Computer Networks 187 (2021) 107788

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Cluster-based flow control in hybrid software-defined wireless sensor networks☆

Qingzhi Liu [a,*], Long Cheng [b], Renan Alves [c], Tanir Ozcelebi [d], Fernando Kuipers [e], Guixian Xu [f], Johan Lukkien [d], Shanzhi Chen [g]

[a] Information Technology group, Wageningen University & Research, The Netherlands
[b] School of Control and Computer Engineering, North China Electric Power University, China
[c] Universidade de São Paulo, Brazil
[d] Interconnected Resource-aware Intelligent Systems (IRIS) group, Eindhoven University of Technology, The Netherlands
[e] Embedded and Networked Systems group, Delft University of Technology, The Netherlands
[f] Electrical Engineering Department, Tampere University, Finland
[g] State Key Laboratory of Wireless Mobile Communications, China Academy of Telecommunication Technology, China

ARTICLE INFO

Keywords:
Software-defined wireless sensor networks
Hybrid SDN
Flow control
Network cluster
Multi-hop communication

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. However, SDN is not widely adopted in constrained networks, such as Wireless Sensor Networks (WSN), due to excessive control overhead, lossy medium, and in-band control channels. Therefore, a key challenge to enable Software-Defined Wireless Sensor Networks (SD-WSN) is to reduce the number of control messages required to configure the data plane. In this paper, we propose a cluster-based flow control approach in hybrid SDNs. Our approach is hybrid in the sense that it takes advantage of distributed legacy routing and centralized SDN routing. In addition, it makes a trade-off between the granularity of flow control and the communication overhead induced by the SDN controller. The approach partitions a network into clusters with minimum number of border nodes. Instead of handling the individual flows of each node, the SDN controller only manages incoming and outgoing traffic flows of clusters through border nodes, while the flows inside each cluster are controlled by a distributed legacy WSN routing algorithm. Our proof-of-concept implementations in both software and hardware show that our approach is efficient with respect to reducing the number of nodes that must be managed and the number of control messages. In comparison to benchmark solutions with and without clustering, our solution reduces communication costs for flow configuration in an SD-WSN at least by 27% and at most by 88% respectively, without degrading packet delay nor delivery rate.

1. Introduction

Software-Defined Networking (SDN), in comparison to traditional networking, provides improved flexibility and reduced complexity when it comes to flow management [2,3]. Given the advantages and large-scale adoption 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 a wireless sensor network (WSN) [4]. However, most SDN research is focusing on wired networks, and only a few initiatives have attempted to extend the benefits of SDN to the wireless domain [5,6].

A WSN typically consists of resource-constrained sensor nodes for monitoring the physical conditions of the environment, while the SDN paradigm provides a simple and flexible control approach to communication networks [7]. The confluence of these techniques is called Software-Defined Wireless Sensor Networks (SD-WSNs) [8–10]. Fig. 1 illustrates a generic architecture of a SD-WSN. In that architecture, the

☆ An earlier 8-page version (Liu et al., 2019, [1]) of this paper was presented at the IEEE Wireless Communications and Networking Conference (WCNC), April 2019. Compared to the earlier version, we have (1) restructured and rewritten the entire paper; (2) redefined our proposed system model; (3) provided examples plus a mathematical proof w.r.t. the performance of our clustering algorithm; (4) performed five additional novel experiments that demonstrate the efficacy of our solution; (5) extended the related work section; and (6) added a section about future work.

* Corresponding author.
E-mail address: qingzhi.liu@wur.nl (Q. Liu).

https://doi.org/10.1016/j.comnet.2020.107788
Received 9 July 2020; Received in revised form 1 December 2020; Accepted 29 December 2020
Available online 5 January 2021

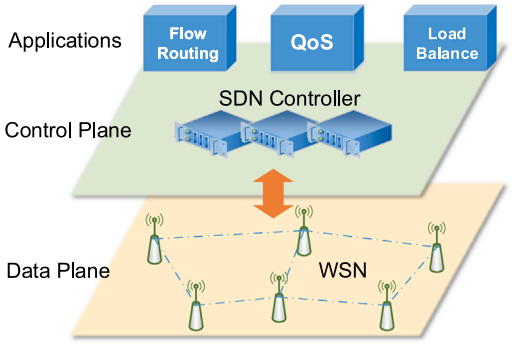


Fig. 1. Architecture of a Software-Defined Wireless Sensor Network (SD-WSN).

sensor nodes only perform packet forwarding, while all the control-plane operations, such as flow routing [11], Quality-of-Service (QoS) control [12], and load balancing [13], are performed by a logically centralized controller. Compared to the distributed control of a WSN, an SDN controller is able to manage and optimize WSN performance, such as energy consumption and communication flow, based on a global view of the entire network.

To implement an SD-WSN, the SDN architecture for wired networks must be mapped to WSN, which involves several difficulties:

- To achieve fine-grained flow control granularity, most existing SDN architectures require frequent message exchange between the data plane and the control plane [14]. Although this overhead is often acceptable in wired networks, the case for WSN is different. In a WSN, the control and data flows share the same wireless channel. Given that most wireless channels have limited bandwidth (in comparison to wired networks), the SDN control flows may significantly interfere with the data flows. For example, a burst of control packets requesting new flow table entries could stress the available wireless bandwidth.
- Nodes in an SD-WSN cannot completely decouple the data plane and control plane. In a typical SD-WSN architecture, the nodes and SDN controllers do not have wired connections. They transmit data via multi-hop wireless communication. Therefore, the nodes have to maintain a distributed local routing table for finding the SDN controller and receiving routing commands.

The observations above imply that the SDN architecture in wired networks cannot directly be applied to a WSN. Instead, to take advantage of the concept of SDN within a WSN, we need to balance the benefits and the communication overhead of SDN. Compared with a pure SDN paradigm, a hybrid SDN contains a mix of centralized SDN control and legacy network control, and thus shows the benefits of both paradigms [15–17]. Therefore, we aim to leverage hybrid SDN solutions to solve the above difficulties.

In this paper, we propose a cluster-based flow control approach called **CluFlow**. CluFlow is a hybrid SDN solution. It takes advantage of distributed legacy WSN routing and centralized SDN routing. Meanwhile, it makes a trade-off between the granularity of flow control and the communication overhead induced by the SDN controller. The properties of CluFlow are twofold. Firstly, CluFlow adopts network clustering to control traffic flows on the cluster level instead of at the level of individual nodes, which decreases the number of nodes and messages that are involved in flow control within an SD-WSN. Secondly, CluFlow makes SDN control work in parallel with distributed routing. 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. Compared with existing SD-WSN solutions [18], the novelty of CluFlow is in two aspects. Firstly, we propose a solution for SD-WSN that combines centralized SDN-based routing among clusters and distributed legacy routing inside clusters. Secondly, to the best of our knowledge, this is the first work that partitions a network into clusters with minimum cluster border nodes.

In this paper, we realize the proposed design and provide the following main contributions:

- We take a graph-theoretic approach for clustering the network with the goal of minimizing the number of border nodes. The SDN controller manages communication by monitoring and controlling the border nodes of clusters.
- We propose a priority scheme to coordinate legacy WSN routing and SDN control, where cluster-level routing performed by the SDN controller has a higher priority than legacy routing. This hierarchical routing decreases the communication overhead of SDN control in WSNs.
- We implement an SD-WSN in simulation and real deployments, in which SDN control operates together with legacy distributed routing protocols.

This paper is organized as follows. The system model is presented in Section 2. Our solution of cluster-based flow control for hybrid SD-WSNs is addressed in Section 3. The simulation and hardware experiments are presented in Section 4 and the related work is discussed in Section 5. The future research directions are discussed in Section 6. Finally, we conclude this article in Section 7.

2. System model

We represent the network as an undirected graph $G = (V, E)$, in which $V = \{v_1, \dots v_i, \dots v_n\}$ represents the set of $n = |V|$ nodes and $E = \{e_1, \dots e_j, \dots e_m\}$ represents the set of $m = |E|$ edges. The nodes in the network transmit data via multi-hop communication. The nodes that share an edge are called neighbors. Suppose the set of nodes V is partitioned into clusters $C = \{c_1, \dots c_k, \dots c_u\}$ with $u = |C|$, we make the following system assumptions with respect to our cluster-based flow control solution:

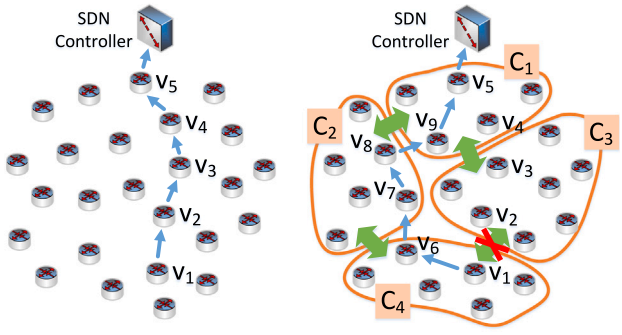
- We assume that there is one central SDN controller that is responsible for partitioning the network (in practice this could be multiple logically centralized controllers). Each node in the WSN reports its neighbor connectivity to the SDN controller. The SDN controller builds the WSN topology and partitions the network.
- Our solution targets a static network topology. Once the topology of the WSN changes, the nodes would report the new connectivity to the controller, and the controller would re-partition the new topology into clusters.

Cluster Head Nodes: To set the number and position of clusters, we specify cluster head nodes $\{h_1, \dots h_k, \dots h_u\}$, in which u is the total number of clusters. Each head node must reside in a cluster. We require that $\{h_1, \dots h_k, \dots h_u\}$ are disconnected, which means there are no edges connecting any pair of cluster head nodes.

Cluster Border Nodes: If node v_i belongs to cluster c_k and one of its neighbor nodes belongs to another cluster, then we call v_i as a *border node* of cluster c_k. We refer to all the border nodes of cluster c_k as node set b_k.

3. Flow control in hybrid SD-WSNs

In this section, we present the design of CluFlow, including the solution overview, an algorithm for minimizing the number of border nodes, and a protocol for cluster-based SDN control.



(a) Distributed legacy routing. (b) Cluster-level routing.

Fig. 2. Cluster-based flow control in a hybrid SD-WSN.

