Improving the performance of load balancing in software-defined networks through load variance-based synchronization

Zehua Guo,*, Mu Su, Yang Xu, Zhemin Duan, Luo Wang, Shufeng Hui, H. Jonathan Chao

**A R T I C L E   I N F O**

Article history:
Received 30 May 2013
Received in revised form 7 November 2013
Accepted 4 December 2013
Available online xxxx

Keywords:
Software-Defined Networking
Load balancing
Multiple controllers
Controller state synchronization

**A B S T R A C T**

Software-Defined Networking (SDN) is a new network technology that decouples the control plane logic from the data plane and uses a programmable software controller to manage network operation and the state of network components. In an SDN network, a logically centralized controller uses a global network view to conduct management and operation of the network. The centralized control of the SDN network presents a tremendous opportunity for network operators to refactor the control plane and to improve the performance of applications. For the application of load balancing, the logically centralized controller conducts Real-time Least loaded Server selection (RLS) for multiple domains, where new flows pass by for the first time. The function of RLS is to enable the new flows to be forwarded to the least loaded server in the entire network. However, in a large-scale SDN network, the logically centralized controller usually consists of multiple distributed controllers. Existing multiple controller state synchronization schemes are based on Periodic Synchronization (PS), which can cause undesirable situations. For example, frequent synchronizations may result in high synchronization overhead of controllers. State desynchronization among controllers during the interval between two consecutive synchronizations could lead to forwarding loops and black holes. In this paper, we propose a new type of controller state synchronization scheme, Load Variance-based Synchronization (LVS), to improve the load-balancing performance in the multi-controller multi-domain SDN network. Compared with PS-based schemes, LVS-based schemes conduct effective state synchronizations among controllers only when the load of a specific server or domain exceeds a certain threshold, which significantly reduces the synchronization overhead of controllers. The results of simulations show that LVS achieves loop-free forwarding and good load-balancing performance with much less synchronization overhead, as compared with existing schemes.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Software-Defined Networking (SDN) is a promising networking technology that provides network operators more control of the network infrastructure. SDN technology decouples the control plane logic from the data plane by...
moving networking control functions from forwarding devices (e.g., switches, routers) to a logically centralized controller so that the networking functions can be implemented by software. One enabler of SDN technology is OpenFlow [1], which provides global visibility of flows in the network by enabling flow-level control over Ethernet switching. Currently, many companies and organizations are working on deploying OpenFlow on backbone and campus networks, such as Internet2 [2], Global Environment for Network Innovation (GENI) [3], and Google’s WAN-B4 [4].

Typically, an SDN network relies on a logically centralized controller using the global network knowledge to operate the network. In the SDN network, the controller conducts packet processing by matching forwarding rules, which can be installed in the flow tables of switches either reactively (e.g., when a new flow arrives) or proactively (e.g., controller installs rules in advance). When a new flow traverses a domain for the first time, the controller selects a forwarding path for the new flow based on the current global network status and configures the forwarding path by reactively installing rules on related switches. The reactive operation using the current global network status makes the improvement of applications possible. For the application of load balancing, when a new flow enters a domain for the first time, the controller conducts Real-time Least loaded Server selection (RLS) for this domain to select the current least loaded server as the destination server of the new flow based on the current network status. After a series of RLSs for related domains, the new flow’s forwarding path to the least loaded server is determined, and the packets of this flow are forwarded via this path. The detail of RLS is elaborated in Section 2.

However, having one centralized controller creates problems including poor scalability, responsiveness, and reliability, which have significantly hindered and detained the deployment of SDN technology in large-scale production and data center networks [5,6]. To overcome the above problems, a simple solution is to use multiple distributed controllers working together to achieve the functionality of the logically centralized controller (e.g., HyperFlow [5] and ASIC [6]), which can benefit from the scalability and reliability of the distributed architecture while preserving the simplicity of the centralized system. In a multi-controller multi-domain SDN network, each controller runs one domain simultaneously, and it only handles the local area switches in its domain. Currently, most of the controllers applied in the SDN network can be categorized as Local State-aware Controllers (LSCs). An LSC could, in real-time, acquire the state of servers in its controlled domain by pulling statistics from edge switches of its domain. However, the LSC obtains the state of servers in domains managed by other controllers through controller state synchronization. The LSC stores the state of servers in its Network Information Base (NIB) and uses its NIB to make forwarding decisions for new flows [7]. A NIB may contain different contents based on the need of network applications [7–9]. In this paper, we focus on the flow-based, web load balancing, where the NIB of each controller contains the load of each server (e.g., the utilization of the link connecting with each server [7]).

One of the major challenges when using multiple controllers in the SDN network is to synchronize the state among controllers. The state desynchronization among controllers significantly degrades the performance of the application. Levin et al. [7] study the controller state synchronization problem for the application of load balancing in the SDN network and propose two controller state synchronization schemes based on Periodic Synchronization (PS): Link Balancer Controller (LBC) and Separate State Link Balancer Controller (SSLBC). Both of these schemes initiate state synchronization among controllers within a fixed period. Using LBC, each controller directs newly arrived flows to the current least loaded server in the entire network based on its NIB, which is updated and synchronized periodically by controller state synchronizations. However, good load-balancing performance using LBC requests frequent synchronizations, which overly burden controllers. To achieve good performance with less synchronization overhead of controllers, SSLBC is further developed. In SSLBC, each controller divides its NIB into two parts: the local NIB containing the load of servers in the local domain and the remote NIB containing the load of servers in domains except the local domain. Using SSLBC, the remote NIB is updated periodically by controller state synchronizations, while the local NIB is updated in real time. The local domain’s controller is aware of the real-time load variation of servers in the local domain by the real-time updated local NIB, which enables the controller to select the least loaded server based on accurate local domain status. Thus, SSLBC achieves better load-balancing performance with the same number of synchronizations as LBC. However, as will be detailed in Section 3, SSLBC may lead to several undesired exceptional situations (e.g., forwarding loops [7,10–12] and black holes [7,10,11]) due to state desynchronization among controllers.

In this paper, we investigate the application of load balancing in the SDN network with multiple domains managed by multiple controllers. We propose a new type of controller state synchronization scheme, Load Variance-based Synchronization (LVS), to improve load-balancing performance by eliminating forwarding loops and to lower synchronization frequency. We first introduce load balancing in the SDN network in detail and differentiate it from load balancing in the IP network. Based on the description, we analyze two problems in existing PS-based schemes: the high controller synchronization overhead problem caused by frequent synchronizations and the forwarding-loop problem caused by state desynchronization among controllers during the interval between two consecutive synchronizations. Then, we solve the two problems with two specific LVS-based schemes: Least loaded Server Variation Synchronization (LSVS) and Least loaded Domain Variation Synchronization (LDVS). Compared with PS-based schemes, LVS-based schemes achieve low synchronization frequency by conducting effective controller state synchronizations only when the load of a specific server or domain exceeds a certain threshold. The results of simulations

We use the terms synchronization overhead of controllers and the number of synchronizations interchangeably in this paper.
show that LVS-based schemes achieve loop-free forwarding and good load balancing performance with much fewer synchronizations as compared with existing schemes.

