Ferrying Vehicular Data in Cloud Through Software Defined Networking

Prasan Kumar Sahoo, Yoppy Yunhasnawa
Dept. of Computer Science and Information Engineering
Chang Gung University, Guei-Shan, 33302, Taiwan (ROC)
Email: pksahoo@mail.cgu.edu.tw, M0329016@stmail.cgu.edu.tw

***Abstract*—In Vehicular Ad Hoc Network (VANET), huge amount of data requests are made by the users from the vehicles traveling through a city. The requested data need to be transmitted and processed by the cloud service providers to maintain the quality of service. In this paper, data transmission time is computed when vehicular data is transmitted to the cloud though the road side units and OpenFlow switches in a Software Defined Network (SDN). The proposed scheme allows some sorts of traffic engineering methods that can be used to address the connectivity problems involving transmission delay and packet loss of the vehicles in a smart city. Simulation results show that the proposed mechanism can achieve better packet delivery rate and least round-trip time as compared to similar data transmission protocols.**

I. Introduction

The numbers of hand-held devices have increased significantly due the advancement of mobile communication technology. On the other hand, vehicular information technologies such as VANET and navigation system have become more and more sophisticated [1]. These technologies are improving the vehicular connectivity experience by which both of the driver and passenger can have easier data access from the inside of their vehicle. In VANET, Road Side Unit (RSU) is used as an instrument to gather data from the vehicles in the road, which is used as a medium to transfer the data from the city cloud to the vehicles.

Guaranteeing quality of connectivity for every vehicle registered in a smart city vehicular network is not a trivial matter due to several factors such as the variety of vehicle movements and traffic density. In a smart city, traffic condition in each road varies time to time. On the one hand, there are some roads that have less traffic in which the communication between each passing vehicle and the corresponding RSU is established smoothly due to the fact that all of those vehicles only need smaller amount of time connecting with the RSUs as they travel to the next road. Consequently, workload of the corresponding RSU will also be less. However, on the other hand there are roads that have more traffic. For example, red traffic light on a busy road intersection or any roads near the shopping center/mall area in which the workload of the corresponding nearby RSU will become more.

In case of a heavily congested roads, both drivers and passengers might become bored and may interact with their devices inside their vehicle to look for some entertainment programs by requesting various services from the city cloud in form of game, video, advertising, news, social media and others. As a result, the corresponding RSU near the road will become overloaded because of the increasing number of vehicles sending and receiving large amount of data to and from the city cloud, simultaneously. Furthermore, this problem can be worsened, if another section of nearby road also suffers heavy traffic jam. In other words, if more nearby RSUs are being overloaded, the corresponding aggregation switch will also have higher chance of becoming overloaded. Thus, the total value of round trip time (RTT) will increase significantly and finally affects the users' experience as the packets transmission is being delayed.

The rest of the paper is organized as follows. Related works are given in Section II. System model of the proposed work is given in Section III. The proposed round trip time computation is described in Section IV. Performance evaluation of the protocol is made in Section V and concluding remarks are made in Section VI.

II. Related Works

Methods of traffic engineering and load-balancing in a cloud-connected network by utilizing SDN controller have been proposed by some handful papers before. The effectiveness of traffic engineering in an existing network with SDN architecture incrementally applied is evaluated by the author in [4]. In that work they demonstrate how SDN controller can significantly improve utilization of the network while at the same time reducing the rate of packet loss and delay. Their work is able to solve traffic engineering problem in a hybrid network environment consists of traditional and OpenFlow switching devices. In contrast, our work considers a network environment that uses 100% OpenFlow-enabled switches that are controlled by an SDN controller.

Author in [5] proposed a method to significantly reduce the rate of packet loss also in a hybrid network by utilizing network packets' header. In this work, they employed piggybacking method using packet's header to update every path's load in the network whenever every packet arrives at any switching devices and finally the SDN controller will decide how much data rate can enter the network's ingress switch. Their method is proven very effective in avoiding network path congestion. Another work by authors in [6] demonstrates

This work is partly supported by the Ministry of Science and Technology (MOST), Taiwan under the grant number 105-2221-E-182-050.

978-1-5090-0724-0/16/$31.00 ©2016 IEEE

the benefits of traffic-aware VMs placement in enhancing the network scalability. The main goal of their work is to produce the most optimal VMs allocation on a group of physical machines by analysing the characteristic of traffic patterns between all VMs. Yet, the proposed method considers only the network link utilization or network congestion, without any thought about the workload in each VM by which the latency of the service can be seriously affected.