3.1. Solution overview

In the design of CluFlow, centralized SDN control and decentralized legacy routing control coexist in the WSN. On the one hand, each WSN node operates legacy routing protocols. On the other hand, an SDN controller partitions the network to clusters and controls the communication flow among clusters. Specifically, the SDN controller sets routing rules at the cluster border nodes, which is called *cluster-level* routing, e.g. forward data flow from cluster c_i to cluster c_j. In this condition, both legacy routing protocols and SDN routing control are performed in the border nodes.

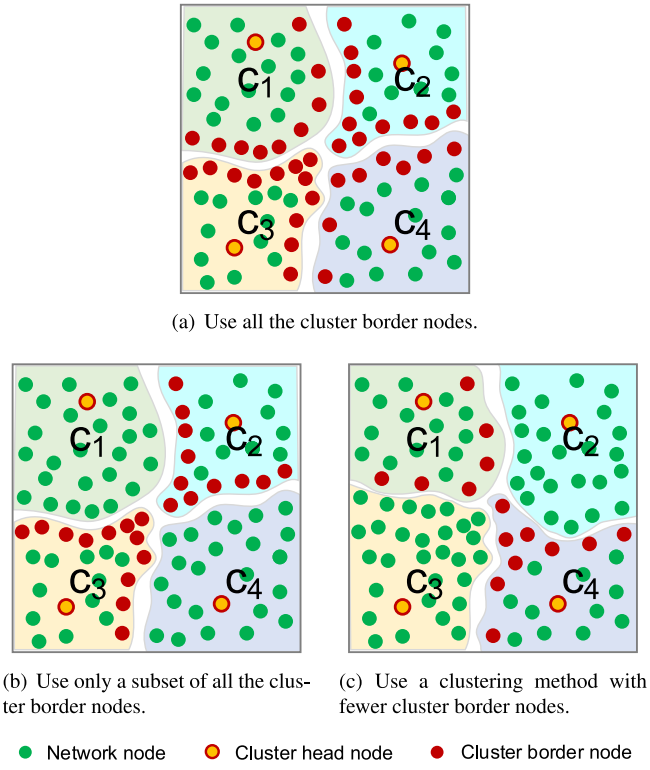
To coordinate the hybrid routing control, we require the cluster-level routing rules to have higher priority than the local routing rules in the border nodes. For example, suppose v_i and v_j are two cluster border nodes, and have a linked edge. $v_i \in c_i$ and $v_j \in c_j$.

- If the cluster-level routing rule allows forwarding packets from c_i to c_j, and the local routing of v_i is "forwarding packets to v_j", then it means the local routing rule fulfills the cluster-level routing rule. Thus v_i is allowed to execute local routing.
- If the cluster-level routing rule prohibits forwarding packets from c_i to c_j, then it means the local routing rule conflicts with the cluster-level routing rule. Thus node v_i removes the route to v_j from its local routing table.

An example of cluster-level flow control is shown in Fig. 2. Suppose the distributed routing from v_1 to the SDN controller is $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5$, as shown in Fig. 2(a). To use the cluster-level flow control, the network is partitioned into four clusters c_1, c_2, c_3, c_4 as shown in Fig. 2(b). The SDN controller sets the cluster-level routing rules in the border nodes of each cluster. The cluster-level routing rules are: (i) traffic flows between c_1 and c_2, c_1 and c_3, c_2 and c_4 are allowed; (ii) traffic flow between c_3 and c_4 is prohibited. So the routing from v_1 to v_2 does not fulfill the cluster-level routing, hence it is blocked. Thereafter, the border nodes of c_4 and c_3 rebuild their local routing tables. Finally, the route from v_1 to the SDN controller becomes $v_1 \rightarrow v_6 \rightarrow v_7 \rightarrow v_8 \rightarrow v_9 \rightarrow v_5$.

Based on the analysis above, we found that it is feasible to control the cluster-level data flow by cluster border nodes. This hybrid SD-WSN control brings benefits to the following perspectives.

- Easy deployment: Only a limited number of WSN devices, i.e. cluster border nodes, need to install SDN control software, which largely reduces the deployment time and cost.
- Fault tolerance: The operation of legacy routing protocols and the cluster-level control are decoupled. If the SDN control flow is congested or the controller has a failure, the WSN devices can still use distributed legacy routing protocols to control data flow.



(a) Use all the cluster border nodes.

(b) Use only a subset of all the cluster border nodes.

(c) Use a clustering method with fewer cluster border nodes.

Fig. 3. Use cluster border nodes for controlling the flow of clusters. The network topology of Fig. (a), (b), and (c) are identical. The network is partitioned to clusters c_1, c_2, c_3, and c_4.

- High scalability: The communication overhead caused by the SDN controller is scalable, which can be tuned by controlling the size of clusters.

However, the existing network clustering solutions cannot optimally partition the network and control the cluster border nodes for two reasons.

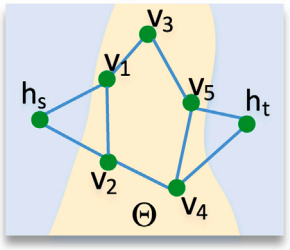
- Firstly, monitoring and controlling the cluster border nodes of all the clusters would cause replicated operations. For example, the network in Fig. 3(a) is partitioned into four clusters c_1, c_2, c_3 and c_4. In these clusters, the incoming flow to c_1 equals the sum of the outgoing flow from c_2 to c_1 and from c_3 to c_1. Therefore, there is no need to monitor all the cluster border nodes of c_1. Instead, we only need to monitor the cluster border nodes of c_2 and c_3 as shown in Fig. 3(b).
- Secondly, fewer border nodes means less control flow with the SDN controller. For example, the number of cluster border nodes in Fig. 3(c) is smaller than Fig. 3(b). Although there are various methods for partitioning a network into clusters, to the best of our knowledge, there is no one suitable for our SD-WSN solution.

To cope with these problems, we present our approach to partition the network to clusters with a minimum number of cluster border nodes in the next section.

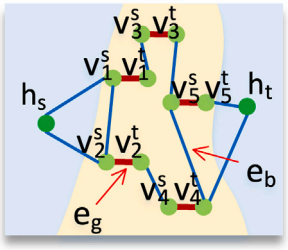
3.2. Minimize cluster border nodes

3.2.1. Problem definition

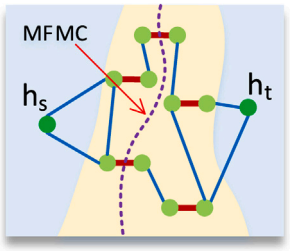
We formally define the problem of clustering with a minimum number of cluster border nodes as follows. Name the set of network nodes excluding the cluster head nodes in $G = (V, E)$ as Θ. Define R as a set of nodes in Θ. We require that the network G is partitioned



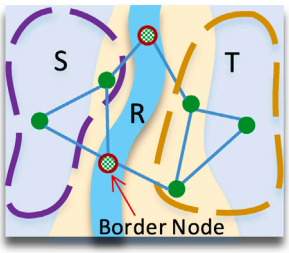
(a) Specify cluster head nodes $\{h_s, h_t\}$ and Θ.



(b) Split the nodes in Θ.



(c) Set edge weights and use MFMC from h_s to h_t.



(d) Specify R as the cluster border nodes.

Fig. 4. Partition a network into two clusters.

into clusters after removing all the nodes in R, such that each cluster contains a cluster head node and any two clusters do not share a single edge. The aim is to select R in Θ with minimum $|R|$. The problem is expressed as

$$\begin{aligned} &\textbf{Objective}: \ Min\,|R| \\ &\textbf{Subject to}: \ (h_k \subset c_k) \wedge (R \subset \Theta) \end{aligned} \tag{1}$$

The problem above is a variant of the k-way node separators (NS) problem, which is known to be NP-hard for general graphs [19] and for which heuristic algorithms, e.g. [20], have been proposed. However, k-way NS algorithms cannot directly be used for our variant. Because, to manage the flow of a SD-WSN, besides requiring to minimize the number of separator/border nodes, the solution must have the following properties:

- The computational complexity must be small to enable the SDN controller to quickly find cluster border nodes after any network changes.
- The sizes of the partitioned clusters do not need to be balanced. We only require that each cluster head node resides in a cluster.

3.2.2. Algorithm

We propose a light-weight k-way node separators solution for partitioning the network into clusters.

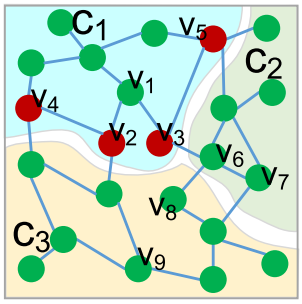
Step I - Partition Network to Clusters. We first introduce a method to partition a network into two clusters. After that, we extend this method to multiple clusters.

Two Clusters: Suppose a network is required to be partitioned into two clusters c_s and c_t. The cluster head nodes h_s and h_t are required to be clustered inside 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 the problem as follows:

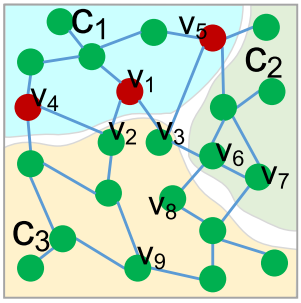
(i) We split each node v_g of Θ into two nodes v_g^s and v_g^t and connect them by an edge e_g. Suppose v_g has a neighbor node v_f in Θ. If the hop distance from v_g to h_s is smaller than from v_f to h_s, then v_g is the previous hop of v_f and we connect v_g^t to v_f^s. Otherwise, we connect v_f^t to v_g^s. If the hop distance from v_g equals that from v_f, we connect v_f^s to v_g^s. If v_g has a connection with h_s, we connect h_s and v_g^s. If v_g has a connection with h_t, we connect h_t and v_g^t. An example to split nodes is shown in Fig. 4(b).

(ii) Denote the edges except e_g as e_b in the new topology. We set the edge weight of e_g to w_g, and the edge weight of e_b 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 total number of edges e_g. After that, we use the Boykov–Kolmogorov Max-Flow-Min-Cut (MFMC) algorithm [21] to cut the edges of the new topology from h_s to h_t as shown in Fig. 4(c).

(iii) The cut edges of MFMC represent the split nodes, which form the node set R to partition the network into two clusters, as shown in Fig. 4(d). The other nodes are separated into two sets 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.



(a) Nodes $\{v_2, v_3, v_4, v_5\}$ are the border nodes of cluster c_1.



(b) Nodes $\{v_1, v_4, v_5\}$ are the border nodes of cluster c_1.

