CluFlow: Cluster-based Flow Management in Software-Defined Wireless Sensor Networks

Liu, Qingzhi; Ozcelebi, Tanir; Cheng, Long; Kuipers, Fernando; Lukkien, Johan

DOI
10.1109/WCNC.2019.8885485

Publication date
2019

Document Version
Accepted author manuscript

Published in
2019 IEEE Wireless Communications and Networking Conference, WCNC 2019

CluFlow: Cluster-based Flow Management in Software-Defined Wireless Sensor Networks

Qingzhi Liu*, Tanir Ozcelebi*, Long Cheng†, Fernando Kuipers‡, Johan Lukkien*

*MCS, Eindhoven University of Technology, The Netherlands
†CS, University College Dublin, Ireland
‡EEMCS, Delft University of Technology, The Netherlands
Email: {q.liu.1, t.ozcelebi}@tue.nl, long.cheng@ucd.ie, f.a.kuipers@tudelft.nl, j.j.lukkien@tue.nl

***Abstract*—Software-defined networking (SDN) is a cornerstone of next-generation networks and has already led to numerous advantages for data-center networks and wide-area networks, for instance in terms of reduced management complexity and more fine-grained traffic engineering. However, the design and implementation of SDN within wireless sensor networks (WSN) have received far less attention. Unfortunately, because of the multi-hop type of communication in WSN, a direct reuse of the wired SDN architecture could lead to excessive communication overhead. In this paper, we propose a cluster-based flow management approach that makes a trade-off between the granularity of monitoring by an SDN controller and the communication overhead of flow management. A network is partitioned into clusters with a minimum number of border nodes. Instead of having to handle the individual flows of all nodes, the SDN controller only manages incoming and outgoing traffic flows of clusters through border nodes. Our proof-of-concept implementations in software and hardware show that, when compared with benchmark solutions, our approach is significantly more efficient with respect to the number of nodes that must be managed and the number of control messages exchanged.**

I. Introduction

Software-Defined Networking (SDN), in comparison to traditional networking, provides improved flexibility and reduced complexity when it comes to flow management [1]. Given the advantages and large-scale implementation of SDN within data-center networks and wide-area networks, a logical question is whether the same advantages can be expected when SDN is introduced within wireless sensor networks (WSN). However, while most SDN research is focusing on wired networks, only few initiatives have attempted to extend the benefits of SDN to the wireless domain [2].

Software-Defined Wireless Sensor Networks (SD-WSN) is proposed with the objective to levarage the benefit of SDN for WSN [3]. Compared with distributed control of WSN, SDN controller is able to manage and optimize the WSN peformance, such as energy consumption, communication flow, etc., based on a global view of the entire network. To implement SD-WSN, the SDN architecture for wired networks must be mapped to WSN, which involves several difficulties:

- Most existing SDN architectures require frequent data exchange, such as request and reply messages, between the data plane and the control plane [4]. This process involves much communication cost. Although this overhead is acceptable in wired networks given the plentiful capacity of wired connections between switches and SDN controllers, the case for WSN is different. WSN requires nodes to relay messages from other nodes, while they have limited resources (e.g., energy and bandwidth). Frequent requests and replies between nodes and the SDN controller would consume precious resources, which would limit the size of SD-WSN.
- Nodes in WSN can not completely decouple the data and control planes. While nodes are not directly connected to the SDN controller, they have to obtain routing commands from it. As a result each node still has to maintain a distributed local routing table for control flow.

To take advantage of the power of SDN within WSN, we need to balance the benefits and the communication overhead of SDN control. We propose a cluster-based flow management approach called **CluFlow** (**Clu**ster **Flow** Management in SD-WSN). The aim of CluFlow is to decrease the number of nodes that are involved in flow management within an SD-WSN. CluFlow makes a trade-off between the granularity of flow management and the communication overhead by SDN controller. Our solution divides the network into clusters and defines the flow on the border nodes of clusters through SDN commands. Within the cluster, a normal routing procedure is used. The properties of CluFlow are twofold. Firstly, we trade off granularity of flow management for less data exchange between the data and control planes. We adopt network clustering and utilize border nodes of clusters to manage traffic flow on cluster level instead of by individual nodes. We monitor flows of only the cluster border nodes, and manage flow on cluster level by controlling the routing tables of the border nodes. Secondly, we enable SDN control to work in parallel with distributed routing.

The key challenges to realize the proposed design are twofold. Firstly, we need to find a way to partition the network into clusters by using a minimum number of border nodes. Secondly, we need to develop a method to install the cluster-based flow rules in SD-WSN. This paper provides the following main contributions:

- We take a graph-theoretic approach for clustering the

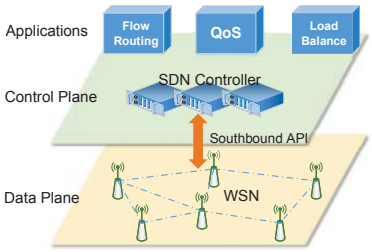
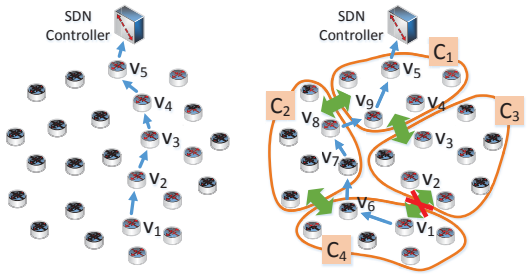


Fig. 1. Architecture of software-defined wireless sensor networks (SD-WSN).



