Towards the Partitioning Problem in Software-Defined IoT Networks for Urban Sensing

Chao Song*†, Jie Wu‡, Xu Chen*, Lei Shi*†, Ming Liu*†
*School of Computer Science and Engineering, University of Electronic Science and Technology of China
†Big Data Research Center, University of Electronic Science and Technology of China
‡Department of Computer and Information Sciences, Temple University
Email: {chaosong, csmliu}@uestc.edu.cn, jiewu@temple.edu

***Abstract*—Software Defined Networks (SDN) have been proposed for use in applications of the Internet of Things (IoT), termed as software-defined IoT (SD-IoT) network, because of the popularity and capability of mobile devices being used for networking in relatively large areas. However, a single controller in SDN has a limited request-processing capability, so a distributed control plane with multiple physical controllers has been used to achieve scalability and reliability for supporting the IoT applications. Accordingly, the data plane of an SDN is partitioned into multiple domains, and each controller just takes over one. When considering both delays and loads of requests to the controllers, a partitioning problem arises. It is required to consider the distributions of flow paths, since inter-domain flow paths will create an extra load of requests to the controllers. In this paper, we investigate the partitioning problem in SD-IoT networks. Since uploading sensing data through the IoT gateways are non-uniform, we utilize a hypergraph to model the relationship between the spatial events and the gateways in IoT for urban sensing. We propose a Partitioning Algorithm for Software-defined IoT Network (PASIN) to partition the SDN by considering both delays and loads of requests from the flow paths. Our extensional simulations verify the effectiveness of our proposed approach.**

I. Introduction

As the Internet of Things (IoT) is fundamental to realizing urban sensing, it should be flexible enough to support various application requirements and the convenient management of infrastructure [1]. Consequently, a Software-Defined IoT (SD-IoT) network has been proposed for smart urban sensing [1], [2], and has received a growing amount of attention from both academic research communities and industrial companies. Recently, the applications of IoT is developing rapidly, and its scale is also growing. For example, Mobike is the worlds first and largest smart bike-sharing company. The company officially launched its service in Shanghai in April 2016 and in less than a year since then has expanded the service to over 80 cities across China and internationally, operating nearly 4.5 million smart Mobikes. In 2017, Qualcomm announced that they plans to support Mobikes Internet of Things (IoT) applications by MDM9206 global multimode LTE modem [3]. This IoT device is designed to help Mobike customers accurately identify an available bike, accelerate the unlock process of the smart lock and assist with real-time management, all while providing Mobike with continuous monitoring of the bikes status. Thus, one important challenge of such large-scale IoT network is the large amount of the asynchronous data transmission among these IoT devices.

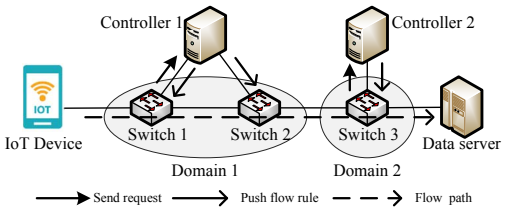


Fig. 1. An illustration of inter-domain flow path scenario.

However, an SDN with a single controller cannot cope with such large-scale IoT system that provides vast amounts of data flows, even under multiple IoT applications. Deploying multiple controllers is a general approach to provide scalability and reliability, as more physical devices and applications are added to the network [1], [2], [4]. In an SDN with multiple controllers, each controller directly handles all the switches in a given domain, as shown in Figure 1. A new data transmission will send a request to the controller for establishing a flow path. Then, the controller will calculate the flow path and push the flow rules to the flow tables of the switches along the path. Thus, the switch-to-controller delays are crucial, and many studies [2], [5] propose to minimize them for partitioning the data plane of SDN into multiple domains. However, such solutions prefer employing more controllers to limit the scope of each domain, because a small-scale domain has both lower total delay and load of requests to its controller. Thus, they make the flow paths in SDN running through more domains, which can increase the load of requests in the whole network. This is because a flow path of data transmissions through two or more domains will send a request to the controller of each domain, and create an extra load of requests to the controllers of these domains. In Figure 1, the data transmission from the host to a data server is through two domains, and therefore will send a total of two requests to these controllers. Therefore, partitioning an SDN should minimize the total load and delay of the requests by considering the distributions of these flow paths.

In this paper, we investigate the partitioning problem in software-defined IoT networks over multiple flow paths. The network traffic along these flow paths is generated by urban

978-1-5386-3224-6/18/$31.00 ©2018 IEEE

sensing applications that detect spatial events. In such applications, sensor nodes (such as vehicles and pedestrians) with IoT devices sense spatial events on the roads (such as potholes [6]) in a pre-defined time and in a pre-defined area. Then, the sensor nodes upload the sensing data to gateways which request to build flow paths to remote data servers in the SDN. The relationship between the spatial events and the uploading gateways is influenced by their spatial locations and the distribution of mobile sensor nodes. We model this relationship as a hypergraph to analyze the non-uniform distribution of input data to SDN. According to the model, we propose a Partitioning Algorithm for Software-defined IoT Network (PASIN) to achieve the minimum total load of requests with the constraint of total switches-to-controller delays in each domain. Therefore, we focus on the partitioning problem for urban sensing applications in an SDN with multiple controllers and we make the following three main contributions:

- We discuss the load of the requests on each controller caused by the flow paths. We take an urban sensing system supported by an SDN to illustrate the data transmissions over the flow paths;
- We model the relationship between spatial events and gateways using a hypergraph. We propose a Partitioning Algorithm for Software-defined IoT Network (PASIN) to partition SDN by considering the flow paths for the urban sensing applications;
- We verify the performance of PASIN using the scalable testbed we developed for SDN simulation.

The remainder of this paper is organized as follows: section II discusses the cost of flow path and introduces the urban sensing system with SDN; section III investigates the partitioning problem in SDN; section IV evaluates the performance of the proposed approach; section V surveys the related work; and the last section concludes this paper and contains our plans for future work.

II. Flow Paths in SD-IoT

In this section, we first discuss the influences of flow paths on the controllers. Then, we introduce the framework of an urban sensing system for detecting spatial events that is supported by an SDN with multiple controllers.