Fig. 5. An example of redundant cluster border nodes.

Multiple Clusters: Assume we have cluster head nodes $\{h_1, \ldots, h_i, h_j, \ldots h_q\}$ with $i, j \in [1, q]$. Based on the method for partitioning two clusters, we partition the network into multiple clusters as follows:

(i) Partition clusters between h_i and the other cluster head node $\{h_j | j \in [1, q], j \neq i\}$ by the method for partitioning two clusters. Name c_i^j and c_j^i as the partitioned clusters containing h_i and h_j, respectively. The border nodes between c_i^j and c_j^i is R_i^j.

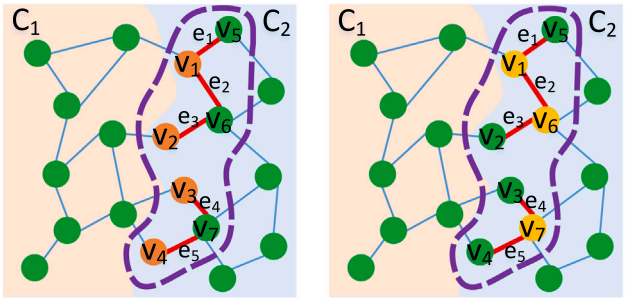
(ii) Calculate the intersection set of $\left\{(c_i^j \cup R_i^j) | j \in [1, q], j \neq i\right\}$ as $\varphi_i = \bigcap_{j \in [1,q], j \neq i} (c_i^j \cup R_i^j)$. We use φ_i as cluster c_i.

(iii) Remove φ_i from the network G. Repeat (i) to (iii) for each cluster head node until all clusters are partitioned.

Step II - Optimize Border Nodes. The intersection set φ_i in Step I is non-optimized. Therefore, we optimize the border nodes of φ_i in this step.

For example, as shown in Fig. 5(a), we cluster the network into c_1, c_2 and c_3 via Step I. Assume the cluster border nodes between c_1 and c_2 are $\{v_3, v_5, v_8, v_9\}$, and the cluster border nodes between c_1 and c_3 are $\{v_2, v_3, v_4, v_6, v_7\}$. The intersection set 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, respectively, the intersection area between c_1^2 and c_1^3 is not optimized. As shown in Fig. 5(b), v_1 can replace $\{v_2, v_3\}$ and the border nodes of c_1 become $\{v_1, v_4, v_5\}$, which further decreases the number of border nodes in c_1.

Proposition 1. *Suppose b_i is the set of border nodes in cluster c_i. Name δ_i as the subset of $c_i - b_i$, in which each node has at least a neighbor in b_i. The minimum vertex cover (MVC) of $b_i \cup \delta_i$ is an alternative to the border nodes b_i for controlling the incoming and outgoing flow of cluster c_i.*



(a) Use $b_2 = \{v_1, v_2, v_3, v_4\}$ as the border nodes.

(b) Use MVC nodes $\{v_1, v_6, v_7\}$ as an alternative to the border nodes.

Fig. 6. An example of an alternative to the border nodes for controlling the incoming and outgoing flow of a cluster.

Proof. Name the set of edges in $b_i \cup \delta_i$ as Z_i^e. Name the set of edges with one endpoint in b_i and another endpoint in δ_i as K_i^e. Based on the property of MVC, each edge in Z_i^e has at least one endpoint in the MVC nodes of $b_i \cup \delta_i$. Thus monitoring the flows of the MVC nodes belonging to $b_i \cup \delta_i$ can capture all the flows in Z_i^e. K_i^e is a subset of Z_i^e. Therefore, monitoring the flows of the MVC nodes belonging to $b_i \cup \delta_i$ can capture all the flows in K_i^e. Assume the data source and sink nodes of cluster c_i are not in b_i. In this condition, all the flows of c_i passes the edges in K_i^e. Therefore, monitoring the flows of the MVC nodes belonging to $b_i \cup \delta_i$ can capture all the flows of cluster c_i. This means the MVC nodes of $b_i \cup \delta_i$ can be used as an alternative set of border nodes to b_i. □

An example of the proposition is shown in Fig. 6. Suppose a network is partitioned to clusters c_1 and c_2, and node set $b_2 = \{v_1, v_2, v_3, v_4\}$ are the border nodes of c_2 as shown in Fig. 6(a). In c_2, the neighbor nodes of b_2 are $\delta_2 = \{v_5, v_6, v_7\}$, and the edges connected to b_2 are $\{e_1, e_2, e_3, e_4, e_5\}$. In this condition, we could manage the incoming and outgoing flows of c_2 by controlling the flows on the edges $\{e_1, e_2, e_3, e_4, e_5\}$ of b_2. At the same time, the MVC nodes of $b_2 \cup \delta_2$ are $\{v_1, v_6, v_7\}$. We could also manage the incoming and outgoing flows of c_2 by controlling the flows on the edges $\{e_1, e_2, e_3, e_4, e_5\}$ of $\{v_1, v_6, v_7\}$ as shown in Fig. 6(b). Therefore, the node set $\{v_1, v_6, v_7\}$ is an alternative to b_2 for controlling the incoming and outgoing flow of cluster c_2.

Based on the analysis above, for optimizing the border nodes of a cluster, we calculate MVC on $b_i \cup \delta_i$ as λ_i. Then, we use λ_i as an alternative to the cluster border nodes b_i selected by Step I. Because λ_i is not necessarily smaller than b_i, we finally select the smaller set between b_i and λ_i as the border nodes of cluster c_i.

3.3. Cluster-based flow control

In this section, we analyze the computational complexity of our clustering solution, and present an SDN control protocol based on the cluster-level control.

3.3.1. Computational complexity

The solution for partitioning networks with a minimum number of border nodes is shown in Alg. 1.

In Step I, we utilize a Max-Flow-Min-Cut (MFMC) method to partition a network into two clusters. In our implementation, we chose 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 [21]. Then we extend this method from partitioning two clusters to multiple clusters. Its complexity becomes $O(mn^2|Cost||C|^2)$, in which $|C|$ is the number of clusters.

In Step II, we optimize border nodes based on a solution to the Minimum Vertex Cover (MVC) problem [22]. Although the MVC problem is NP-complete, its calculation is only performed on a small number of cluster border nodes.

Algorithm 1: Clustering with Minimum Border Nodes

```
for Each h_i in G do
    for Each h_j (j ≠ i) in G do
        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 h_s to h_t.
    Calculate φ_i as cluster c_i.
    Use φ_i as c_i and remove φ_i from G.
for Each c_i do
    Calculate MVC on b_i ∪ δ_i as λ_i.
    Select Min{|b_i|, |λ_i|} as the border nodes of c_i.
```

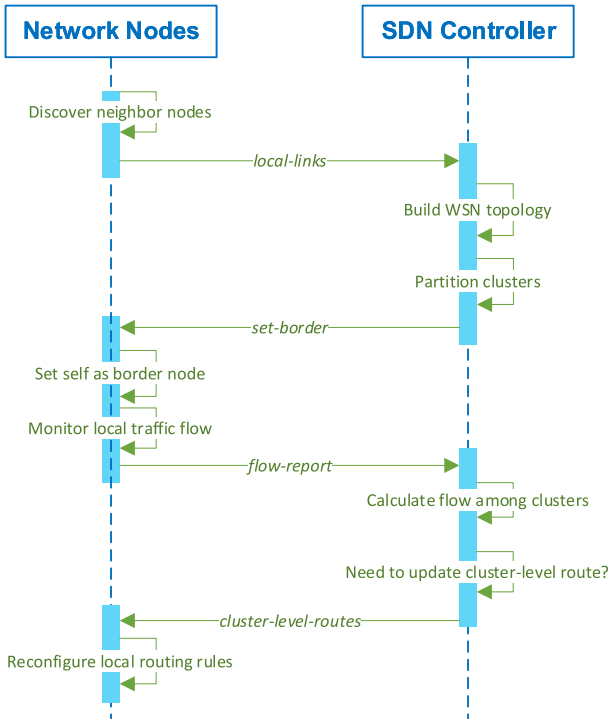


Fig. 7. Sequential diagram of CluFlow in SD-WSN.

3.3.2. Communication protocol

The main protocol of cluster-based flow control in SD-WSN is shown in Fig. 7. The protocol has three phases. In the first phase, each network node sends neighbor connectivity information `local-links` to the controller. The controller builds the topology of the network based on the received neighbor connectivity information and partitions the network into clusters using the algorithm in Section 3.2. Then the controller sends a `set-border` command to the selected cluster border nodes. The network nodes that receive the `set-border`

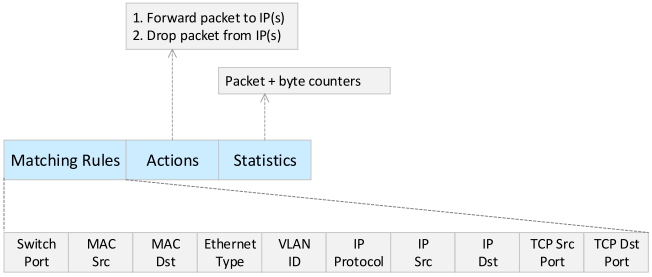
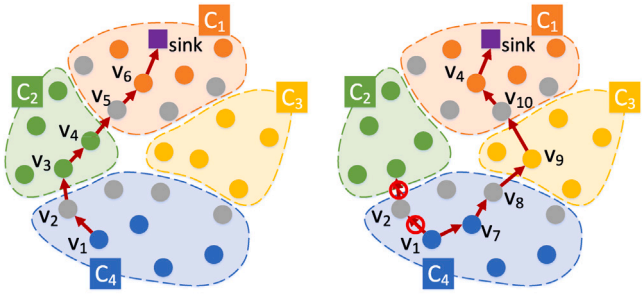


Fig. 8. An example of SD-WSN flow table that could be used for CluFlow.



(a) There is no flow blockage between c_2 and c_4.

(b) There is flow blockage between c_2 and c_4.

Fig. 9. An example of CluFlow protocol execution. The network is partitioned into four clusters with various colors. The gray nodes represent cluster border nodes.

