

As an example shown in Fig. 6, data flows F_1 and F_2 are both generated from surveillance camera S_1, and F_3 comes from S_2. F_1 is requested from V_1. F_2 and F_3 are destined to V_2. Suppose that V_1's current location is C and V_2 is at D, F_1 and F_2 need at least two rules at switches A and B in unicast, and F_2 and F_3 need two rules at least at switches A and B in multicast. This is not efficient since F_1 and F_2 come from the same node, and F_2 and F_3 have the same destination. We modify the destination address in the header of packets in switch B (as a branching node) for F_1 and F_2. By setting the destination address of packets in F_1 and F_2 ($F_1 \rightarrow C$ and $F_2 \rightarrow D$), the number of rules in switches is reduced and so is the size of the flow table for more services. Let's assume V_2 sends a request to S_1 in the beginning. Each switch only needs one rule to carry F_2. When V_2 asks data from S_2, the rules at switches still suit. Finally, when V_1 requires data from S_1, it just increases one rule at switch B via matching (S_1, D) with an action of modification. Here we use two parameters to represent the packet header, namely $(sourceaddress, destinationaddress)$, and so are the match conditions in the flow table. This destination-driven mode emphasizes the destination address as matching conditions in the flow table, and generalizes the data flow demands of the same destination for reducing the size of flow tables in switches. In Fig. 4, when V_3 joins, it needs address modification at the branching switch to optimize rule installation.

Next, we give an analysis of improved rule installation for two cases: (a) one server connects to multiple vehicles and (b) one vehicle connects to multiple servers. For the first one, the traditional rule installation that forwards packets to given ports simply based on the source address needs one rule at each switch along the data path. In our proposed rule installation method, we need one rule at each switch for forwarding and an extra action for modifying data header at the branch nodes. Since the modifying address action can be combined with the forwarding action in OpenFlow as a set of actions of rule, the number of rules does not change compared with the traditional rule installation. For the second one, we assume that there is one vehicle (v) connecting to multiple servers $(s_1, s_2, ...s_n)$, and s_i is h_i hops away from v. Let L_i denote a set of links used for the data flow between v and s_i. Hence, we have $|L_i| = h_i$. Then the traditional approach needs $\sum_{i=1}^{n} h_i$ rules, while by our improved rule installation, we need $|L|$ rules where $L = \bigcup(L_1, L_2, ...L_n)$. Considering one extreme scenario that any two L_i and L_j have no sharing link. Then we have $|L| = \sum_{i=1}^{n} h_i$. In this scenario, the number of rules needed by the proposed approach is the same as the traditional method. For many other scenarios, $|L| < \sum_{i=1}^{n} h_i$, which means that the use of the proposed one reduces the number of rules. In summary, the worst case of our improved rule installation including the case one and the extreme scenario of case two does not change the number of rules while the best case could reduce many rules when there is one vehicle connecting to multiple servers.

D. *Examples*

We give four examples to show how our approach works. We then describe the details of rule installation at switches.

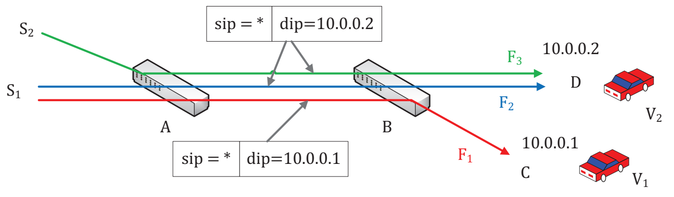


Fig. 6. Data flow F_1 (red line) comes from S_1 and its destination is C; Data flow F_2 (blue line) comes from S_1 and its destination is D; Data flow F_3 (green line) comes from S_2 and its destination is D. The matching conditions only depend on the destination address.

By analyzing the process of data transmission in a real-time query service, we summarize four patterns as shown in Fig. 7.

1) one-to-one case: In the simplest situation, there is only one vehicle asking for a real-time query service from one surveillance camera. As shown in Fig. 7(a), V sends a request to camera A and then receives data flow from A. The packets from A is (A, C), and matching conditions at B and C are both $(*, C)$. The packets multicast at C is $(*, A^{'})$ where $A^{'}$ is the multicast address of A. In this scenario, the path found by $PathFind(s, d, v)$ is $C \rightarrow B \rightarrow A$, which is the most likely path that the driver would choose. Then we install rules for multicast with $(*, A^{'})$ at B previously. When V arrives at B, it can receive data without interruption.