A. Overview of Software-Defined IoT System

With an increase in the number of mobile applications, many of the applications are getting a lot of attention from both academic researchers and industries [7]. Many cities have deployed sensor platforms to support urban sensing [1]. In addition to dedicated sensor platforms, smart phones equipped with a rich set of sensors (like cameras, digital compasses, GPS, etc.) can also be exploited to realize urban sensing. This is referred to as mobile crowd-sensing (MCS) [8]. Mobile crowdsensing for detecting spatial events, such as pothole detection [6], is a typical and popular urban sensing application. The ubiquity of smartphones has led to the emergence of mobile crowdsensing tasks that make use of the way that smartphone users move around in their daily lives [9]. In a typical urban sensing application, the system is comprised of a central platform and a collection of IoT devices.

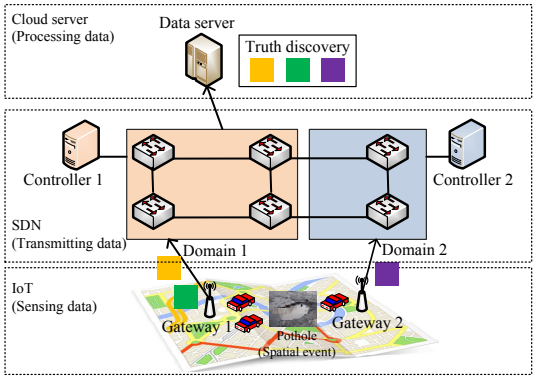
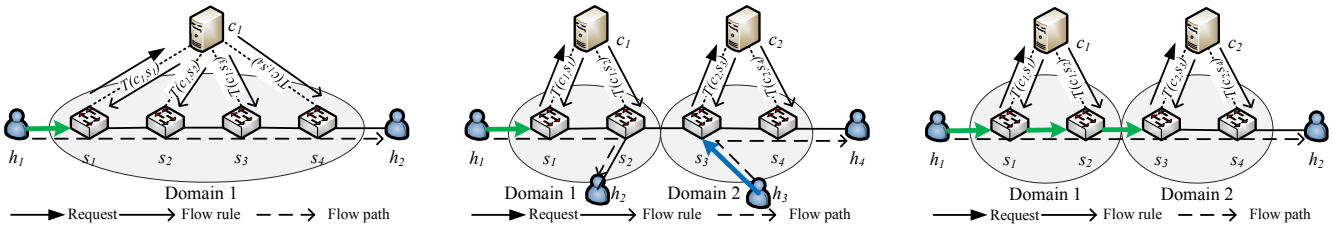


Fig. 2. The framework of a SD-IoT system.

Figure 2 demonstrates the typical framework of a software-defined IoT (SD-IoT) system for urban sensing applications. The system can be logically divided into three subsystems: IoT for sensing data, SDN for transmitting data, and cloud server for processing data. In this example, the IoT devices in the bypassing vehicles sense the spatial events on the roads. When a mobile device senses an event, it generates sensing data to describe it. Then, the sensing data is required to be uploaded to the data server for further aggregation or to find the truth of the event. Usually, the IoT device in a vehicle first transmits the data to a gateway which then transmits the data to the remote server through the networks. The IoT devices can immediately communicate with a gateway via the cellular network (3G/4G); otherwise, it can transmit the report delayed offloading through an access point act as gateway [10]. Therefore, a large number of data flows is transmitted through the gateways.

SDN in such a system takes responsibility for transmitting the sensing data from the gateways to the data servers. It separates the network into a control plane with a collection of network-attached servers and a data plane with programmable and packet-flow switches. The control plane makes decisions about how traffic is managed based on a global, logically-centralized network view. The data plane actually forwards the data traffic to the desired destinations, and the flow paths are handled by the SDN controller. Each gateway or data server directly connects a switch in the data plane. The sensing data is uploaded from the gateways to the data servers and is forwarded in the data plane. The data plane of SDN is partitioned into several domains, and each controller takes control of a domain. Because the switches have limited buffers and the static flow entries cannot adapt the dynamic networks, the switches cannot store all the possible flow rules for the paths between the gateways and the server. The data server harvests all the sensing data and processes them according to the algorithm of truth discovery [9]. In this paper, we focus on the problem of data transmission in such urban sensing applications. The issues of sensing data [6] and processing data [9] are not within the scope of this paper.



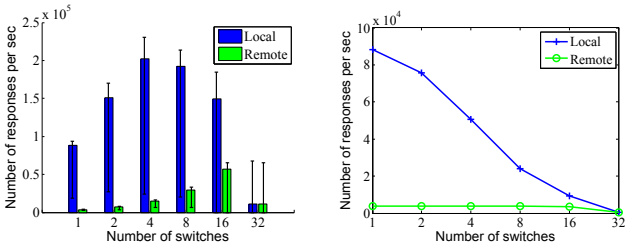
(a) One intra-domain path (single controller) (b) Two intra-domain paths (double controllers) (c) One inter-domain path (double controllers)

Fig. 3. The influences of flow paths on the controllers.

B. Cost of Flow Path

In SDN, the load on the controller is caused by the request of each data transmission. As illustrated in Figure 3(a), the network contains a single controller c_1, and four switches (s_1, s_2, s_3, and s_4). The host h_1 sends a data packet to another host h_2. When the first switch s_1 receives the data packet, it will check its flow table. If it misses, i.e there is no entry for flow the data packet from h_1 to h_2, the switch s_1 will send a request to its controller c_1. Many reasons result in missing the requested entries on the flow table, such as the timeout of an old entry or the limited buffer of a switch. Then, the controller calculates the flow path, and sends the flow rules to the switches along the path in order to update their flow tables. This flow path is called an *intra-domain* path since both its source and destination directly connect to the same domain and its data transmission is also in this domain.

