Computer Networks 96 (2016) 48–57

Contents lists available at ScienceDirect

Computer Networks

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

Efficient switch clustering for distributed controllers of OpenFlow network with bi-connectivity

Junichi Nagano, Norihiko Shinomiya*

Graduate School of Engineering, Soka University, Tokyo, Japan

ARTICLE INFO

Article history:
Received 6 September 2014
Revised 16 October 2015
Accepted 20 October 2015
Available online 6 November 2015

Keywords:
OpenFlow
Distributed control plane
Clustering
Bi-connected components

ABSTRACT

Control planes of Software Defined Networking have faced a scalability issue because they have been widely applied for large scale networks. To address this issue, control planes with multi-controllers such as Onix and HyperFlow have been proposed in which controllers share global network view. Since the global network view includes topology, states and events of a network, shared information could become larger when governed network is large and complicated. Thus, Onix have proposed reduction method of the information by abstracting switches as a single virtual switch. However, there is a lack of discussion about reliability of virtual switches that depends on topology of abstracted networks. To improve reliability of the virtual switches, there are two viewpoints: (1) administrative areas of controllers, and (2) limitation of abstraction areas. Hence, this paper proposes construction method of more dependable virtual switches focusing on bi-connectivity and reduction of shared information using the virtual switches. Our experimental results show that our reduction method can improve reliability of virtual switches and curb the number of edges in a federation graph that is regarded as shared information.

1. Introduction

Our lives have become increasingly dependent on the Internet technologies, which has made the volume of data traffic growing significantly. Then, even a short interval of failure or performance degradation might cause enormous influences. Network management focusing on high-speed failure recovery and load balancing of flows has been getting more attention. These management techniques require a redundant topology with at least bi-connectivity which contains two or more disjoint paths in order to improve reliability of networks by establishing a spare path for a failed or congested path [1,2].

OpenFlow has been widely used as a standard protocol to actualize Software Defined Networking (SDN) and facilitates to create testbed environments of realistic network for new control mechanisms [3]. In an OpenFlow network, a controller is in charge of creating data forwarding rules while a switch node is responsible for forwarding data. In this way, an OpenFlow network emploies a centralized architecture where the controller regulates the data forwarding of switches in an integrated fashion. As a result, because there would not be necessary to configure network devices individually, it could be expected to simplify and reduce the burdens related to the network operation and management by programmable networking.

However, an OpenFlow network dominated by a single controller has been expected to raise scalability problems [4–6]. Although to tackle the scalability problems, HyperFlow [7] and Onix [8] have been proposed as control platforms of multiple controllers, the multi-controllers could have two issues: additional control delay and increase of the amount of shared information.

* Corresponding author. Tel./fax: +81 426 919419.
E-mail addresses: jnagano@soka-u.jp, j1nagano0925@gmail.com (J. Nagano), shinomi@soka.ac.jp, shi-nomi@soka.ac.jp (N. Shinomiya).

http://dx.doi.org/10.1016/j.comnet.2015.10.017
1389-1286/

The additional delay would be mainly caused by inter-controller messaging when the controllers establish flows or control decisions. For reducing the delay, all controllers should share information to deal with any arriving event by a distributed storage system [7]. This might cause a surge of shared information among controllers, because the information contains not only topology information, but also events such as flow arrivals, link failures and network-wide statistics [9].

To alleviate a burden on controllers from the shared information, Onix [8] indicates the abstraction method in which a controller can regard switches in its administrative area as a single virtual switch. This method can enclose topology information, but it might cause a malfunction of a virtual switch for particular topologies that do not include enough information for establishing a spare path for a failure. The malfunction of a virtual node should be curbed because it has negative effect for the other network elements.

There are two viewpoints for creating more reliable virtual switches: (1) administrative areas of controllers, and (2) limitation of abstraction areas. From the viewpoint (1), reliable virtual switches are constructed by improving reliability of administrative areas. However, it could be difficult to obtain administrative areas whose networks are wholly reliable. From the viewpoint (2), in order to obtain more reliable virtual switches, it is useful to limit abstraction areas with highly connected networks such as bi-connected networks, but this limitation could lead to increase of amount of shared information. Hence, in case (2), it is desirable to decrease amount of shared information.

Therefore, this paper proposes a construction method of administrative areas, which is defined as switch clustering, for reliable virtual switches and reduction method of shared information in terms of two viewpoints.

Section 2 discusses some issues and related works on distributed OpenFlow controllers, and Section 3 defines reliability of virtual switches and the amount of topology and event information. Then, Section 4 describes a clustering algorithm that reduces the amount of information by using cycles. Some numerical experiments are demonstrated in Section 5, and then Section 6 is concluding remarks.

2. Issues and related works on distributed OpenFlow controllers

This section discusses some possible issues and relevant research works in deploying multiple OpenFlow controllers in a distributed manner from the perspective of failure recovery and load balancing.

Tootoonchian et al. [7] represented that some switches might encounter longer latency for setting up the flow entries when a network has a larger diameter. Thus, they proposed a method gathering requisite information to a local controller in order to alleviate the latency by a distributed storage system. HyperFlow enables controllers to share state information and events of a network by a distributed storage system called WheelFS. The WheelFS employs publish-subscriber patterns and contains all information of the network so that a controller can access sufficient information for controlling switches quickly. However, the larger amount of shared information especially in a wide-ranging and complex network could enlarge workloads of controllers for maintaining the distributed storage system [10,11].

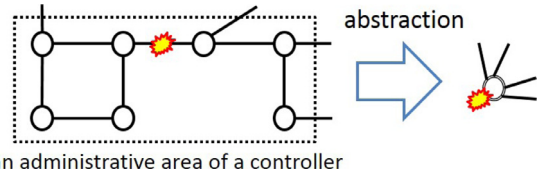


Fig. 1. An example of a virtual node in an administrative area of a controller.