2) N-to-one case: When a camera connects to multiple vehicles, there is the need to modify the destination address at the branching nodes, e.g., B in Fig. 7(b). For OpenFlow switches, it supports a set of actions (e.g. packet header modification and forwarding action) at one rule. Hence the modifying action does not need an extra rule. At the beginning, A sends packets (A, C) since V_1 at C send the request earlier than V_2 at D. When V_2 sends a request, the controller (not shown in this figure) installs a new rule with a modification action which can be implemented in OpenFlow. As a result, switch B forwards two packets (A, C) and (A, D) to different ports. The data flows forwarded (by AP) at B, C and D are all $(*, A^{'})$.

3) one-to-N case: A vehicle needs to connect to multiple cameras simultaneously. As descried in Fig. 7(c), V wants to receive data from A and B at the same time. The packets generated from A and B are (A, D) and (B, D). At switches C and D, the matching conditions are both $(*, D)$. There only needs *one* rule to meet the requirements for different packets. When a packet (A, D) comes to C, the rule $(*, D)$ matches it and then forwards to D. Switch D follows the same procedure as C. C and D also need to change the destination address of their packets for multicast. The packets for multicast at C and D are $(*, A^{'})$ and $(*, B^{'})$, where $A^{'}$ and $B^{'}$ are the multicast address.

4) N-to-N case: We combine the *one-to-N* and *N-to-one* for a more common scenario *N-to-N* in Fig. 7(d). V_1 requires real-time data from A and B, and V_2 requires data from A only. There is a modification at C for packets matching $(A, *)$, since C is a branching node when data source A transfers packets to D and E at the same time. The packets from B are forwarded to the given port without modification.

The *N-to-N* scenario is a general case that gives details about how to implement improved rule installation. We ob-