However, the request-processing capability of a single controller is limited [2], and in a large-scale network, the number of requests can exceed the limitation of a single controller. The request-processing capability of a single controller is limited; for example, NOX can process about 30K requests per second [2]. We utilized CBench [11] to test the request-processing capability of the controller FloodLight [12] which runs on the physical server has the CPU of Intel i5-6500 3.20GHz with 4 cores and 12 GB memory. CBench simulates the specified number of OF switches which connect to the controller to test its request-processing capability. Our experiments included two modes: in local mode CBench runs on the same physical server with the controller, and in remote mode CBench runs another computer which connects to the controller through a 100 Mbps Ethernet link. As shown in Figure 4(a), while increasing the number of connected switches, the amount of the processed requests by the controller first increases when less than a certain number of switches, and then the amount of the processed requests decreases. Under the local mode, the amount deceases when the number of the connected switches is more than 4. Under the remote mode, the amount deceases when the number of the connected switches is more than 16, because of the bandwidth of the link between the controller and CBench. When the number of switches is 32, the amount of the requests reduce the lowest under both of the two modes. As shown in Figure 4(b), while the number of connected switches is increasing, the average number of responses received by each switch is decreasing under both of local mode



(a) Total responses of all switches (b) Average responses per switch

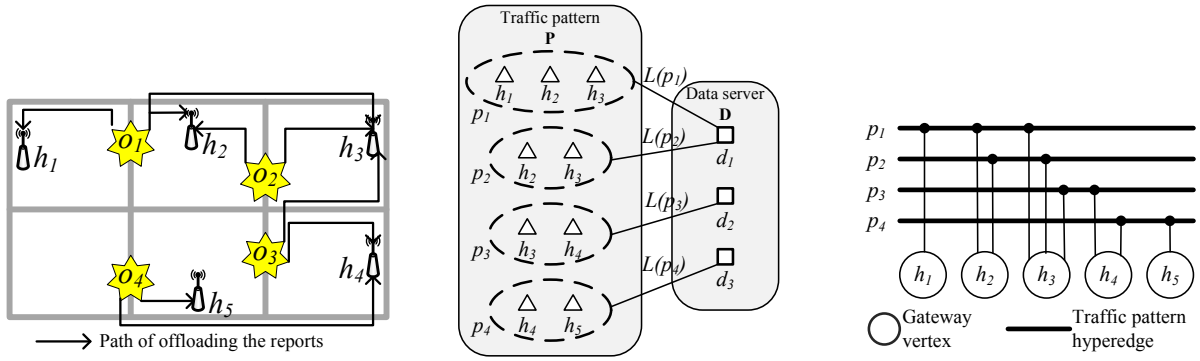
Fig. 4. The responses from a controller by CBench.

and remote mode. This implies that the capability of providing service to each switch by the controller is decreasing while increase the number of connected switches.

Thus, building a distributed control plane with multiple physical controllers is necessary to a large-scale network. As shown in Figure 3(b), two hosts h_1 and h_3 send their data packets to hosts h_2 and h_4, respectively. When the switches s_1 and s_3 receive the data packets and miss their flow tables, both will send requests to their controllers. If all the switches are in the same domain with a single controller, the controller will receive two requests from the two data transmissions. If the data plane is partitioned into two domains, the switches s_1 and s_2 are in the domain with the controller c_1, and the switches s_3 and s_4 are in the domain with the controller c_2. The two data transmissions ($h_1 \rightarrow h_2$ and $h_3 \rightarrow h_4$) are intra-domain. Each controller will receive only 1 request. This example implies that partitioning the data plane can reduce the load of requests for each controller.

Multiple controllers can reduce the total switch-to-controller delay. Let $T(c_i, s_j)$ denote the delay of a link between the controller c_i and the switch s_j. The average delay in Figure 3(a) can be calculated by $\frac{\sum_{j=1}^{4} T(c_1,s_j)}{4}$, and the average delay in Figure 3(c) can be calculated by $\frac{\sum_{j=1}^{2} T(c_1,s_j)+\sum_{j=3}^{4} T(c_2,s_j)}{4}$. We assume that the switches s_1 and s_2 are geographically closer to the controller c_1 and that the switches s_3 and s_4 are closer geographically to the controller c_2. Thus, the total delay of links between the controller c_2 and the two switches (s_3 and s_4), calculated by $\sum_{j=3}^{4} T(c_2, s_j)$, is less than the total delay between the controller c_1 and the two switches s_3 and s_4, calculated by $\sum_{j=3}^{4} T(c_1, s_j)$.

We call the flow path of the data transmission running



(a) Scenario of multiple events and APs (b) Bipartite graph (c) Hypergraph

Fig. 5. An example of uploading the sensing reports from the four events through the five gateways.

through two or more domains the *inter-domain* flow path. An inter-domain flow path can result in an extra load of requests in the whole network. As shown in Figure 3(c), the host h_1 sends a data packet to the host h_2. The data plane is partitioned into two domains, i.e. the switches s_1 and s_2 are in the domain with the controller c_1, and the switches s_3 and s_4 are in the domain with the controller c_2. When the switch s_1 receives data packet that does not match with any entry in its flow table, it will send a request to its controller c_1. The controller c_1 only updates the flow tables of the two switches (s_1 and s_2). Then, when the first switch s_3 in another domain receives the data packet, it will also send a request to its controller c_2. The controller c_2 only updates the flow tables of the other two switches, s_3 and s_4. Compared to the example in Figure 3(a), the total number of requests increases by 2. Therefore, partitioning a flow path can reduce the average delay, but will also increase the total load of requests in the network.

Therefore, we investigate the problem of partitioning SDN with multiple controllers to minimize the total load of requests, which considers the following factors: (1) the request-processing capability of a controller (denoted by L_{max}); (2) load balance among the multiple controllers; (3) the extra loads caused by the inter-domain flow paths; (4) the total switch-to-controller delay. Among these factors, the flow paths in SDN are the essential part, which are generated by uploading the sensing data from the gateways to the data servers.