The rest of the paper is organized as follows. Section 2 distinguishes load balancing in the SDN network from load balancing in the IP network. Section 3 presents the problems of existing PS-based schemes. Section 4 proposes two specific LVS-based schemes to improve the load-balancing performance by addressing the problems presented in Section 3. Section 5 compares two proposed LVS-based schemes with the existing PS-based schemes. Section 6 discusses a practical issue in LVS-based schemes implementation. Section 7 concludes the paper.

2. Problem background

2.1. Notations

We first summarize notations used in this paper in Table 1.

2.2. Load balancing in the IP network

Load balancing [6,7,13,14] is a typical network application. The goal of load balancing is to enable the load of each server to approach the global optimal balanced server load \( U_{\text{global,opt}} \) by dynamically dispatching new flows to multiple servers in the network. \( U_{\text{global,opt}} \) is expressed as follows:

\[
U_{\text{global,opt}} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M_i} U_{ij}}{NM_i}
\]

where \( N \) denotes the number of domains, \( M_i (1 \leq i \leq N) \) denotes the number of servers in domain \( i \), and \( U_{ij} (1 \leq j \leq M_i, 0 \leq U_{ij} \leq 1) \) denotes the load of the \( j \)-th server in domain \( i \).

In the traditional IP network, the function of load balancing is achieved by the Load Balancing Router (LBR). Specifically, when a new flow \( f_{\text{new}} \) enters the network, \( f_{\text{new}} \) will first go through LBR. In LBR, a specific server (e.g., the least loaded server) is selected as the destination server of \( f_{\text{new}} \) based on the current network status. The IP address of the destination server is then written into the header of \( p_{\text{new}} \) (i.e., the packet of \( f_{\text{new}} \)). Subsequently, \( p_{\text{new}} \) will be strictly forwarded to its destination server via the path calculated by routing protocols (e.g., OSPF). Obviously, in the IP network, the forwarding decision is made only based on the network status at the time of selecting the destination server. However, the network status may change over time. In such conditions, during the forwarding process, there might be some servers that are more suitable than the destination server selected by LBR before. Therefore, the load-balancing performance in the IP network is not optimal.

2.3. Load balancing in the SDN network

In an SDN network, a series of Real-time Least loaded Server selections (RLSs) is conducted by the controller to achieve load balancing. RLS is used to determine a new flow’s destination server and to calculate a path heading to the destination server when the new flow enters a domain for the first time. The intuition behind RLS is to make the forwarding decision for each new flow based on a time-varying network status. Specifically, assume a new flow \( f_{\text{new}} \) enters the SDN network from a domain \( D_i \). Since \( f_{\text{new}} \) enters \( D_i \) for the first time, it does not match any rules in the flow tables of switches in \( D_i \) and the controller cannot find \( f_{\text{new}} \)’s forwarding path. Thus, similar to the IP network, RLS is conducted to select the destination server of \( f_{\text{new}} \). In RLS, the controller uses its current NIB to select the least loaded server \( S_{\text{least}} \) in the entire network as the destination server of \( f_{\text{new}} \). Unlike in the IP network where LBR writes the IP address of \( S_{\text{least}} \) into the header of \( f_{\text{new}} \)’s packets, the controller only configures a forwarding path to direct \( f_{\text{new}} \) from the current domain \( D_i \) to the domain \( D'_i \), which is closer to \( S_{\text{least}} \). If \( f_{\text{new}} \) arrives at \( D'_i \) and the controller cannot find its forwarding path in \( D'_i \), the controller also activates RLS to select the destination server of \( f_{\text{new}} \) based on its current NIB. The result of RLS is \( S_{\text{least}} \). Because the network status varies over time, \( S_{\text{least}} \) could be different from \( S_{\text{least}} \). If so, \( f_{\text{new}} \) is directed to a domain that is closer to \( S_{\text{least}} \) via a new calculated path and \( S_{\text{least}} \) is updated to \( S_{\text{least}} \). Otherwise, \( f_{\text{new}} \) is forwarded to the next domain that is closer to original \( S_{\text{least}} \). Once a packet of \( f_{\text{new}} \) arrives at the destination server, subsequent packets of \( f_{\text{new}} \) will be strictly forwarded via the same path to the same destination server until the path is removed by the timeout scheme or the controller.

RLS enables the controller to make better forwarding decisions for new flows using real-time network status.

---

Table 1

<table>
<thead>
<tr>
<th>Notations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>Number of domains</td>
</tr>
<tr>
<td>( M_i )</td>
<td>Number of servers in domain ( i )</td>
</tr>
<tr>
<td>( T )</td>
<td>Synchronization period</td>
</tr>
<tr>
<td>( D_i )</td>
<td>Domain ( i (1 \leq i \leq N) )</td>
</tr>
<tr>
<td>( C_{LC} )</td>
<td>Logically centralized controller of an SDN network</td>
</tr>
<tr>
<td>( C_i )</td>
<td>Controller of domain ( i )</td>
</tr>
<tr>
<td>( f_{\text{new}} )</td>
<td>Newly arrived flow</td>
</tr>
<tr>
<td>( p_{\text{new}} )</td>
<td>Packet of ( f_{\text{new}} )</td>
</tr>
<tr>
<td>( S_{ij} )</td>
<td>( j )-th server of domain ( i (1 \leq j \leq M_i) )</td>
</tr>
<tr>
<td>( S_{\text{least}} )</td>
<td>Least loaded server selected by the controller of the domain a flow just passed by</td>
</tr>
<tr>
<td>( S_{\text{least}} )</td>
<td>Least loaded server selected by the controller of the current domain</td>
</tr>
<tr>
<td>( U_{ij} )</td>
<td>Load of ( S_{ij} )</td>
</tr>
<tr>
<td>( U_{\text{load}} )</td>
<td>Load of ( S_{\text{least}} )</td>
</tr>
<tr>
<td>( U_{\text{global,opt}} )</td>
<td>Global optimal load</td>
</tr>
<tr>
<td>( D_{\text{least}} )</td>
<td>Least loaded domain</td>
</tr>
<tr>
<td>( THS )</td>
<td>Threshold of LSVS</td>
</tr>
<tr>
<td>( THS )</td>
<td>Threshold of LDVS</td>
</tr>
<tr>
<td>( S_{\text{load}} )</td>
<td>A specific server used in ( THS )</td>
</tr>
<tr>
<td>( D_{\text{load}} )</td>
<td>Score of ( S_{\text{load}} )</td>
</tr>
<tr>
<td>( \Delta U )</td>
<td>Load margin</td>
</tr>
</tbody>
</table>

Please cite this article in press as: Z. Guo et al., Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Netw. (2014), http://dx.doi.org/10.1016/j.comnet.2013.12.004
as compared with the forwarding decision only based on the network status at the time of selecting the destination server in the IP network. In [7], SSLBC applies RLS and improves load-balancing performance.

