A Temporal-information-based Adaptive Routing Algorithm for Software Defined Vehicular Networks

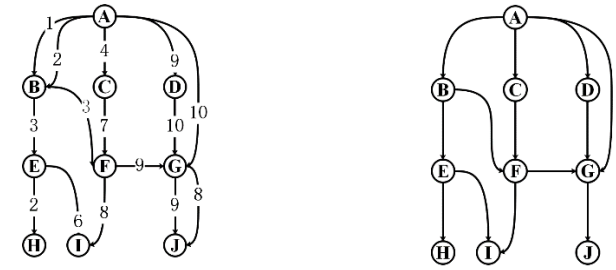
Liang Zhao[1], Zhuhui Li[1], Jiajia Li[1], Ahmed Al-Dubai[2], Geyong Min[3] and Albert Y. Zomaya[4]
[1] *School of Computer Science, Shenyang Aerospace University, Shenyang, China*
[2] *School of Computing, Edinburgh Napier University, UK*
[3] *College of Engineering, Mathematics and Physical Sciences, University of Exeter, UK*
[4] *School of Information Technologies, University of Sydney*
Corresponding Author: Liang Zhao (lzhao@sau.edu.cn)

***Abstract*—Most recent studies on Software Defined Vehicular Networking (SDVN) consider the vehicular network as a static graph and compute the flow table based on the static information. However, a static graph could only contain partial network data. Computation based on the static graph could be inefficient by missing the important temporal information since vehicular networks are temporal graphs in fact. Thus, in this paper, we propose a novel routing algorithm based on the Markov model and temporal graph. Unlike conventional routing algorithms, the proposed algorithm adopts the concept of a temporal graph where every edge has its specific temporal information. We apply the Markov model to predict the future routing of the network and use prediction data to get the optimal routing. This is achieved by running the temporal graph optimal path algorithm. The algorithm runs on the temporal graph constructed for SDVNs without generating additional routing overhead. In addition, based on the information of the vehicular network which is collected from the data plane, the controller can enhance the Markov model as time flows. By applying the above mechanisms, the flow table (route) could be calculated more precisely to enable efficient vehicular communication. The simulation experiments demonstrate the superiority of the proposed algorithm over its counterparts in high-density vehicular networks.**

Keywords-VANET, SDVN, Controller, Temporal Graph, Optimal Path

I. INTRODUCTION

Although distributed management fashion has been widely adopted, the vehicular ad hoc network (VANET) is still unable to optimally manage and fulfill the demands of Intelligent Transport System (ITS). Instead, as an emerging networking paradigm, Software Defined Vehicular Network (SDVN) breaks the limitation caused by the current architecture of vehicular communication [1]. Software Defined Vehicular Networking (SDVN) is a novel network architecture that aims to facilitate management of vehicular communication and enable programmatically efficient network configuration. The main idea of SDVN is to decouple the data plane from the control plane, which binds the interaction of vehicles more closely. SDVN employs a logically centralized network controller, refers to as the controller, which is mainly able to manage, mediate, and facilitate communication among network elements, and get the global network information from them. In VANETs, nodes forward the packets, and compute routing selection simultaneously. In contrast, SDVN decouples the control logic (by control plane) from the underlying routers and switches that forward the information (by data plane) [2]. In this context, the underlying networking devices are only required to focus on the forwarding packets. Hence, the controller collects the status of the network, and calculate the globally best routing path. This architecture allows more flexibility and quick response for network management. However, this architecture has still numerous challenging issues in implementing SDVN for central control and management of vehicular networking. One of the open problems is how to efficiently compute the most proper routing path for every vehicle in real-time and adjust the computing strategy according to the current network situation [3]. There are still requirements to meet in the control plane to calculate the optimal routing. For example, the massive data exchange among vehicles also brings the unbounded delay, packet loss, and network congestion during the packet transmission. [4]



(a) Temporal Graph (b) Static Graph

Figure 1. Temporal Graph G and its corresponding Static Graph G'