III. Partitioning Algorithm for SD-IoT Network

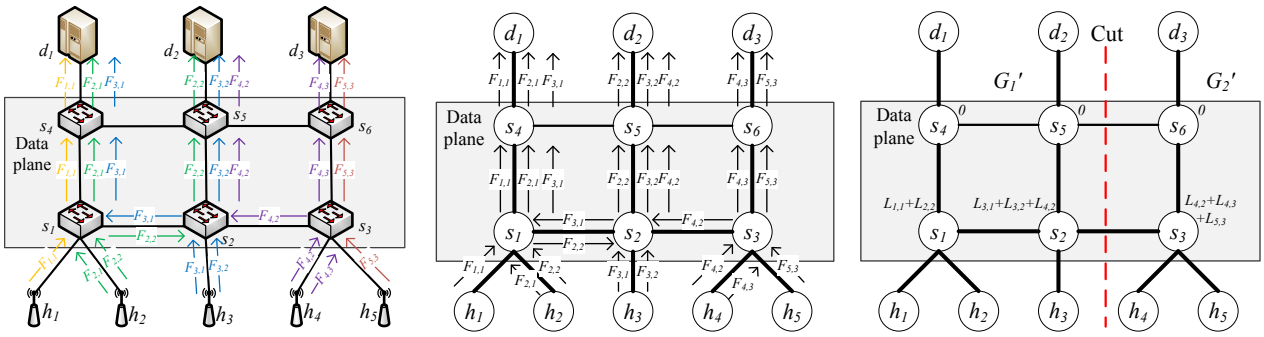
In this section, we model the relationship between spatial events and gateways to analyze the non-uniform data transmissions through the flow paths in SDN. Then, we propose a Partitioning Algorithm for Software-defined IoT Network (PASIN) to partition SDN by considering the flow paths.

A. Traffic Pattern

In the urban sensing applications, an event, denoted by o, has a location marked by latitude and longitude. When a vehicle moves through the location of event o, the vehicle's mobile device senses the event o and generates a report to describe it. The probing vehicle first transmits the data to a gateway (denoted by h), and then the gateway transmits data to the remote server through the networks. The set of all the gateways is denoted by $\mathbf{H}$. We assume all the reports buffered in the vehicle will be uploaded during one contact with a gateway. For simplicity, the way of uploading sensing data through cellular network is regarded as immediately offloading the sensing data without delay. Therefore, once a spatial event occurs, a large amount of data flow will arrive at the relative gateways. As shown in Figure 5(a), four events (o_1, o_2, o_3, and o_4) occur in such an area. The sensing reports from these events will be carried by the vehicles in the area to offload via the five gateways (h_1, h_2, h_3, h_4, and h_5). The load of requests generated by the data transmission from event o_i to gateway h_j in a predefined time is denoted by $L(o_i, h_j)$. We assume that the system obtains the load of requests from each event based on the statistics of historical sensing tasks near their locations.

Reports of the same event can be uploaded to, at most, g different gateways via either the cellular network (3G/4G) or delayed offloading. Thus, each event o_i has a *traffic pattern* denoted by p_i, which is the set of the gateways upload the sensing reports from event o_i. We denote the space of traffic patterns by $\mathbb{P} = \{\mathbf{H}, \emptyset\}^g$. The gateways in a practical system actually fall in a subset of the space, denoted by $\mathbf{P} \subset \mathbb{P} = \{\mathbf{H}, \emptyset\}^g$. As illustrated in Figure 5(a), the example shown has four traffic patterns: $p_1 = \{h_1, h_2, h_3\}$, $p_2 = \{h_2, h_3\}$, $p_3 = \{h_3, h_4\}$, and $p_4 = \{h_4, h_5\}$.

We assume that sensing reports of the same event are transmitted to the same data server (denoted by d) for processing, i.e., a map from traffic pattern to data server: $p \to d$. The set of all the data servers is denoted by $\mathbf{D}$. As shown in Figure 5(b), we utilize a bipartite graph to illustrate the relationship between the traffic patterns and the data servers. In this bipartite graph, the traffic patterns and the data servers are the two sets of vertices, and the edges between these two sets are weighted by their loads of requests, denoted by $L(p, d)$ or $L(p)$. The load of requests by the data transmission $L(o, h)$ can also be denoted as $L(p, h)$. Then, the load of requests from a pattern is equal to the sum of the load from all the gateways in it, i.e., $L(p, d) = \sum_{h \in p} L(p, h)$. We can calculate the total



(a) Multiple flow paths (b) Graph of data plane G (c) Loads of switches after partitioning

Fig. 6. Partition data plane over multiple flow paths.

load of requests using the data transmission from the gateway h to the data server d by

$$L(h,d) = \sum_{p \in \mathbf{P}} [L(p,h)\mathbf{1}(h \in p)\mathbf{1}(L(p,d) > 0)] \quad (1)$$

where $\mathbf{1}(h \in p)$ indicates whether h is a member of the traffic pattern p, returning 1 if true or 0 otherwise. $\mathbf{1}(L(p,d) > 0)$ indicates whether there exists a load of requests using the data transmission from traffic pattern p to data server d, returning 1 if true or 0 otherwise.

The partitioning problem for balancing the loads among multiple controllers is modeled as a hypergraph k-partitioning problem, where the objective function is to minimize the total costs of cuts on the hypergraph. A hypergraph $H(X,Y)$ is a further generalization of a graph; the hypergraph allows each of its hyperedges to involve multiple vertices while the edge of an ordinary graph can only involve two vertices at most [13]. Figure 5(c) shows a bus representation of a hypergraph [14]. We set up the vertex set X with all the gateways, i.e. $X = \{h \mid h \in \mathbf{H}\}$. The hypergraph set Y contains all the traffic patterns. Each traffic pattern hyperedge involves multiple gateways, which is the main reason to introduce a hypergraph. Formally, $Y = \{p \mid p \in \mathbf{P}\}$.

Each hyperedge $y \in Y$ is assigned a weight which is equal to the load of the traffic pattern $L(p)$. Since all the input data of SDN come from the gateways, we consider that each gateway $h \in \mathbf{H}$ sends the requests of input data to a unique controller $c \in \mathbf{C}$. Thus, the gateway-to-controller mapping function is defined as $\mathcal{M} : h \rightarrow c$, which specifies the controller c according to each gateway h. Fundamentally, our work focuses on designing a partitioning scheme that provides a reasonable solution to $\mathcal{M}$. However, as discussed in Section II-B, both the cutting cost and the load of each subgraph should consider the flow paths in the data plane. Next, we discuss the problem of partitioning the flow paths along the switches between the gateways and the data servers.