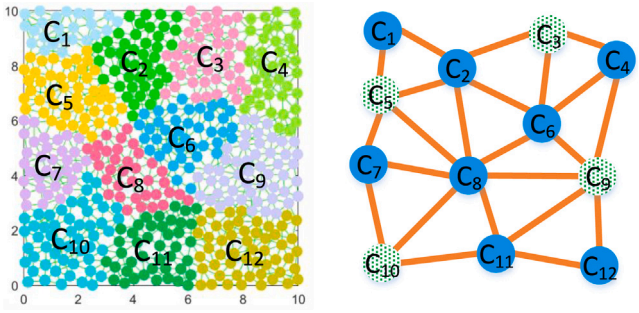
command set themselves as cluster border nodes. In the second phase, the cluster border nodes send local flow information `flow-report` to the controller. The controller calculates the flow among clusters based on the aggregated flow information. In the third phase, the controller checks whether it needs to update the cluster-level route based on the flow among clusters, and sends `cluster-level-routes` to the cluster border nodes as needed.

CluFlow can be deployed as a network management service, which is connected to the SDN northbound APIs [23]. In such a system, CluFlow requests network information, including neighbor connectivity and data flow of each node, from the SDN controller. At the same time, the SDN controller interacts with the forwarding plane of WSN nodes through southbound APIs of communication protocols. In this way, the SDN controller adds and adjusts routing entries in the internal flow-table of cluster border nodes. To control cluster-level flow, SDN controllers configure the action of cluster border nodes mainly through two actions, i.e., forwarding packets to a destination or dropping packets of a source.

CluFlow is able to work with the standard OpenFlow protocol [14], but it is not tied to any specific southbound protocol. An example of OpenFlow-based flow table that could be used for CluFlow is shown in Fig. 8. For example, the flow table entry could match the IP address of the source node, the IP address of the destination node, and the ports of the service. The detailed design about how to translate the routing policies of CluFlow to flow table entries is implementation-specific, and will be part of our future work.

Fig. 9 illustrates two examples of how SDN controls the flow among clusters. In the initialization stage, an SDN controller first gathers topological information of the WSN to build a local network representation. After that, the SDN controller calculates the network clustering and sends control messages to the borders nodes.

- *Without Blockage on Border Nodes:* Suppose node v_1 requests to transmit packets to the data sink as shown in Fig. 9(a). In the first place, node v_1 uses local routing to calculate the next hop, which is node v_2. It should be noted that v_1 does not receive flow configurations from the SDN controller, because it is not a border node. At the same time, the SDN controller configures the border node v_2 to allow traffic flow from c_4 to c_2. In this way, node v_2 forwards the packet from v_1 to v_3. The remainder of the route from v_1 to the sink node is configured in the same way.
- *With Blockage on Border Nodes:* Suppose the SDN controller is requested to re-configure the route from v_1 to the sink node. Fig. 9(b) shows the new SDN policy. To block traffic from c_4 to c_2, the SDN controller instructs node v_2 to drop packets from c_4 to c_2. Once node v_1 discovers the blockage on v_2, it removes node v_2 from its neighbor node table. After that, v_1 uses distributed routing in c_4, and builds another route to v_8. At the same time, the SDN controller configures the border node v_8 to allow traffic flow from c_4 to c_3. In the same way, the SDN controller re-configures the remainder of the route from v_1 to the sink node.



(a) The network nodes are partitioned into clusters.

(b) The clusters are abstracted into a cluster-level topology.

Fig. 10. Build a cluster-level network based on a partitioned network.

4. Experimental setup and results

In this section, we test and evaluate CluFlow in simulation and a real deployed WSN. Firstly, we examine the validity of Alg. 1 (Section 4.2). Secondly, we test the practicality of protocol shown in Fig. 8 (Section 4.3). Thirdly, we compare the number of border nodes between CluFlow and the benchmark approaches (Section 4.4 and Section 4.5). After that, we examine how the search space of clustering affects the number of cluster border nodes (Section 4.6). Then, we measure the communication load of CluFlow using real communication protocol stacks in an SD-WSN simulator (Section 4.7). Finally, we evaluate the number of border nodes and communication cost in a real deployed WSN (Section 4.8).

4.1. Benchmark approaches

We compare the performance of CluFlow with the following four benchmark solutions.

Minimum Vertex Cover Nodes (MVC): This benchmark solution monitors and calculates the communication flow belonging to the minimum vertex cover (MVC) nodes in the network. Then we calculate the flows on the edges based on the incoming and outgoing flows on the MVC nodes.

Cluster Border Nodes of Voronoi Clustering (CB): Based on cluster head nodes, we partition the network into Voronoi clusters [24]. We monitor the traffic flow of every cluster border node of all the clusters. The incoming and outgoing flows of the clusters is the sum of the incoming and outgoing flows of cluster border nodes, respectively.

Cluster Border Nodes of Minimum Vertex Cover Voronoi Clustering (MVC-CB): We first partition the network into Voronoi clusters using the solution ***CB***. After that, we change the network into a cluster-level topology as shown in Fig. 10. Specifically, we use a cluster-level

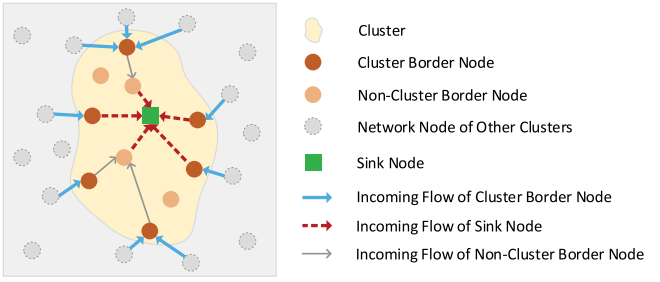


Fig. 11. Validity test of cluster border nodes. Every incoming flow of the cluster passes the cluster border nodes, so that the sum of the incoming flows in all the cluster border nodes equals the incoming flows of the sink node.

node to represent a cluster. If there exist edges between two clusters as in Fig. 10.(a), we connect the corresponding cluster-level nodes as in Fig. 10.(b). 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, we use the flows of MVC clusters to calculate the flows of the other clusters.

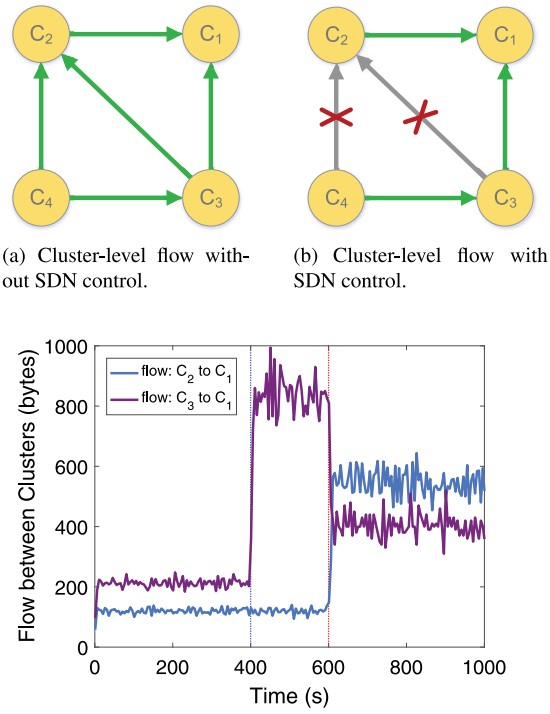
Balanced Graph Partition (METIS): This benchmark solution adopts the widely used METIS algorithm [25] of balanced network partitioning. For balanced partitioning problem [26], the objective is to partition V of G into $k, (k > 1)$ subsets, such that (i) the subsets have equal size and are disjoint; (ii) the number of edges with endpoints in two subsets is minimized. METIS only sets the number of clusters, while does not set the cluster head nodes. We set the key parameters of METIS as follows. The scheme for partitioning is multilevel k-way partitioning. The scheme for computing the initial partitioning is to grow a bisection using a greedy strategy. Each partitioning subset is contiguous.

4.2. Validity test of cluster border nodes

The purpose of this experiment is to validate that the cluster border nodes selected by Alg. 1 (Section 3.2) can correctly capture all the incoming and outgoing flows of clusters. In the experiment, the head nodes are specified as the sink nodes of each cluster. Each network node sends packets to all the head nodes. We measure: (i) the total number of incoming packets received by the head node (named as I_i in cluster c_i); (ii) the total number of incoming packets received by all the border nodes of each cluster (named as O_i in cluster c_i). Then we compare these two values. If I_i equals O_i, it means the cluster border nodes capture all the incoming flow of a cluster. So that, cluster border nodes selected by Alg. 1 (Section 3.2) can correctly capture all the flows of clusters. The diagram of the experimental design is shown in Fig. 11.

We implement the experiment in Matlab. The deployment area is 100 m×100 m, and the nodes are randomly deployed. The number of nodes in the experiments is set to 60, 80, 100, 120, and 140, respectively. The transmission range of each node is identical within a single experiment. For different experiments, we reset the transmission range, which always has an average of 6 nodes within the transmission range. We assume a perfect wireless channel without packet loss. We randomly select cluster head nodes in the network. The number of these head nodes equals to the number of required clusters. These head nodes are at least 5 hops away from each other. The network is partitioned into 6 and 9 clusters separately using Alg. 1. The transmission speed of each node is randomly set in the initialization and constant during the testing. The routes from each node to the head nodes are built via the shortest path routing. For every set of testing parameters, including the number of nodes and clusters, we make 50 rounds of testing.

The experimental results illustrate that I_i and O_i are equal in every cluster for each round of the test. This experiment demonstrates that the cluster border nodes selected by Alg. 1 can capture all the incoming and outgoing flows of clusters, which can be used to correctly calculate the flows among clusters.



(a) Cluster-level flow without SDN control.

(b) Cluster-level flow with SDN control.

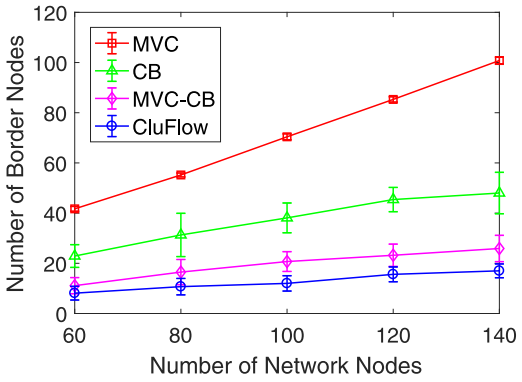
(c) Real-time cluster-level flow from c_2 to c_1, and from c_3 to c_1.