Authors in [7] proposed an idea to enable zero over-head communication between users and servers by considering jobs arrival rate. While their scheme successfully eliminates any overhead caused by scheduling in each job's critical path, they do not separate the jobs type and treat them equally. Finally, the author of [8] proposed a method to reduce a cloud data center network's Round Trip Time. They consider different type of request as well as the load of every path in the network. In the work they collected all available routing paths in the entire network and then estimate the paths' load based on packets' arrival rate and switching devices processing capability. All of those paths then sorted and filtered according to the least load. They also filter the destination VM based on the request type and estimate all VM load to find the correct VM with the least load. The combination of the least loaded network path and VM resulting in the minimization of overall RTT. Although they successfully reduce the RTT in a fixed network, they do not consider the dynamic back-trip path. For any given packet, they will use the same path used by incoming request packet for the outgoing response packet. In a mobility-enabled network, it is very crucial as requesting nodes can move at any time to any place.

III. System Model

In this system model we assume in a smart city vehicular network there are many vehicles $\{V_1, V_2, ...V_m\}$ and each of the vehicle V_i has different position in latitude V_i^x and longitude V_i^y. Those vehicles are traveling along many different roads which have their area fully covered by communication range with radius RSU_i^r of a set of road-side units (RSUs) $\{RSU_1, RSU_2, ...RSU_m\}$ installed across the entire city. All of those RSUs are grouped into some clusters $\{ClusRSU_1, ClusRSU_2, ...ClusRSU_m\}$ based on their geographical adjacency. For each RSU cluster $ClusRSU_i$ there will be single SDN switch SW_i acting as the aggregation switch for all RSU_i member of that cluster such that the total number of RSU clusters is equal with the total number of SDN switches installed around the city. Each and every SDN switch is assumed to be connected in MESH topology with all available VM $VM_{i,j}$ serving different request types in the smart-city cloud. Every link that allows a packet to be delivered from any vehicle through RSU to SDN switch, until it reaches a specific destination VM we named it as packet route $Path_i$ and every packet route will have different value of round-trip time RTT_i.

IV. Proposed Protocol

Both of the transmission delay and packet loss problems will be considerably difficult to handle by conventional aggregation switches because that kind of switching devices work individually in nature as they adopted distributed non-collective routing algorithm instructed in each device's logic board. This independent work causing each switching device ignoring the work of other switching devices thus they become blind of the global view of the entire network topology, furthermore preventing advanced technique of congestion control and load balancing to be applied on the city's already complicated vehicular network.

In this research work we propose a way to work out the problem by considering the usage of SDN switches as the replacement of conventional aggregation switches installed in the smart-city's vehicular communication network. SDN switch allows centralized communication network management, thanks to its separated data and control plane [2]. With the help of OpenFlow protocol, the routing logic of the switching device can be moved to a separate controlling entity away from the physical network itself [3]. Consequently, those controlling entity is able to gather the information of all available routing paths from the entire network resulting in the complete understanding of the global view of the network.

Based on the known location information of a moving vehicle and the comprehensive knowledge of all available routing paths across the entire city's vehicular network, we can then easily select and sort the most optimal routing path for the vehicle at a specific time in a specific transmission grid. In this work, we try to select the best routing path to be assigned to each registered vehicle by considering the value of RTT, a chunk of time needed by a request to travel from its sender until it reaches a specific destination VM in the cloud plus the time needed by the consecutive response packet to travel from the VM back to the requesting vehicle.

Knowing the estimated value of RTT precisely is very important because RTT itself is proportional to how much load in a routing path, moreover the load value is also proportional with the possibility of congestion. Finally, the most important contributions of this work consists of an analysis about the total value of RTT of a certain routing path in the smart-city vehicular network as well as a set of protocols that aims to reduce the probability of transmission delay and packet loss in between vehicle and city cloud data exchange.

A. VANET Layer

In VANET layer we divide a smart-city area into a group of smaller rectangular area based on the communication radius of every RSU installed across the city. This rectangular area, we name it as transmission grid. Transmission grid is a logical grid in the shape of equal-sided rectangle inside a circle formed by rotating an RSU communication radius.

If an RSU_i has the communication radius RSU_i^r, then RSU_i will be the center of a transmission grid TG_i that has

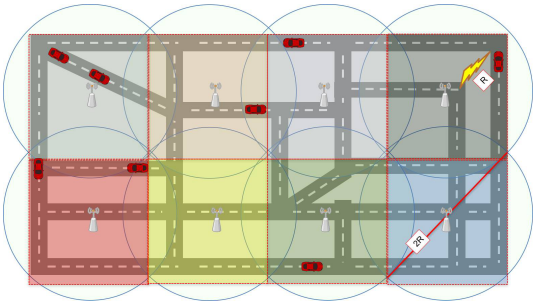

Fig. 1. Transmission Grid.