B. Partitioning Algorithm for Software-defined IoT Network

In the data plane, reports are transmitted from the gateways to the data servers by a specific openflow protocol. In this paper, we take the shortest path as an example to illustrate our partitioning scheme. Let $F_{i,j}$ denote the flow path from the gateway h_i to the data server d_j, and the number of requests from the path is denoted by $L(F_{i,j})$. A flow path F is an alternating sequence of switches and links, and $F^{(i)}$ represents the i^{th} switch in it. The set of all the flow paths is denoted by $\mathbf{F}$. Figure 6(a) shows the flow paths among the switches for the previous example. We construct a graph $G(V,E)$ for such data plane (see in Figure 6(b)). The set of vertices V represents the set of the switches as: $V = \{s \mid s \in \mathbf{S}\}$. The edge set E represents the set of the links among the switches as: $E = \{e_{ss'} \mid s, s' \in \mathbf{S}\}$.

Since each controller takes control of one domain, the data plane of the network is required to be partitioned into k domains for the k controllers in a set $\mathbf{C} = \{c_1, \cdots, c_k\}$. We model the problem as a k-way graph partitioning problem (GPP) on the graph $G(V,E)$. The graph is partitioned into k smaller subgraphs, which are denoted by $G_1', G_2', \cdots, G_k'$. Each subgraph $G_i'(V_i', E_i')$, with V_i' vertices and E_i' edges, is handled by the controller c_i. The load of edge e is defined as the total load of requested flow paths through it, as follows:

$$L(e) = \sum_{F_{i,j} \in \mathbf{F}} L(F_{i,j})\mathbf{1}(e \in F_{i,j}) \quad (2)$$

where $\mathbf{1}(e \in F_{i,j})$ indicates whether the flow path $F_{i,j}$ is through the edge e, returning 1 if true or 0 otherwise.

The load of a vertex s is defined as the total load of requests from the flow paths through it, as follows:

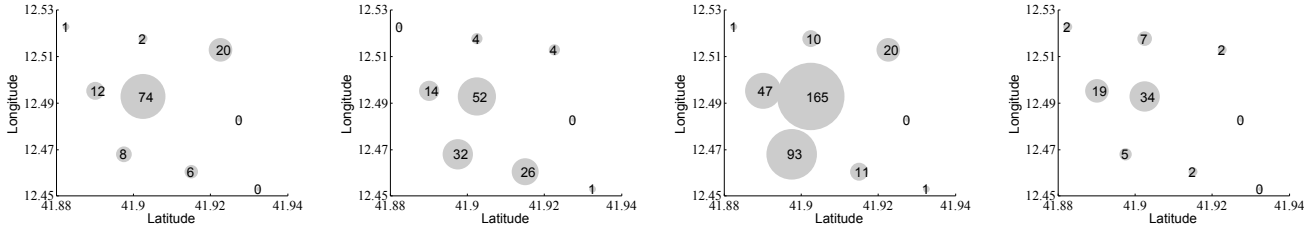
$$L(s) = \sum_{F_{i,j} \in \mathbf{F}} L(F_{i,j})\mathbf{1}(s \in F_{i,j}) \quad (3)$$

where $\mathbf{1}(s \in F_{i,j})$ indicates whether the flow path $F_{i,j}$ runs through the switch s, returning 1 if true or 0 otherwise.

The total delays of the links between switches and their controller in a subgraph G_i' is calculated by the load and the delay of each vertex in the subgraph as follows:

$$T(G_i') = \sum_{s \in G_i'} T(s, c_i)L(s) \quad (4)$$

Initially, we assume all switches connect to a single controller (denoted by c_0) in set $\mathbf{C}$, which has the minimum total delay to all the switches. Thus, the initial delay of a vertex s_i



(a) Northeast event (b) Southeast event (c) Southwest event (d) Northwest event

Fig. 7. Offloading data reports from the events at different places.

to its controller is defined as the function of $T(s_i, c_0)$. After partitioning, the vertex s_i is connected to a new controller c_j. Compared with c_0, the difference between the two delays can be calculated by the function:

$$\Delta T(s_i, c_j, c_0) = T(s_i, c_j) - T(s_i, c_0) \tag{5}$$

In order to combine the load of the controller and the delay to the controller from a vertex as one metric, we transform the transmission delay into the number of requests to the controllers. We regard the transmission delay as the queuing delay on the controller. We take a simple D/D/1 queuing model as an example, and the queuing delay of each request at a controller is equal to q. Thus, the difference in transmission delay is transformed to the number of requests as follows:

$$\Delta n(s_i, c_j) = \frac{\Delta T(s_i, c_j, c_0)}{q} \cdot L(s_i) \tag{6}$$

A cut $cut = (V_1, V_2)$ is defined as a partition of V of a graph $G = (V, E)$ into two subsets V_1 and V_2, as shown in Figure 6(c). The set of all the cuts is denoted by CUT. The cut cut is the set $\{e_{uv} \in E \mid u \in V_1, v \in V_2\}$ of edges that have one endpoint in V_1 and the other endpoint in V_2. According to the extra load of inter-domain flow paths and the delays of links between the switches and their controllers, the cost of cut is defined as follows:

$$cost(cut) = \sum_{e_{uv} \in cut} L(e_{uv}) + \sum_{u \in V_1} \Delta n(u, c_1) + \sum_{v \in V_2} \Delta n(v, c_2) \tag{7}$$

A flow path $F_{i,j}$ is partitioned into m segments, denoted by $f_{i,j}^1, \cdots, f_{i,j}^m$, and each segment has a head vertex and a tail vertex. Each segment belongs to a subgraph and has the same load $L_{i,j}$ on that domain. A vertex has the load $L_{i,j}$ only if it is the head of a segment that belongs to the flow path $F_{i,j}$ in such a subgraph. In Figure 6(c), the flow path $F_{4,2}$ is partitioned by the cut, and the vertices s_2 and s_3 become the heads in both of the partitions. Therefore, they both have the load $L_{4,2}$. As a result, the weights of vertices s_1, s_2, and s_3 can be calculated by $[L(F_{1,1}) + L(F_{2,2})]$, $[L(F_{3,1}) + L(F_{3,2}) + L(F_{4,2})]$, and $[L(F_{4,2}) + L(F_{4,3}) + L(F_{5,3})]$, respectively. Since the vertices s_4, s_5, and s_6 are not the head of any flow paths in their subgraphs, their weights are all equal to zero. Therefore, the total load of switch s to controller c is:

$$L(s, c) = \sum_{F_{i,j} \in \mathbf{F}} L(F_{i,j}) \mathbf{1}(s = F_{i,j}^{(1)}) \tag{8}$$

where $\mathbf{1}(s = F_{i,j}^{(1)})$ indicates whether the switch s is the head of the flow path denoted by $F_{i,j}^{(1)}$, returning 1 if true or 0 otherwise.