Fig. 12. Case study of cluster-based flow control by CluFlow. The SDN controller balances the cluster-level flows (from c_2 to c_1 and from c_3 to c_1) by re-configuring the cluster-level routes (from c_3 to c_2 and from c_4 to c_2).

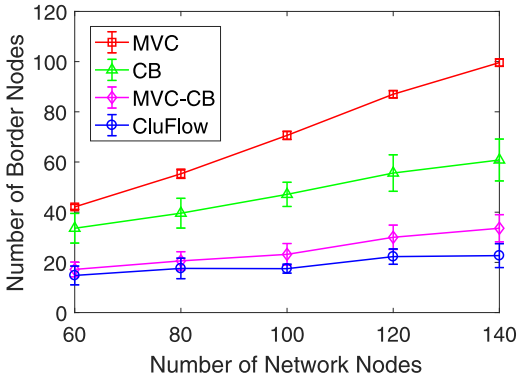
4.3. Practicality test of cluster-based flow control

We show the practicality of the protocol shown in Fig. 8 (Section 3.3) by controlling the cluster-level traffic flow in a case study. We implemented the experiment in Matlab. The deployment area is 100 m×100 m, and the nodes are randomly deployed. The network consists of 200 nodes and is partitioned into 4 clusters. There are 6 nodes on average within the transmission range of each node. We assume a perfect wireless channel without packet loss. The head node of c_1 is set as the sink node. Every node of the network sends packets to the sink via the shortest path routing. The cluster-level topology and flow without CluFlow control are shown in Fig. 12(a). The time interval between the present and the next sending time of every node is uniformly distributed in [1, 8] seconds. The nodes in c_1 and c_3 send packets of 10 bytes in the whole experiment. The nodes in c_2 and c_4 send packets of 10 bytes before 400 s, and packets of 50 bytes after 400 s. The SDN controller sets cluster-level routing rules to block the flows between c_2 and c_3, c_2 and c_4 after 600 s, as shown in Fig. 12(b).

The real-time traffic flows from c_2 to c_1 and from c_3 to c_1 are shown in Fig. 12(c). In the experimental results, the flows from c_2 to c_1 and from c_3 to c_1 are quite unbalanced between 400 s to 600 s. The main reason is that the traffic generated by the nodes inside c_2 and c_4 increases significantly after 400 s and they all pass through c_2. After 600 s, the flows from c_2 to c_1 and from c_3 to c_1 are better balanced. The main reason is that the controller resets the cluster-level routing rules, in which the traffic generated by the nodes inside c_4 are prohibited to pass through c_2. So, the traffic generated by the nodes inside c_4 must pass through c_3. Compared with using only local distributed routing, cluster-level SDN control makes the flow from c_2 to c_1 and flow from c_3 to c_1 more balanced. This case study shows the practicality of cluster-based flow control.

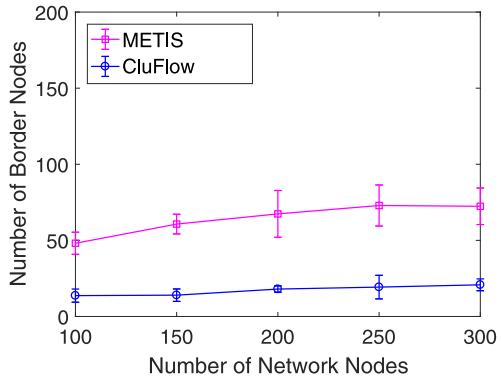


(a) 6 clusters.

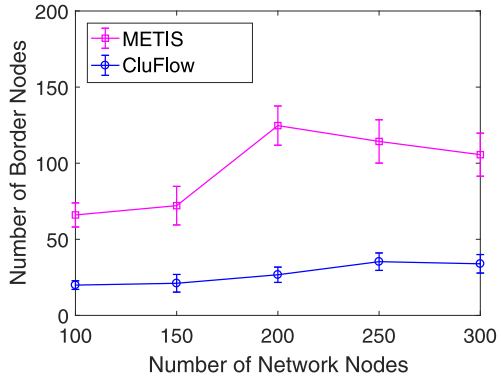


(b) 9 clusters.

Fig. 13. The number of border nodes using CluFlow and the benchmark approaches MVC, CB, and MVC-CB with 6 clusters and 9 clusters.



(a) 6 clusters.



(b) 9 clusters.

Fig. 14. The number of border nodes using CluFlow and the balanced cluster partitioning approach METIS with 6 clusters and 9 clusters.

4.4. Number of border nodes in unbalanced clustering

We compare the number of cluster border nodes created by CluFlow to the unbalanced clustering solutions MVC, CB, and MVC-CB. A smaller number of cluster border nodes means fewer communication costs between nodes and the SDN controller.

In the experiment, the number of nodes in the network is set to 60, 80, 100, 120, and 140, respectively. The network is partitioned into 6 and 9 clusters, respectively. The other settings of the network are the same as in Section 4.3. For each set of parameters, we make 10 rounds of testing. The experimental results are illustrated in Fig. 13. The results 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, because the state space for partitioning clusters is larger in larger networks. In the testing with 140 nodes and 6 heads, CluFlow has 83%, 65%, 34% fewer border nodes than MVC, CB and MVC-CB, respectively.

Compared with MVC and CB, the number of border nodes selected by MVC-CB is smaller. The main reason is that 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. But MVC-CB only uses Voronoi cluster partition. So CluFlow, using cluster partition Alg. 1, has fewer cluster border nodes than MVC-CB. Meanwhile, as the number of clusters increases from 6 to 9, the number of cluster border nodes increases in both CluFlow and benchmark solutions. This means the cost for flow control of cluster border nodes increases as the number of clusters becomes larger.

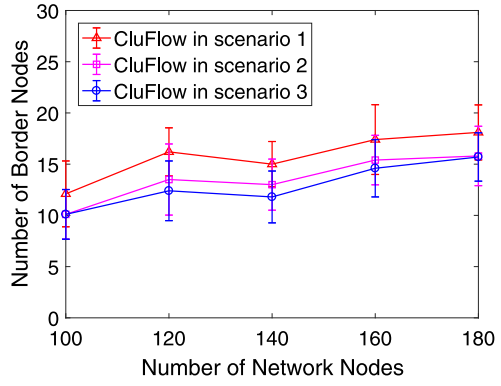
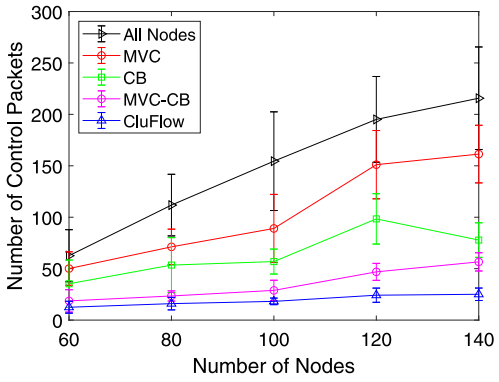


Fig. 15. The number of cluster border nodes using CluFlow with different sizes of Θ.

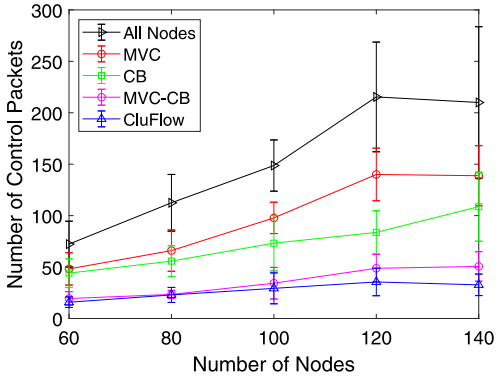
4.5. Number of border nodes in balanced clustering

We compare CluFlow with balanced clustering solution METIS. To increase the state space for clustering, we increase the number of network nodes (compared with the experiments in unbalanced clustering) to 100, 150, 200, 250, and 300. The values of other experimental parameters are the same as the experiments in unbalanced clustering.

The experimental results are shown in Fig. 14. The number of border nodes produced by METIS is much higher than CluFlow. In the testing of 300 nodes, CluFlow has 71% and 68% fewer border nodes than METIS with 6 and 9 clusters, respectively. The main reason is that METIS needs to balance the cluster size while minimizing the number of cut edges, which produces more cluster border nodes. Compared with

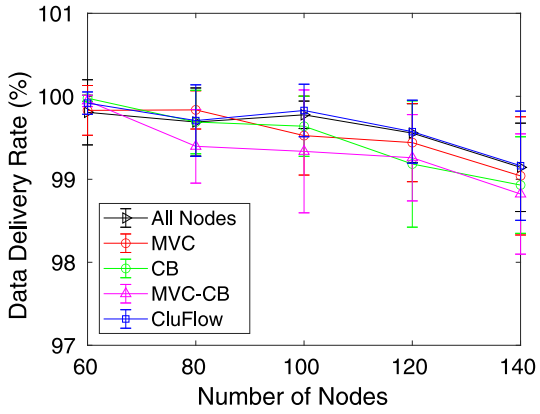


(a) 6 clusters.

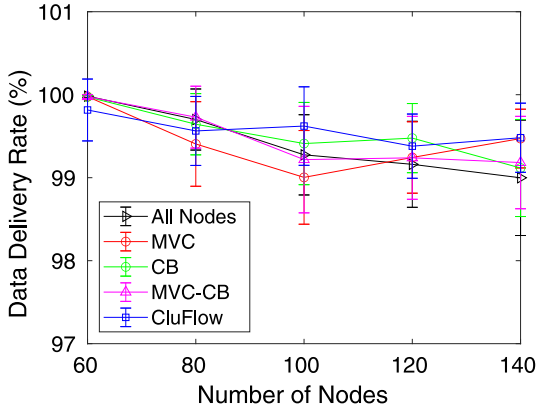


(b) 9 clusters.

Fig. 16. The number of flow configuration packets with 6 clusters and 9 clusters in an SD-WSN. "All Nodes" represents traditional SD-WSN without clustering, in which all the network nodes communicate with the SDN controller.



(a) 6 clusters.



(b) 9 clusters.

Fig. 17. Data delivery rate with 6 clusters and 9 clusters in an SD-WSN. "All Nodes" represents traditional SD-WSN without clustering, in which all the network nodes communicate with the SDN controller.

METIS, CluFlow aims to minimize the number of border nodes without requirement on balanced partitioning.