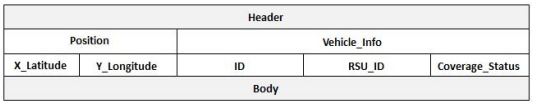

Fig. 2. Proposed Packet Header.

equal length in all of its sides TG_i^s where:

$$TG_i^s = \frac{1}{2}\sqrt{2RSU_i^r} \tag{1}$$

The formed transmission grid TG_i should have a maximum and minimum location range based on the Cartesian position of road-side unit RSU_i that become its center. The maximum/minimum X position $TG_i^{x-max/min}$ is the maximum/minimum latitude for any vehicle to be considered inside the coverage area of transmission grid TG_i and the same thing applies for the maximum/minimum Y $TG_i^{y-max/min}$ as longitude. If coordinate of the RSU that consists of latitude RSU_i^x and longitude RSU_i^y are known, then:

$$TG_i^{x-max/min} = RSU_i^x \pm \sqrt{2RSU_i^r} \tag{2}$$

And,

$$TG_i^{y-max/min} = RSU_i^y \pm \sqrt{2RSU_i^r} \tag{3}$$

Respectively.

1) Registration and De-registration: In order to make sure that every vehicle registered in the smart-city network does not experience any packet loss, their movements should be well monitored. So, we assume that SDN controller should have a capability in storing and maintaining the location information attached in data packet sent by each and every vehicle from the entire city network. The monitoring process consists of two different processes called registration and de-registration. Each of the process will be enforced every time a vehicle entering and exiting a transmission grid.

Registration process is a location recording process enforced at the time a vehicle V_i is entering the area of a transmission grid TG_i. In this process, the corresponding road-side unit RSU_i will inform the SDN controller that the vehicle V_i is currently under its 'jurisdiction'. Road-side unit RSU_i will modify the header of all data packets sent by vehicle V_i filling the rsu_id slot with its ID. At the moment the packet forwarded and then retrieved by an SDN switch, SDN controller will read the packet header and then updating the $registered\_vehicle$ table in its memory by adding a new record containing the ID of vehicle V_i and ID of RSU RSU_i exactly the same as written in in the packet header. In addition, the record will also contain 'IN' flag for $coverage\_status$ column.

De-registration process happens at the time a vehicle V_i is exiting the area of transmission grid TG_i. The overall steps are similar to the registration process, however for de-registration process the corresponding RSU RSU_i will inform SDN controller that the vehicle V_i is no longer in its jurisdiction. In respond to this signal, SDN controller will also make a new entry in $registered\_vehicle$ table containing ID of vehicle V_i and the ID of RSU RSU_i but with 'OUT' flag for $coverage\_status$ column.

One of the most important benefits from these two processes is that the SDN controller will always know where is exactly the vehicle V_i currently resides. Accordingly, the SDN controller will be able to prevent any packet loss whenever a response packet sent from the service provider VM back to the requesting vehicle at the time the packet arrived at any SDN switch by performing location checking in its memory.

Following is the complete vehicle handoff mechanism in an algorithm form describing how a vehicle movement from one transmission grid to another transmission grid will be handled.

B. SDN Layer

This layer covers communication aspect that occurs in the city wired network along the way from one transmission grids RSU to the very first SDN switches connected to it until finally the very last SDN switch connected to the cloud. The protocol in this layer heavily depends on the analysis of how much average time needed by packets to travel from an RSU to a final SDN switch. That chunk amount of time we named it DTT (Data Transfer Time) will determine whether a certain path is in heavily loaded, congested, or normal condition. The main purpose of this layer is to minimize overall DTT for the packet as well as avoiding network congestion while maintaining balanced load at the same time.

1) Packet Time in the Network: In this work, DTT is regarded as one of two important RTT components because the value of DTT will considerably affects the total value of RTT as packets may spent most of their living time in the chosen network paths. DTT itself is a result of combination between average packet sojourn time tSW_{wait} in the network and packet processing time by the switch $tSW_{process}$ in all network nodes.

$$DTT = tSW_{wait} + tSW_{process} \tag{4}$$

In a smart-city network, a packet route $Pahth_j$ can be formed by connecting several network nodes comprises of different number of RSUs and switching devices. In our VANET topology as described in Layer 1 part, RSU can be connected to other RSU, but RSU connections will always ended up in

the first SDN aggregation switch. So, once the packet reaches an SDN switch, it will not go into any other RSU, instead it will go to the next switching devices until finally reaches its destination VM. Each RSU node $RSU_{i,j}$ in $Path_j$ will have different number of vehicles nV_i that are currently resided in their respecting transmission grid TG_i. Using that packet route $Path_j$, each packet from each of those vehicles will be processed one time in each network node before they reach the final SDN switch. If the packet processing policy is first come first serve, then packets traversing this packet route can be modeled as M/M/1 queue. If nV_i in each $RSU_{i,j}$ is known and the processing capability of every RSU $\mu RSU_{i,j}$ and SDN switch $\mu SW_{i,j}$ serving as the path's network nodes are also known, then the average packet sojourn time in the path will be,

$$tSW_{wait} = \left(\sum_{j=1}^{nRSU_k} \frac{\rho RSU_{j,k}}{\mu RSU_{j,k} - \left(\sum_{i=1}^{nV_j} V_{i,j} * \lambda pV_{i,j} \right)} \right) + \frac{\rho SW_k}{\mu SW_k - \mu RSU_{nRSU_k - 1,k}} \tag{5}$$