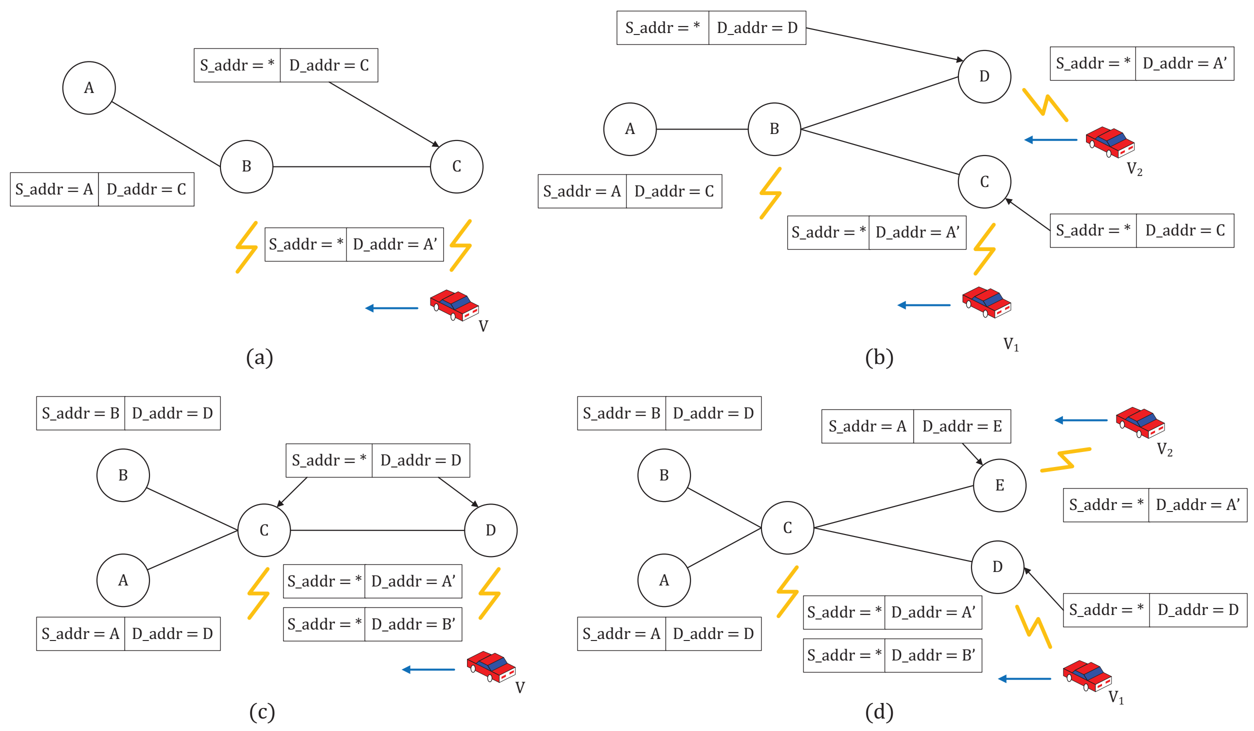


Fig. 7. (a): a *one-to-one* pattern that V wants to receive data from A; (b): a *N-to-one* pattern that V_1 and V_2 want to receive data from A; (c): a *one-to-N* pattern that V wants to receive data from A and B at the same time; (d): a *N-to-N* pattern that V_1 wants to receive data from A and B while V_2 wants to receive data from A.

serve that the theoretical upper bound of the table size of the proposed improved rule installation approach is related with the number of devices that the driver is connecting to at the same time. N_d denotes the number of devices, R_0 denotes the number of rules for the traditional approach and R denotes the number of rules for the proposed improved rule installation approach. We have $R = R_0/N_d$ as the upper bound of the table size. In this case, there are all devices connecting to the same switches.

V. Performance Evaluation

A. Testbed

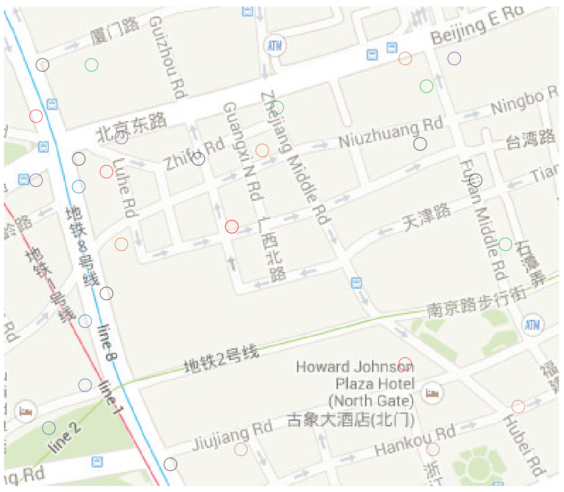
We use Floodlight [6] as the controller and Mininet [7] to build an SDN environment so as to evaluate the proposed improved rule installation approach in SDIV. We run Floodlight on a server, with 16 AMD Opteron(tm) processor 6172 and 16GB memory. Our server is installed with Linux kernel version 2.6.32. We run Mininet on a separate server. Servers are connected by a 10Gbps Ethernet network.

B. Effects of Improved Rule Installation

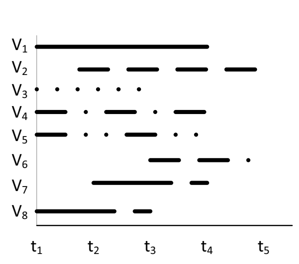
We use the real data of traveling traces in Shanghai city [30] as a common scenario to show the benefit of improved rule installation. Fig. 8(a) shows a snapshot around People's Square in Shanghai. Traveling traces of vehicles are composed of GPS data at different time. The number of vehicles is limited since we are using the real data in Shanghai city but it is enough to show the effects of the proposed approach. At time t_1, there are only five vehicles in this area. At t_2, two more vehicles join. At t_3, V_6 appears in the area. The appearing time and disappearing time of vehicles are illustrated in Fig. 8(b). Interval time between any two moments is 120 seconds. Since these vehicles cannot appear at the exact time, like t_1 and t_2 between which the interval time is 120 seconds, we make the vehicle appear at t_i if $|t_i - t_{real}| < 30$ where t_{real} is the real appearing time of the vehicle. Although these GPS marks at the same moment may not have exactly the same timestamp, one can say that these vehicles move to these locations very closely at that moment by limiting the difference between the timestamps of GPS marks and the time of moments to a certain range. In this case, we set it to 30 seconds to ensure that every GPS mark can reflect the real location at that moment.