In order to reduce such a large amount of shared topology information, Onix offers an abstraction function that creates a virtual node from a set of nodes in an administrative area of a controller. This function can enclose the whole topology information in a virtual switch, but there is no discussion about reliability of virtual switches created by this method. As described above, virtual switches could be fragile when an administrative area of a controller is unreliable. For example, a single link failure could cause a malfunction of the virtual node as shown in Fig. 1.

On the other hand, Kandoo has proposed an event aggregation method in order to diminish the burden of distributed database, which is mainly caused by frequent events such as flow setup requests and network statistics collections [9]. Although the local controllers lessen the frequency of notifying events to the root controller by aggregating assorted events, the issue on clustering of a controller remains to be dealt with.

The paper [12] tries to diminish the amount of shared information by aggregating all switches in an administrative area of a controller and treating the switches as a single virtual switch. The paper compares the characteristics of two clustering methods and concludes that the minimum cut clustering can curb the amount of shared information but it has no discussion about the reliability of an aggregated switch.

Consequently, there might be no precise discussion on clustering switches from the view of the reliability of virtual switches and the amount of shared information. Hence, this paper clarifies reliability of virtual switches and global network state information, and then discusses a clustering method for minimizing failures of virtual switches and the shared information and its implementation technique with multiple controllers.

3. Definition

This paper describes topology information as graph $G = (V, E)$. V denotes a set of nodes, and a node indicates a network equipment such as a router or a switch. E represents a set of links connecting to two network equipment. Determining administrative areas of controllers is identical with obtaining the clustering in graph theory [13]. The clustering $\mathscr{C} = \{V_i \mid 1 \le i \le j\}$ of G is a partition of the node set V into non-empty subsets called cluster V_i.

3.1. Reliability of virtual nodes

From the viewpoint (1), to decrease occurrence of failures of virtual nodes, it is necessary to expand highly connected

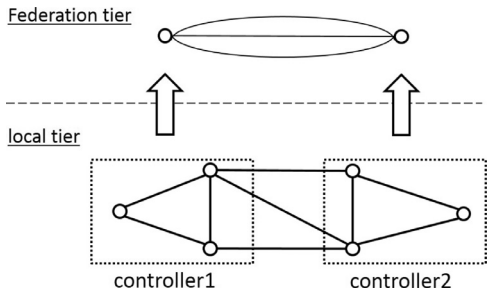


Fig. 2. An example of the federation tier and the local tier.

elements in administrative areas. This section describes *TotalAP* that is a reliability index of virtual nodes by counting outgoing ports called Affected Ports (*AP*) of failed virtual nodes focusing on a single link failure.

Let *AP* be the number of outgoing links of a virtual node which was failed by a link failure. When the failure causes no malfunction of virtual nodes, *AP* is set to zero. *TotalAP* is defined as

$$TotalAP = \sum_{e \in E} AP(\mathscr{C}, e). \quad (1)$$

3.2. Shared Information among local controllers

In an OpenFlow network with multi-controllers, each controller plays two roles: governance of a part of switches and federation of a whole network by communications with controllers. We call the interior controlling of the separated parts a local tier and federating functions a federation tier, respectively. This section discusses information of each tier for the case (2).

3.2.1. Topology information

When each controller handles many switches, it solely deals with failures or congestion that can be treated using paths governed by the controller; otherwise, it cooperates with other controllers for the treatment. In order to recognize the necessity of cooperation for protection, bi-connected components in a network becomes important because it always provides a backup path for any single link failure, and load balancing also requires bi-connectivity to find a new path for defusing a congestion. Therefore, a single controller cannot deal with a failure of a link that is not in any bi-connected components, so in the federation tier, controllers should share information of such links. In contrast, the local tier should have information of internal bi-connected components. Fig. 2 indicates an example of the federation tier and the local tier. The local tier has two controllers governing three nodes, respectively, and each controller aggregates its bi-connected component, which is triangle, to a single node of the federation tier.

When a graph G and a clustering $\mathscr{C}$ are given, the local tier contains local subgraphs $G_i = (V_i, E_i)$ of a graph G where $V_i \in \mathscr{C}$. On the other hand, the federation graph $G^f = (V^f, E^f)$ of the federation tier has all bi-connected components of local graphs as V^f and links connecting the local graphs in E. G^f is bi-connected when a graph G is bi-connected, and a link in E is included by the federation graph or the local graphs.

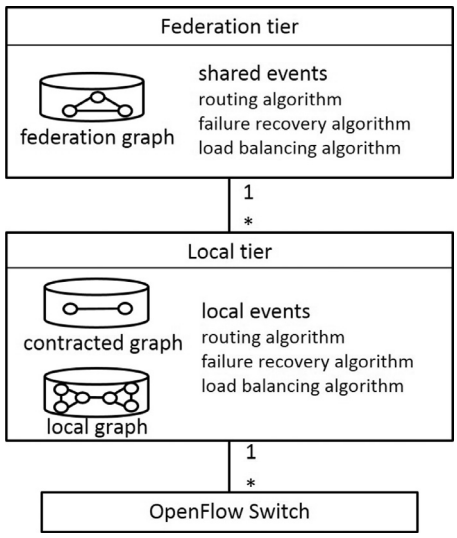


Fig. 3. Information of a federation tier and a local tier.

Hence, a failure recovery or load balancing method using G^f or local graphs can deal with all link failures in each graph, so all links in G can be recoverable.

As described above, a federation tier and a local tier have a federation graph and a local graph respectively as topology information. Fig. 3 shows the information in each tier. In the local tier, each switch connects to a controller, and controllers construct local graphs. Then, the controllers also create contracted graphs to share it with other controllers. In the contracted graph, all bi-connected components of the local graphs are abstracted as a single node. The federation tier gathers the contracted graphs from all controllers, and discovers links connecting nodes in the contracted graphs in order to create a federation graph.

The federation tier and local controllers also have a routing algorithm that can decide flows using a graph.