Fig. 1 illustrates how RLS works for a new flow in a domain of an example SDN network. In the figure, the SDN network is composed of \( N \) isolated domains \( \{D_1, ..., D_N\} \) and is controlled by a logically centralized controller \( C_{LC} \). In each domain, edge switches connect with web servers, which provide web services to users outside the SDN network. Let \( S_{ij} \) denote the \( j \)-th server in domain \( i \) and \( S_{\text{least}} \) denote the least loaded server of the entire SDN network. RLS processing includes seven steps. When a new flow \( f_{\text{new}} \) enters \( D_1 \) for the first time, the first switch receiving \( f_{\text{new}} \) asks \( C_{LC} \) where to forward \( f_{\text{new}} \), as shown by steps 1 and 2. After searching its global NIB, \( C_{LC} \) finds that \( S_{ij} \) is the current \( S_{\text{least}} \) and then determines \( f_{\text{new}} \)'s path to \( S_{\text{least}} \) by looking up its routing table, as shown by steps 3, 4, and 5. In the next step, \( C_{LC} \) sets up \( f_{\text{new}} \)'s path to \( S_{\text{least}} \) by installing rules in the flow tables of related switches in \( D_1 \). Finally, \( f_{\text{new}} \) is directed to the next domain that is closer to \( S_{\text{least}} \) via the configured path.

The description above is based on the assumption that an SDN network only contains a centralized controller, which stores the entire network status and conducts packet possession for all new flows. However, in the real deployment, the centralized controller is usually implemented by multiple distributed controllers to improve limited scalability, responsiveness, and reliability over using one controller. Fig. 2 shows the deployment of multiple distributed controllers in a large-scale SDN network. In this figure, \( C_i \) is physically composed of multiple distributed controllers \( C_i \), which manages domain \( D_i \) and maintains the NIB stored in \( C_i \). The combination of \( C_i \) works together to achieve the function of \( C_{LC} \). When a new flow \( f_{\text{new}} \) enters the network from \( D_1 \), \( C_1 \) is queried by the first switch of \( D_1 \) receiving \( f_{\text{new}} \), and \( S_{\text{least}} \) is selected as the destination server of \( f_{\text{new}} \) based on the NIB of \( C_1 \). If \( S_{\text{least}} \) is located in \( D_1 \), \( C_1 \) configures the related switches in \( D_1 \) with a path to \( S_{\text{least}} \) and forwards \( f_{\text{new}} \) via this path. Otherwise, \( C_1 \) uses routing protocols to calculate a path, which directs \( f_{\text{new}} \) to the next domain heading to \( S_{\text{least}} \), and configures the related switches in \( D_1 \) with this path. In the rest of this paper, controller is short for distributed controller, unless stated otherwise.

3. Problems of existing schemes

In the above discussion, load balancing in the SDN network is achieved by selecting the forwarding paths for new flows based on controllers’ NIBs. The update of controllers’ NIBs is determined by the controller state synchronization scheme, which significantly impacts load-balancing performance. The existing PS-based controller state synchronization schemes have two major problems: the high synchronization overhead of controllers and the forwarding loop, which will be detailed in the following subsections.

![Fig. 1. Example of Real-time Least loaded Server selection (RLS) in a SDN network.](image-url)
3.1. Term definitions

For ease of understanding the proposed schemes in this paper, we highlight several important terms. All terms are defined from the view of $C_i$.

Local domain: the domain controlled by $C_i$, i.e., $D_i$.
Remote domain: the domain controlled by a controller except $C_i$.
Local NIB: a part of $C_i$’s global NIB, which stores the load of servers in $D_i$.
Remote NIB: a part of $C_i$’s global NIB, which stores the load of servers in the remote domains of $C_i$.

Update: the action to obtain the real-time load of servers in the specific domain(s) and to store those loads into a specific part of the NIB.
Update of local NIB: the action to get the real-time load of servers in $D_i$ by pulling the statistics from edge switches of $D_i$ and to store those loads in the local NIB of $C_i$.
Update of remote NIB: the action to get the real-time load of servers in the remote domains of $C_i$ by controller state synchronization and to store those loads in the remote NIB of $C_i$.

Synchronize: the action to conduct the update of the local and remote NIBs.

3.2. High synchronization overhead of controllers

LBC [7] is a PS-based controller state synchronization scheme. Several current works are based on LBC, such as consensus routing [15] and consensus algorithms [16].

Using LBC in the SDN network, both the local and remote NIBs are updated and synchronized periodically and simultaneously. We illustrate the impact of LBC on load-balancing performance by the load variation of servers in an interval between two consecutive synchronizations under a specific topology named SimSDN1. Similar to the simulation topology in [7], SimSDN1 is a simple case of Fig. 2. SimSDN1 contains two domains $D_1$ and $D_2$, each of which is composed of one edge switch connecting with one server. Figs. 3, 4, 7 show the load variation of servers when different schemes are used for synchronizing the state among controllers. In those figures, $U_{10}$ and $U_{21}$, respectively, denote the real-time loads of $S_{11}$ and $S_{21}$, $U_{10}'$ and $U_{21}'$, respectively, denote the estimated loads of $S_{11}$ and $S_{21}$, which are stored in $C_1$ and updated by the specific synchronization scheme. For $C_1$, $U_{10}'$ belongs to the local NIB of $C_1$, while $U_{21}'$ belongs to the remote NIB of $C_1$. The synchronization period of LBC is $T$ seconds, so that the controllers’ NIBs are only updated and synchronized at $t_0 + nT$ ($n \in \mathbb{N}$). In Fig. 3, we assume that $S_{11}$ and $S_{21}$ are homogeneous servers, and the loads of $S_{11}$ and $S_{21}$ are 50% and 70% before $t_0$, respectively. Two new flows $f_1$ with the load of 10% and $f_2$ with the load of 20%, respectively, enter SimSDN1 at $t_1$ and $t_2$. Two existing flows $f_3$ with the load of 10% and $f_4$ with the load of 30%, respectively, terminate at $t_3$ and $t_4$. Using LBC, at $t_0 + T$, $U_{10}'$ is updated by pulling $U_{11}$ from the edge switch connected with $S_{11}$, and $U_{21}'$ is updated by synchronizing $U_{21}$ from $C_2$. At any time between $t_0$ and $t_0 + T$, the local and remote NIBs of $C_1$ do not change even if the load of servers in $D_1$ and $D_2$.