In SDVNs, most existing studies treat the vehicular network as a sequence of static graphs. Each static graph represents the nodes and links at a specific timestamp. Then, the flow table is calculated by applying the static routing algorithm (normally a static shortest path algorithm like Dijkstra or Bellman-Ford) based on a single static graph. However, real vehicular networks are actual temporal graphs [5]. The nodes are highly dynamic, and the existence of links may only last for a short period. In such networks, the node communicates with another node at a specific time while each edge of a vehicle temporal graph has its specific temporal information. Therefore, computing routes based on a static graph without real-time network information could lead to packet loss and heavy delays in the later forwarding stage. For example, Fig.1(a) shows a temporal graph G of a vehicular network, while Fig.1(b) shows its corresponding static graph. As stated above, each edge has its temporal information which is represented by (u, v, t, d) indicating that the edge from u to v starts at time t, and it survives for d timestamps until the packet has successfully arrived at v [6]. In fact, there could be

multiple edges between u and v indicating their relationships at different timestamps. For simplicity, let us assume that the duration of each edge in Fig.1(a) is 1 while the number on each edge is its starting time. Now, we can calculate the shortest path from A to J. In Fig.1(b), one of the shortest paths is $< A, D, G, J >$ with distance 3 obviously. However, in Fig.1(a), it is impossible to find any accessible path from A to J. It is because that we can only get J from G but the latest departure time from G is time 9 and the earliest-arrival time at G is time 10. The difference makes J unreachable if we start from A. Fig.1 shows the traditional static graph can produce misleading information in dynamic traffic topology. This also works when we calculate the routing path (flow table) in the next timestamp in SDVNs. First, we assume that each node is a vehicle, or an infrastructure and the edge represents a vehicle transmitting the packet to another vehicle. The edge (or link, that was valid in the past) could be broken before the packet arrives at the destination (receiver). If the above takes place, the new route to retransmit the packet is required to generate in the controller of SDVN which causes packet loss, routing overhead, and delay [7]. Hence, it is essential to keep the temporal information in SDVNs.

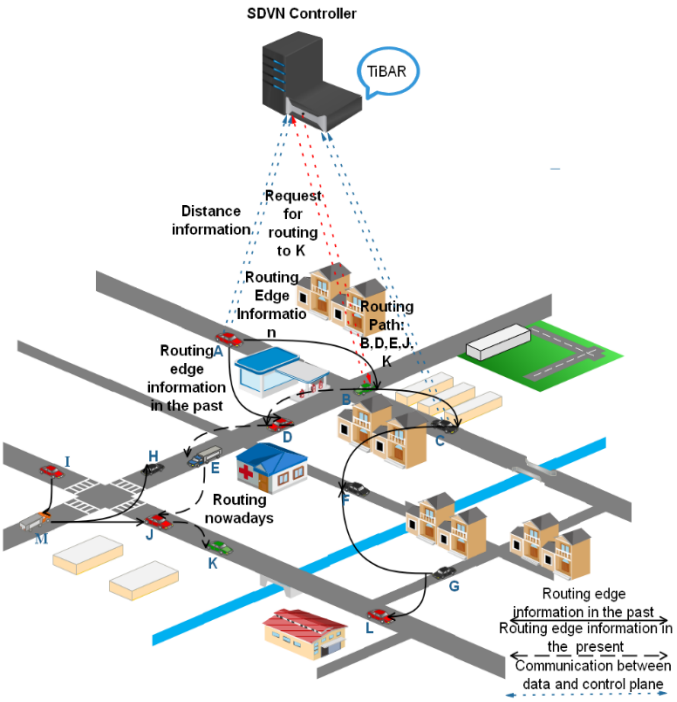


Figure 2. A typical urban scenario for the vehicle B requesting for transmitting traffic information to K in our architure.