Where,

$$\rho RSU_{j,k} = \frac{\left(\sum_{i=1}^{nV_j} V_{i,j} * \lambda pV_{i,j} \right)}{\mu RSU_{j,k}} \tag{6}$$

And,

$$\rho SW_k = \frac{\mu RSU_{nRSU_k - 1,k}}{\mu SW_k} \tag{7}$$

Assuming the processing capability of all RSUs is uniform as well as the SDN switches, the value of processing time $tSW_{process}$ with the number of RSU nodes and SDN switch nodes known can be calculated as,

$$tSW_{process} = \left(\sum_{i=1}^{nRSU_k} \frac{1}{\mu RSU_{i,k}} \right) + \left(\sum_{j=1}^{nSW_k} \frac{1}{\mu SW_{j,k}} \right) \tag{8}$$

2) Addressing Request Priority Issue: In a smart-city network, there might be a set of different request types $SReq$ = $\{Req_1, Req_2, ...Req_j\}$ that have different bandwidth requirements $\{RBw_1, RBw_2, ...RBw_j\}$ and different priority values $\{Pv_1, Pv_2, ...Pv_j\}$. This different priority values can be made based on the service policy of the network or to satisfy the quality of service for paying users above the non-paying ones. If all of the request types are treated as if they have the same priority value Pv, all of the Req_j in $SReq$ will compete for limited bandwidth of packet route $Path_{bw}^k$ whenever they traverse the route $Path_k$ at the same time. This may cause some undesirable delay for higher priority user if lower priority requests win the bandwidth competition and this is most likely to occur because in most case, lower priority requests will be more in number compare to the higher ones. To combat this problem, SDN controller should not allow any set of requests $SReq_k$ to traverse certain path $Path_k$ at the same time directly. Instead, it should specify which requests have highest priority and how many number of other lower priority requests can be put in a set $SReq_k$ at a time such that the total value of priority Pv_{tot}^k can be maximized without the total bandwidth of the set $SReq_k$ exceeding the total bandwidth $Path_{bw}^k$ of the selected packet route $Path_k$. We see this problem is equivalent to classical knapsack 0/1 problem that can be solved using a number of handful existing algorithms. For our protocol, we consider the usage of dynamic programming approach to solve the problem because the simplicity nature of the algorithm.

The proposed mechanism works by dividing the knapsack main problem into sub-problems. First of all, a two-dimensional place-holder $matrix$ with the size of row j equals to number of distinct item + 1 and the size of column w equals to the maximum path bandwidth $Path_{bw}^k$ + 1 is created. Then, it will initialize the values of first row and column of the matrix into 0. After that, the algorithm will try to find the maximum value of priority Pv_{tot}^k that can be hold by the path $Path_k$. This is done by iterating through $matrix$ rows representing request types $SReq$ where in each row the algorithm will also also compare the bandwidth requirement RBw_j of current request type Req_j with incremental bandwidth $Path_{bw}^{k,i}$; i = $\{0, 1, 2...Path_{bw}^k\}$ of the path bandwidth $Path_{bw}^k$ represented in $matrix$ column. If current request Req_j of the request set $SReq$ has total bandwidth requirement exceeding the current iteration of $Path_{bw}^{k,i}$, the Req_j can not be a part of the solution. Hence, we put the value of last request type Req_{j-1} into current $matrix$ cell. Otherwise, we choose the greater value between request type in previous iteration Req_{j-1} and current request type Req_j. Finally, by finishing all the iteration, the maximum priority value Pv_{tot}^k the sack $Path_k$ able to hold can be found in the bottom-right corner of the $matrix$. To find the combination of requests that sum into the maximum value, the place-holder $matrix$ needs to be traced back. While iterating back from the bottom-right corner of the $matrix$, the algorithm compares current cell with one-cell above it. If the current cell value equals to the value of on cell above, the current cell value will be marked as 'added' and all of the 'added' cell value is the request type Req_j that is added into final set of the solution $SReq_k$.

3) Congestion Avoidance and Load Balancing: In this protocol, a packet route or a path is defined as a wired network link that connects one RSU in a transmission grid to any final switching device that is connected to any VM in the city cloud. A packet route can be consisted of several to many connected network nodes (more than one RSUs and switches) and we assume that the SDN controller has the ability to maintain all available packet routes information from the entire city network. That information includes the address of every network nodes that forms each of those packet routes. Every packet route will be stored as a record in a data table and each record should be associated with one unique path ID. Each of the recorded packet routes will have different DTT value and this value must be continuously updated by the

SDN controller periodically every certain amount of time. This ability is very important to be possessed by SDN controller in order to guarantee that network data is relevantly reflected the physical vehicle traffic condition in city road.