(a) (b)

Fig. 2. (a) The communication flow is managed by each nodes. (b) The SD-WSN controller inserts cluster-level routing rules.

network and selecting the border nodes. The SDN controller manages communication flow by monitoring and controlling these border nodes.

- We propose a priority scheme where cluster-level routing, performed by the SDN controller, has higher priority than node-level routing. This hierarchical routing decreases communication overhead of SDN control in WSN.
- We build an SDN controller to manage the communication flow in WSN and demonstrate that it works with traditional distributed routing protocols in a real deployment.

The paper is organized as follows. Our system model is presented in Section II. Cluster-based SDN flow management is addressed in Section III. Simulations and hardware experiments are presented in Section IV. Related work is discussed in Section V and we conclude in Section VI.

II. System Model

WSN typically consists of resource-constrained sensor nodes for monitoring the physical conditions of the environment. The SDN paradigm could potentially provide a simple and flexible control approach to WSN [5]. Fig. 1 illustrates the general architecture of SD-WSN. In such an architecture, the sensor nodes only perform packet forwarding and all the control plane operations, including flow routing, Quality of Service (QoS) control, load balancing, etc., are performed by a (logically) centralized controller.

A. Research Aim

One of the challenges in SD-WSN is that the control and data flows share the same wireless network channel, while in a typical wired SDN, the control flow relies on a dedicated wired channel. Given that most WSNs are resource-constrained, the SDN control flow becomes a high burden to data flow. In addition, as the topology of a WSN changes or grows, a burst of control packets requesting new flow table entries could surpass part of the already limited bandwidth.

To cope with this challenge, some research limits the flow table request rate [6], and some utilize multiple controllers to build a hierarchical SD-WSN architecture [7]. These approaches alleviate the communication burden caused by control flows, and increase the scalability of SD-WSN. Complementary to these existing approaches, we propose a clustering approach CluFlow to manage SD-WSNs.

B. Solution Overview of Cluster Flow Control

We represent the network by an undirected graph $G = (V, E)$ with $V = \{v_1, ...v_i, ...v_n\}$ representing the set of nodes with $n = |V|$ and $i \in [1, n]$ and $E = \{e_1, ...e_j, ...e_m\}$ representing the edges with $m = |E|$ and $j \in [1, m]$. Nodes that share an edge are called *neighbors*. Suppose the set of nodes V is partitioned into clusters $C = \{c_1, ...c_k, ...c_u\}$ with $u = |C|$ and $k \in [1, u]$.

We assume that each node in the WSN reports its neighbor connectivity to the SDN controller. There is one central SDN controller that is responsible for partitioning the network. The routing rules set by SDN controller with CluFlow are on cluster level, called *cluster-level* routing. The cluster-level routing rules have higher priority than local routing rules in the border nodes. Suppose v_i and v_j have a linked edge. $v_i \in c_i$ and $v_j \in c_j$. If the cluster-level routing rule allows forwarding packets from c_i to c_j, and the local routing of v_i is able to forward packets to v_j, then we state that the local routing rule fulfills the cluster-level routing rule and allow v_i to execute local routing. If the cluster-level routing rule prohibits forwarding packets from c_i to c_j, then node v_i removes the route to v_j from its local routing rules.

An example of cluster-level flow control is shown in Fig. 2. Suppose the distributed routing protocol sets the traffic flow route from v_1 to the SDN controller as $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5$. The network is partitioned into four clusters c_1, c_2, c_3, c_4. The SDN controller sets the cluster-level routing rules to the border nodes of each cluster. Traffic flow between c_1 and c_2, c_1 and c_3, c_2 and c_4 is allowed, while flow between c_3 and c_4 is prohibited. Routing from v_1 to v_2 does not fulfill the cluster-level routing, hence it is blocked. The border nodes of c_4 and c_3 rebuild their local routing tables. Finally, the traffic flow route from v_1 to the SDN controller becomes $v_1 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow v_9 \rightarrow v_5$.

III. Cluster Flow Management in SD-WSN

We present the design of CluFlow in this section, including an approach to partition the network into clusters with a minimum number of border nodes, and a protocol for cluster-based SDN control.

A. Basis for Monitoring Flows Across Clusters

Assume E_i represents the edges incident to node v_i. Let I_e^i and O_e^i be the incoming and outgoing flows of node v_i on the edge $e \in E_i$. Node $v_i \in c_k$ is named as a *border node* of cluster c_k if v_i has a neighbor not in c_k. Denote the set of border nodes of c_k by b_k and let Γ^i denote the set of border edges of border node v_i, viz. the edges in E_i that cross clusters.

We first assume v_i does not generate or consume flows of packets, i.e. it only forwards traffic generated by other nodes. The unit of flow is defined as the number of packets forwarded in a time unit. According to the *flow-conservation law* [8], the sum of incoming flows equals the sum of outgoing flows in node v_i, which means $\sum_{e\in E_i} O_e^i - \sum_{e\in E_i} I_e^i = 0$. We extend the flow-conservation law from a single node to a cluster of nodes. The sum of incoming flows to the cluster c_k (through border nodes) equals the sum of outgoing flows, which is denoted as $\sum_{v_i\in b_k}\sum_{e\in\Gamma^i} O_e^i - \sum_{v_i\in b_k}\sum_{e\in\Gamma^i} I_e^i = 0$.