4.6. Search space of cluster border nodes

In this experiment, we observe how the search space of clustering affects the number of cluster border nodes. Θ is the search space of cluster border nodes. We set Θ as follows. Firstly, we randomly select cluster head nodes in the network. These cluster head nodes are at least 8 hops away from each other. Secondly, we make Voronoi clusters in the network based on the cluster head nodes. Name the border nodes of all the Voronoi clusters as Θ_b. Name the nodes that reside outside Θ_b and have 1 hop distance to any node in Θ_b as Θ_{b1}. Name the nodes that reside outside Θ_b and have 2 hop distance to any node in Θ_b as Θ_{b2}. Finally, we create Θ in the following three scenarios.

- Scenario 1: Θ includes Θ_b and Θ_{b1}.
- Scenario 2: Θ includes Θ_b, Θ_{b1}, and Θ_{b2}.
- Scenario 3: Θ includes all the nodes except the cluster head nodes.

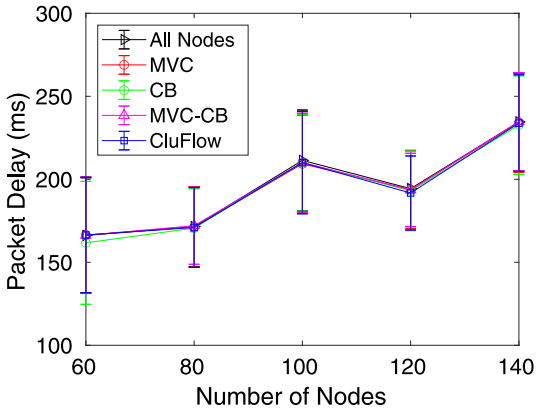
In the three scenarios, the size of Θ in scenario 1 is the smallest, and the size of Θ in scenario 3 is the largest. We set the number of nodes in different experiments to 100, 120, 140, 160, and 180, and the number of clusters to 4. The other settings are the same as in Section 4.3. For each setup, we perform 10 rounds of testing. The testing results are shown in Fig. 15. As the size of Θ increases, the number of border nodes decreases. The main reason is that larger Θ provides a bigger search space to partition clusters, so that the possibility to find fewer border nodes increases.

4.7. Communication cost

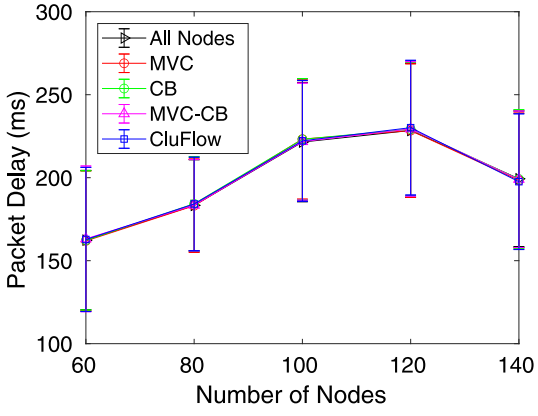
To back up our claim that a smaller number of border nodes leads to less control traffic, we perform experiments to assess the number of flow configuration messages in an SD-WSN simulated scenario. We use the Cooja simulator [27] with sky motes. We use IT-SDN [28] as the southbound protocol, since it is tailored to WSNs. The version of IT-SDN is 0.4.1, which is configured to use source-routed control packets. A simple custom neighbor discovery protocol is employed, which gathers neighborhood information at the beginning of the simulation by periodic beacons. We set the number of nodes in different experiments to 60, 80, 100, 120, and 140, and the number of clusters is 6 and 9. The other settings are the same as in Section 4.3.

We select a sink node in the network and the controller is located at the sink node. The SDN controller interacts only with the cluster border nodes, while the other nodes route packets according to a distributed routing algorithm. Since our goal is to study the behavior of SDN control messages in the face of different clustering algorithms, the nodes are configured with static routing tables instead of dynamic distributed local routing. Every node in the network transmits one 10-byte data packet to the data sink per minute. Fig. 16 displays the average number of flow configuration messages for each clustering solution and for a traditional SD-WSN without clustering.

Our approach CluFlow yields the least amount of control messages. In comparison to not using clustering, the reduction ranges from 78% to 88%. MVC-CB is the closest to CluFlow, however it produces on average 75% and 27% more control messages, for 6 and 9 clusters, respectively. We observe that the number of control messages increases



(a) 6 clusters.



(b) 9 clusters.

Fig. 18. Packet delay with 6 clusters and 9 clusters in an SD-WSN. "All Nodes" represents traditional SD-WSN without clustering, in which all the network nodes communicate with the SDN controller.

as the number of clusters becomes larger. This is mainly because the amount of border nodes tends to increase with the number of clusters.

The goal of cluster-level routing is to reduce the control overhead. While the experiments above show CluFlow mitigates the control cost of SD-WSN in comparison to other clustering algorithms, it is crucial to investigate whether this gain comes at the expense of degrading other metrics or not. To this end, we measure two important communication metrics, which are the data delivery rate as shown in Fig. 17 and packet delay as shown in Fig. 18.

The experimental results show that all the tested clustering approaches have achieved over 98.5% delivery rate, and there is no statistical difference among them. In addition, CluFlow presents the highest delivery rate in half of the scenarios. Regarding the packet delay, the time difference between the highest value and the lowest value in each testing point is always less than 5 ms. This means none of the solutions presents significantly lower or higher delay than the others. The small variations observed are likely to arise from the underlying simulation randomness.

Based on the above experimental results, we conclude that, in comparison to the benchmark solutions, CluFlow significantly mitigates the communication cost of SD-WSN, while it does not degrade the other important communication metrics.

4.8. Performance in a real indoor WSN

We set up a real indoor WSN in a university building to test CluFlow. We measure the number of border nodes and the communication cost.

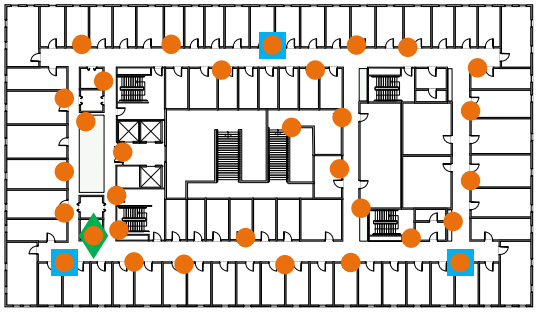
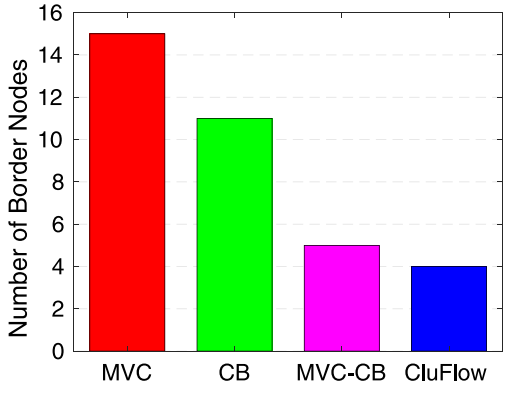
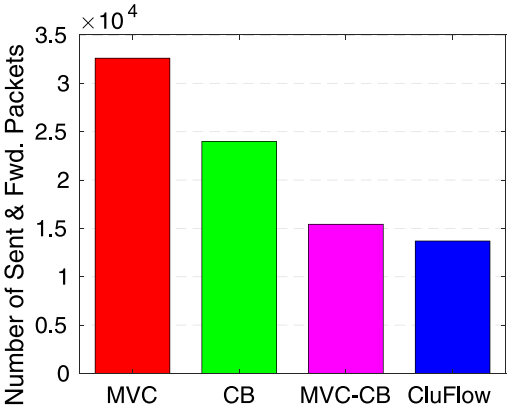


Fig. 19. The deployment of a WSN in a building. Orange circles ● represent the positions of the deployed nodes. The node with green diamond background ♦ is the SDN controller. The nodes with blue square background ■ are the heads of clusters.



(a) The number of border nodes.



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

Fig. 20. Performance of SD-WSN in a Real Indoor WSN.

The deployed nodes are CC2650STK SensorTag motes [29], using Contiki 3.0 OS [30], IEEE 802.15.4 MAC standard [31]. We use CSMA/CA collision avoidance, Contiki-Mac radio duty cycle, and RPL [32] routing protocol. The Tx power of each node is set to 0dBm, and Rx sensitivity is -100dBm. 32 nodes are deployed in an area of 65m×38m as shown in Fig. 19. The sink node is attached to a SensorTag Debugger DevPack, which links to a computer by a USB cable. The SDN controller runs on a 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 s. 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 s. 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 s 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. 20. Compared with the benchmark approaches, CluFlow utilizes the smallest number of border nodes and communication costs. Compared with the results in Section 4.4 and Section 4.7, the improvement of CluFlow to MVC, CB and MVC-CB is smaller. The main reason is that the total number of nodes is smaller in the real network deployment. Therefore the difference in clustering using different solutions is smaller.

5. Related work

Most existing WSN structures utilize distributed control solutions. They face the same difficulties as traditional wired networks, such as lack of a high-level abstraction and ossified protocol stack. Dynamically changing control policy in a WSN becomes increasingly difficult as the size of the WSN increases [8]. For example, as the communication flow pattern or environment changes, if a WSN needs to achieve better performance, the control plane of each sensor node must be (re)programmed. In a large-scale WSN, this task is difficult to handle. Therefore, a WSN needs a high-level centralized SDN control.

There are already various types of research on SDN in wired networks. These solutions provide improved flexibility and reduced complexity for flow control. For example, in [33], a hybrid mechanism is presented to control distributed routing by centralized management. The SDN controller injects routing guidance, e.g. fake nodes, to networks. In [34], the wired SDN is partitioned into clusters. Only border switches are connected and controlled by the SDN controller. SDN switches forward all the received messages to the SDN controller. After receiving messages, the SDN controller changes the routing information and sends it back to SDN switches.

Hybrid SDN is a networking paradigm where both centralized SDN control and distributed networking paradigms coexist [1,15–17]. Although using SDN technologies on top of legacy networking devices poses several challenges [35], hybrid SDN is gradually adopted by industry and academia. For example, the research in [36] presents a service-based hybrid SDN model in a wireless mesh backhaul for the coexistence of network services SDN controller and distributed network services. [37] proposes an incremental deployment strategy and a throughput-maximization routing for deploying a hybrid SDN. [38] addresses the efficient deployment problem of hybrid SDN devices through the maximum coverage problem.