However, existing studies on SDVN controllers mainly focus on calculating the flow table based on the static topology whereas the road network is, in fact, a temporal graph. Based on the recent study of temporal graphs [8,9,10], the optimal path algorithms demonstrate the time complexity of $O(m+n)$ and the space complexity of $O(n)$. Meanwhile, if we apply a greedy strategy to calculate the optimal path in a similar way to Dijkstra's algorithm, with the requirement of a minimum priority queue, the greedy strategy could be highly inefficient compared to the optimal path algorithm of the temporal graph as they achieve a time complexity of $O(m \log \pi + m \log n)$ and a space complexity of $O(M+n)$ [8]. Thus, it is necessary to consider temporal information in the routing of SDVNs. Therefore, in this paper, we propose a routing algorithm for SDVNs, namely Temporal-information-based Adaptive Routing (TibAR).

At first, to compute the routes for the packet transmission in the next timestamp, the controller needs to predict the required temporal information for such computation. Here we define the history temporal information which we use for the prediction in SDVNs

DEFINITION 1 (Routing Edge Information). *During a packet transmission in SDVNs, a packet on the optimal route sent from vehicle i reaches the next vehicle j using optimal routing. The temporal information of this routing edge is collectively called routing edge information. The routing edge information, like the edge of temporal graph, include the sender node i, the receiver node j, the starting time t and the duration d.*

With the information collected from the data plane, we construct a Markov model based on the historical Routing Edge Information (REI) and distance information. Then we predict the possible REI from the source to the receiver. Finally, as the input of an efficient optimal path algorithm, the predicted REI is used to calculate the optimal routing. The contributions of this paper can be summarized as follows.

- To the best of our knowledge, the current work is the first to adopt temporal graphs for routing in SDVNs.
- A Markov routing prediction model is proposed by considering historical REI and distance information. When the traffic topology changes from time to time, the model will be updated adaptively.
- An efficient optimal path algorithm with a linear computation time for the temporal SDVNs is proposed by applying the properties of temporal graphs.

The rest of the paper is organized as follows. Section II presents the application scenarios and the proposed architecture, model, and algorithms that work in the scenarios. Section III evaluates the performance of our proposal. We conclude the paper in Section IV.

II. APPLICATION SCENARIOS, SYSTEM MODEL AND ALGORITHMS

A. System Model

As for specific application scenarios, Fig.2 shows the system for routing calculation of the proposed algorithm. Vehicles are equipped with GPS for geographic location management. In this model, vehicles periodically send beacon messages to their one-hop neighbors. The neighbors receive the beacons and calculate the distance to the sender based on the GPS position information carried in the beacon messages. Then the receiver forwards the distance information to the SDVN controller. When a vehicle node on the optimal routing path receives the packet, it is required to send the present REI to the controller. When the controller receives the routing request from a vehicle, our proposed optimal path algorithm is then applied to compute the optimal routing path (flow table) efficiently. This algorithm is

fully based on the distance and history REI received from the data plane (vehicles).

B. Algorithm components and details

Temporal graphs record two types of temporal information. The first one is temporal information which has occurred such as all REI during the last hour. The second one is routine flight information like an air-transportation network. In order to calculate the routing at the next timestamp, we should predict enough REI in the future. Hence, we use the Markov model constructed by the historical REI and distance. Here we need to explain why we consider distance as a vital factor. Distance is directly connected with the quality and the delay of this edge in SDVNs. The longer the edge normally leads to a more fragile link. Moreover, the delay grows with the increase of the distance. It is worth mentioning that, in this paper, we will consider the duration of an edge as the distance between two nodes of this edge since the delay of a data packet is highly related to the distance.