3.2.2. Event information

This section mentions other information of each tier including events, such as topology changes, failures and flow setup requests. Considerable events of both tiers are listed below. At the events F1, F2, F3 and F4, each local controller should access a federation graph or shared events of a federation tier in distributed database.

Events of a federation tier

(F1) arrival of statistic information from a local controller
(F2) arrival of a packet_in from a local controller
(F3) discovery of a change of contracted graph in a local controller
(F4) discovery of a link failure in the federation graph

Events of a local tier

(L1) arrival of a packet_in from a switch
(L2) arrival of a flow setup request from the federation tier
(L3) arrival of a packet_out request from the federation tier
(L4) discovery of a change of a local graph
(L5) discovery of a failed or a congested link
(L6) discovery of a statistic change of a federation link

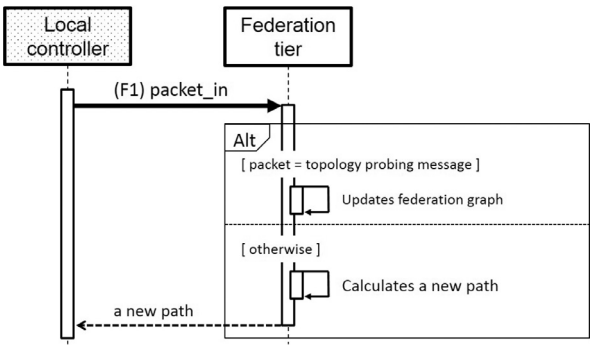


Fig. 4. A behavior of a federator in a packet_in event.

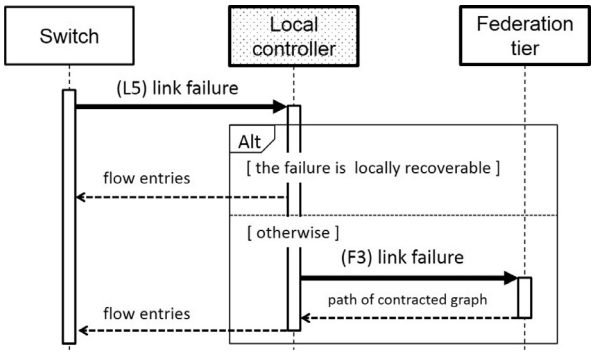


Fig. 5. The recovery process of a link failure.

When a local controller receives statistic information from a switch (L6), the controller invokes a control function of the federation tier (federator) if the information is not about local graph (F1).

When a local controller receives a packet_in message (L1), the local controller decides a path by its routing algorithm if the local graph contains enough information. Otherwise, it transfers the message to federator.

Receiving the packet_in (F2), a federator creates a flow by its routing algorithm and sends setup requests to local controllers following the flow. When the flow setup request is reached to a local controller (L2), the local controller translates the flow to a path on the local graph and then sends flow entries to switches in order to establish the path.

The federator can send packet_out messages to a port of a local controller for particular purposes, such as topology probing of inter-local controller links. Receiving the messages (L3), a local controller transfers the message to a corresponding port connecting to the other controllers. Then, the message is caught by another local controller as packet_in (L1) from a switch, and the controller informs the arrival of the packet_in to a federator (F2). Fig. 4 indicates behaviors of the federator. The federator updates federation graph if packet_in contains a topology probing message; otherwise it establishes a new working path and sends the new path to the local controller.

When varied topologies is found by a local controller (L4), the controller updates a local graph and a contracted graph. If the update changes the contracted graph, the controller sends it to a federator. Finding a varied link in the federation graph (F3), a federator updates federation graph. Fig. 5 shows that at the L5 event, the local controller deals with locally recoverable failures and does not send such failures to federators; otherwise, requires a backup path to a federator (F4).

The periodical changes in statistic information about load and probes of links are likely to pull out the events F1 and F2 frequently [9]. Thus, the amount of information of events is proportional to the number of federation nodes and links.

4. Clustering algorithm

This section proposes a clustering algorithm with cycles that are basic bi-connected components.

It is desirable to reduce *TotalAP* in the case (1) and decrease $|E^f|$ in the case (2). As previously stated, a larger bi-connected component in a cluster reduces *TotalAP*. In another words, increase of the number of links not in bi-connected components, that is $|E^f|$, improves *TotalAP*. Since reduction of $|E^f|$ in the case (2) also decreases *TotalAP*, the objective functions for improving reliability of virtual nodes and minimizing the size of federation graph is

$$\text{Minimize } |E^f| = f(G, \mathscr{C}) \qquad s.t. |V_i| < k, V_i \in \mathscr{C} \tag{2}$$

where k is the upper bound of a cluster size which indicates controller capability to govern switches and the size should be less than the maximum capability of controllers.

Generally, minimum cut or conductance is known as clustering indices focusing on intra-cluster density or inter-cluster sparsity [13]. these indices only consider the number of links in clusters and out of clusters, so they might have possibilities to fail to curb the size of federation graph. For instance, the lower graphs of Fig. 6(a)–(c) depict clustering examples in which the same graph are separated into two clusters. Although clustering in both Fig. 6(a) and (b) has the same number of inter-cluster links, the federation graph of (b) could include the larger number of nodes than (a). Because difference between (a) and (b) is balance of the number of inner-cluster edges,it may be a reasonable index for diminishing the federation graph. However, balancing the number of inner-cluster edges does not always decrease the number of bi-connected components like Fig. 6(c). These examples imply that a clustering method should focus on the number of bi-connected components in order to decrease the size of a federation graph. The following of this section proposes a clustering method considering cycles that are basic bi-connected components.

Let $\mathcal{L}$ be a set of cycles in a graph $G = (V, E)$, and the union of all cycles in $\mathcal{L}$ includes all nodes in V. Then, a cycle graph $\underline{G} = (\mathcal{L}, \underline{E}, \omega)$ is defined as a meta-graph whose node is a cycle and a link between two cycles indicates existence of more than one shared nodes of the two cycles. A link cost function $\omega : \mathcal{L} \times \mathcal{L} \to \mathcal{P}(V)$ indicates shared nodes in V of two cycles on both ends of the link provided $\mathcal{P}(V)$ is the power set of V. An example of cycles L_i in a graph and its cycle graph are shown in Fig. 7.