In the research of hybrid SDN, some works focus on managing SDN control by network clustering. In [39], an SDN enabled 5G vehicular ad-hoc network is proposed. Due to the mobility features of vehicles, the solution utilizes vehicle clustering for reducing the overhead of cellular networks and providing better communication quality. The aim of this work is to manage the network of mobile vehicles, but the control for SD-WSN is not researched. [40] proposes an active network management QoS scheme for managing data flow in SD-WSN. In its implementation, a WSN is partitioned into clusters, and multiple base stations are used as cluster heads. However, this solution uses SDN controllers to manage all the cluster member nodes of clusters. Although this solution is easy to be implemented, the communication cost of managing WSN nodes is much higher than CluFlow. The research in [41] proposes an SDN-based clustering mechanism, which considers power, trust, secure centrality, mobility, priority, and heterogeneity in Internet of Things. Compared with CluFlow, this paper aims to provide secure clustering by adaptive cluster head selection instead of decreasing the communication cost on SDN control. Meanwhile, the paper assumes that the cluster heads are the main communication bridges, and the SDN controller does not manage the routing of multi-hop communication as in CluFlow. The research in [42] provides a two-level hybrid SDN control mechanism for Internet of Things. In the mechanism, a routing protocol based on multi-hop clustering is used on the first level, and an SDN for managing the global network is leveraged on the second level. Compared to the SD-WSN structure of CluFlow, this paper assumes a hierarchical network structure, in which the communication among clusters is through SDN switches. In addition, the aim of this paper is to meet QoS requirements, while CluFlow aims to reduce the communication cost of SDN control.

SD-WSN is one of the most important research directions in hybrid SDN. TinySDN is an early effort in developing an SD-WSN [43]. Its experiments use a 7-node network, focusing on the delay metric. CORAL-SDN aims at exploring the impact of discovery algorithms on SD-WSN performance [44]. The authors assessed metrics related to topology discovery on a variety of 25-node topologies. Control overhead, however, is not measured. [45] proposes μSDN, an IPv6-compatible SD-WSN stack. Their experiments show that approximately 15% of the traffic in a 30-node network is composed of SDN control packets. [28] uses IT-SDN to perform an SD-WSN scalability study. Its experiments are based on networks with up to 289 nodes under varying networking conditions. The results indicate that control traffic grows linearly with the network size, suggesting the need for control traffic reduction mechanisms. [46] proposes a mechanism for on-line metric assessment on SD-WSN systems. However, the mechanism further increases the control overhead in larger networks. The research in [47] provides a solution to utilize OpenFlow in wireless networks. It uses the OpenFlow centralized controller for routing data traffic. SDN-WISE [48] 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 [49] creates an SDN framework for IoT systems based on SDN-WISE and Open Network Operating System (ONOS) [50]. To connect IoT and SDN, it extends the functionality of ONOS as the controller in a WSN, while the communication protocol relies on SDN-WISE. Generally, all the above wireless SDN structures do not completely decouple the control plane and data plane. 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 periodically exchange request and reply messages over multiple hops. This process increases communication delay and control overhead in wireless networks.

Besides researching SD-WSN architectures, some works focus on increasing the performance of WSNs using an SDN structure, such as energy efficiency, task scheduling, and routing. SDN-ECCKN [51] proposes an SDN-based energy management system for WSN. The system reduces the total transmission time to increase the network lifetime. [52] minimizes energy consumption on sensors with guaranteed quality-of-sensing in a multi-task SD-WSN. It utilizes a centralized SDN to formulate the minimum-energy sensor activation by jointly considering sensor activation and task mapping. The work in [53] presents an energy-efficient routing algorithm based on the SD-WSN framework. To minimize the transmission distance and the energy consumption of sensor nodes, the algorithm partitions the WSN into clusters and dynamically assigns tasks to the intra-cluster nodes by a cluster control node.

6. Limitations and future research directions

In this work, we open up a new solution for communication flow control in hybrid SD-WSNs, which leverages the benefits of network clustering and legacy distributed routing in WSN. At the same time, we are aware of some key points of improvement as follows.

- It is unclear how to take advantage of our solution to dynamic WSNs. In a dynamic WSN, the network topology changes frequently. Therefore, if CluFlow is directly used in a dynamic WSN, the SDN controller has to calculate new clusters frequently, which will increase the communication cost.
- It is important that cluster-level routing rules and distributed routing rules coordinate correctly. For example, cluster-level routing rules and distributed routing rules must avoid livelock routing [54]. The detailed design of cluster-level routing rules, and the coordination of cluster-level routing and distributed legacy routing will be part of our future work.
- In this paper, we propose an effective network clustering solution. The clustering algorithm is generic to any kind of networks, and not specific to WSNs. Meanwhile, clustering is a widely used method in communication networks. So it is valuable to evaluate whether our solution is effective in the other wired and wireless networks.

7. Conclusion

In this work, we have presented CluFlow, a cluster-based hybrid SD-WSN architecture. The key idea is to manage communication flows using central SDN control on the cluster border nodes. Consequently, the control overhead required for programming the network can be significantly reduced. To this end, we have provided a clustering algorithm tailored for minimizing the number of cluster border nodes. In this way, CluFlow can effectively take advantage of distributed control at node-level and centralized control at cluster-level. Based on simulations and testbed experiments, we have demonstrated that CluFlow can significantly decrease the number of border nodes and the communication load for controlling/monitoring cluster-level communication compared to benchmark solutions.

CRediT authorship contribution statement

Qingzhi Liu: Conceived and designed the analysis, Collected the data, Contributed data or analysis tools, Performed the analysis, Wrote the paper. **Long Cheng:** Conceived and designed the analysis, Contributed data or analysis tools, Performed the analysis, Wrote the paper. **Renan Alves:** Conceived and designed the analysis, Collected the data, Contributed data or analysis tools, Performed the analysis, Wrote the paper. **Tanir Ozcelebi:** Conceived and designed the analysis, Performed the analysis, Wrote the paper. **Fernando Kuipers:** Conceived and designed the analysis, Performed the analysis, Wrote the paper. **Guixian Xu:** Conceived and designed the analysis, Performed the analysis, Wrote the paper. **Johan Lukkien:** Conceived and designed the analysis, Performed the analysis, Wrote the paper. **Shanzhi Chen:** Conceived and designed the analysis, Wrote the paper.

Acknowledgments

Renan C. A. Alves was supported by grants #2016/21088-1 and #2018/11295-5, São Paulo Research Foundation (FAPESP), Brazil.

Declaration of competing interest

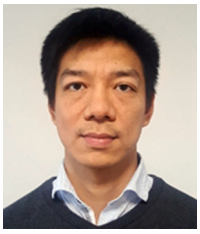
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Q. Liu, T. Ozcelebi, L. Cheng, F. Kuipers, J. Lukkien, Cluflow: Cluster-based flow management in software-defined wireless sensor networks, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2019, pp. 1–8.
[2] H. Kim, N. Feamster, Improving network management with software defined networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.
[3] B.A.A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634.
[4] K.M. Modieginyane, B.B. Letswamotse, R. Malekian, A.M. Abu-Mahfouz, Software defined wireless sensor networks application opportunities for efficient network management: A survey, Comput. Electr. Eng. 66 (2018) 274–287.
[5] I.T. Haque, N. Abu-Ghazaleh, Wireless software defined networking: A survey and taxonomy, IEEE Commun. Surv. Tutor. 18 (4) (2016) 2713–2737.
[6] K. Sood, S. Yu, Y. Xiang, Software-defined wireless networking opportunities and challenges for internet-of-things: A review, IEEE Internet Things J. 3 (4) (2016) 453–463.
[7] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: From concept to implementation, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2181–2206.
[8] T. Luo, H.-P. Tan, T.Q. Quek, Sensor OpenFlow: Enabling software-defined wireless sensor networks, IEEE Commun. Lett. 16 (11) (2012) 1896–1899.
[9] H. Mostafaei, M. Menth, Software-defined wireless sensor networks: A survey, J. Netw. Comput. Appl. 119 (2018) 42–56.
[10] H.I. Kobo, A.M. Abu-Mahfouz, G.P. Hancke, A survey on software-defined wireless sensor networks: Challenges and design requirements, IEEE Access 5 (2017) 1872–1899.
[11] D. Sajjadi, Z. Zheng, R. Ruby, J. Pan, Randomized single-path flow routing on SDN-aware Wi-Fi mesh networks, in: International Conference on Mobile Ad Hoc and Sensor Systems (MASS), IEEE, 2018, pp. 184–192.
[12] M. Rezaee, M.H.Y. Moghaddam, SDN-based quality of service networking for wide area measurement system, IEEE Trans. Ind. Inf. (2019).
[13] A.A. Abdelltif, E. Ahmed, A.T. Fong, A. Gani, M. Imran, SDN-based load balancing service for cloud servers, IEEE Commun. Mag. 56 (8) (2018) 106–111.
[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM 38 (2) (2008) 69–74.
[15] Y. Sinha, K. Haribabu, et al., A survey: Hybrid sdn, J. Netw. Comput. Appl. 100 (2017) 35–55.
[16] R. Amin, M. Reisslein, N. Shah, Hybrid SDN networks: A survey of existing approaches, IEEE Commun. Surv. Tutor. 20 (4) (2018) 3259–3306.
[17] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, S. Hu, A survey of deployment solutions and optimization strategies for hybrid SDN networks, IEEE Commun. Surv. Tutor. 21 (2) (2018) 1483–1507.
[18] B.B. Letswamotse, R. Malekian, C.-Y. Chen, K.M. Modieginyane, Software defined wireless sensor networks (SDWSN): a review on efficient resources, applications and technologies, J. Internet Technol. 19 (5) (2018) 1303–1313.
[19] M.R. Garey, D.S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1990.
[20] P. Sanders, C. Schulz, D. Strash, R. Williger, Distributed evolutionary k-way node separators, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2017, pp. 345–352.
[21] Y. Boykov, V. Kolmogorov, An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision, IEEE Trans. Pattern Anal. Mach. Intell. 26 (9) (2004) 1124–1137.
[22] G. Karakostas, A better approximation ratio for the vertex cover problem, in: International Colloquium on Automata, Languages, and Programming, Springer, 2005, pp. 1043–1050.
[23] W. Zhou, L. Li, M. Luo, W. Chou, REST API Design patterns for SDN northbound API, in: 2014 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE, 2014, pp. 358–365.
[24] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405.
[25] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[26] K. Andreev, H. Racke, Balanced graph partitioning, Theory Comput. Syst. 39 (6) (2006) 929–939.
[27] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-level sensor network simulation with cooja, in: IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp), 2006.
[28] R.C. Alves, D.A. Oliveira, G.A.N. Segura, C.B. Margi, The cost of software-defining things: A scalability study of software-defined sensor networks, IEEE Access 7 (2019) 115093–115108.