Thus, we define the load of a subgraph G_i' as the sum of the loads of all the vertices in it as follows:

$$L(G_i') = \sum_{s \in G_i'} L(s, c_i) \tag{9}$$

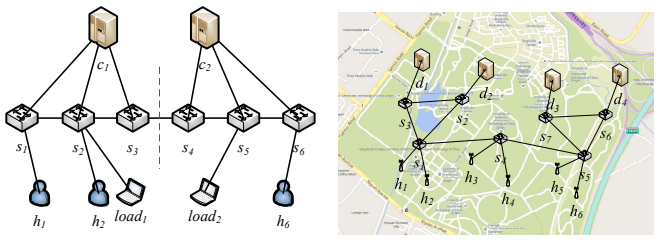
A 2-way graph partitioning problem can be formulated as a minimum cut optimization problem as follows:

$$\begin{aligned} \min_{CUT} \quad & \sum_{cut \in CUT} cost(cut) \\ \text{s.t.} \quad & \mid L(G_1') - L(G_2') \mid \leq \varepsilon, \\ & L(G_1'), L(G_2') \leq L_{max}, \\ & T(G_1'), T(G_2') \leq T_{max}. \end{aligned} \tag{10}$$

where ε is an imbalanced parameter that satisfies $\varepsilon \in \mathbb{R}_{\geq 0}$. L_{max} and T_{max} denote the maximum load and the maximum delay of a subgraph, respectively. The graph-partitioning problem is proved to be NP-hard as shown in [13]. We adopt the Kernighan-Lin algorithm , with a computation complexity of between $O(\mid V \mid^2)$ and $O(\mid V \mid^2 \log \mid V \mid)$, to solve it.

The k-way partitioning problem is most frequently solved by recursive bisection [15]. That is, after obtaining a 2-way partitioning of V, we recursively obtain a 2-way partitioning of each resulting partition. After $\log k$ phases, graph G is partitioned into k disjoint subgraphs. Thus, the problem of performing a k-way partitioning is reduced to that of performing a sequence of bisections. A multilevel recursive bisection (MLRB) algorithm has emerged as a highly effective method for computing a k-way partitioning of a graph. The complexity of the MLRB for producing a k-way partitioning of a graph $G = (V, E)$ is $O(|E| \log k)$ [16].

Furthermore, PASIN not only partitions the data plain into multiple domains with minimum cost, but also provides a reasonable map between the gateways and the controllers, i.e., $\mathcal{M} : h \to c$, discussed in previous subsection. As previously discussed, the gateways that transmit the sensing reports from the same event are grouped into a traffic pattern, and the switches which connect the gateways also send the requests of the flow paths with the data server act as destination. An intra-domain flow path in a domain means that both the source (gateway) and the destination (data server) connect to this



(a) Linear topology (b) Campus of UESTC

Fig. 8. Scenarios of the experiments.

domain. By contrast, an inter-domain flow path in a domain means that either the source (gateway) or the destination (data server) is unconnected to this domain. None the flow paths from a traffic pattern to its data server bring any extra loads of requests if they are covered by one domain as intra-domain flow paths. This should requires that all gateways in the traffic pattern and the data server act as their destination are connected to this domain. On the contrary, if some of the flow paths from a traffic pattern to its data server are inter-domain, they will result in an extra load of requests. That means at least one of the gateways in the traffic pattern or the data server does not connect to this domain.

IV. Evaluation Results

In this section, we demonstrate the relationship between the spatial events and the gateways using a real dataset. Then, we evaluate the limitation of the capability of the controller. At last, we evaluate the performance of our proposed Partitioning Algorithm for Software-defined IoT Network (PASIN).

We have developed a scalable framework of an SDN testbed with multiple controllers [17]. The architecture of this testbed contains five module components: control plane, data plane, host, monitor, and analysis. Each component can coordinate with other simulators to accomplish the simulation tasks. In the simulations for this paper, the module data plane utilizes Mininet [18], a popular SDN simulator with OpenFlow module. The module of multiple controls is implemented through a combination of Floodlight [12] and FlowVisor [19].

A. Offloading Data at APs

To evaluate the relationship between the spatial events and the gateways, we do experiments on the Taxi-ROMA dataset [20]. This dataset contains real mobility traces of taxi cabs in Rome, Italy. It contains GPS coordinates of approximately 320 taxis collected over 30 days. We select a dataset containing traces collected on Feb. 5, 2014, which contains information from 172 taxis. The traces cover an area with a range of 66km $\times$ 59km. Our experiment consists of 9 access points serving as gateways (center, north, east, south, west, northeast, southeast, southwest, and northwest) and 4 events (northeast, southeast, southwest, and northwest) at different places.

Figure 7 depicts the number of reports offloaded at different access points; each circle represents an access point with a number of reports corresponding to their positions in the map. We notice that the access points near the event can receive more reports. For example, in Figure 7(a), the northeast access point receives more reports from the northeast event than other events. In Figure 7(b), both the south access point and the east access point receive more reports from the southeast event than other events. We notice that the center access point receives the most reports overall (see in Figure 7(c)) since it has a heavy vehicular traffic density. In contrast, an event that is far from the center will offload fewer reports to the access points, such as the northwest event in Figure 7(d). In consequence, all the results imply that the distributions of data from events to access points are non-uniform.

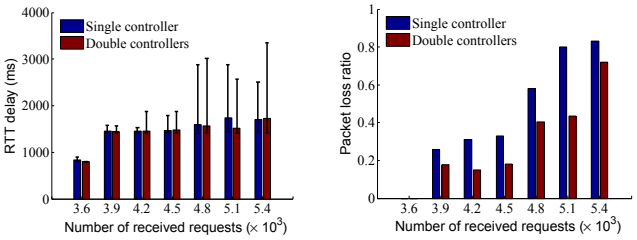
B. Limitation of Controllers

To verify the limitation of the capability of the controller, we evaluate the performance of SDN with linear topology, shown in Figure 8(a), where the data plane contains 6 switches and the control plane contains 2 controllers. Each of the two hosts ($load_1$ and $load_2$) connected to the switches s_2 and s_5 generates a load of 50 UDP packets with unreachable destinations per second, and these packets are flooded to the whole network. The size of each UDP packet is 67 Bytes. We test the RTT (round trip time) delay of two different data transmissions ($h_1 \rightarrow h_6$ and $h_1 \rightarrow h_2$) to compare the approaches of the single controller and the double controllers. Under the single-controller approach, the controller connects to all the switches. Under the double-controllers approach, each controller connects to 3 switches.