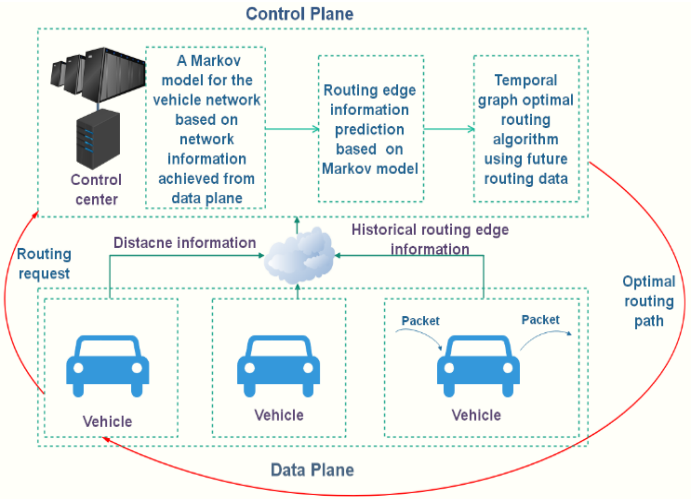


Figure 3. The framework of Temporal-information-based Adaptive Routing (TibAR) Algorithm in a SDN architecture.

Hence, this section shows how to construct a Markov model, how to apply the model to get enormous REI, and how to calculate the optimal routing based on the REI. To construct the Markov model for predicting REI, the controller needs to get distance and historical REI of the entire network from the data plane. Fig.3 shows the communication between two agents of an SDVN and the application of our proposal.

In this section, we first present the process of constructing the Markov model by achieving the transition index between two nodes, which possibly communicate with each other in the future. Then, we propose the Forward Prediction algorithm and the Reverse Prediction algorithm to generate enough REI for the future routing. Among the predicted REI, the requester and the destination are connected. Finally, we introduce an efficient optimal path algorithm to get the optimal routing for temporal SDVNs.

1) Constructing the Markov Model

To fully utilize historical REI, a prediction matrix of the Markov model is constructed by linking each state to each vehicle in the network. We define the possibility of transiting state i to state j as transition index from i to j. The transition index from i to j is denoted by i_{ij}. With all transition index between each node pair in the vehicular network, a Markov model is constructed [11]. Now the question is how to compute each transition index between every two nodes who had or could have a connection with each other in the future. We consider that there are two factors: historical REI and distance. We shall not state the importance of distance between two vehicles in detail as it is very clear that the closer distance means a higher success rate of delivering data packets, higher data rate, and less overhead. Meanwhile, the historical REI indicates there was a successful delivery in the past. This link has a big chance to be stable and reliable now. Of course, the longer time it has passed, the less important the historical REI will be. Then the weight of this REI will be smaller. Thus, i_{ij} between i and j is calculated as follow:

$$i_{ij} = \frac{k}{d_{ij}} + \frac{l}{\sum_{h=0}^{p}(t_p - t_h) * r_{ij}^{t_h}} \quad (1)$$

In equation (1), i_{ij} denotes the transition index from node i to node j. k and l are the weight of distance and historical REI respectively. The sum of k and l is always 1. The bigger the k value is, the more important the distance information is. On the contrary, the importance of historical REI grows with the increase of the value of l. d_{ij} represents the distance between node i and node j at the present time, t_i means the time value at time i. $i = 0$ means the time that at which the first historical REI happened. Meanwhile, $i = \mathrm{p}$ denotes the present time value and $r_{ij}^{t_h}$ is the routing valid value from i to j at time h. If at time h, node i was transmitting packets to node j; i.e., this routing from i to j was happening, the value of $r_{ij}^{t_h}$ is 1, otherwise this routing valid value is 0.

With the historical REI (e.g., the REI in the last hour) and the current distance information of vehicular network obtained from the data plane, we can compute the transition indexes for each pair of the nodes in the network at the initialization phase. That means that we only need to do it once at the beginning of each time period. These computed indexes are stored as entries of a two-dimensional N × N matrix where one dimension means the node of the current state and the other dimension corresponds to the state in the next timestamp. At this point, the initial state of the Markov model has been constructed.