Given the scenario of a certain congested road sector in a city network, we want to minimize the round-trip time of every packet in that sector. First of all the SDN controller should find all available packet routes $\{Path_1, Path_2, ...Path_k\}$ in that congested sector's transmission grid TG_i by selecting all packet routes that are originated from the transmission grid's respective RSU RSU_i. Those set of packet routes then labeled as the $admissible\_paths$. Prior to those filtering, for each path $Path_k$ in the set, SDN controller then count the number of vehicle $nVPath_k$ currently using the path. Based on those known traffic in particular path, the SDN controller will be able to estimate the current DTT_k of every path $Path_k$.

With the value of all DTT_k known, SDN controller will be able to calculate the average DTT avg_dtt across all packet routes in $admissible\_paths$ set. The next step, SDN controller will select every path $Path_k$ from $admissible\_paths$ that has DTT_k that is more than 150% of avg_dtt and then labels all of those paths as $overloaded\_path$ and grouped into $overloaded\_paths$ set. After that, the SDN controller will select every path $Path_k$ from $admissible\_paths$ set that has DTT less than 50%of avg_dtt, labels them as $underloaded\_path$ and put it in $underloaded\_paths$ set respectively.

Now two different paths 'buckets' have been categorized by the SDN controller, the $overloaded\_paths$ bucket and the $underloaded\_paths$ bucket. Upon the formation of those buckets, for each path $current\_path$ in $overloaded\_paths$ bucket, the SDN controller will find the most under-loaded path in $underloaded\_paths$ bucket. The 'most under-loaded' means a $Path_k$ that has smallest value of DTT_k and this path then labeled as the $recipient\_path$. Next, the SDN controller will move a portion of traffic $traffic\_portion$ from $current\_path$ to $recipient\_path$. This is done by changing every vehicle's $VPath_{q,k}$ packet route assignment that is included in the $traffic\_portion$. SDN controller then will recalculate both of $current_path$s and $recipient\_path$'s DTT values. If $recipient_path$s DTT_{recip} is still below avg_dtt, SDN controller should continue adding another $traffic\_portion$ from $current\_path$ traffic and recalculate DTT_{recip} until its value become equals or exceeds 100% of avg_dtt. Whenever that condition is satisfied, the $recipient\_path$ is removed from $underloaded\_paths$ bucket. As long as the $current\_path$'s DTT_{cur} still not less than or equals to 100% avg_dtt, the SDN controller will continue to find again another most under-loaded path, set it as new $recipient\_path$ and finally apply the knapsack function to allow prioritization of the incoming requests.

The mechanism works by continuously monitoring and recalculating the value of DTT. Using this mechanism, congestion should be always avoided, as there will be no path that has load more than the average load. Also at the same time, because of DTT is estimated by considering the processing capability and current job in every network node, the overall load in all SDN switched can be balanced. All of the benefits can be achieved provided that, the estimation of DTT can be done in short amount of time.

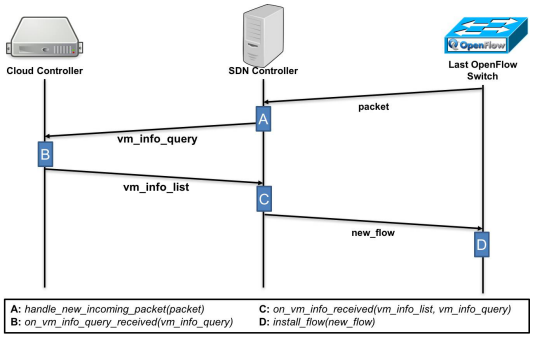

Fig. 3. Communication sequence between controllers.

C. Cloud Layer

Layer 3 explains the scenario in handling request packets journey from the last SDN switch toward a specific destination VM. These packets will be processed in the VM until finally the corresponding response packets are issued and their destination RSU is decided by the SDN controller. In this layer, the collaboration between SDN controller and cloud controller is very important. SDN controller and cloud controller is assumed to have communication link between them. This means that they must be able to exchange bidirectional stream of data directly. This information sharing between 2 controllers will help SDN controller make better decision about which VM should receive the request packet and which RSU should receive the corresponding response packets.

Other than that, this layer of the protocol also needs the cloud controller to have sufficient ability in maintaining vm_info_table just like the SDN controller's ability to maintain previously explained $path\_info\_table$. The vm_info_table will store all useful information of every usable VM in the cloud. These information including, but not limited to, VM ID, current CPU load, and request type served. All off the information are very essential for the SDN controller in making correct VM selection.

So, upon the arrival of the first packet $Packet_i, j$ of request Req_j to the last SDN switch on top of the wired network, the switch will check its flow tables entry. If there is no forwarding rule that matches packet's sender address, source RSU, and request type, the OpenFlow switch will ask for new flow by sending the packet to the SDN controller. The SDN switch then will determine the packet's request type $ReqType_j$ from packet's header. Subsequently, the SDN controller will form a message vm_info_query containing $ReqType_j$ information and pass it to the cloud controller. This is the starting point of the Layer 3. After the cloud controller received those vm_info_query the following 2 main processes will be started.