A division of a graph into some clusters must break some other cycles because nodes cannot belong to two or more clusters. For instance, when L_2 in Fig. 7 is in a cluster,

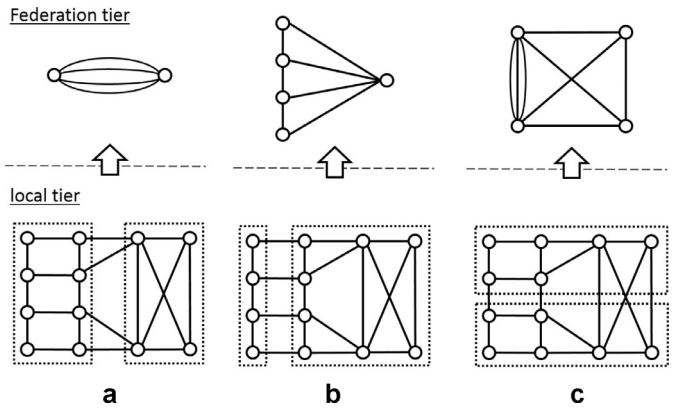


Fig. 6. Differences of federation graphs by clustering.

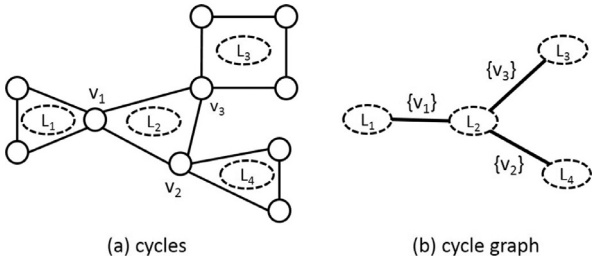


Fig. 7. An example of cycle graph.

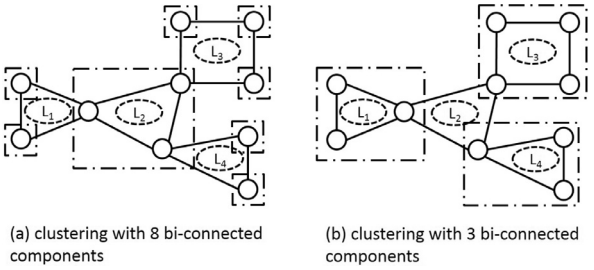


Fig. 8. Differences of bi-connected components by cycle selection.

cycles L_1, L_3 and L_4 are decomposed. Moreover, the cluster including L_2 could result in larger number of bi-connected components depicted in Fig. 8(a) although the graph can be decomposed three bi-connected components like Fig. 8(b). Hence, the cluster selection method should consider the decomposed cycles. Therefore, the selection method counts the number of nodes isolated from all cycles because the nodes on a decomposed cycle can be included in the other unchanged cycles. The *Isolated Nodes* $IN(L_a)$ are indicated as

$$\cup_{L_i \in N_{L_a}} \left(V(L_i) - \cup_{L_j \in N_{L_i} - N_{L_a}} \omega(L_i, L_j) \right) \tag{3}$$

where L_i is a selected cycle, and N_{L_i} is adjacent cycles of L_i in $\underline{G}$. In Fig. 8, L_1, L_3 and L_4 have no isolated nodes whereas L_2 has seven.

Containing a selected cycle, a cluster can be expanded if the number of nodes in the cluster is less than k. The expansion of a cluster can insert isolated nodes $IN(L_i) \cap L_j$ in $L_j \in N_{L_i}$ into the bi-connected component $L_i \cup L_j$. However, this expansion could causes different isolated nodes, so *Isolated Nodes after Expansion (INE)* of L_j is

$$INE(L_i, L_j) = IN(L_i) \cup IN(L_j) - V(L_i) - V(L_j). \tag{4}$$

Algorithm 1 shows a clustering method (cycle clustering) with a cycle graph, IN and INE. First, it selects a cycle L_a by IN from the cycle graph and inserts L_a into a cluster in the lines 5 and 6. Then, in order to expand the size of nodes in the cluster up to k, L_b with minimum INE are chosen from *candidates* that represent the neighbors of cycles in the current cluster in the lines 9. Then, L_b are added to the cluster, and finally, the expanded cluster is obtained.

5. Experimental results

This section presents experimental results about comparison on *TotalAP* and the number of edges in federation graph.

Algorithm 1 Cycle clustering.

Require: $\underline{G} = (\mathcal{L}, \underline{E}, \omega)$ and k
1: Let N_L be adjacent cycles of L in the graph $\underline{G}$.
2: k is upper bound of a cluster size
3: Clustering $\mathscr{C} \leftarrow \phi$
4: **while** $\mathcal{L} \neq \phi$ **do**
5: Select a cycle L_a with minimum $IN(L_a)$ from $\mathcal{L}$
6: $C \leftarrow L_a$
7: $Candidates \leftarrow N_{L_a}$
8: Delete cycles with larger than $k - |V(C)|$ nodes from *Candidates*
9: **while** $Candidates \neq \phi$ **do**
10: Select a cycle L_b with minimum $INE(C, L_b)$ from *Candidates*
11: Combine L_b to C
12: $Candidates \leftarrow N_C$
13: Delete cycles with larger than $k - |V(C)|$ nodes from *Candidates*
14: **end while**
15: Add $V(C)$ to $\mathscr{C}$
16: Add $IN(C)$ to $\mathscr{C}$
17: Delete C from $\underline{G}$
18: **end while**
19: **return** $\mathscr{C}$

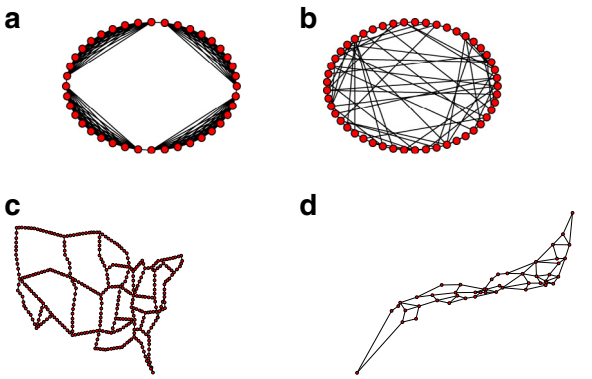


Fig. 9. An example of graphs. (a) Connected caveman. (b) NWS. (c) Real network model 1 [14]. (d) Real network model 2 [15].