[29] Texas Instruments, Multi-Standard CC2650 SensorTag Design Guide, 2019, URL: https://www.ti.com/lit/ug/tidu862/tidu862.pdf.
[30] Contiki OS, 2018, URL: http://www.contiki-os.org/.
[31] E. Callaway, P. Gorday, L. Hester, J.A. Gutierrez, M. Naeve, B. Heile, V. Bahl, Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks, IEEE Commun. Mag. 40 (8) (2002) 70–77.
[32] T. Winter, P. Thubert, A. Brandt, T. Clausen, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.P. Vasseur, RPL: IPv6 Routing Protocol for Low Power and Lossy Networks, ROLL Working Group, 2011.
[33] S. Vissicchio, O. Tilmans, L. Vanbever, J. Rexford, Central control over distributed routing, ACM SIGCOMM 45 (4) (2015) 43–56.
[34] M. Caria, A. Jukan, M. Hoffmann, SDN partitioning: A centralized control plane for distributed routing protocols, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 381–393.
[35] S. Vissicchio, L. Vanbever, O. Bonaventure, Opportunities and research challenges of hybrid software defined networks, ACM SIGCOMM Comput. Commun. Rev. 44 (2) (2014) 70–75.
[36] J. Nunez-Martinez, J. Baranda, J. Mangues-Bafalluy, A service-based model for the hybrid software defined wireless mesh backhaul of small cells, in: 2015 11th International Conference on Network and Service Management (CNSM), IEEE, 2015, pp. 390–393.
[37] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, H. Wang, Incremental deployment and throughput maximization routing for a hybrid SDN, IEEE/ACM Trans. Netw. 25 (3) (2017) 1861–1875.
[38] B. Kar, E.H.-K. Wu, Y.-D. Lin, The budgeted maximum coverage problem in partially deployed software defined networks, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 394–406.
[39] X. Duan, Y. Liu, X. Wang, SDN enabled 5G-VANET: Adaptive vehicle clustering and beamformed transmission for aggregated traffic, IEEE Commun. Mag. 55 (7) (2017) 120–127.
[40] B.B. Letswamotse, R. Malekian, C.-Y. Chen, K.M. Modieginyane, Software defined wireless sensor networks and efficient congestion control, IET Netw. 7 (6) (2018) 460–464.
[41] K. Kalkan, SUTSEC: SDN utilized trust based secure clustering in IoT, Comput. Netw. (2020) 107328.
[42] A. Ouhab, T. Abreu, H. Slimani, A. Mellouk, Energy-efficient clustering and routing algorithm for large-scale SDN-based IoT monitoring, in: ICC 2020-2020 IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.
[43] B.T. De Oliveira, L.B. Gabriel, C.B. Margi, TinySDN: Enabling multiple controllers for software-defined wireless sensor networks, IEEE Lat. Am. Trans. 13 (11) (2015) 3690–3696.
[44] T. Theodorou, L. Mamatas, Software defined topology control strategies for the internet of things, in: IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2017, pp. 236–241.
[45] M. Baddeley, R. Nejabati, G. Oikonomou, M. Sooriyabandara, D. Simeonidou, Evolving SDN for low-power IoT networks, in: IEEE Conference on Network Softwarization and Workshops (NetSoft), IEEE, 2018, pp. 71–79.
[46] T.C. Luz, G.A. Nunez, C.B. Margi, F.L. Verdi, In-network performance measurements for software defined wireless sensor networks, in: International Conference on Networking, Sensing and Control (ICNSC), IEEE, 2019, pp. 206–211.
[47] A. Detti, C. Pisa, S. Salsano, N. Blefari-Melazzi, Wireless mesh software defined networks (wmSDN), in: WiMob, IEEE, 2013, pp. 89–95.
[48] L. Galluccio, S. Milardo, G. Morabito, S. Palazzo, SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks, in: INFOCOM, IEEE, 2015, pp. 513–521.
[49] A.-C.G. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, S. Palazzo, Towards a software-defined Network Operating System for the IoT, in: World Forum on Internet of Things (WF-IoT), IEEE, 2015, pp. 579–584.
[50] 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.
[51] Y. Wang, H. Chen, X. Wu, L. Shu, An energy-efficient SDN based sleep scheduling algorithm for WSNs, J. Netw. Comput. Appl. 59 (2016) 39–45.
[52] D. Zeng, P. Li, S. Guo, T. Miyazaki, J. Hu, Y. Xiang, Energy minimization in multi-task software-defined sensor networks, IEEE Trans. Comput. 64 (11) (2015) 3128–3139.
[53] W. Xiang, N. Wang, Y. Zhou, An energy-efficient routing algorithm for software-defined wireless sensor networks, IEEE Sens. J. 16 (20) (2016) 7393–7400.
[54] L. Gravano, G.D. Pifarre, P.E. Berman, J.L.C. Sanz, Adaptive deadlock-and livelock-free routing with all minimal paths in torus networks, IEEE Trans. Parallel Distrib. Syst. 5 (12) (1994) 1233–1251.

Qingzhi Liu is a Lecturer at the Information Technology Group, Wageningen University & Research, The Netherlands. He received the B.S. degree in Telecommunication and the M.Eng. degree in Software Engineering from Xidian University, China in 2005 and 2008 respectively. He received the M.Sc. (with cum laude) and the Ph.D. from Delft University of Technology, The Netherlands in 2011 and 2016 respectively. He was a Postdoctoral Researcher at Eindhoven University of Technology, The Netherlands from 2016 to 2019. His research interests include Internet of Things and Artificial Intelligence.



Long Cheng is a Professor in the School of Control and Computer Engineering at North China Electric Power University in Beijing. He received the B.E. from Harbin Institute of Technology, China in 2007, M.Sc from University of Duisburg-Essen, Germany in 2010 and Ph.D from National University of Ireland Maynooth in 2014. He was an Assistant Professor at Dublin City University, and a Marie Curie Fellow at University College Dublin. He also has worked at organizations such as Huawei Technologies Germany, IBM Research Dublin, TU Dresden and TU Eindhoven. His research focuses on high-performance data analytics, distributed systems, cloud computing, and process mining.

Renan C. A. Alves is a postdoctoral fellow at Universidade de São Paulo, Brazil. He obtained his PhD degree (2020), M.Sc degree (2014) and graduated in Electrical Engineering with emphasis in Computer and Digital Systems (2011), all at Universidade de São Paulo. His main research interests include network protocol modeling and performance analysis, Wireless SensorNetworks protocols and applications.

Tanir Ozcelebi received the Ph.D. degree in Electrical Engineering from Koc University, Istanbul in 2006. Since 2006, he joined the Interconnected Resource-aware Intelligent Systems (IRIS) group at Eindhoven University of Technology (TU/e), The Netherlands. He is currently an associate professor at IRIS of TU/e. His main research interests are architectures and life-cycle management of resource constrained for smart spaces and intelligent Internet of Things (IoT), including the relevant data analytics aspects.

Fernando A. Kuipers is an associate professor and head of the Lab on Internet Science at Delft University of Technology (TU Delft). In 2004, he obtained his Ph.D. degree cum laude, the highest possible distinction at TU Delft. His research focus is on network optimization, network resilience, Quality of Service, and Quality of Experience and addresses problems in Software-Dened Networking, Tactile Internet, Internet-of-Things, and critical infrastructures. His work on these subjects include distinguished papers at IEEE INFOCOM 2003, Chinacom 2006, IFIP Networking 2008, IEEE FMN 2008, IEEE ISM 2008, ITC 2009, IEEE JISIC 2014, NetGames 2015, and EuroGP 2017. Fernando Kuipers is senior member of the IEEE, was a visiting scholar at Technion - Israel Institute of Technology (in 2009) and Columbia University in the City of New York (in 2016), and is Vice-Chair of the IFIP Working Group 6.2 on Network and Internetwork Architectures. He co-founded Do IoT and PowerWeb and is part of the board of the TU Delft Safety & Security institute.

Guixian Xu received his Ph.D. degree in communications and information systems from Beijing University of Posts and Telecommunications (BUPT), China, in 2017. He was a visiting Ph.D. student with the National Tsing Hua University, Hsinchu, Taiwan, from 2015 to 2016. He was a postdoc with Aalborg University and Aarhus University, Denmark, in 2018 and 2019, respectively. Since 2020, he is a postdoc research fellow with Tampere University, Finland. His research interests are in signal processing in 5G and beyond, machine learning, and convex optimization.

Johan Lukkien is a Full Professor in Interconnected Resource-aware Intelligent Systems (IRIS) group at the Department of Mathematics and Computer Science, Eindhoven University of Technology (TU/e). Since 2002 he has been the Chair of the Interconnected Resource-aware Intelligent Systems (IRIS) group. Contributions of the IRIS group are in the area of component-based middleware for resource constrained devices, distributed coordination, Quality of Service in networked systems and schedulability analysis in real-time systems. The key areas of his expertise include embedded software, algorithms, resource constrained systems, real-time systems and system architecture.

Shanzhi Chen received his Ph.D. degree from Beijing University of Posts and Telecommunications, China, in 1997. He joined the Datang Telecom Technology and Industry Group and the China Academy of Telecommunication Technology (CATT) in 1994, and has been serving as the EVP of Research and Development since 2008. He is currently the Director of the State Key Laboratory of Wireless Mobile Communications, CATT, where he conducted research and standardization on 4G TD-LTE and 5G. He is an IEEE fellow, have served as the area editor of IEEE Internet of Things, an editor for IEEE Network, and a guest editor of IEEE Wireless Communications, the IEEE Communications Magazine, and IEEE Transactions on Vehicular Technology, as well as a member of TPC Chairs of many international conferences. His achievements have received multiple top awards and honors by the China central government, especially the Grand Prize of the National Award for Scientific and Technological Progress, China, in 2016 (the highest Prize in China). His current research interests include 5G mobile communications, network architectures, vehicular communication networks, and Internet of Things.