![Fig. 2. Deployment of multiple distributed controllers in a large-scale SDN network.](https://www.tarjomano.com/order)
change. $C_2$ works exactly as $C_1$. The states of $C_1$ and $C_2$ are always consistent with their NIBs updated at the last synchronization.

Obviously, by applying LBC, load-balancing performance depends on $T$. If $T = 0$, the real-time state synchronization among controllers enables good load-balancing performance, but it also incurs a large number of synchronizations. Due to the direct proportional relationship between the synchronization overhead of controllers and the number of synchronizations [7], controllers consequently suffer from high synchronization overhead. If $T$ is large (e.g., 16 s), the load variation of servers in one domain cannot be timely updated and synchronized to other controllers in the SDN network. Thus, in the interval between two consecutive synchronizations, controllers make consistent forwarding decisions for new flows based on the out-of-date NIBs, which would degrade load-balancing performance and, even worse, lead to packet loss. For example, new flows are forwarded to $S_{out}$ even if $S_{out}$ reaches 100% load, and some new flows are dropped. This phenomenon could be exacerbated when a large number of new flows enter the network in a short period.

### 3.3. Forwarding loop

In [7], SSLBC is proposed to improve load-balancing performance with the same synchronization overhead of LBC. Compared with the eventual consistency of LBC, SSLBC is an inconsistent PS-based controller state synchronization scheme. In SSLBC, the load of servers in the remote NIB is updated every $T$ seconds by synchronization, while the load of servers in the local NIB is updated in real time. The real-time update of the local NIB mitigates load unbalancing among servers caused by a time gap between two consecutive synchronizations. Fig. 4 shows the load variation of servers under the same network and flow configuration described in Section 3.2 when SSLBC is used as the controller state synchronization scheme. In Fig. 4, for $C_1$, $U_{C_1}$ is strictly consistent with $U_{t_1}$ at $t_1$, $t_2$, and $t_4$ due to the real-time update of the local NIB. For $C_2$, $U_{C_2}$ is also updated at $t_2$ to reflect the real-time value of $U_{C_2}$.

However, the real-time update of the local NIB could incur temporary state inconsistency among controllers. Thus, in the interval between two consecutive synchronizations, different controllers may have different understandings on the current least loaded server in the network, which could cause forwarding loops. In Fig. 4, during the interval between $t_2$ and $t_4$, $C_1$ thinks that $S_{least}$ is $S_{out}$ of the two domains since the inquiry result of $C_1$’s NIB shows $U_{C_1} > U_{C_2}$. However, in $D_2$, $S_{least}$ is recognized as $S_{out}$ of the two domains by $C_2$ because the NIB of $C_2$ shows $U_{C_2} > U_{C_1}$. If a new flow $f_{new}$ enters the network during this interval, $f_{new}$ will be pushed back and forth among the two domains. The loop forms. After a certain period of time, the packets of $f_{new}$ are dropped and packet loss occurs.

In the traditional IP network, the forwarding loop can be eliminated after the NIBs of controllers become consistent again (e.g., after $t_0 + T$ in Fig. 4). This is because packets in the IP network are routed individually, and the new NIB will become effective for all newly arrived packets. However, in the SDN network, once the forwarding rules are installed in the flow tables of switches, flows are strictly forwarded until the rules are removed by the timeout scheme or controllers. The forwarding loops may last tens of seconds, which could cause significant packet loss and communication disruptions.

If an SDN network only contains two domains, this forwarding-loop problem can be easily solved. For example, we can forbid sending $p_{new}$ back to the port where $p_{new}$ comes from. However, the forwarding-loop problem becomes complicated if an SDN network is composed of three or more domains since controllers have no idea which domains $p_{new}$ went through except the domain $p_{new}$ just passed by. Fig. 5 describes the forwarding-loop problem in the SDN network containing three domains. When $f_{new}$ enters the network from $D_1$, $C_1$ searches its current NIB and finds that $S_{2,2}$ is $S_{out}$. Then, $C_1$ configures switches in $D_1$ to forward $f_{new}$ heading to $S_{2,2}$ via the configured path. When $D_2$ receives $f_{new}$ sent from $D_1$, $C_2$ finds that $f_{new}$ does not match any rules in the flow tables of switches in $D_2$. Thus, $C_2$ selects $S_{2,1}$ as $S_{least}$ based on its current NIB. Since $S_{least}$ is different from $S_{out}$, $S_{least}$ is updated to $S_{2,1}$, and $f_{new}$ is directed to $S_{2,1}$ by configuring the related switches in $D_2$. When $D_3$ receives $f_{new}$ sent from $D_2$, $C_3$ acts similarly to $C_1$ and $C_2$, and selects $S_{out}$ based on its current NIB. Unfortunately, $S_{least}$ is selected by $C_3$ as $S_{least}$, so that $C_3$ updates $S_{least}$.
to $S_{1,2}$ and makes the forwarding configuration on switches in $D_3$ to direct $f_{\text{new}}$ heading to $S_{1,2}$. As a result, $f_{\text{new}}$ is sent back to $D_1$, where $f_{\text{new}}$ comes from, and the forwarding loop forms. In this example, even if a packet is forbidden to be sent back to the port where it comes from, $C_3$ also sends $f_{\text{new}}$ back to $C_1$. This is because $C_3$ only knows that $f_{\text{new}}$ comes from $C_2$, but has no idea about other domains that $f_{\text{new}}$ went through before.

4. Load variance-based synchronization

The forwarding loop is caused by the inconsistent information of the least loaded server among controllers, which could lead to significant packet loss and communication disruptions. In this section, we propose two specific LVS-based schemes to eliminate forwarding loops and to achieve good load-balancing performance with low synchronization overhead.

4.1. Least loaded server variation synchronization (LSVS)

LVS is a trigger-based synchronization. The intuition behind LVS are two points: (i) make sure that new flows are forwarded heading to the same least loaded server/domain and (ii) use as few synchronizations as possible to update all controllers with the crucial variance of network status that could lead to forwarding loops. The first point enables loop-free forwarding. The second point trades off load-balancing performance and the synchronization overhead of controllers.

In LSVS, there are two roles for each controller: Active Controller (AC) and Passive Controller (PC). An SDN network only contains one AC, which manages the domain containing the least loaded server $S_{\text{least}}$ selected at the last synchronization. The remaining domains of the SDN network are managed by PCs. In the role of PC, the controller is assumed to update its local NIB in real time by pulling statistics from underlying switches in its local domain. AC not only updates its local NIB in real time but also activates synchronizations to update the current network status to all controllers when the load of $S_{\text{least}}$ (i.e., $U_{\text{least}}$) exceeds the threshold $TH_{S}$. $TH_{S}$ is set as the load of a specific server $S_{\text{thd}}$ (i.e., $U_{\text{thd}}$) plus a fixed load margin $\Delta U$, that is $TH_{S} = U_{\text{thd}} + \Delta U$. The load margin is used to prevent frequent synchronizations when the difference between $U_{\text{least}}$ and $U_{\text{thd}}$ is very small.