1) Data Processing Time (DPT): In our protocol, we define data processing time abbreviated as DPT, as the time needed by a specific destination VM to process certain type of request. DPT is important as combining the best packet route with the least DTT and the best VM with the least DPT as well as CPU load will ensure minimum RTT for each request sent by users. To choose the best VM instance for handling current request type, we need to find each VM's DPT value. Based on our protocol flow, we observed that each VM in the city cloud that is capable to handle a specific request type Req_j may have different computational power compare to each other, therefore each of them will also have different number of requests that can be handled per unit time. This is what we define as the VM processing capability $\mu VM_{i,j}$. On the other hand, there will always be a number of new requests per unit time redirected to each VM based on the decision made by the SDN controller, this is what we define as the request arrival rate $\lambda VM_{i,j}$. Based on the known processing capability and request arrival rate, we can also get the traffic intensity of each VM defined as $\rho VM_{i,j}$. With all the three factors known, we can calculate the average response time $tVM^{i,j}_{process}$ for each VM as follows.

$$tVM^{i,j}_{process} = \frac{1}{\mu VM_{i,j} - \lambda VM_{i,j}} \tag{9}$$

Because overall DPT is a product of combination between VM's response time and the time spent by each request in their queue $tVM^{i,j}_{wait}$ where,

$$tVM^{i,j}_{wait} = \frac{\rho VM_{i,j}}{\mu VM_{i,j} - \lambda VM_{i,j}} \tag{10}$$

Hence, the final value of data processing time in i-th VM instance serving request type j should be,

$$DPT_{i,j} = \frac{1 + (\lambda VM_{i,j}/\mu VM_{i,j})}{\mu VM_{i,j} - \lambda VM_{i,j}} \tag{11}$$

2) Handling Incoming Request: As a control entity, cloud controller should have the ability to retrieve necessary information from all workable VMs in the cloud. All of those information are stored as entries of vm_info_table in the cloud controller's memory. This table has 3 essential columns: vm_id, $service\_type$, and cpu_load. The vm_info_table will keep the record of information about every virtual machine's ID, service type and current CPU load that are periodically updated every certain amount of time. This is done in order to ensure VM information relevancy. vm_id and $service\_type$ information might be obtained in a longer interval, for example, on each time a VM instance is created, or even just statically provided as the service type assigned to a specific VM usually is fixed and rarely changed. While the information about VM load must be gathered in shorter interval, or if possible, every time cloud controller received vm_info_query message because VM load is very dynamic in nature.

The cloud controller will always wait for any vm_info_query that comes from SDN controller. Whenever it comes, cloud controller will immediately extract the $ReqType_j$ data. Proceeding to the next step, cloud controller will search for any VM_i record from its vm_info_table that currently serving the same $ReqType_j$ and also checking their cpu_load info. Each of those VM_i record along with its cpu_load info then stored into a data structure called vm_info and then pushed into a prepared list vm_info_list. This list is then encoded together with received vm_info_query and finally sent back to the SDN controller.

As the time when SDN controller receives vm_info_list sent back by the cloud controller, it counts all $DPT_{i,j}$ of all VMs and ultimately find the average avg_dpt. Following the process, SDN controller will select all VM_i that has $DPT_{i,j}$ more than 75% of avg_dpt and put it into new list called $underloaded\_vm\_info\_list$. Then the SDN controller will extract the information about cpu_load_i for each VM_i in the $underloaded\_vm\_info\_list$. The SDN controller then select any VM that has smallest CPU load and then set it as the $destination\_VM$. Following this decision, SDN controller will encode the information of the selected VM $destination\_vm$ into a new flow and then installed the flow to all OpenFlow switches that are connected to all VMs in the cloud so that whenever any packets that are parts of the current request Req_j are come they will be forwarded to the $destination\_vm$.

3) Handling Outgoing Requests: At any time when the first response packet is sent by VM to the first outgoing OpenFlow switch SW_{out}, the switch will forward that packet to the SDN controller to determine which path should be used. This is very important because at the time the request is completely processed by the destination VM, it is very likely that the vehicle sending those request already moved to another transmission grid which means another RSU. If the packet is sent using the same incoming packet route then it will never reach the requesting vehicle.

SDN controller will decide which outgoing path should be used to send the rest of the response packets by searching the location information of the requesting vehicle based on its address $destination\_addr$ extracted from the response packet header. The location information can be precisely retrieved just by reading the last rsu_id which has $coverage\_status$ equals to 'IN' from the $vehicle\_table$ previously recorded in Layer 1. Following the discovery of the present RSU_i the vehicle V_i being connected, the SDN controller will then select all packet routes from its $path\_info$ table that is originated from SW_{out} and ended at RSU_i. Every record of the selected packet routes has a field that stores the last Data Processing Time (DTT) estimation previously done by the SDN controller at Layer 2. Whenever the fields does not contain any value, the SDN controller will estimate the path's DTT and save it to $path\_info\_table$ for later usage.