Suppose the cluster c_k generates or consumes flows. We assume the border nodes have the destination and source addresses of each flow, and the addresses of the nodes in their clusters. To calculate the flow over clusters, we name the incoming flow, which is via border node v_i and routed to the sink node inside the cluster, as T_e^i; and the outgoing flow, which is via border node v_i and originated from source nodes inside the cluster, as S_e^i. Then we have $\sum_{v_i\in b_k}\sum_{e\in\Gamma^i}(O_e^i - S_e^i) - \sum_{v_i\in b_k}\sum_{e\in\Gamma^i}(I_e^i - T_e^i)) = 0$. Based on this formula, the SDN controller could evaluate the incoming and outgoing flows of the cluster by observing O_e^i, S_e^i, I_e^i, and T_e^i.

B. Minimize the Number of Cluster Border Nodes

Different partitioning of a network into clusters produces different number of border nodes. Based on our cluster level flow control, fewer border nodes means less communication flow of control with the SDN controller. In this section, we aim to partition a network into clusters with minimum number of border nodes.

1) Requirements on Clusters: Before partitioning the network into clusters, we assume there exist requirements about the number, position, the minimum required size and range of partitioned clusters. We require that the borders of clusters must be selected inside a specified sub-network area of the graph. We call this sub-network through which the cluster border can be selected as the *border-selection-belt* Θ. Correspondingly, the sub-networks $\{h_1, ...h_k, ...h_u\}$ are the areas which must reside within clusters, where u is the required total number of clusters. We refer to these areas as *cluster-contained-subnets*. It is required that $\{h_1, ...h_k, ...h_u\}$ are disconnected, which means there are not edges passing between any pair of nodes belonging to two different cluster-contained-subnets. These *cluster-contained-subnets* fulfill the requirements about the number, position and the minimum size of partitioned clusters. The size of these pre-specified sub-network fields depends on the user requirements. As shown in the example of Fig. 3(a), the WSN is categorized into two types of fields $\{h_1, h_2, h_3, h_4\}$ and Θ. The requirement could specify each cluster-contained-subnet to contain only one single node or several nodes.

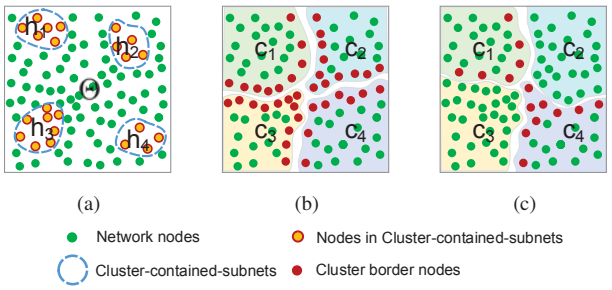


Fig. 3. (a) An example of specifying border-selection-belt and cluster-contained-subnets. (b) Monitor all the border nodes of every cluster. (c) Monitor the minimized number of border nodes of selected clusters.

2) Example of Cluster Border Nodes: Suppose the network is required to be partitioned into four clusters c_1, c_2, c_3 and c_4. We make two types of cluster partitioning patterns. The first cluster partitioning pattern is shown in Fig. 3(b). All the border nodes of every cluster are used to monitor the communication flow. These flows of cluster border nodes are further used to calculate the flow across clusters. The second partitioning pattern is shown in Fig. 3(c). This approach partitions the network in a different cluster pattern, and we only monitor the flow of the border nodes in clusters c_1 and c_4. The approaches in both Fig. 3(b) and Fig. 3(c) can calculate the flow among clusters, but the approach in Fig. 3(c) uses much less number of cluster border nodes.

3) Solution to Minimize Cluster Borders Nodes: We formally define the problem as follows. Define R as any set of nodes in Θ. We require that the network $G = (V, E)$ is partitioned into clusters $\{c_1, ...c_k, ...c_u\}$ after removing the nodes R and the edges connected with R, such that any two clusters do not have connected edges, and all the flows passing among clusters completely pass across border nodes R. The aim is to select R in Θ with minimum number of nodes. We express the problem as

$$\textbf{Objective}: Min\,|R| \qquad (1)$$
$$\textbf{Subject to}: (h_k \subset c_k) \wedge (R \subset \Theta)$$

The problem above is a variant of the k-way node separators (NS) problem, which is known to be NP-hard for general graphs [9] and for which heuristic algorithms, e.g. [10], have been proposed. However, the existing k-way NS algorithms cannot be used for our variant. Because, to manage the flow of SD-WSN, besides requiring to minimize the number of separator nodes, the solution must have the following properties:

- The computing complexity must be small to enable the SDN controller to quickly find cluster border nodes after any network changes.
- The size of partitioned clusters do not need to be balanced. We only require that node separators are inside the border-selection-belt Θ.