LSVS can be explained through the process of controller role exchange mechanism. Specifically, after the last synchronization, one controller becomes the AC because one of its servers is selected as $S_{\text{least}}$ of the entire network, and the remaining controllers are PCs. Each controller is aware of the load variation of servers in its local domain by updating its local NIB in real time. When $U_{\text{least}}$ exceeds $TH_{S}$, AC initiates a synchronization to update the current network status to all the controllers. During this synchronization,
the previous AC changes its role to a PC. If a server in a domain managed by a previous PC becomes the new S\textsubscript{least}, the controller of this domain changes its role from a PC to the new AC. After this synchronization, there is still one AC in the SDN network, and new flows will be forwarded to the new S\textsubscript{least} until the next synchronization occurs. Fig. 6 shows the controller role exchange in LSVS. The controller role exchange mechanism ensures that one SDN network only contains one AC, and all the controllers have the consistent information of the least load server. Compared with PS-based schemes, LSVS conducts effective synchronizations and eliminates the forwarding loop by controller role exchange mechanism. Note that the proposed controller role exchange mechanism can incorporate any signaling protocols.

Fig. 7 shows the load variation of servers under SimSDN1 with the same flow arrival configuration in Section 3.2 when LSVS is used as the controller state synchronization scheme. We assume $\Delta U = 5\%$, TH\textsubscript{S} is set as the load of the second least loaded server in the last synchronization plus $\Delta U$. In Fig. 7, before $t_1$, $C_1$ is AC, $S_{1,1}$ is S\textsubscript{least}, and TH\textsubscript{S} is 75% ($TH_S = U_{1,1}^{C_1} + \Delta U = 70\% + 5\% = 75\%$). At $t_1$, $S_{1,1}$ receives $f_1$, and both $U_{1,1}$ and $U_{2,1}^{C_1}$ increase to 60%. At $t_2$, $S_{1,1}$ receives $f_2$, and both $U_{1,1}$ and $U_{2,1}^{C_1}$ increase to 80%. Hence, a synchronization is initiated by $C_1$ since $U_{2,1}^{C_1}$ is larger than the current TH\textsubscript{S}. During the synchronization, $S_{1,2}$ is selected as the new S\textsubscript{least}. $C_2$ becomes the new AC, $C_1$ changes to a PC, and TH\textsubscript{S} is updated to 85% ($TH_S = U_{1,1}^{C_2} + \Delta U = 80\% + 5\% = 85\%$). $C_1$ and $C_2$ update S\textsubscript{least}, AC, and TH\textsubscript{S} by calculating the three variables using their updated NIBs. At $t_3$, $U_{2,1}$ decreases to 60% when $f_1$ terminates, and $U_{2,1}^{C_1}$ is updated accordingly. At $t_4$, $U_{1,1}$ decreases to 50% when $f_2$ terminates, and $U_{2,1}^{C_1}$ is updated accordingly. The next synchronization will not occur until $U_{2,1}$ exceeds 85%. Thus, if a new flow $f_{\text{new}}$ enters the network from either $D_1$ or $D_2$ during the interval between $t_2$ and $t_4$, $f_{\text{new}}$ is forwarded to $S_{2,1}$ because both $C_1$ and $C_2$ think that $S_{2,1}$ is S\textsubscript{least} in the entire network. The forwarding is loop-free. Since LSVS only initiates necessary synchronizations to eliminate forwarding loops, the synchronization is effective.

To achieve good load-balancing performance, LSVS also uses RLS. As Section 2.3 explains, when RLS is applied, a new flow $f_{\text{new}}$ could be dynamically absorbed by a server S\textsubscript{least}, which is in the domain on the forwarding path to S\textsubscript{least}, only if $U_{\text{least}}$ is less than $U_{\text{least}}$. Specifically, when $f_{\text{new}}$ enters the network from $D_d$ ($1 \leq d \leq N$), $C_d$ selects S\textsubscript{least} based on its current NIB and directs $f_{\text{new}}$ heading to S\textsubscript{least}. During the procedure of $f_{\text{new}}$ forwarding, if $f_{\text{new}}$ traverses $D_e$ ($1 \leq e \leq N$, $e \neq d$) for the first time and the real-time load of S\textsubscript{least} selected by $C_e$ is the least one in $C_e$'s local and remote NIBs, $f_{\text{new}}$ is absorbed by S\textsubscript{least}. In Fig. 7, after $t_4$, $C_2$ is still AC and $S_{2,1}$ is still S\textsubscript{least} since no synchronization occurs. However, in the NIB of $C_1$, $U_{2,1}^{C_1}$ is larger than $U_{1,1}^{C_1}$, that is $U_{2,1}^{C_1} > U_{1,1}^{C_1}$. Thus, if a new flow $f_{\text{new}}$ enters $D_1$ after $t_4$, $C_1$ directs $f_{\text{new}}$ to $S_{1,1}$ rather than to $S_{2,1}$. After a period of time, $U_{2,1}^{C_1}$ exceeds $U_{2,1}^{C_1}$, and then new flows will start to be forwarded to $S_{2,1}$ again. By applying RLS, a controller directs a new flow either to a server, which is in the local domain managed by this controller, or to S\textsubscript{least}, which is selected by all controllers during the last synchronization and will not be changed until the next synchronization occurs. Thus, the combination of LSVS and RLS does not cause any forwarding loops. RLS enables controllers to make forwarding decisions for new flows considering the real-time load variation of servers in those controllers' local domains, which further improves load-balancing performance.

4.2. Least loaded domain variation synchronization (LDVS)

LSVS solves the forwarding-loop problem, but it triggers synchronizations based on the load variation of a single server, which could also overly burden controllers with frequent synchronizations under specific cases. For example, due to the limited processing capacity of a single server, when a large number of new flows enter the network in a short period of time, S\textsubscript{least} could easily exceed the threshold TH\textsubscript{S}, and frequent synchronizations will be activated to select the new S\textsubscript{least}. In addition, if the load of certain servers belonging to different domains is very close to U\textsubscript{least}, a large number of synchronizations will also be initiated to frequently select the new AC. The number of synchronizations can be decreased by using a large $\Delta U$, but load-balancing performance would be inevitably degraded, as shown by simulation results in Section 5. The high synchronization overhead of controllers could increase the packet forwarding latency, reduce the synchronization accuracy, and lead to some inevitable loops [17].