Now that all selected paths have their DTT value, the SDN controller will sort list of all selected packet info in ascending order based on the DTT value. Then the first index of the list will be selected as the target path $Path_i$ in which the rest of response packets will travel to reach the destination

TABLE I
REQUEST TYPES

Type	Size of Request (KBytes)	VM Processing Rate
1	2	3500
2	10	15000
3	50	90000
4	100	180000

vehicle V_i. The information of $Path_i$ then encoded as a new flow that finally installed by the SDN controller into flow table of SW_{out}. Prior to this flow installation, the rest of the response packets will be forwarded to V_i via $Path_i$. This will not change until all packets completely transferred and new distinct response packets arrived at the SW_{out}.

V. PERFORMANCE EVALUATION

The performance of our protocol is evaluated by using NS-3 simulator. We created a custom node as the RSU object as well as SDN Controller, Cloud Controller and VM. Three RSU groups are created with three RSU in each group. For each group, all of the RSU members are connected to a single OpenFlow aggregation switch and all of the switches are connected to a single core switch. From the core switch, dispatched four links are connected to four instances of the VMs serving four different types of requests. All OpenFlow switches are controlled by a single SDN Controller, while all of the VMs are controlled by also a single cloud controller. Finally, random numbers of requests are set to come from each RSU in each one second.

We compare our protocol with two existing protocols namely OSPF [9] and WARRA. OSPF works by selecting the shortest path of the network to reach the destination VM, while WARRA works as explained in our Related Work part. We also simulate our protocol using four different data types where each data type differentiated based on its request size and the time needed to handle certain size of the request by certain destination VM. Table I describes the details of the differences between all data types.

Fig. 4 and Fig. 5 show the comparison of DTT (Data Transfer Time) and DPT (Data Processing Time) for different types of requests. Overall, our protocol Smart Mobility Vehicular Communication Protocol (SMVCP) achieves significantly less DTT and DPT (below 4 seconds) for any type of request up to 2000 requests, simultaneously.

In Fig. 6, we can see the overall RTT of SMVCP as compared to other protocols. OSPF algorithm suffers from the highest average RTT. This is happened because OSPF will always choose the same shortest path resulting in the increasing number of queuing packets over time in each switch along the path. As the path load becomes more and more peak, a large number of packet will stay in the queue as the request continues to come. Furthermore, the condition is also worsen by the absence of target VM selection in the algorithm. On the other side, both of the WARRA and SMVCP do not experience

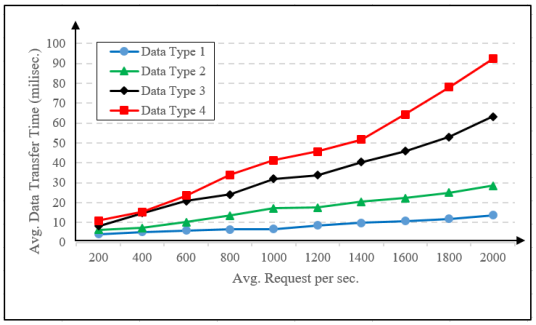

Fig. 4. Data Transfer Time (DTT).

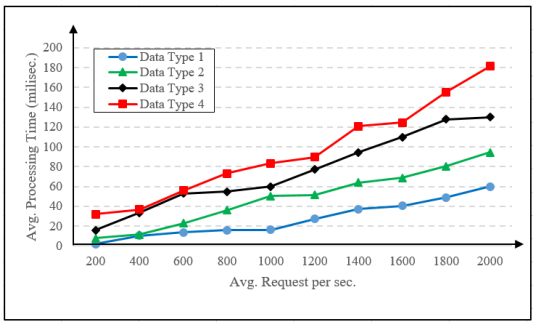

Fig. 5. Data Processing Time (DPT).

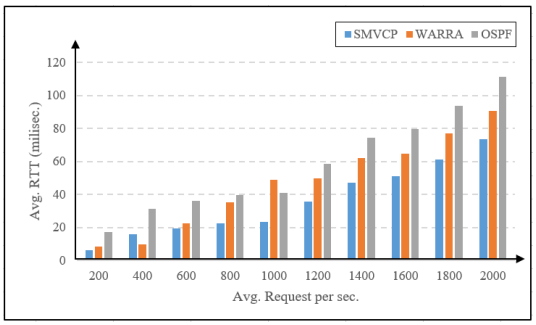

Fig. 6. RTT Comparison.

the same problem because both of the algorithm always check for the network path load and target VMs. Once, a path seen to be saturated the algorithms will quickly assign other paths to handle the rest of incoming request and the same thing also applied for each VM in the cloud. Consequently, the packets queue will be far shorter and the overall RTT also lessen.