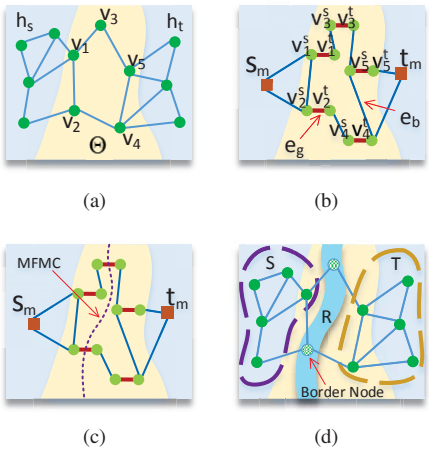


Fig. 4. (a) Specify the border-selection-belt Θ and cluster-contained-subnets h_s and h_t. (b) Merge h_s and h_t. Split the nodes in Θ and set edge weights. (c) Use MFMC from s_m to t_m. (d) Specify R as border nodes.

Algorithm 1: Select Border Nodes of Clusters

for *Each* h_i *in* G **do**
  for *Each* h_j $(j \neq i)$ *in* G **do**
    Merge h_i as node s_m and h_j as node t_m.
    for *Each node* v_g *in* Θ **do**
      Split into two nodes v_g^s and v_g^t.
    Connect v_g^s to previous hop.
    Connect v_g^t to next hop.
    Set edge weight of e_g to w_g and others to w_b.
    Make MFMC from s_m to t_m.
  Calculate overlapping set φ_i as cluster c_i.
  Remove φ_i from G.

Therefore, we propose a light-weight k-way node separators solution, which includes three steps. Step I: we utilize a Max-Flow-Min-Cut (MFMC) algorithm to solve NS for two clusters (Section III-B3a). In our implementation, we choose the Boykov-Kolmogorov MFMC algorithm with a worst-case complexity of $O(mn^2|Cost|)$, in which $|Cost|$ is the sum of the costs of boundary edges [11]. Step II: we extend the approach of Step I to multiple clusters (Section III-B3b). Its complexity is $O(|C|^2)$, in which $|C|$ is the number of clusters. Step III: we reassign the border nodes into minimum number of clusters, and further reduce some redundant border nodes (Section III-B3c). In this step, we utilize the solution of Minimum Vertex Cover (MVC) problem [12] to eliminate redundant separator nodes. Although it involves an NP-hard problem, the calculation is only performed on a small number of cluster border nodes.

*a) **Step I – Partition to Two Clusters**:* Suppose a network is required to be partitioned into two clusters c_s and c_t. The border of c_s and c_t is required to be inside the border-selection-belt Θ. The cluster-contained-subnets h_s and h_t should be a sub-set of clusters c_s and c_t, respectively. As shown in Fig. 4(a), $h_s \subset c_s$, $h_t \subset c_t$, and $h_s \cup h_t \cup \Theta = V$. We solve NS for two clusters as follows.

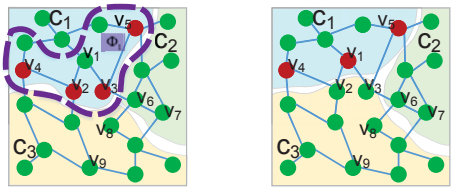
(i) Merge h_s into node s_m and h_t into node t_m. We split each node v_g of Θ into two nodes v_g^s and v_g^t and connect them by an edge e_g. We use e_g as the common notation for the link edge of each pair of split node. For the traffic flow from s_m to t_m, if v_g has a previous hop node v_f in Θ, we connect v_f^t with v_g^s. If v_g has a connection with s_m and t_m, we connect s_m and v_g^s, t_m and v_g^t. In this new topology, we denote each edge except e_g as a common notation e_b. In our implementation, we use the hop distance to s_m to determine the relative flow direction between v_g and v_f. Suppose v_g and v_f are two neighbor nodes in Θ. If the hop distance from v_g is smaller than from v_f, then v_g is the previous hop of v_f and we connect v_g^t to v_f^s. Otherwise, we connect v_f^t to v_g^s. If the hop distance from v_g equals from v_f, we connect v_f^s to v_g^s.

(ii) We set the edge weight of each e_g to w_g, and all the other edge weights to w_b. The value of w_g is set to 1. The value of w_b is set to a constant value that is larger than the number of edges e_g. The operation to split nodes and set edge weights is shown in Fig. 4(b).

(iii) We use Boykov-Kolmogorov MFMC algorithm [11] to cut the edges of the new topology from s_m to t_m as Fig. 4(c). The cut edges e_g represent the split nodes, which form the border R to partition the network into two clusters as shown in Fig. 4(d). The other nodes are separated into two sets of nodes S and T. To form clusters c_s and c_t, the border nodes R combine with either S or T. If R combines with S, then $c_s = S \cup R$ and $c_t = T$. If R combines with T, then $c_s = S$ and $c_t = T \cup R$. The border nodes R are used to monitor and control the flow between the two clusters c_s and c_t.