Fig. 8(c) is a sketch map of Fig. 8(a). Arrows with different features illustrate the directions of vehicles according to the order of these timestamps in GPS marks as shown in Fig. 8(a). We assume that there is always one road-side AP (switches in Fig. 8(c)) around the GPS marks for data transmission available at any time. We make every vehicle in the sketch map have a data transfer demand at the first moment they appear. Therefore, V_1 and V_2 near the intersection of lines 8 and 2 want to receive data from both B and C, and also have the same path along line 8. V_3 locating at the convergence of two roads demands data from B and C. V_4-V_5-V_6 have the same target A but choose different paths. V_7 asks the data from D. V_8 has the target B only. All these vehicles require a persistent data flow service until they move to their destinations. We need to install rules at every necessary switches to meet their requirements.

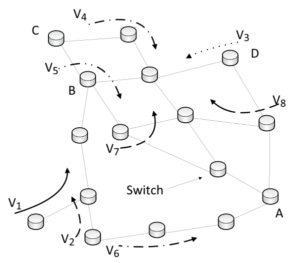
The baseline method with which we compare is a general method that forwards packets to given ports simply based on the source address. By improved rule installation, we merge



(a)



(b)



(c)

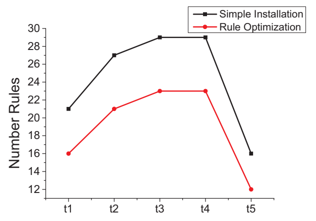
Fig. 8. (a): A snapshot around People's Square in Shanghai; (b): The appear and disappear time of vehicles; (c): A sketch map of (a) with the direction of vehicles.

the rules that have the same destination (destination-driven mode) in *one-to-N* pattern like V_3 connecting to B and C, and modify headers of packets at branching nodes in *N-to-one* pattern. The result is shown in Fig. 9(a). The decline at t_4 is due to the timeout in rules we set. Fig. 9(b) shows the number of vehicles at different moments. Fig. 9(c) shows the delay time of different vehicles in two approaches. The delay time we evaluate here is the longest one if a vehicle connects two cameras at the same time. We choose C instead of B for evaluating V_1's delay time.

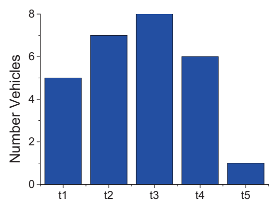
Also, we can observe from Fig. 9(c) that our improved rule installation approach hardly affects the performance of data transmission though it needs address modification in the header of packets. We reduce the number of rules and save the space at the flow table for manageability and scalability. Especially, from Fig. 9(b), the number of rules has reduced by 60% comparing with the baseline method.

C. Delay Analysis

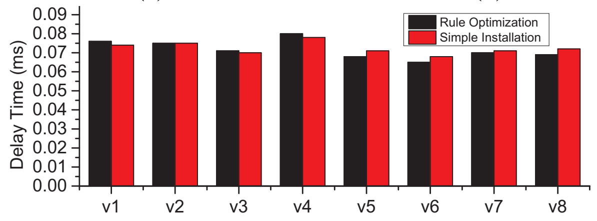
We study two patterns (*one-to-N* and *N-to-one*) in details and show how they influence results differently. We compare the number of rules and the delay time with the baseline method to show that it does not affect the performance of data transmission. Fig. 10(a) shows *N-to-one* mode that there are multiple vehicles connecting to one surveillance camera (B). We pick A that has the longest path to evaluate delay time. For the baseline method, it needs to install one rule at each switch with the match of source address. And for proposed approach,



(a)



(b)



(c)

Fig. 9. (a): Different number of rules for two strategies; (b): The number of vehicles at different moments; (c): The delay time of vehicles in two strategies.

every packet transferred from B to A modifies the headers (destination IP address) at each switch and then forwards to the next switch. Fig. 10(b) shows that, as the number of switches increases, delay time of both methods gets larger but with no statistically significant difference. We conclude that the packet modification process in improved rule installation does not influence performance. Fig. 10(c) shows the number of rules at switches. The red (right side) line represents the baseline method that node B transfers data flow across the network to every node at the same time. It only needs one rule at each switch with $(*, B')$ (B' is the multicast address). The black (left side) line depicts improved rule installation approach that modifies the destination address at every branching node and then forwards the packets to given ports. It needs only one rule at each switch comparing to the the baseline method that requires multiple rules, the number of which equals to the data sources even they have the same destination. For example, the switches along V_1 to B need two rules since V_1 also sends a request to C.