Figure 9 shows the results of the data transmission from h_1 to h_6. Figure 9(a) shows delays when different requests are received by the controller. We notice that when the number of received requests is more than 4500, the RTT delay of data transmissions obviously increases. In most cases, the delay under the double controllers is less than under the single controller. Figure 9(b) shows the loss ratio of data packets with different requests received by the controller. We notice that when the number of received requests is more than 4500, the loss ratios increase. Moreover, the loss ratio under double controllers is much less than under the single controller. Figure 10 shows the results of the data transmission from h_1 to h_2. We notice that the delay of the intra-domain data transmission (see Figure 10(a)) is less than that of $h_1 \rightarrow h_6$, and the loss ratio of intra-domain data transmission (see in Figure 10(b)) is also less than that of $h_1 \rightarrow h_6$. Both the RTT delay and the loss ratio under the double controllers are less than under the single controller. This implies that multiple controllers can improve the performance of the network by balancing loads among controllers.

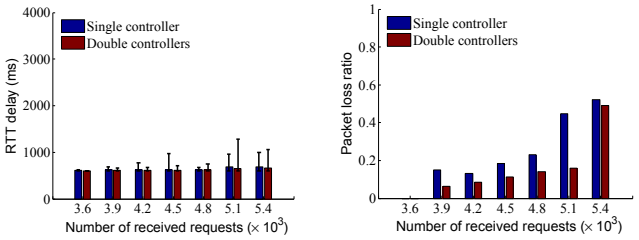
C. Partitioning

To verify the performance of PASIN, we compare it with two other approaches: (1) Single: the whole network contains only one controller and one domain for all the switches; (2) UbiFlow [2]: multiple controllers are deployed to divide the network into several partitions, which represent different geographical areas. Our simulation is based on the UESTC



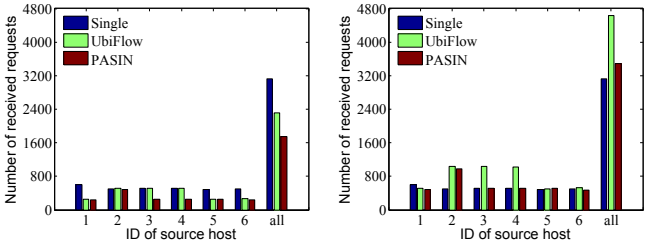
(a) RTT delay (b) Loss ratio

Fig. 9. Inter-domain data transmission ($h_1 \rightarrow h_6$).



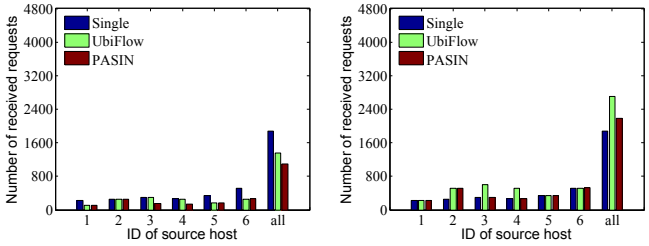
(a) RTT delay (b) Loss ratio

Fig. 10. Intra-domain data transmission ($h_1 \rightarrow h_2$).



(a) Mean of each controller (b) Total of whole control plane

Fig. 11. Average number of received requests under uniform data rate.



(a) Mean of each controller (b) Total of whole control plane

Fig. 12. Average number of received requests under non-uniform data rate.

campus in China, as shown in Figure 8(b). The backbone topology consists of 4 data servers (each Ethernet link between a switch and a server has 1Gbps), 7 switches (each Ethernet link between two switches has 1Gbps, and each switch is directly controlled by one controller), and 6 access points as the gateways (each access point has one 100Mbps Ethernet link to one switch). In our simulation, we set six data flows from the access points to the data servers, and each access point owns one path. Compared with the single-controller approach, the network under both UbiFlow and PASIN contains two controllers. We set the timeout of flow entries at the switches as 50 ms, and the interval time for sending the data packets from the hosts is no less than 100 ms. Under the uniform data rate strategy, all the hosts have the same sending interval of 100 ms. In contrast, under the non-uniform data rate strategy, the hosts have different sending intervals between 100 ms and 300 ms. We record the average number of received requests by the controllers every 60 seconds.

Figure 11 shows the number of requests received under the uniform data sending rate. We notice that the mean number of received requests by the controllers under the single-controller strategy is much higher than the mean of either UbiFlow or PASIN, both of which have two controllers (see in Figure 11(a)). PASIN has the lowest mean number of received requests, since it balances the loads of intra-domain flow paths. The total number of requests received by the hosts under PASIN is much less than under UbiFlow or under the single-controller approach (see in Figure 11(b)), since PASIN reduces the extra loads by making use of inter-domain flow paths. Figure 12 shows the number of requests received under the non-uniform data sending rate. We notice that PASIN also has the lowest mean number of requests received by the controllers (see in Figure 12(a)), and has a much lower total number of requests received from all the hosts than UbiFlow (see in Figure 12(b)). The reason is that PASIN partitions the network according to the distribution of data traffic to reduce the total load of requests with a constraint of load balance among domains. This implies that PASIN is better equipped than UbiFlow to handle non-uniform data traffic.

V. Related Work

Internet of Things: The advent of the Internet of Things (IoT) has inspired a large variety of new applications that can provide ubiquitous services to make the lives of existing industrial systems and people more intelligent, e.g., industrial automation, smart grids, intelligent transportation systems, smart healthcare, smart home, etc [21]. The development and management of IoT cloud systems and applications have received a lot of attention lately. In [22], the authors mostly deal with IoT infrastructure virtualization and its management on cloud platforms. Large-scale network environments (e.g. IoT applications in smart cities) have the potential to provide vast amounts of data flows.