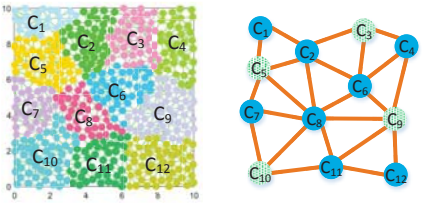
*b) **Step II – Partition to Multiple Clusters**:* Assume we have cluster-contained-subnets $\{h_1, ..., h_i, h_j, ...h_q\}$ with $i, j \in [1, q]$. First, calculate NS as in Section III-B3a between h_i and every other cluster-contained-subnet $\{h_j | j \in [1, q], j \neq i\}$. Name c_i^j and c_j^i as the partitioned clusters containing h_i and h_j respectively. The cluster border of cluster c_i^j and c_j^i is R_i^j. As explained in Section III-B3a, we combine R_i^j into cluster c_i^j, so $R_i^j \subset c_i^j$. Second, calculate the intersection set of $\left\{c_i^j | j \in [1, q], j \neq i\right\}$ as $\varphi_i = \bigcap_{j \in [1,q], j \neq i} c_i^j$. We use φ_i as the partitioned cluster c_i. Third, remove φ_i from the network G. This three-step process continues for each cluster-contained-subnet until all clusters are partitioned. Finally, each node is categorized to a cluster with a *cluster-ID*. The operation to select border nodes of multiple clusters base on Step I and Step II is shown in Alg. 1.

*c) **Step III – Reduce Redundant Border Nodes**:* In Section III-B3b, we select the minimum number of border nodes between each pair of clusters, while the border nodes of the intersection set φ_i is not optimized. In this step, we eliminate some redundant border nodes. For example, as shown in Fig. 5(a), we make the cluster c_1 via Steps I and II as follows. Firstly, the cluster border nodes between c_1 and



(a) (b)

Fig. 5. The network nodes are partitioned into cluster c_1, c_2 and c_3. (a) The nodes $\{v_2, v_3, v_4, v_5\}$ are the border nodes of cluster c_1 after Step II of Section III-B3b. (b) Node v_1 replaces v_2 and v_3 as the border node.



(a) (b)

Fig. 6. (a) The network nodes are partitioned into clusters with different colors. (b) The clusters are abstracted into a cluster-level topology. The nodes in solid blue are vertex cover clusters.

Algorithm 2: Reduce Redundant Border Nodes

```
for Each cluster c_i do
    Set cluster weight as 1/(|b_i ∪ β_i|).
Calculate MVC on cluster-level topology.
if Border node in non-VC cluster then
    Change to the cluster-ID of neighbor VC cluster.
Calculate MVC on b_i^m ∪ δ_i^m as λ_i.
Select Min{|b_i^m|, |λ_i|} as the border nodes of c_i^m.
```

c_2 are $\{v_3, v_5, v_8, v_9\}$, and the cluster border nodes between c_1 and c_3 are $\{v_2, v_3, v_4, v_6, v_7\}$. Secondly, the intersection area between c_1^2 and c_1^3 becomes cluster c_1 with border nodes $\{v_2, v_3, v_4, v_5\}$. Although we select the minimum number of border nodes for c_1^2 and c_1^3, v_1 can replace v_2 and v_3 as the border node of c_1 as shown in Fig. 5(b) to further decrease the total number of border nodes. We formalize the approach to reduce redundant border nodes as follows. The main operation flow is shown in Alg. 2.

(i) We reset the cluster-ID of all the border nodes selected in Section III-B3b to a minimum number of clusters. We convert it to the Minimum Vertex Cover (MVC) problem [12] as follows. In the first place, we abstract the clusters into a cluster-level topology as shown in Fig. 6, where each cluster-level node represents a cluster. If there exists edges between two clusters as in Fig. 6.(a), we connect the two cluster-level nodes. Next, we set the weight of each node in the cluster-level topology. The border nodes in cluster c_i are b_i after Alg. 1, and we call all the other border nodes that have edge connections with cluster c_i as β_i. The nodes set $b_i \cup \beta_i$ represents the maximum set of border nodes in c_i if re-categorizing the cluster-ID of β_i. To concentrate more border nodes in fewer clusters using MVC, we set weight value to each cluster. If the number of all the possible border nodes $|b_i \cup \beta_i|$ is high, we set a low weight value to the cluster c_i. In the implementation, we set the weight of cluster c_i to $1/(|b_i \cup \beta_i|)$. After that, we run the MVC algorithm on the cluster-level topology. If a border node belongs to a non-VC cluster, it changes its cluster-ID to the neighbor VC cluster. To differentiate with the notations before this step, c_i changes to c_i^m after re-categorizing the border nodes, and b_i changes to b_i^m.

(ii) Name δ_i^m as the subset of $c_i^m - b_i^m$, in which each node has at least a neighbor in b_i^m. To simplify the analysis, we suppose there are not sink nodes at b_i^m or δ_i^m. The nodes in b_i^m and δ_i^m are not in cluster-contained-subnets. Name $\Phi_i = b_i^m \cup \delta_i^m$. Fig. 5(a) illustrates the example Φ_1 of c_1. Then we calculate MVC on Φ_i as λ_i. We use λ_i as alternative border nodes to b_i^m. Because each edge in Φ_i has at least one endpoint in the MVC nodes λ_i, so monitoring λ_i can capture all the flows passing over Φ_i. The number of λ_i is not necessarily smaller than b_i^m. Therefore, we select Min$\{|b_i^m|, |\lambda_i|\}$ as the new border nodes of c_i^m. If λ_i are selected as the border nodes of cluster c_i^m, the non-VC nodes in b_i^m do not need to monitor the flow, and their cluster-ID are set to the neighbor cluster.

C. Protocol for Cluster based SD-WSN

CluFlow makes the SDN controller estimate flow among clusters by monitoring the flow at border nodes. The SDN controller controls traffic flow by injecting cluster-level routing rules to the border nodes. The management procedure of the SDN controller and nodes in WSN is shown in Alg. 3.