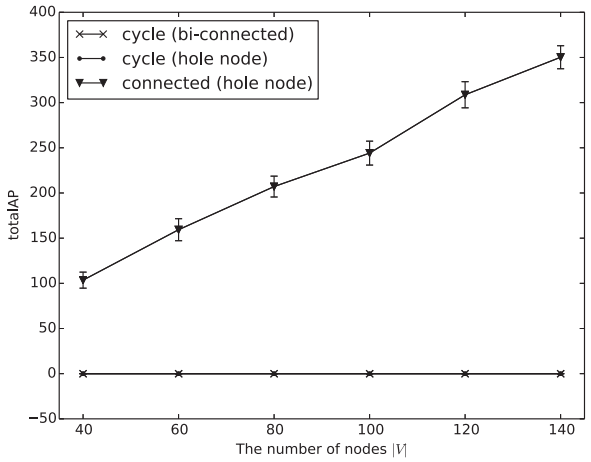
Algorithm 2 Connected clustering.

Require: $G = (V, E)$ and k
Clustering $\mathscr{C} \leftarrow \phi$
while $V \neq \phi$ **do**
  Select a *root* $\in V$
  $C \leftarrow root$
  while $|C| < k$ and C has adjacent nodes $v \in V$ **do**
    $C \leftarrow v$
  end while
  Clustering $\mathscr{C} \leftarrow C$
  Delete C from V
end while
return $\mathscr{C}$

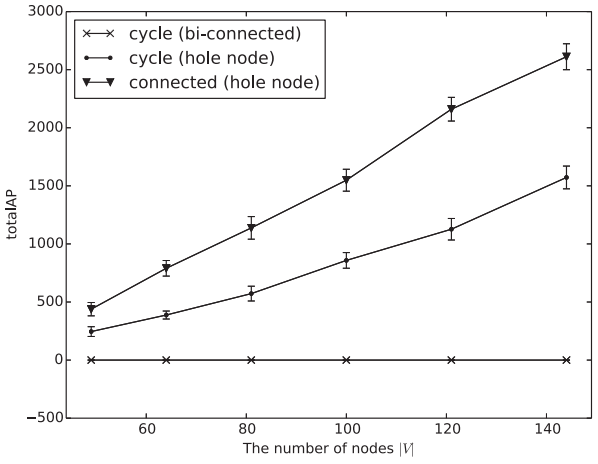
Algorithm 3 HCS clustering.

Require: $G = (V, E)$ and k
function HCS(G, k)
  $(H_1, H_2, C) \leftarrow MinimumCut(G)$
  if G is highly connected and $|V(G)| \leq k$ **then**
    $\mathscr{C} \leftarrow V(G)$
  else
    $HCS(H_1, k)$
    $HCS(H_2, k)$
  end if
end function
Clustering $\mathscr{C} \leftarrow \phi$
HCS(G, k)
return $\mathscr{C}$

This experiment compares cycle clustering algorithm with connected clustering and HCS (Highly Connected Subgraph) clustering [16]. The connected clustering is a simple clustering method that randomly selects a node as a cluster seed and expands it by using a tree search algorithm BFS (Breadth First Search). This clustering seek to shorten distances between controllers and switches when the controllers are randomly deployed. Such a cluster construction method focusing on distance is shown in [17]. The pseudo-code of this algorithm is shown in Algorithm 2. HCS clustering shown in Algorithm 3 repeatedly separates a graph by minimum cut in order to deminish the number of inter-cluster edges.



(a) Connected caveman.



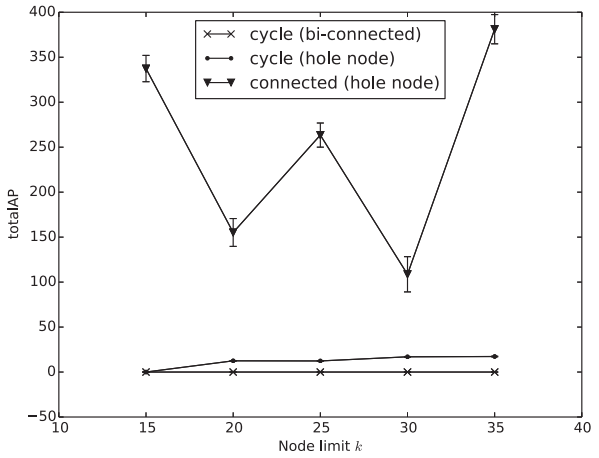
(b) NWS.

Fig. 10. The # of nodes $|V|$ vs. the # of affected ports.

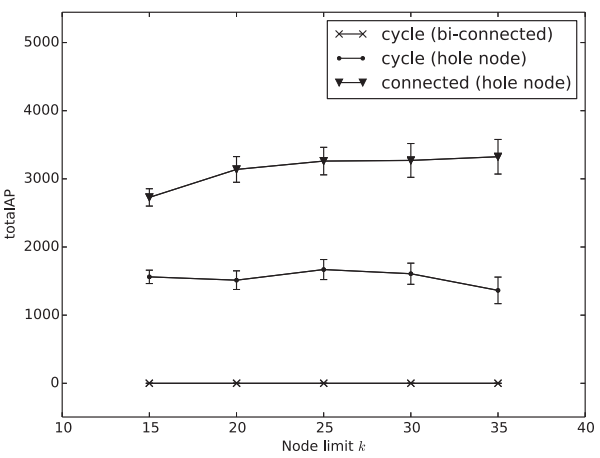
These three clustering algorithms are implemented by using Python and NetworkX.