To reduce the synchronization overhead of controllers as well as to maintain good load-balancing performance, we further propose LDVS, the controller state synchroniza-
tion scheme based on the load variation of a domain. In LDVS, domain $D_i$’s load (i.e., $U_{i}^{D}$) is defined by the average load of servers in $D_i$, that is:

$$U_{i}^{D} = \frac{\sum_{j=1}^{M_i} U_{i,j}}{M_i}$$  

(2)

The domain with the least domain load is defined as the least loaded domain $D_{\text{least}}$. The load of $D_{\text{least}}$ is $U_{\text{least}}^{D}$. Using LSVS, an SDN network still contains one AC and multiple PCs. AC controls $D_{\text{least}}$, and PCs control other domains except $D_{\text{least}}$. In the interval between two consecutive synchronizations, all new flows are forwarded heading to $D_{\text{least}}$. To maintain good load-balancing performance, AC further distributes new flows received by $D_{\text{least}}$ to different servers in $D_{\text{least}}$ based on its real-time updated local NIB. The controller state synchronization only occurs when $U_{\text{least}}^{D}$ exceeds the threshold $TH_B$, which is set as the load of a specific domain $D_{\text{ind}}$ (i.e., $U_{\text{ind}}^{D}$) plus a fixed load margin $\Delta U$, that is $TH_B = U_{\text{ind}}^{D} + \Delta U$. Obviously, $D_{\text{least}}$ receives more new flows than $S_{\text{least}}$ does before a synchronization is activated. Thus, the interval between two consecutive synchronizations of LDVS would be longer than that of LSVS. For a certain period, the number of synchronizations using LDVS is reduced, as compared with using LSVS.

We illustrate an example to differentiate LSVS and LDVS under a topology SimSDN2. SimSDN2 is another simple SDN network similar to that in Fig. 2. SimSDN2 contains two domains, and each of the two domains is composed of one switch connecting with three servers. In this example, $TH_B$ is set as the load of the second least loaded server in the last synchronization plus a fixed load margin $\Delta U$, and $\Delta U = 5\%$. The network and flow arrival configuration are listed as follows: at $t_5$, the load of each server is 60% and $C_1$ is AC; at $t_5 + 1$, the existing flows $f_{\text{ext}}$, $f_{\text{ex}}$, with the load of 45%, $f_{\text{ex}}$, with the load of 39%, and $f_{\text{ex}}$ with the load of 42%, respectively, terminate from $S_1$, $S_1$, and $S_1$; the new flows $f_{\text{new}}$ with the load of 9% and $f_{\text{new}}$ with the load of 9%, respectively, enter the network at $t_5 + 2$ and $t_5 + 3$. Table 2 shows the variation of server load and the threshold $TH_B$ when LSVS is used as the controller state synchronization scheme under the above network and flow arrival setup. Specifically, LSVS works as follows: at $t_5 + 1$, the three existing flows $f_{\text{ext}}$, $f_{\text{ex}}$, and $f_{\text{ex}}$ terminate, $U_{11}$, $U_{i2}$, and $U_{i3}$, respectively, decrease to 15%, 21%, and 18%, $TH_B$ changes to 23% ($U_{21} + \Delta U = 18\% + 5\%$). At $t_5 + 2$, when $f_{\text{new}}$ enters $D_1$, it is forwarded to $S_1$ since $S_1$ is $S_{\text{least}}$ in the entire network. Then, $U_{11}$ increases to 24%, and a synchronization is initiated by $C_1$. After the synchronization, $S_{21}$ becomes $S_{\text{least}}$, $C_2$ becomes AC, and $TH_B$ changes to 26% ($U_{12} + \Delta U = 21\% + 5\%$). At $t_5 + 3$, $S_{21}$ receives $f_{\text{new}}$ and $U_{21}$ increases to 27%. Thus, $C_2$ initiates a synchronization, in which $S_{21}$ becomes $S_{\text{least}}$. $C_1$ becomes AC, and $TH_B$ changes to 29% ($U_{11} + \Delta U = 24\% + 5\%$). In the last two seconds, two synchronizations occur when using LSVS. However, LDVS could not incur any synchronizations during the same time interval. In this example, $TH_B$ is set as the load of the second least loaded domain in the last synchronization plus a fixed load margin $\Delta U$ ($\Delta U = 5\%$). Table 3 shows the variation of domain load and the threshold $TH_B$ when LDVS is used as the controller state synchronization scheme under the same network and flow arrival setup as Table 2. Specifically, LDVS works as follows: at $t_5 + 1$, $U_{i1}$ equals 32% ($\left[U_{i1} + U_{i2} + U_{i3}\right]/3 = \left[15\% + 21\% + 60\%\right]/3$), and $U_{i2}$ equals 46% ($\left[U_{i2} + U_{i3}\right]/3 = \left[18\% + 60\% + 60\%\right]/3$). Since $U_{i1}^{D}$ is less than $U_{i2}^{D}$, $D_1$ is $D_{\text{least}}$. $C_1$ is AC, and $TH_B$ is 51% ($U_{i2}^{D} + \Delta U = 46\% + 5\%$). At $t_5 + 2$, $D_1$ receives $f_{\text{new}}$ and $U_{i1}^{D}$ increases to 35% (32% + $9\%/3$). At $t_5 + 3$, since $D_1$ is still $D_{\text{least}}$, $f_{\text{new}}$ is still forwarded to $D_1$ and $U_{i1}^{D}$ increases to 38% (35% + $9\%/3$). By applying LDVS, no synchronization occurs in the interval between $t_5 + 1$ and $t_5 + 3$, and the synchronization will not be initiated until $U_{i1}^{D}$ exceeds $TH_B$. Thus, using the domain load as the trigger variable of synchronization enables the reduction on the synchronization overhead of controllers.

In addition, LDVS could use few synchronization information. In LDVS, the forwarding decision of a new flow $f_{\text{new}}$ includes two steps: (1) $f_{\text{new}}$ is first forwarded to $D_{\text{least}}$ based on the load of domains and (2) $f_{\text{new}}$ is then forwarded to a server in $D_{\text{least}}$ based on the local NIB of $D_{\text{least}}$. Since the two steps do not use the load of servers in the remote domains, LDVS could reduce synchronization information by using two types of NIBs: the domain NIB and the server NIB. The server NIB stores and updates the load of servers in the local domain in real time, but the domain NIB only stores the load of local and remote domains. The load of the local domain is calculated in real time and updated from the server NIB, while the loads of remote domains are updated by the controller state synchronization. Using the domain NIB, a controller only synchronizes its local domain load instead of the load of servers in its local domain to other controllers during the controller state synchronization, which reduces the amount of synchronization information.