There are at least two benefits of utilizing SDN control in cluster-level routing. Firstly, compared with SDN management for every node in WSN, CluFlow trades granularity of SDN control for less communication load. Only cluster border nodes communicate with the SDN controller. The number of nodes that communicate with the SDN controller decreases. Secondly, cluster-level routing and local routing are decoupled. The nodes inside the clusters use only distributed local routing and do not need to request flow table entries from the SDN controller. The communication delay caused by requesting flow table entries therefore decreases.

IV. Experimental Setup and Results

In this section, we test and evaluate CluFlow in simulation and a real deployed WSN.

A. Benchmark Approaches

To evaluate the performance of CluFlow, three benchmark approaches are implemented to calculate the communication flow among clusters.

Algorithm 3: Cluster-based Flow Control in SD-WSN

▷ **Network Nodes:**

while *True* **do**
  if Discover new neighbor nodes. **then**
    Send "`local-links`" to SDN controller.
  if Receive "`set-border`" command. **then**
    Set self as border node
  if Self is border node. **then**
    Monitor local traffic flow.
    if Period of reporting **then**
      Send "`flow-report`" to SDN controller.
    if Receive "`cluster-level rules`". **then**
      Reconfigure local routing rules.

▷ **SDN Controller:**

while *True* **do**
  if Receive "`local-links`" from nodes. **then**
    Build topology of WSN.
    Partition clusters. ▷ *See Sec.III-B*
    Send "`set-border`" command to border nodes.
  if Receive "`flow-report`". **then**
    Calculate flow among clusters.
  if Update cluster-level route to border nodes. **then**
    Send "`cluster-level rules`".

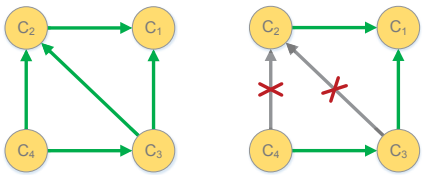
*1) Minimum Vertex Cover Nodes (**MVC**):* We monitor the traffic flow belonging to the minimum vertex cover (MVC) nodes in the network. The flows on all the other nodes are calculated based on the monitored flows in MVC nodes. The traffic flow of a cluster is calculated by the sum of the border nodes flows.

*2) Cluster Border Nodes of Voronoi Clustering (**CB**):* This approach requires cluster-contained-subnets and border-selection-belt as CluFlow. We first merge all the nodes in each cluster-contained-subnet as a single header node. Then, the network is partitioned into Voronoi clusters [13] based on the header nodes. We monitor the traffic flow of every cluster border node. The sum of incoming and outgoing flow of cluster border nodes is the flow of the cluster.

*3) Cluster Border Nodes of Minimum Vertex Cover Voronoi Clustering (**MVC-CB**):* We first partition the network into Voronoi clusters following the approach of CB. Next, the clusters are abstracted into a cluster-level topology. Then, we select the MVC clusters in the cluster-level topology. The border nodes of MVC clusters are used to monitor and control the communication flow. Finally, by the flow of MVC clusters, we calculate the traffic flow between each pair of clusters.

B. Cluster Level Flow Control

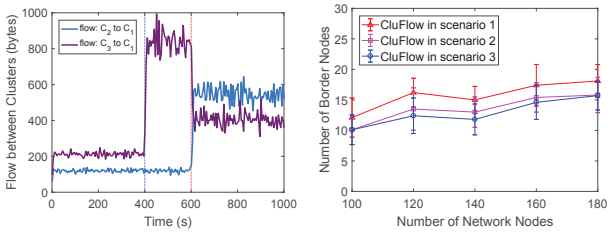
We now show the practicality of CluFlow Alg. 3 by controlling the cluster level traffic flow in a case study of Matlab simulation. The deployment area is 100m×100m, and the nodes are randomly deployed. The network consists of 200 nodes, and is partitioned into 4 clusters. There are 6.4 nodes on average within the transmission range. We assume a perfect wireless channel without packet loss. The sink node resides in c_1. Each node sends packets to the sink by the shortest path routing. The time interval between the present and the next sending time is uniformly distributed in (1, 8) seconds. The cluster level topology and flow without CluFlow control are shown in Fig. 7(a). The nodes in c_1 and c_3 send packets of 10 bytes. The nodes in c_2 and c_4 send packets of 10 bytes before 400s, and packets of 50 bytes after 400s. The SDN controller sets cluster level routing rules to block the flow between c_2 and c_3, c_2 and c_4 after 600s as shown in Fig. 7(b).



(a) Cluster level flow before cluster level SDN control.

(b) Cluster level flow after cluster level SDN control.

Fig. 7. Case study of traffic flow control among clusters by CluFlow. The SDN controller blocks the flow from c_3 to c_2, and from c_4 to c_2 in cluster level route at 600s.



(a) (b)

Fig. 8. (a) Realtime flow from c_2 to c_1, and from c_3 to c_1 (Section IV-B). (b) The number of border nodes of partitioned clusters with different size of border-selection-belt Θ (Section IV-C).