2) Prediction of REI with Markov Model

In this part, we denote how to use this model to predict enough REI values for the shortest-path algorithm. There are two main problems for the predicted data in our proposal. First, the destination (receiver of a data packet) must be reachable for the requester among these future data, i.e., the reachability between the source vehicle and the receiver vehicle. We need to guarantee that we can extract at least one connected routing with right time sequence among all the predicted REI. Second, the number of predicted REI need to be within an appropriate range. In our later simulation, the maximum number of nodes is 4000. According to that our Markov model is a 4000×4000 matrix, if we predict the REI iteratively for each involved node for certain times, the number of REI could be too big to compute

further. If the number of predicted REI is small, we cannot guarantee the reachability at all.

To solve these two problems, we present a Bidirectional Recursive Prediction Method Based on the Markov model. We use the pseudo-code in Algorithms 1 and 2 to show the procedures described below. Before the prediction, we need to create an empty sorted list L to store the prediction result ordered by the REI starting time. And then the algorithm officially starts. First, we linearly scan M[s], i.e., the transition index for start point s to any other point. If the transition index between s and j is above the average value of the sum of transition index start with s (Line 5), we consider this REI is qualified and insert this edge into L (Line 6 and 7). Then, we use the Forward Prediction recursively, the starting point is the endpoint of this REI j and the starting time is $t_a + d_{sj}$. In addition, iteration times plus 1. Eventually, when the routing calculation reaches at the endpoint or the iteration time equals MaxTime, this branch then reaches its end. After all recursion have finished, this algorithm is over, and all the predicted REI are stored in L ordered by their starting time.

Algorithm 1: **Forward Prediction** $(\mathrm{M}, \mathrm{s}, \mathrm{d}, t_a, t_w, t, L)$

Input: A Markov model M include N nodes in its $\mathrm{N} \times \mathrm{N}$ matrix representation, start point s, end point d,time interval $[t_a, t_w]$,iteration time t and a sorted list L to store predicted REI ordered by their starting time; initially, L is empty;
Output: a series of predicted REI in the future ordered by their start time

```
1 if s = d or t ≥ MaxTime then
2         return;
3 end if
4 for each node j in the M do
5         if M[s][j] > AveValue[s] then
6                 edge e ← (s, j, t_a, d_sj);
7                 insert e into L;
8                 Forward Prediction (M, j, d, t_a + d_sj, t_w, t + 1, L)
9         end if
10 end for
```

The Forward Prediction can generate the proper amount of REI. However, we cannot guarantee reachability. In this case, we can predict the REI reversely from the endpoint to the start point. In that way, if these two parts of REI could be linked together in the right time sequence, there must be at least one optimal and linked routing among the data. Therefore, we design Algorithm 2 to solve this problem.

The main body of this Reverse Prediction algorithm is similar to the Forward Prediction algorithm. Compared with the latter, there are two main differences in order to increase the reachability between two nodes. (1) the Reverse Prediction starts at the endpoint. Each recursion takes the starting point of the previous step as the endpoint. It is like that we predict the REI reversely from the end to the beginning. Its termination condition is that the end meets the starting or the iteration time equals MaxTime, same as which of the Forward Prediction. (2) we set the termination time (the starting time of an edge + its duration) of the last edge end with d. Then we execute this algorithm post orderly. Once we meet the edge whose starting point has happened in L as the endpoint of the edge (Line 7). Note that, the edges in L are the edges we have predicted in the Forward Prediction. We adjust the starting time of the linked edge to the minimum end time of the involved edges in L (Line 8). As we do it post orderly, the previous prediction can get the temporal information from the back one which is closer to the starting point. In this method, we can link the REI predicted from the Forward Prediction algorithm and the Reverse Prediction algorithm together. By running these two algorithms and adjust the MaxTime value, we can guarantee the reachability as well as the proper amount of REI.