Connected caveman graph, Newman Watts Strogatz (NWS) random graph and graphs of two real network model are used for our experiments. A connected caveman graph is a graph formed by cycle of caves that are subgraphs removed one link from cliques. They can be used as benchmark graphs of clustering algorithms because the caveman graph is most sparse graph whose clustering coefficient is nearly one (one is maximum), that coefficient is the measure of the degree to which nodes in a graph tend to cluster together [18]. Fig. 9 describes five types of graphs used in experiments with showing the characteristics in Table 1.

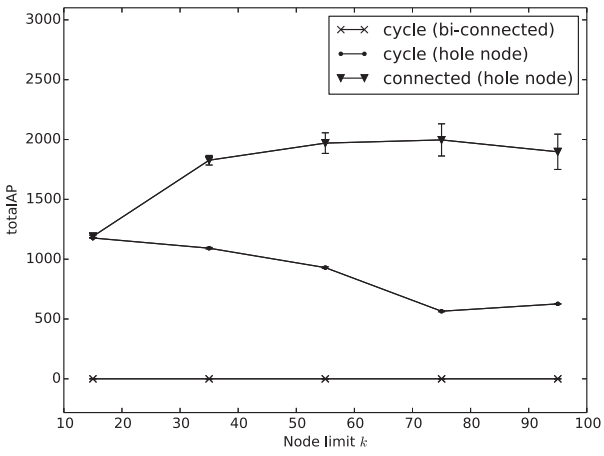
For *TotalAP* and the number of federation edges, there are two types of experiments: the first varies the number of nodes in a graph and the second varies the number of node limitation k of a cluster which is important factor as same as the size of a graph for clustering methods. In the first, the node limitation k is set to 15. In the second, the number of nodes is set to 140 for connected caveman graphs and 144 for NWS graphs.



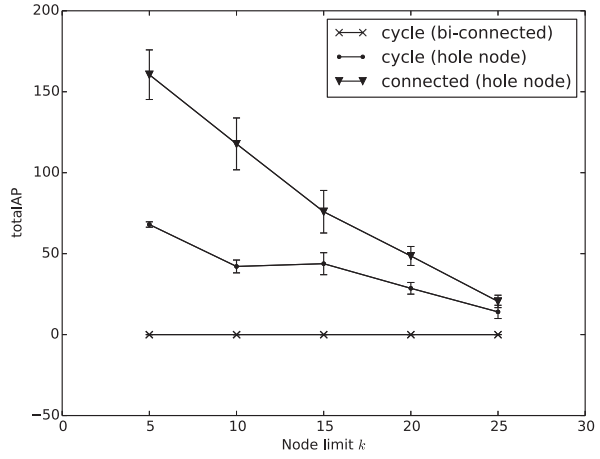
(a) Connected caveman.



(b) NWS.



(c) Real network model 1.



(d) Real network model 2.

Fig. 11. The node limit k vs. the # of affected ports.

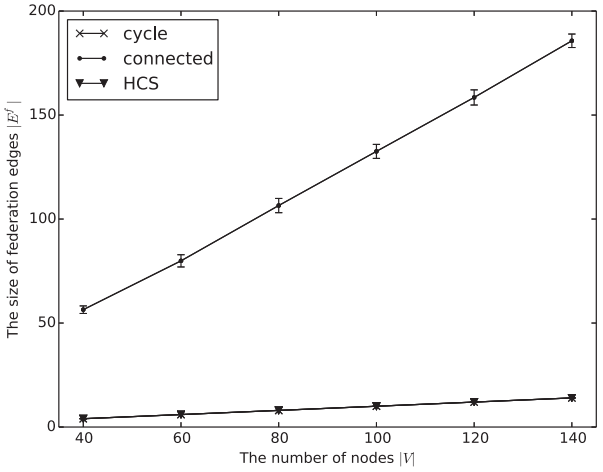
Fig. 10 describes *TotalAP* for the first experiments and Fig. 11 depicts *TotalAP* for the second experiments. *Cycle* and *connected* in these figures indicate clustering, and *bi-connected* and *hole node* mean limitation of abstraction area. Figs. 10 and 11 depict that abstracted nodes of cycle clustering are more reliable than connected clustering in each graph. These figures also indicate that virtual nodes abstracting bi-connected components are truly reliable in any situation. In connected caveman graphs, cycle clustering significantly decreases *TotalAP* while connected clustering depends on the node limitation. Fig. 11(c) shows that connected clustering cannot decrease the *TotalAP* when the node limitation is large in real network model 1 which is sparse graph. In contrast, in a real network model 2 that is more dense than network model 1, connected clustering can decrease the *TotalAP* as shown in Fig. 11(d). It is considerable that connected clustering can efficiently work in dense graphs.

Figs. 12 and 13 show $|E^f|$ of the first experiments and the second experiments, respectively. Fig. 12(a) shows that cycle and HCS can decrease the number of federation edges $|E^f|$ in connected caveman graphs. Fig. 12(b) shows that cycle clustering has the smallest federation edges in NWS graphs. From the above, cycle clustering can curb the number of federation edges in each type of graphs.

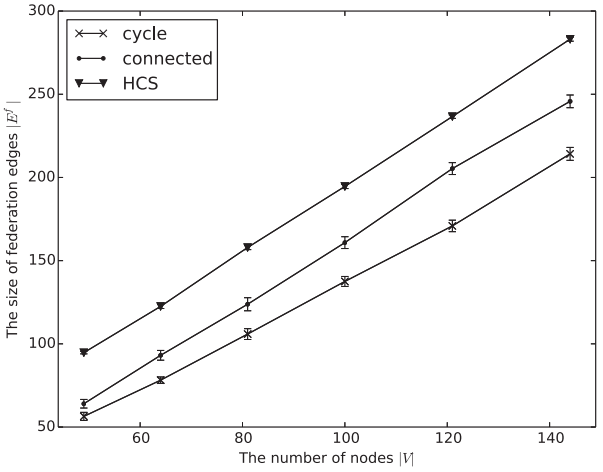
Table 1
The number of nodes and edges of graphs in experiments.