In Fig. 7 we can see the throughput of SMVCP compared with other 2 protocols. Throughput in the simulation is affected by the rate of packet loss, means how many packets are missing their target. In the figure, both OSPF and WARRA experience a significant proportion of packet loss. This is because both of the protocols does not consider the location factor of the sender nodes (in this case, vehicles in city network). Hence, whenever any sender node disconnected from their RSU and then connected to any other RSU, the

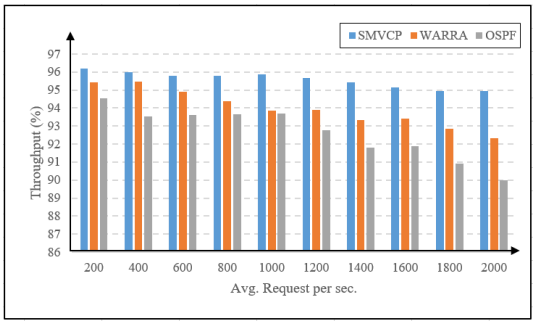

Fig. 7. Throughput Comparison.

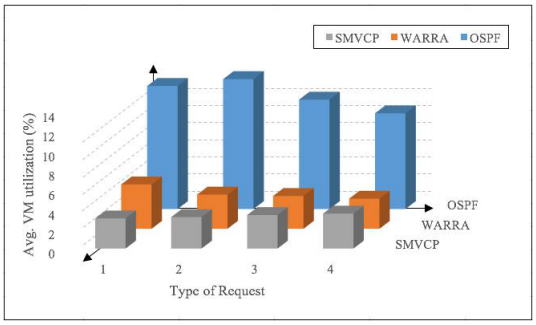

Fig. 8. VM Utilization Rate.

remaining response packet will fail to reach the sender node. Therefore both protocol's throughput is significantly reduced. However the such thing is rarely happened in SMVCP case because our proposed protocol always monitors the location of every sender node. As a result, there will be very less discarded packets.

Finally, Fig. 8 shows the variation of average VMs utilization. From the figure we can see that OSPF algorithm suffers from highest utilization percentage for all request type. This is understandable due to the fact that the OSPF algorithm works by selecting the shortest path only without considering the load of the VM. In contrast, both WARRA and SMVCP obtains almost utilization percentage load across all request types because both of the protocols considers also the load of the corresponding VMs.

VI. Conclusion

The existing SDN solutions for the cloud environment mainly work by minimizing latency via load balancing and traffic engineering in the area surrounding the network paths. However without considering the load of the target VMs, the significant amount of latency will still exist, which affects the overall performance of the service. In order to address the problem, protocol is designed to transmit data from vehicles to the city clouds through the Openflow switches. The proposed protocol minimizes the overall round-trip time by finding the best routing path and the most proper destination VM for specific type of requests. Furthermore, the protocol also guarantees the delivery of the corresponding response packets by determining current position of the sender vehicle. In the future, we would like to design a better solution to further support different parameters of request type such as request privilege and security.

References

[1] B. Kovacevic, M. Kkovacevic, T. Maruna, D. Rapic. *Android4Auto: a Proposal for Integration of Android in Vehicle Infotainment Systems*. IEEE International Conference on Consumer Electronics (ICCE), 2016: 99-100.

[2] M. Casado, M. J. Freedman, J. Pettit, et al. *Ethane: Taking Control of the Enterprise*. SIGCOMM Computer Communication Review, 2007, 37(4): 1-12.

[3] N. McKeown, T. Anderson, H. Balakrishnan, et al. *Openflow: Enabling Innovation in Campus Networks*. SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.

[4] S. Agarwal, M. Kodialam, T. V. Lakshman. *Traffic Engineering in Software-Defined Networks*. Proc. of the 32nd IEEE International Conference on Computer Communications, 2013: 2211-2219.

[5] S. Fang, Y. Yu, C. H. Foh, K. M. M. Aung. *A Loss-Free Multipathing Solution for Data Center Network Using Software-Defined Networking Approach*. IEEE Transactions on Magnetics, 2013: 2723-2730.

[6] X. Q. Meng, V. Pappas, L. Zhang. *Improving the Scalability of Data Center Networks with Traffic-Aware Virtual Machine Placement*. Proc. of the 29th IEEE International Conference on Computer Communications, 2010: 1-9.

[7] A. Nahir, A. Orda, D. Raz. *Schedule First, Manage Later: Network-aware Load Balancing*. Proc. of the 32nd IEEE International Conference on Computer Communications, 2013: 510-514.

[8] Haitao Yuan, Jing Bi, Bo Hu Li. *Workload-Aware Request Routing in Cloud Data Center Using Software-Defined Networking*. Journal of Systems Engineering and Electronics, 2014: 72-82.

[9] S. U. Malik, S. K. Srinivasan, S. U. Khan. Convergence time analysis of open shortest path first routing protocol in Internet scale networks, Electronics Letters, 2012, 48(19): 1188-1190.