The realtime traffic flows from c_2 to c_1 and from c_3 to c_1 are shown in Fig. 8(a). The flow from c_2 to c_1 increases significantly from 400s to 600s, while the flow from c_3 to c_1 keeps at the same level. This is because the traffics generated by the nodes inside c_2 and c_4 all pass over c_2, which causes unbalanced flow among clusters. After 600s, by setting the cluster level routing rule, the traffic generated by the nodes inside c_4 passes over c_3. The cluster level SDN control above makes the flow from c_2 to c_1 and flow from c_3 to c_1 more balanced from 600s to 1000s.

C. Size of Border-Selection-Belt

We test how the size of border-selection-belt Θ affects the number of cluster border nodes by CluFlow. The number of

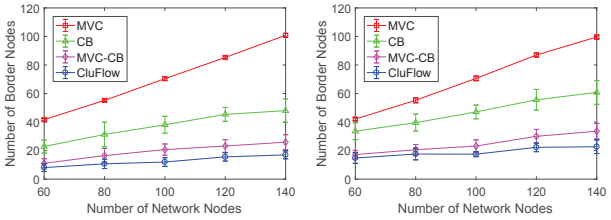


Fig. 9. The number of border nodes using CluFlow and the benchmark approaches with (a) 6 clusters and (b) 9 clusters.

nodes in different experiments are set to 100, 120, 140, 160 and 180, and the number of partitioned clusters is 4. The other setups of the network are the same as in Section IV-B. For each setup, we make 10 rounds of testing. We create Θ as follows. Firstly, we randomly select cluster header nodes in the network which have at least 8 hops away from each other. Secondly, Voronoi clusters are created based on the cluster header nodes. Name the border nodes set of all the Voronoi clusters as Θ_b. Thirdly, we create Θ in three scenarios. Scenario 1: the nodes set including Θ_b and the nodes which have 1 hop distance to Θ_b form Θ. Scenario 2: the nodes set including Θ_b and the nodes which have 2 hop distance to Θ_b form Θ. Scenario 3: all the nodes except the cluster header nodes form Θ.

The testing results are shown in Fig. 8(b). As the size of Θ increases, the number of border nodes decreases. The main reason is that larger Θ provides more flexibility to partition clusters, so that the possibility to partition clusters with less border nodes increases.

D. Number of Cluster Border Nodes

We count the number of cluster border nodes created by CluFlow and the other benchmark approaches. A smaller number of cluster border nodes means less requests and replies between nodes and the SDN controller, which could further benefit the performance of WSN, such as energy consumption. In the experiment, the number of nodes in the network are set to 60, 80, 100, 120 and 140 respectively. The network is partitioned to 6 and 9 clusters respectively. The other setups of the network are the same as in Section IV-B. For each setup, we make 10 rounds of testing.

Our results, as illustrated in Fig. 9, show that the number of border nodes created by CluFlow is much smaller than the benchmark approaches. As the total number of network nodes increases, the percentage of improvement increases. With 140 nodes and 6 headers, CluFlow has 83%, 65%, 34% less border nodes than MVC, CB and MVC-CB, respectively.

MVC-CB inherits some properties of CluFlow, including (i) abstracting the network to cluster-level topology and (ii) controlling the border nodes of MVC clusters. Therefore, compared with MVC and CB, the number of border nodes using MVC-CB is reduced. But MVC-CB only uses Voronoi cluster partition. So CluFlow using cluster partition Alg. 1 and Alg. 2 has the smallest number of cluster border nodes. Meanwhile, as the number of clusters increases from 6 to 9, the number of cluster border nodes becomes larger using CB, MVC-CB and CluFlow respectively. This means larger number of clusters has a smaller control granularity, while the cost for flow management of cluster border nodes will increase.

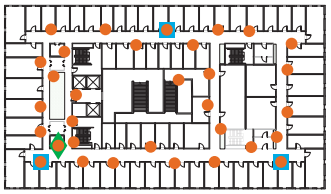


Fig. 10. The deployment of the WSN in the building. Orange circles ● represent the positions of the deployed nodes. The node with green diamond background ♦ is the SDN controller. The nodes with blue square background ■ are the headers of clusters.

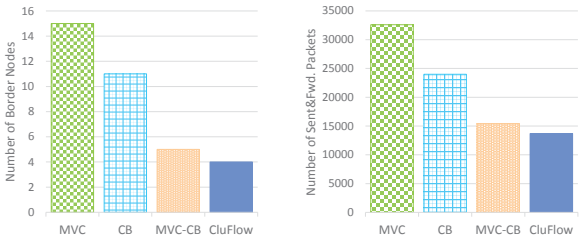


Fig. 11. (a) The number of border nodes. (b) The number of sent and forwarded IP packets in border nodes.

E. Performance in Real Indoor WSN

We setup a real indoor WSN in a university building to test CluFlow. We measure the number of border nodes and the communication cost. The deployed nodes are CC2650STK SensorTag motes, using Contiki 3.0 OS, IEEE 802.15.4 MAC standard. We use CSMA/CA collision avoidance, Contiki-Mac radio duty cycle, and RPL [14] routing protocol. The Tx power of each node is set to 0dBm, and Rx sensitivity is -100dBm. 32 nodes are deployed in the area of 65m×38m as shown in Fig. 10. The sink node is attached to a SensorTag Debugger DevPack, which links to a computer by a USB cable. The SDN controller runs on the computer, and communicates with the WSN through the sink node.