Connected caveman graph						
The # of nodes	40	60	80	100	120	140
The # of edges	180	270	360	450	540	630
The # of nodes in a cave	10	10	10	10	10	10
NWS graph						
The # of nodes	49	64	81	100	121	144
The # of edges	98	128	162	200	242	288
Real network model 1						
The # of nodes	365					
The # of edges	772					
Real network model 2						
The # of nodes	48					
The # of edges	82					

On the other hand, the second experiment changes the node limitation of a cluster. Fig. 13(d) shows that cycle

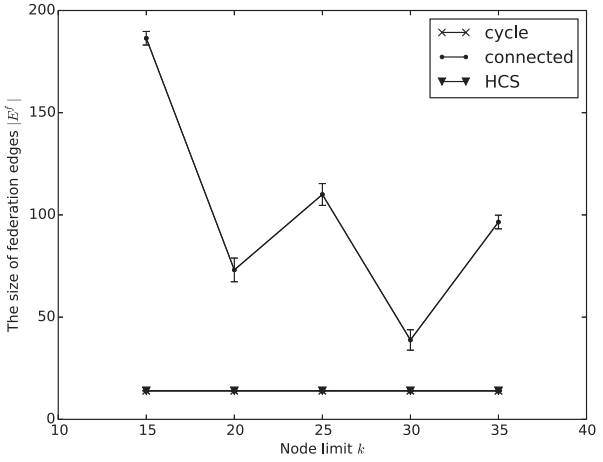


(a) Connected caveman.

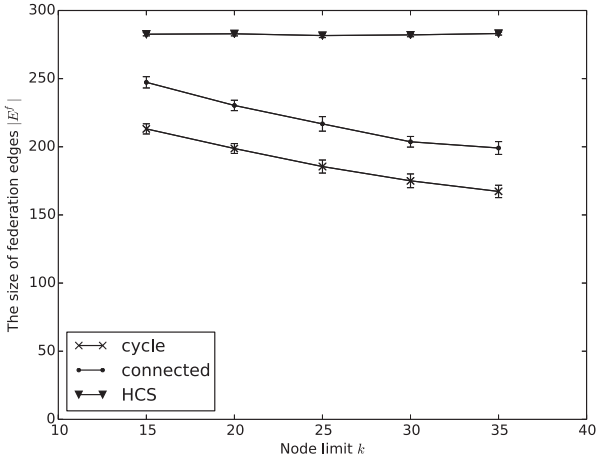


(b) NWS.

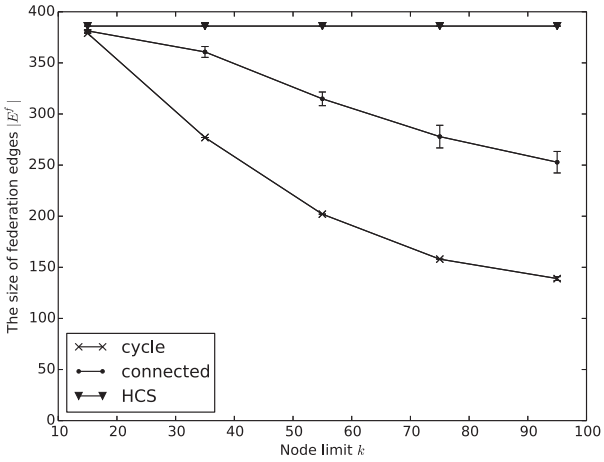
Fig. 12. The # of nodes $|V|$ vs. the # of federation edges $|E^f|$.



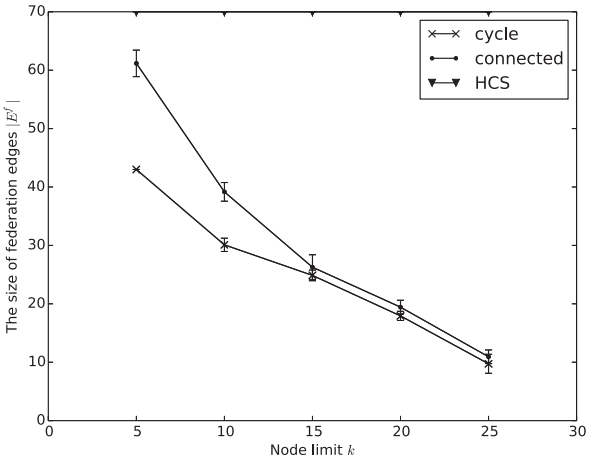
(a) Connected caveman.



(b) NWS.



(c) Real network model 1.



(d) Real network model 2.

Fig. 13. The node limit k vs. the # of federation edges $|E^f|$.

clustering can decrease the number of federation edges when the node limitation is small. In addition, Fig. 13(a)–(c) indicate that cycle clustering decreases the federation edges in connected caveman, NWS and real network model 1 graphs. In Fig. 13(a), the results of connected clustering sharply fall at 20 and 30 nodes. This is caused by higher possibilities to construct a cluster containing whole nodes in a cave when the node limitations are equal to integral multiple of cave size (10 nodes).

6. Discussion on implementation issues

This chapter describes how to deploy the results of clustering by using Onix or HyperFlow. To deploy the clustering results, each Onix/HyperFlow instance should govern a subnetwork consisting of nodes in a cluster; thus, the topology of the subnetwork correspond to a local graph. The instances also share the federation graph by its distributed database that is the NIB (Network Information Base) or WheelFS. To construct the federation graph, the instance should share aggregated topology information such as the contracted graph with other instances and have an observation function of inter-cluster links.