Fig. 10(d) shows *one-to-N* patterns where vehicle V connects multiple cameras (B, C, D) at the same time. Fig. 10(e) shows the delay time according to the different number of switches. Without improved rule installation, three rules are needed for each source node at each switch, $(B, *), (C, *), (D, *)$, which is a source-driven mode (forwarding packets based on data source). With improved rule installation, we need one rule only, i.e., $(*, A)$ at each switch and it can still support *N-to-one* pattern. Fig. 10(f) shows the number of rules of two methods.

We draw two conclusions from the results. First, the modification of packets at branching nodes has no impact on the performance of data transmission. Second, for the best case of improved rule installation (*one-to-N*), the number of reduced rules is proportional to the number of data sources. For the worst case of improved rule installation (*N-to-one*), the number of rules at switches is equal to that of the baseline method.

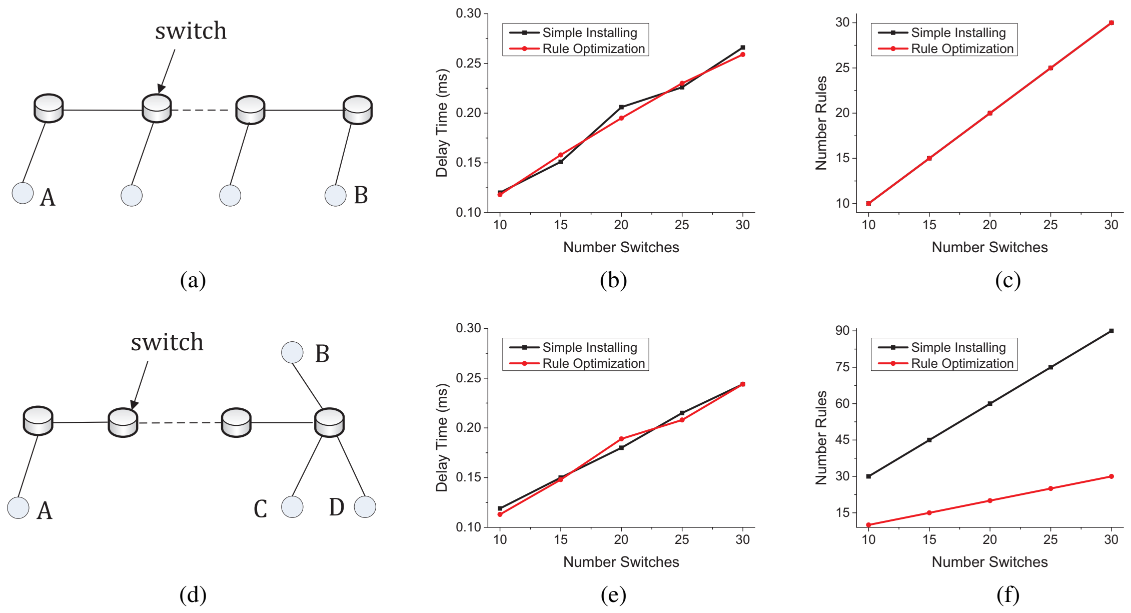


Fig. 10. (a): A topology of *N-to-one* mode; (b): Delay time on different number of switches in (a); (c): Number rules on different number of switches in (a); (d): A topology of *one-to-N* mode; (e): Delay time on different number of switches in (d); (f): Number rules on different number of switches in (d)

VI. Conclusion and Future Work

This paper proposes a new architecture SDIV to address the proprietary and closed way issue in the traditional network design in IoV. The baseline method of rule installation is not efficient due to the characteristics of IoV. Improved rule installation for reducing flow table size is thus necessary in SDIV. We design an improved rule installation approach that reduces the size of a flow table without degrading the performance of data transmission for real-time query services. The separation of a wired data plane from wireless data plane and the proposed destination-driven mode are suited well for the characteristics of IoV. Evaluation based on testbed implementation with real traces shows that our improved rule installation approach significantly reduces the number of rules without degrading the performance of data transmission. In our future work, we plan to make the controller collect more information from switches, e.g., the counter in the OpenFlow rule which equals the number of matched packets for the rule, for further data transfer optimization.