Algorithm 2: **Reverse Prediction**$(\mathrm{M}, \mathrm{s}, \mathrm{d}, t_a, t_w, t, L)$

Input: A Markov model M include N nodes in its $\mathrm{N} \times \mathrm{N}$ matrix representation, start point s, end point d, time interval $[t_a, t_w]$, iteration time t and a sorted list L to store predicted REI ordered by their starting time after **Forward Prediction**;
Output: a series of predicted REI in the future ordered by their start time

```
1 if s = d or t ≥ MaxTime then
2         return;
3 end if
4 for each node j in the M do
5         if M[j][d] > AveValue[j] then
6                 Reverse Prediction (M, s, d, t_a, t_w − d_jd, t + 1, L);
7                 if there are edges es end with j in the L then
8                         t_w ← min end time tt in es + d_jd;
9                 end if
10                edge e ← (s, j, t_w − d_jd, d_jd);
11                insert e into L;
12        end if
13 end for
```

3) Temporal Graph Routing Algorithm

After predicting enough REI in the future, our method uses the optimal path algorithm based on temporal information to calculate the optimal routing we need. As for the optimal path for the temporal graph, Wu et al. propose four shortest path algorithms in [8]. Here we mainly choose the first one to compute the earliest-arrival time, which has the same goal with routing. The main idea of this algorithm is described as followed. First, we scan the edges by the sequence of their starting times. We have achieved this goal as L in the prediction procedure. L is a REI sequence ordered by their starting time. Then we set a list $t[N]$ to record and manage the earliest-arrival time of each node. Finally, we check the temporal graph to scan each edge $e(u, v, t, d)$ in L by the right sequence of their starting times. If the starting time of e is latter than the present earliest-arrival time of u and the end time $(s + d)$ is earlier than the present earliest-arrival time of v, we consider this edge is a part of the optimal route at the present stage and update the earliest-arrival time of v. Since the edges are scanned ordered by the right time sequence, we certainly will not miss any possible optimal routing.

Hence the main problem of this algorithm is the input, which is an edge sequence order by their starting times. We have already achieved this as L in the prediction procedure. However, except for the earliest-arrival time, we also need the optimal path node information. So, we bring a list for each node in our algorithm to record previous hop nodes on the optimal path. Because there may be multiple previous hop nodes on the optimal path. i.e. there are multiple optimal paths with the same earliest-arrival time. Therefore, we will record multiple previous hop nodes instead of one. After the algorithm, we will get the earliest-arrival time and its corresponding previous hop nodes list for each node on the optimal path. As we scan the previous hop nodes list, we can get the node sequence on the optimal path easily. Then we present the detail of the algorithm as following in Algorithm 3.

Algorithm 3: **Computing Optimal Routing**

Input: A sorted edge list L ordered by their starting time, start point x, time interval $[t_a, t_w]$;

Output: The earliest-arrival time from x to every node, the previous nodes list for each node on their specific optimal routing.

```
1 initialize t[x] = t_a, and t[v] = ∞ for each node v except x;
2 initialize prev[v] for each node v to make sure each one is empty;
3 for each edge e = (u, v, t, d) in L do
4       if t ≥ t[u] then
5             if t[u] + d < t[v] then
6                   initialize prev[v];
7                   insert u into prev[v];
8                   t[v] ← t + d;
9             else
10                  if t[u] + d = t[v] then
11                        insert u into prev[v];
12                  end if
13            end if
14      end if
15 end for
16 return t[v] and prev[v] for each node v;
```

Line 1 and 2 are to initialize the earliest-arrival time and previous node for each node; we use array $t[N]$ and $prev[N]$ to represent them. We filter the edges for the first time at Line 4. If it departure at u before the earliest-arrival time of u, we will not take this edge into account. Then we will update the earliest-arrival time and rebuild the previous nodes if the end time of this edge is earlier than the earliest-arrival time of v at Line 5-8. The reason of rebuilding is that this could be a brand-new routing. If the end time equals the earliest-arrival time, we just add u into $prev[v]$. At last, we end this algorithm when all edges have been scanned. The $t[N]$ and $prev[N]$ are the optimal routing result.