5. Simulation

5.1. Simulation setup

In this section, we evaluate the performance of different controller state synchronization schemes for the SDN network. We assume that one SDN contains $N$ domains, and each domain is composed of $M$ servers connected by edge

Table 2

<table>
<thead>
<tr>
<th>Time</th>
<th>State (%)</th>
<th>$U_{11}$</th>
<th>$U_{12}$</th>
<th>$U_{13}$</th>
<th>$U_{21}$</th>
<th>$U_{22}$</th>
<th>$U_{23}$</th>
<th>$TH_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_5 + 1$</td>
<td>15</td>
<td>21</td>
<td>60</td>
<td>18</td>
<td>60</td>
<td>60</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>$t_5 + 2$</td>
<td>24</td>
<td>21</td>
<td>60</td>
<td>18</td>
<td>60</td>
<td>60</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>$t_5 + 3$</td>
<td>24</td>
<td>21</td>
<td>60</td>
<td>27</td>
<td>60</td>
<td>60</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Time</th>
<th>State (%)</th>
<th>$U_{i1}^{D}$</th>
<th>$U_{i2}^{D}$</th>
<th>$TH_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_5 + 1$</td>
<td>32</td>
<td>46</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>$t_5 + 2$</td>
<td>35</td>
<td>46</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>$t_5 + 3$</td>
<td>38</td>
<td>46</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

Please cite this article in press as: Z. Guo et al., Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Netw. (2014), http://dx.doi.org/10.1016/j.comnet.2013.12.004
switches. Each server has approximately the same processing ability. The controller state synchronization is achieved by a method similar to [18]. Each controller maintains a communication channel with every other controller. A controller sends its local information to other controllers through these communication channels during synchronizations. We conducted the simulation experiments in two modes: packet mode and flow mode. In the simulation, we evaluate three performance metrics: the packet loss rate, the load-balancing performance, and the synchronization overhead of controllers.

We simulate three specific cases for the two modes: Case 1 for the packet model, Cases 2 and 3 for the flow model. In the packet-model simulation, we use Mininet [19] to build the SDN network and use POX [20] to simulate multiple controllers. With the packet-mode simulation, we are able to obtain the three performance metrics. However, because of its long simulation time, only relatively small SDN networks can be evaluated in a reasonable period. In Case 1, similar to the simulation topology in [7], the SDN network consists of two domains, and each domain contains two switches connecting with three servers. The traffic load with the deterministic amount (i.e., 10% of a load server) randomly ingresses one of the two domains. For Case 1, we ran our simulation 100 times with a duration of 240 s for each scheme.

Flow-mode simulation captures flow arrival and termination events without injecting packets as in the packet mode. Specifically, in flow-mode simulation, a realistic workload is used. New flows randomly enter different domains, and the arrival rate and durations of each flow are driven by a Poisson distribution with the parameter \( \lambda \). Due to its coarser granularity, flow-mode simulation can be used to evaluate the load-balancing performance and the synchronization overhead of controllers for large-scale SDN networks with high traffic loads. In Case 2, we expand the network to six domains, each of which contains five switches connecting with two servers. In Case 3, we simulate a more complex scenario: the SDN network still contains six domains, and each domain contains five switches. However, in each domain, one switch connects with ten servers. Compared with Case 2, the number of traffic flows in Case 3 increases as the network scales. \( \lambda \) equals 80 and 400 for Cases 2 and 3, respectively. In flow-mode simulation, we use two types of flows: mice flows and elephant flows. For Cases 2 and 3, 80% of the flows are mice flows, whose sizes randomly vary in the range of 0.01% and 0.1% of a server load, and 20% of the flows are elephant flows, whose sizes randomly vary in the range of 0.1% and 1% of a server load. For Cases 2 and 3, we ran our simulation 100 times with a duration of 24,000 s for each scheme. Table 4 shows the parameters of the three cases in detail.

For PS-based schemes, we evaluate the impact of different synchronization periods on the three performance metrics. In Section 5.2.1, five different synchronization periods are used. In Sections 5.2.2 and 5.2.3, three synchronization periods are used. For LVS-based schemes, we first use one specific threshold for each scheme to compare with PS-based schemes, and then evaluate the impact of different thresholds on performance metrics. In Sections 5.2.1, 5.2.2, and 5.2.3, \( TH_1 \) is set as the load of the second least loaded server in the last synchronization plus a fixed load margin \( \Delta U \), and \( TH_2 \) is set as the load of the second least loaded domain in the last synchronization plus a fixed load margin \( \Delta U \). In Section 5.2.4, six specific servers, three specific domains, and six load margins are used. In our simulations, we use star topology for the intra-domain connectivity and ring topology for the inter-domain connectivity. The simulations we ran on the line and ring topologies for the intra-domain connectivity exhibit very similar behavior.

5.2. Simulation results

First, we validate our simulation environment by matching the results of LBC and SSLBC reported in [7] under the similar network and workload configuration. Then, we compare the proposed LVS-based schemes against PS-based schemes with the performance metrics under deterministic workload and Poisson distribution workload. We also evaluate the impact of different thresholds on the performance of LVS-based schemes. For ease of reading, PS-based schemes plus synchronization periods are short for PS-based schemes with synchronization periods. For example, LBC-4s is short for LBC with a synchronization period of 4 seconds.

5.2.1. Case 1

In Case 1, we evaluate the ideal scheme (i.e., LBC-0s), two LVS-based schemes (i.e., LSVS and LDVS) and two PS-based schemes with different synchronization periods (i.e., LBC-1s, LBC-2s, LBC-4s, LBC-16s, and SSLBC-16s). Table 5, Fig. 8(a), and (b), respectively, show the average number of synchronizations per minute, the packet loss rate, and the load-balancing performance of these schemes under Case 1. In Fig. 8(a), SSLBC causes packet loss and the packet loss rate increases as the synchronization period increases. In contrast, LSVS and LDVS do not cause any packet loss since they eliminate forwarding loop.

The load-balancing performance is evaluated by server Root Mean Squared Error (RMSE) of all the servers in the entire network [7]. A smaller RMSE means a better performance. If all the servers have the same load, RMSE will be 0. Fig. 8(b) describes the server RMSE for different controller synchronization schemes with different synchronization periods in the form of box-plot. In Fig. 8(b), the box represents the center half of the data and the red line represents the median data. The whiskers include 95% data while

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Parameters of simulation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Number of domain (N)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

2 For interpretation of color in Fig. 8(b), Fig. 9, Fig. 10, Fig. 11, and Fig. 12, the reader is referred to the web version of this article.
the 5% outliers are drawn as separate red crosses. The ideal theoretical performance is shown by LBC-0s in which each controller distributes new flows based on the real-time updated local and remote NIBs. The load-balancing performance using PS-based schemes is represented by LBC-1s, LBC-2s, LBC-4s, LBC-16s, and SSLBC-16s. LBC-1s performs better than other schemes, but it synchronizes 60 times per minute. LBC-4s synchronizes the approximately same times as LSVS and LDVS do, but it performs much worse than those two schemes. The server RMSE of LBC-16s is much worse than other schemes because the long synchronization period significantly degrades the load-balancing performance. The server RMSE of SSLBC-16s is close to the ideal one, but it is not practical due to the packet loss caused by forwarding loops. LSVS and LDVS achieve the mean RMSE comparable to the ideal one (LBC-1s), but they only initiate 14 and 11 synchronizations per minute on average, respectively. Since the synchronization overhead of controllers is proportional to the number of synchronizations, LSVS and LDVS achieve good load-balancing performance with much less synchronization overhead, as compared with LBC-1s. Thus, under the deterministic workload, the proposed LSVS and LDVS eliminate the packet loss caused by forwarding loops and achieves the trade-off between the load-balancing performance and the synchronization overhead of controllers.