References

[1] "CVIS." http://www.cvisproject.org/.

[2] H. Makino, "Smartway project," *Development*, vol. 2005, 2005.

[3] "NDN." http://named-data.net/.

[4] "MobilityFirst." http://mobilityfirst.winlab.rutgers.edu/.

[5] "NEBULA." http://nebula-fia.org/.

[6] "Floodlight." http://www.projectfloodlight.org/floodlight/.

[7] "Mininet." http://mininet.org/.

[8] L. Du and H. Dao, "Information dissemination delay in vehicle-to-vehicle communication networks in a traffic stream," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 1, pp. 66–80, 2015.

[9] J. Ott and D. Kutscher, "A disconnection-tolerant transport for drive-thru internet environments," in *Proc. IEEE INFOCOM 2005*. Miami, FL, USA: IEEE, 2005, pp. 1849–1862.

[10] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. N. Levine, and J. Zahorjan, "Interactive wifi connectivity for moving vehicles," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, 2008, pp. 427–438.

[11] A. U. Joshi and P. Kulkarni, "Vehicular wifi access and rate adaptation," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 423–424, 2011.

[12] F. Xu, S. Guo, J. Jeong, Y. Gu, Q. Cao, M. Liu, and T. He, "Utilizing shared vehicle trajectories for data forwarding in vehicular networks," in *Proc. IEEE INFOCOM 2011*. Shanghai, China: IEEE, 2011, pp. 441–445.

[13] P. Sahu, H. Wu, J. Sahoo, and M. Gerla, "Bahg: Back-bone-assisted hop greedy routing for vanet's city environments," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 1, pp. 199–213, 2013.

[14] L. Zhang, B. Yu, and J. Pan, "Geomob: A mobility-aware geocast scheme in metropolitans via taxicabs and buses," in *Proc. IEEE INFOCOM 2014*. Toronto, ONT, CA: IEEE, 2014, pp. 1279–1787.

[15] H. Zhu, M. Dong, S. Chang, Y. Zhu, M. Li, and X. Shen, "Zoom: Scaling the mobility for fast opportunistic forwarding in vehicular networks," in *Proc. IEEE INFOCOM 2013*. IEEE, 2013, pp. 2832–2840.

[16] J. Nzouonta, N. Rajgure, G. Wang, and C. Borcea, "Vanet routing on city roads using real-time vehicular traffic information," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 7, pp. 3609–3626, 2009.

[17] J. Cheng, J. L. Cheng, M. C. Zhou, Q. Zhang, C. Yan, Y. Yang, and C. Liu, "Routing in internet of vehicles: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2339–2352, 2015.

[18] J. Hare, L. Hartung, and S. Banerjee, "Policy-based network management for generalized vehicle-to-internet connectivity," in *Proc. ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*. Helsinki, Finland: ACM, 2012, pp. 37–42.

[19] A. Abadi, T. Rajabioun, P. Ioannou *et al.*, "Traffic flow prediction for road transportation networks with limited traffic data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 653–662, 2015.

[20] Z. Wu, M. Yao, H. Ma, and W. Jia, "Improving accuracy of the vehicle attitude estimation for low-cost ins/gps integration aided by the gps-measured course angle," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 553–564, 2013.

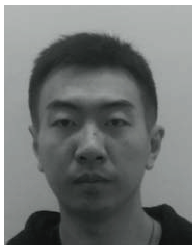
[21] J. Ahn, Y. Wang, B. Yu, F. Bai, and B. Krishnamachari, "Risa: Distributed road information sharing architecture," in *Proc. IEEE INFOCOM 2012*. Orlando, FL, USA: IEEE, 2012, pp. 1494–1502.

[22] C. T. Calafate, G. Fortino, S. Fritsch, J. Monteiro, J.-C. Cano, and P. Manzoni, "An efficient and robust content delivery solution for ieee 802.11 p vehicular environments," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 753–762, 2012.