The experimental results indicate that the cycle clustering can diminish the number of federation links, which allows our method to generate a larger cluster including biconnected components. The clustering also enables a local controller to register the flow entries to switches without using a globally shared graph when a link failure occurs in the administrative area. In the case of traffic balancing, the burden of construction of consistent flow entries can be reduced because the controllers can locally distribute traffic in their bi-connected components.

The computation time of cycle clustering is 13 seconds on average for the real network model 1 which is the largest graph in our experiments. The computer used for experiments has intel corei7-960 processor, 24 GB RAM and ubuntu 14.04 installed. The calculation time is sufficiently short because this paper focuses on calculating the initial settings of switch clustering.

7. Conclusion

This paper presents issues of multi-controllers of OpenFlow: (1) reliability of abstracted virtual nodes and (2) increase of the amount of shared information when more reliable virtual nodes are constructed. In order to improve the reliability of virtual nodes and reduce surge of shared information among controllers, this paper proposes effective clustering method focusing on cycles. Experimental results show that our cycle clusterin gmethod can improve reliability of virtual nodes and curb the number of edges in a federation graph that is regarded as shared information.

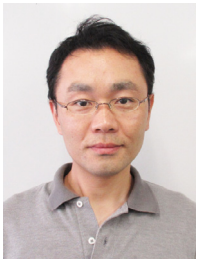
Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 26330120.

References

[1] J.-P. Vasseur, M. Pickavet, P. Demeester, Network Recovery, Morgan Kaufmann, 2004.
[2] M. Todinov, Flow Networks, 1st, Elsevier, 2013.
[3] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. van Reijendam, P. Weissmann, N. McKeown, Maturing of OpenFlow and software-defined networking through deployments, Comput. Netw. 61 (2014) 151–175, doi:10.1016/j.bjp.2013.10.011.
[4] S.H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Commun. Mag. 51 (2) (2013) 136–141, doi:10.1109/MCOM.2013.6461198.
[5] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in software defined networks, Comput. Netw. 71 (2014) 1–30, doi:10.1016/j.comnet.2014.06.002.
[6] M. Jammal, T. Singh, A. Shami, R. Asal, Y. Li, Software defined networking: State of the art and research challenges, Comput. Netw. 72 (2014) 74–98, doi:10.1016/j.comnet.2014.07.004.
[7] A. Tootoonchian, Y. Ganjali, HyperFlow: a distributed control plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, 2010, pp. 1–6.
[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: a distributed control platform for large-scale production networks, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, 10, 2010, pp. 1–6.
[9] S.H. Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot topics in Software Defined Networks – HotSDN '12, ACM Press, New York, USA, 2012, pp. 19–24, doi:10.1145/2342441.2342446.
[10] O. Papapetrou, W. Siberski, W. Nejdl, PCIR: combining DHTs and peer clusters for efficient full-text P2P indexing, Comput. Netw. 54 (12) (2010) 2019–2040, doi:10.1016/j.comnet.2010.03.025.
[11] E. Meshkova, J. Riihijärvi, M. Petrova, P. Mähönen, A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks, Computer Networks 52 (11) (2008) 2097–2128, doi:10.1016/j.comnet.2008.03.006.
[12] H. Aoki, J. Nagano, N. Shinomiya, Layered control plane for reducing information sharing in OpenFlow networks, in: Proceedings of IEEE International Conference on Communication Workshop (ICCW), IEEE, 2015, pp. 369–374, doi:10.1109/ICCW.2015.7247207.
[13] U. Brandes, T. Erlebach, Network Analysis, Springer Berlin Heidelberg, 2005.
[14] N. Shinomiya, T. Hoshida, Y. Akiyama, H. Nakashima, T. Terahara, Hybrid link/path-based design for translucent photonic network dimensioning, J. Lightw. Technol. 25 (10) (2007) 2931–2941, doi:10.1109/JLT.2007.905224.
[15] T. Sakano, Y. Tsukishima, H. Hasegawa, T. Tsuritani, Y. Hirota, S. Arakawa, H. Tode, A study on a photonic network model based on the regional characteristics of Japan (in Japanese), in: IEICE Technical Report of Photonic Network, 113, Fukushima, Japan, 2013, pp. 1–6.
[16] E. Hartuv, R. Shamir, A clustering algorithm based on graph connectivity, Inf. Process. Lett. 76 (2000) 175–181, doi:10.1016/S0020-0190(00)00142-3.
[17] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the first workshop on Hot topics in software defined networks – HotSDN '12, ACM Press, New York, USA, 2012, pp. 7–12, doi:10.1145/2342441.2342444.
[18] D.J. Watts, Networks, dynamics, and the SmallWorld phenomenon, Am. J. Sociol. 105 (2) (1999) 493–527, doi:10.1086/210318.

Junichi Nagano recieved his M.E. degree in information system science from Soka University, Tokyo, Japan in 2007.
He is currently working towards the Ph.D. degree in system modeling at Soka University. His research interests include graph theory, network modeling and software defined networking.



Norihiko Shinomiya received the B.E. degree, the M.E. degree and the Ph.D. degree in information systems science in 1995, 1997 and 2001, respectively, from Soka University in Tokyo, Japan. In 2000, he joined the Network Systems Laboratories, Fujitsu Laboratories Ltd., as a research engineer, where he engaged in development of a planning and design tool for wavelength division multiplexing (WDM) networks and a control method of IP networks.
Since 2005, he has been an associate professor of department of information systems science, faculty of engineering, Soka University, where he has been engaged in the research and development of design method, control architecture and management system based on the graph theoretical algorithms. He received the Best Paper Award at IEEE ICUMT in 2010 and two other Best Paper Awards at IARIA ICONS in 2014 at a time.
Dr. Shinomiya is a member of IEEE CAS, ComSoc and Computer societies. He served as a secretary of IEEE CAS Society Japan Chapter from 2012 to 2013.