4) Improvement of Temporal-information-based Adaptive Routing

The calculation in our algorithm are mainly concentrated in the construction of the Markov model. If once the controller receives the routing request, it computes a specific Markov model for the entire network. The algorithm becomes too complicated. We need to adjust the original Markov model adaptively [12]. Thus, we propose an improved method. At the beginning, similar to the conventional model one, we construct the Markov model; predict REI and compute the optimal routing. The difference is, in the new method, once the Markov model is constructed, we do not have to construct a brand-new one when the latest information comes. We can update the related information of the Markov model instead of deleting the past one and creating a new one. Once the Markov model obtains the distance information update between vehicle u and vehicle v, the transition index updates as defined in (2).

$$i_{uv} = i'_{uv} + \frac{k}{k+l}(d_{uv} - d'_{uv}) \tag{2}$$

i_{uv} denotes the goal transition index between u and v we need. i'_{uv} is the transition index we already have, and it is out of date. k and l are the weight parameters for distance and historical data mentioned before. d_{uv} and d'_{uv} are the present and historical distance information between u and v. We adjust the transition index adaptively according to the distance in the present and past. Similarly, we adjust the transition index when packet transmission from u to v happens, as described in (3).

$$i_{uv} = i'_{uv} * \frac{l}{k+l} * de \tag{3}$$

de refers to the withering factor which is in the range of 0 and 1. It is not wise to let one edge takes over too much transmission work which may cause unbalanced load. Therefore, at one timestamp each time packet transmission happens between u and v, the importance of edge from u to v decrease. the transition index declines too. At last, the average value of transition indexes of u is required to change correspondingly.

$$AveValue[u] = \frac{AveValue[u]' * N + (i_{uv} - i'_{uv})}{N} \tag{4}$$

To guarantee the accuracy, after a fixed time interval, we restart a process to recalculate the Markov model based on the latest distance and REI. These data have been collected by the control plane already.

III. SIMULATION

In this section, we present the detail of the simulation including parameters and evaluations. We ran all the experiments on a PC with Windows 10 on an Intel 2.4 GHz CPU and 16 GB RAM. We build an SDVN simulation platform with C++ programming language based on the changes of the positions of vehicles. The position is produced from SUMO [13]. Moreover, the historical REI is produced by NS-2 under the protocol GPSR. In the simulation, four different numbers of nodes have been considered. The main parameters of the vehicular network are described in Table 1.

TABLE I. PARAMETERS OF SIMULATION

Parameter	Value
Number of vehicles	500/1000/2000/4000
Vehicle velocity	1 - 60 km/h
Vehicle movement range	1200 km * 850 km

Parameter	Value
MAC protocol	IEEE 802.11p
Transmission Range	250 m
The simulation duration	100 s

TibAR is compared with the existing scheme of the Dijkstra algorithm. We send the routing request for the random vehicle to the SDVN controller from a random vehicle once per second. We mainly consider two performance parameters to evaluate the performance of TibAR. First, the average calculating time for routing is used as the average time for the controller to complete the calculating the optimal routing for each routing which includes the construction of Markov model and updating. Second, the average delay of routing is applied as the average delay of each routing form the source node to each other nodes (we use a relative value to represent the level of latency in this paper).