[23] G. Grassi, D. Pesavento, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang, "Vanet via named data networking," in *Proc. IEEE Computer Communications Workshops (INFOCOM WKSHPS)*. Toronto, ONT, CA: IEEE, 2014, pp. 410–415.

[24] K.-K. Yap, R. Sherwood, M. Kobayashi, T.-Y. Huang, M. Chan,

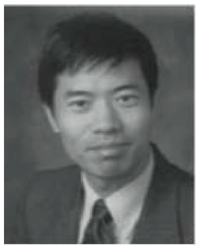
N. Handigol, N. McKeown, and G. Parulkar, “Blueprint for introducing innovation into wireless mobile networks,” in *Proc. The Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*. New Delhi, India: ACM, 2010, pp. 25–32.
[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[26] H. Kim and N. Feamster, “Improving network management with software defined networking,” *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.
[27] Y. Kanizo, D. Hay, and I. Keslassy, “Palette: Distributing tables in software-defined networks,” in *Proc. IEEE INFOCOM 2013*. Turin, Italy: IEEE, 2013, pp. 545–549.
[28] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: Simplifying sdn programming using algorithmic policies,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, 2013, pp. 87–98.
[29] “Open Networking Foundation.” https://www.opennetworking.org/.
[30] https://www.dropbox.com/s/xbx0iw954a9m7kk/Shanghai%20taxi%20data-08.zip.



Xin Wang received the BS degree from the Department of Computer Science, Tongji University in Shanghai, China. He is currently working toward the PhD degree in the Department of Computer Science and Engineering, Tongji University. His research interests include computer network, Software Defined Network, and distributed computing.

Cheng Wang received MS degree at Department of Applied Mathematics from Tongji University in 2006 and the PhD degree in Department of Computer Science at Tongji University in 2011. He is currently a research professor in the Department of Computer Science at Tongji University. His research interests include software-defined networking, wireless communications and networking, and intelligent transportation system.

Junqi Zhang received his Ph.D. in Computing Science from Fudan University, China, in 2007. He became a Post-Doctoral Research Fellow and a Lecturer at the Key Laboratory of Machine Perception, Ministry of Education in Computer Science at Peking university, Beijing, China in 2007, and was the recipient of the Outstanding Post-Doctoral Award from Peking university. He is currently a Research Fellow and an Associate Professor with Department of Computer Science and Technology, Tongji University, Shanghai, China. His current research interests include machine learning, intelligent and learning automata, particle swarm optimization, high-dimensional index and multimedia data management.



Mengchu Zhou (S’88-M’90-SM’93-F’03) received his B.S. degree in Control Engineering from Nanjing University of Science and Technology, Nanjing, China in 1983, M.S. degree in Automatic Control from Beijing Institute of Technology, Beijing, China in 1986, and Ph. D. degree in Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY in 1990. He joined New Jersey Institute of Technology (NJIT), Newark, NJ in 1990, and is now a Distinguished Professor of Electrical and Computer Engineering. His research interests are in Petri nets, Internet of Things, big data, web services, manufacturing, transportation, and energy systems. He has over 620 publications including 12 books, 300+ journal papers (240+ in IEEE Transactions), and 28 book-chapters. He is the founding Editor of IEEE Press Book Series on Systems Science and Engineering. He is a recipient of Humboldt Research Award for US Senior Scientists, Franklin V. Taylor Memorial Award and the Norbert Wiener Award from IEEE Systems, Man and Cybernetics Society. He is a life member of Chinese Association for Science and Technology-USA and served as its President in 1999. He is a Fellow of International Federation of Automatic Control (IFAC) and American Association for the Advancement of Science (AAAS).

Changjun Jiang received the PhD degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995 and conducted postdoctoral research at the Institute of Computing Technology, Chinese Academy of Sciences, in 1997. Currently he is a professor with the Department of Computer Science and Engineering, Tongji University, Shanghai. He is also a council member of China Automation Federation and Artificial Intelligence Federation, the vice director of Professional Committee of Petri Net of China Computer Federation, the vice director of Professional Committee of Management Systems of China Automation Federation, and an Information Area Specialist of Shanghai Municipal Government. His current areas of research are concurrent theory, Petri net, and formal verification of software, concurrency processing, and intelligent transportation systems. He is a member of the IEEE.