Mobile Crowdsensing: Zhou *et al.* in [23] investigate prediction applications that reduce initial construction overheads, applying their work to the prediction of bus arrival times. Yang *et al.* in [24] design incentive schemes for mobile phone sensing, with two system models: the platform-centric model, in which the platform provides a reward to be shared by participating users, and the user-centric model, in which users have more control over the payment they will receive. He *et al.* in [25] research the optimal task allocation and show that the allocation problem is NP hard, and also discuss how to decide fair prices of sensing tasks.

Software Defined Network: To cope with huge numbers of data flows in a large-scale network, oponen *et al.* in [26] propose a distributed SDN architecture, Onix, where the network view is distributed among multiple controller instances. Phemius *et al.* in [27] propose DISCO, a distributed SDN Control plane to cope with the distributed and heterogeneous nature of modern overlay networks. Hassas *et al.* in [28] propose a control plane with two layers where the bottom layer is a group of controllers with no interconnection, and the top layer is a logically centralized controller that maintains the network-wide state. Jang *et al.* in [29] propose an SDN-based WLAN monitoring and management framework called RFlow$^+$, which considers the trade-off between measurement accuracy and network overhead.

VI. Conclusion and Future Work

Because of the limited request-processing capability of a single controller, many researchers propose and study SDN with multiple physical controllers to achieve scalability and reliability in a network. However, a partitioning problem arises when considering the limited capability of each controller and the load balance among multiple controllers. In this paper, we investigate the partitioning problem over multiple flow paths in SDN, with the urban sensing applications for detecting spatial events. The data flow is generated by mobile devices sensing the spatial events and is uploaded via gateways to connect to the data server. We analyze the non-uniform distribution of sensing data from the events to the data server using a hypergraph. Further, we propose a Partitioning Algorithm for Software-defined IoT Network (PASIN) to achieve our objective of minimum total load of requests, with the constraint of total switch-to-controller delays in each domain. Our extensional simulations verify the effectiveness of our proposed scheme. This paper only considers the influence of the spatial distribution of events on partitions of the data plane; we will investigate the events in a dynamic manner with a spatio-temporal distribution in our future work.

Acknowledgment

This work is supported by NSFC grants No. 61370204, 61572113; the Fundamental Research Funds for the Central Universities No. ZYGX2015J155, ZYGX2016J084, ZYGX2016J195; and the Applied Basic Program of Sichuan Province of China No. 2014JY0192.

References

[1] J. Liu, Y. Li, M. Chen, and W. Dong, "Software-defined internet of things for smart urban sensing," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 55–63, 2015.

[2] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "Ubiflow: Mobility management in urban-scale software defined iot," in *Proc. of IEEE INFOCOM*, 2015.

[3] "Qualcomm, China Mobile Research Institute and Mobike Plan to Commence First of its Kind LTE IoT Multimode Field Trials in China." [Online]. Available: https://www.qualcomm.com/news/releases/2017/05/22/qualcomm-china-mobile-research-institute-and-mobike-plan-commence-first-its

[4] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A distributed and robust sdn control plane for transactional network updates," in *Proc. of IEEE INFOCOM*, 2015.

[5] B. Heller, R. Sherwood, and N. Mckeown, "The controller placement problem," in *Proc. of ACM HotSDN*, 2012.

[6] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: using a mobile sensor network for road surface monitoring," in *Proc. of ACM MobiSys*, 2008.

[7] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application," in *Proc. of ACM SenSys*, 2008.

[8] H. Ma, D. Zhao, and P. Yuan, "Opportunities in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 29–35, 2014.

[9] R. W. Ouyang, M. Srivastava, A. Toniolo, and T. J. Norman, "Truth discovery in crowdsourced detection of spatial events," in *Proc. of ACM CIKM*, 2014.

[10] F. Mehmeti and T. Spyropoulos, "Is it worth to be patient? analysis and optimization of delayed mobile data offloading," in *Proc. of IEEE INFOCOM*, 2014.

[11] "Cbench." [Online]. Available: https://sourceforge.net/projects/cbench

[12] "Floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight

[13] C. Bichot and P. Siarry, Eds., *Graph Partitioning*. Wiley, 2011.

[14] S. Q. Zheng and J. Wu, "Dual of a complete graph as an interconnection network," *Journal of Parallel and Distributed Computing*, vol. 60, no. 8, pp. 1028–1046, 2000.

[15] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, 1998.

[16] K. George and K. Vipin, "A fast and highly quality multilevel scheme for partitioning irregular graphs," *Proc. of ICPP*, 1999.

[17] X. Chen, C. Song, Y. Qi, X. Dai, and M. Liu, "A scalable framework of testbed for sdn simulation with multiple controllers," in *in Proc. of IEEE ISPA*, 2017.

[18] "Mininet." [Online]. Available: http://mininet.org/

[19] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "Can the production network be the testbed?" *in Proc. of OSDI*, 2010.

[20] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD data set roma/taxi (v. 2014-07-17)," Downloaded from http://crawdad.org/roma/taxi/, Jul. 2014.

[21] A. Al-Fuqaha, M. Guizani, M. Mohammadi, and M. Aledhari, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, p. Fourthquarter 2015, 2015.

[22] S. Nastic, H. L. Truong, and S. Dustdar, "Sdg-pro: a programming framework for software-defined iot cloud gateways," *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–17, 2015.

[23] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: predicting bus arrival time with mobile phone based participatory sensing," in *Proc. of ACM MobiSys*, 2012.

[24] D. Yang, G. Xue, X. Fang, and J. Tang, "Crowdsourcing to smartphones: incentive mechanism design for mobile phone sensing," in *Proc. of ACM MobiCom*, 2012.

[25] S. He, D.-H. Shin, J. Zhang, and J. Chen, "Toward optimal allocation of location dependent tasks in crowdsensing," in *Proc. of IEEE INFOCOM*, 2014.

[26] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, and T. Hama, "Onix: A distributed control platform for large-scale production networks." *in Proc. of OSDI*, 2010.

[27] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *in Proc. of IEEE NOMS*, 2014.

[28] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *in Proc. of ACM HotSDN*, 2012.

[29] R. Jang, D. Cho, Y. Noh, and D. Nyang, "Rflow$^+$: An sdn-based wlan monitoring and management framework," in *in Proc. of IEEE INFOCOM*, 2017.