In the experiment, each mote reports the connectivity of neighbor nodes to the SDN controller every 30 seconds. The controller builds the topology of the network. The network is partitioned into 3 clusters based on 3 header nodes. The selected border nodes send monitoring data and routing requests to the controller every 3 seconds. Once the controller receives a request, it sends a reply back. The controller uses the monitoring data of all the border nodes to calculate the traffic flow among clusters. We do not instantiate cluster-level routing in this test. The border nodes do not change local routing rules after receiving the reply messages from the SDN controller.

The load size of each packet is 64 bytes. The experiment lasts for 600 seconds using CluFlow and each benchmark approach.

We count the number of cluster border nodes and the communication cost, i.e., the number of sent and forwarded IP packets in the border nodes. The results are shown in Fig. 11. Compared with the benchmark approaches, CluFlow utilizes the smallest number of border nodes and communication cost.

V. Related Work

Most existing WSN structures utilize a distributed control system. They are facing the same difficulties as traditional wired networks. Existing WSN management does not provide high-level abstraction. Dynamically changing control policy in WSN becomes increasingly difficult as the scale of WSN increases [6]. The research in [15] provides a solution to utilize OpenFlow in wireless networks. It uses the OpenFlow centralized controller for routing data traffic. SDN-WISE [16] designs and implements a complete SDN system in a real multi-hop wireless network. Its SDN components consist of SDN controller, topology manager, protocol stacks, and wireless motes. It provides a stateful solution and reduces the amount of communication between nodes and SDN controllers. The research in [17] creates an SDN framework for IoT systems based on SDN-WISE and Open Network Operating System (ONOS) [18]. To connect IoT and SDN, it extends the functionality of ONOS as the controller in WSN, while the communication protocol relies on SDN-WISE. In these frameworks, the SDN controller must rely on distributed routing to setup control flow in the nodes that are several hops away. To update flow table entries, the nodes and the SDN controller have to exchange request and reply messages over multiple hops periodically. This process causes much communication delay and overhead in wireless networks.

Some researches focus on increasing the performance of WSN, such as energy efficiency, task scheduling, routing, etc., using SDN structure. SDN-ECCKN [19] proposes an SDN-based energy management system for WSN. The system reduces the total transmission time to increase the network lifetime. [20] minimizes energy comsumption on sensors with guaranteed quality-of-sensing in multi-task software defined WSN. It utilizes a centralized SDN to formulate the minimum-energy sensor activation by jointly considering sensor activation and task mapping. The work in [21] presents an energy-efficient routing algorithm based on the framework of software defined WSN. To minimize the transmission distance and the energy consumption of sensor nodes, the algorithm partitions WSN into clusters and dynamically assigns tasks to the intra-cluster nodes by a cluster control node.

VI. Conclusion

We have presented a cluster-based SDN architecture CluFlow to manage communication flow in WSN, by controlling and monitoring the incoming and outgoing flow of cluster border nodes. CluFlow minimizes the number of border nodes and the communication overhead used for SDN control. Based on the simulations and the experiments in a real network, we have demonstrated that CluFlow significantly decreases the number of nodes and communication load needed by the SDN controller to control and monitor cluster-level communication flow compared with benchmark solutions.

References

[1] 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 Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.

[3] H. Mostafaei and M. Menth, "Software-defined wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 119, pp. 42–56, 2018.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM*, vol. 38, no. 2, pp. 69–74, 2008.

[5] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[6] T. Luo, H.-P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[7] R. Rahmani, H. Rahman, and T. Kanter, "On performance of logical-clustering of flow-sensors," *International Journal of Computer Science Issues*, vol. 10, no. 5, pp. 1–13, 2013.

[8] Y. Breitbart, C.-Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz, "Efficiently monitoring bandwidth and latency in ip networks," in *INFOCOM*, vol. 2. IEEE, 2001, pp. 933–942.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[10] P. Sanders, C. Schulz, D. Strash, and R. Williger, "Distributed evolutionary k-way node separators," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 345–352.

[11] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1124–1137, 2004.

[12] G. Karakostas, "A better approximation ratio for the vertex cover problem," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2005, pp. 1043–1050.

[13] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[14] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, and J. P. Vasseur, "Rpl: Ipv6 routing protocol for low power and lossy networks." ROLL Working Group, 2011.

[15] A. Detti, C. Pisa, S. Salsano, and N. Blefari-Melazzi, "Wireless mesh software defined networks (wmsdn)," in *WiMob*. IEEE, 2013, pp. 89–95.

[16] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Sdn-wise: Design, prototyping and experimentation of a stateful sdn solution for wireless sensor networks," in *INFOCOM*. IEEE, 2015, pp. 513–521.

[17] A.-C. G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "Towards a software-defined network operating system for the iot," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 579–584.

[18] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[19] Y. Wang, H. Chen, X. Wu, and L. Shu, "An energy-efficient sdn based sleep scheduling algorithm for wsns," *Journal of Network and Computer Applications*, vol. 59, pp. 39–45, 2016.

[20] D. Zeng, P. Li, S. Guo, T. Miyazaki, J. Hu, and Y. Xiang, "Energy minimization in multi-task software-defined sensor networks," *IEEE transactions on computers*, vol. 64, no. 11, pp. 3128–3139, 2015.

[21] W. Xiang, N. Wang, and Y. Zhou, "An energy-efficient routing algorithm for software-defined wireless sensor networks," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7393–7400, 2016.