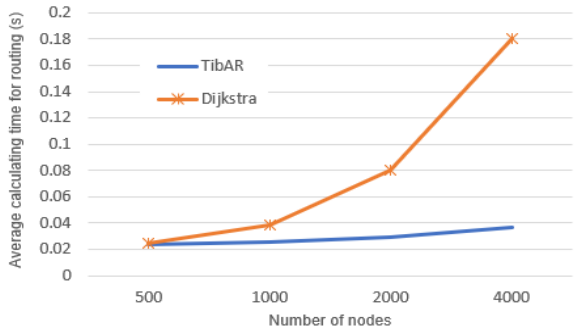


Figure.4 Average calculating time for routing v.s. the number of nodes

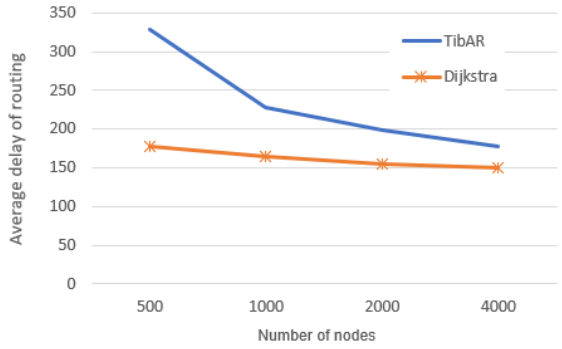


Figure.5. Average delay of routing v.s. the number of nodes

As shown in Fig.4, the difference between the efficiency of TibAR and Dijkstra is not apparent in the low-density traffic scenario (500 nodes). However, with the increase of nodes, TibAR gradually outperforms Dijkstra. In particular, the efficiency of TibAR could be six times compared to Dijkstra as more edges are scanned. With the increase of nodes, Dijkstra requires to deal with the number of edges growing at $O(n^2)$ speed. Meanwhile, in TibAR, by applying the proposed Forward Prediction and Reserve Prediction algorithm, the number of scanning edges is within a proper range. As shown in Fig.5, the Dijkstra algorithm calculates the absolute optimal path in the vehicular networks, while Dijkstra outperforms TibAR in the routing delay. However, with the increase of nodes, TibAR could obtain much more training data in the constructing process which results in a reduction of latency. As we can see, the routing delay produced by TibAR is very close to the absolute optimal path when the number of nodes reaches 4000. We observe that TibAR outperforms the main existing routing schemes in computing efficiency and routing latency in the high-density traffic scenarios of SDVNs.

IV. CONCLUSIONS AND FUTURE WORK

Our study reveals that existing routing schemes do not consider the temporal information. They view the vehicular network as a sequence of static graphs. This could lead to the missing of temporal information and cause the additional overhead. On the other hand, the shortest path algorithm in the temporal graph has a better performance in efficiency. To fill in this gap, we proposed a SDVN routing scheme namely, TibAR. Instead of calculating the static graph, TibAR used the properties of the temporal graph to compute optimal routing. We also construct a Markov model based on the REI and distance information to produce the REI as the input of optimal path algorithm. Our simulation results show that TibAR outperforms existing SDVN routing schemes in the high-density traffic scenarios. For future work, we would like to optimize the Markov model to achieve more precise predictions and allow our work to adapt to dynamic topology.

REFERENCES

[1] Nadeau, Thomas D., and Ken Gray. SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies. O'Reilly Media, Inc.", 2013.

[2] A. Y. Al-Dubai, L. Zhao, A. Y. Zomaya and G. Min, "QoS-Aware Inter-Domain Multicast for Scalable Wireless Community Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 3136-3148, 1 Nov. 2015.

[3] D. Kreutz *et al.,* "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[4] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[5] P. Holme and J. Saramäki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97–125, 2012.

[6] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[7] L. Zhao, A. Al-Dubai, X. Li, G. Chen, and G. Min, "A new efficient cross-layer relay node selection model for Wireless Community Mesh Networks," *Computers & Electrical Engineering*, vol. 61, pp. 361-372, 2017.

[8] H. Wu *et al.,* "Path problems in temporal graphs," *Proc. VLDB Endow.*, vol. 7, no. 9, pp. 721–732, 2014.

[9] H. Wu, Y. Huang, J. Cheng, J. Li, and Y. Ke, "Reachability and time-based path queries in temporal graphs," in the *Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, Helsinki, Finland, May. 2016 - May. 2016, pp. 145–156.

[10] B. B. XUAN, A. FERREIRA, and A. JARRY, "Computing shortest, fastest, and foremost journeys in dynamic networks," *Int. J. Found. Comput. Sci.*, vol. 14, no. 02, pp. 267–285, 2003.

[11] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[12] H. Li, K. Ota and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," *IEEE Network*, vol. 32, no. 1, pp. 96-101, Jan.-Feb. 2018.

[13] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal on Advances in Systems and Measurements*, 5 (3&4):128-138, December 2012.

[14] L. Zhao et al., "Vehicular Communications: Standardization and Open Issues," *IEEE Communications Standards Magazine*, 2019