5.2.2. Case 2

In Case 2, we evaluate LSVS, LDVS, LBC-1s, LBC-2s, and LBC-4s. The average number of synchronizations per minute and the load-balancing performance of these schemes under Case 2 are shown in Table 5 and Fig. 9, respectively. In Fig. 9, the load-balancing performance of LBC improves, as compared with Case 1. There are three reasons. First, Case 2 simulates much longer than Case 1, which enables the collection of more precise statistical characteristics. Second, the workload of Case 2 varies more frequently than Case 1, which makes a more balanced load on servers possible. Third, the SDN network of Case 2 contains many more servers than that of Case 1. Thus, more servers can be candidates for $S_{\text{least}}$.

However, LVS-based schemes perform much better than PS-based schemes. The server RMSEs of LSVS and LDVS is very small, as compared with the server RMSE of LBC in Case 2. LSVS and LDVS update the local NIB in real time and update the remote NIB only when the load of $S_{\text{least}}/D_{\text{least}}$ exceeds $THS/THD$. The NIB update scheme enables each controller to make forwarding decisions for new flows based on the real-time load status of the local domain. In addition, as the number of servers in the network increases, the function of RLS becomes more effective. It is worth noting that in this case, even the performance of LBC-1s is not comparable with the performance of LSVS and LDVS. Thus, under a realistic Poisson variance-based synchronization, Comput. Netw. (2014), http://dx.doi.org/10.1016/j.comnet.2013.12.004
distribution workload, LSVS and LDVS achieve good load-balancing performance with low synchronization overhead.

5.2.3. Case 3

In Case 3, we evaluate LSVS, LDVS, LBC-1s, LBC-2s, and LBC-4s. The average number of synchronizations per minute and the load-balancing performance of these schemes under Case 3 are shown in Table 5 and Fig. 10, respectively. Compared with Case 2, the load-balancing performance of LBC degrades significantly because of the use of out-of-date NIBs to make consistent forwarding decisions for new flows. In the interval between two consecutive synchronizations, new flows are forwarded to $S_{\text{least}}$. However, the network status varies over time. For example, a server’s real-time load could be much less than $U_{\text{least}}$ if some flows terminate from this server. $S_{\text{least}}$ could reach 100% load after it receives a large number of new flows. Without considering those critical changes, consistently forwarding new flows to $S_{\text{least}}$ significantly degrades the load-balancing performance and, even worse, leads to packet loss. Compared with Case 2, the number of traffic flows in Case 3 increases as the network scales. Thus, the problem of using out-of-date NIBs is exacerbated.

In contrast, both LSVS and LDVS outperform LBC-1s, LBC-2s, and LBC-4s on server RMSE. As the number of servers increases, the advantage of RLS becomes more obvious. During the procedure of flow forwarding, more flows are absorbed by the current least loaded server, which is selected by controllers using those controllers’ real-time updated local NIB. As a result, LSVS and LDVS not only prevent a server from being fully loaded in a short period, but also do not significantly increase the number of synchronizations. Therefore, LSVS and LDVS achieve good load-balancing performance with low synchronization overhead in the large-scale SDN network with high traffic loads.

5.2.4. The impact of threshold

In LVS-based schemes, the thresholds $TH_S/TH_D$ are composed of two components: $S_{\text{thd}}$, load margin $D_{\text{thd}}$, load margin $U_{\text{thd}}$, and load margin $\Delta U$. The selection of components impacts the performance of schemes. In the following subsections, we evaluate the impact of the two components on the load-balancing performance and the average number of synchronizations under Case 2.

First, we set $\Delta U = 0$ and use different specific servers/domains for the two proposed LVS-based schemes. For LSVS, we use the second, sixth, tenth, twentieth, and fiftieth least loaded server (i.e., $S^2_{\text{least}}$, $S^6_{\text{least}}$, $S^{10}_{\text{least}}$, $S^{20}_{\text{least}}$, and $S^{50}_{\text{least}}$) as $S_{\text{thd}}$. For LDVS, we use the second, third, and fourth least loaded domain (i.e., $D^2_{\text{least}}$, $D^3_{\text{least}}$, and $D^{4}_{\text{least}}$) as $D_{\text{thd}}$. Obviously, a server/domain with a larger superscript has a larger load. Tables 6 and 7, respectively, show the average number of synchronizations per minute for LSVS/LDVS using different $S_{\text{thd}}/D_{\text{thd}}$ under Case 2. Fig. 11 shows the load-balancing performance of LSVS/LDVS using different $S_{\text{thd}}/D_{\text{thd}}$ under Case 2. For LVS, when $S_{\text{thd}}$ varies from $S^2_{\text{least}}$ to $S^{50}_{\text{least}}$, the load-balancing performance degrades very slightly and the average number of synchronizations per minute decreases slightly, as shown in Fig. 11(a). The reason is by applying LSVS the load difference among those servers is very small. However, the load of $S^0_{\text{least}}$ is much larger than the load of $S^{10}_{\text{least}}$. If $S^0_{\text{least}}$ is used in $S_{\text{thd}}$, $S_{\text{least}}$ receives more flows before it reaches $TH_S$, as compared with using $S^{10}_{\text{least}}$. Thus, when using $S^0_{\text{least}}$ in $S_{\text{thd}}$, the average number of synchronizations per minute decreases significantly, which degrades the load-balancing performance degrades as compared with using $S^{10}_{\text{least}}$. The selection of using the specific server in $TH_S$ trades off the load-balancing performance and the synchronization overhead of controllers. For LDVS, as the load of $D_{\text{thd}}$ increases, the load-balancing performance varies very slightly and the synchronization overhead of controllers decreases very slightly. This is because by applying LDVS the load difference among domains is very small.

Second, we set $S_{\text{thd}} = S^2_{\text{least}}$ and $D_{\text{thd}} = D^2_{\text{least}}$ and use six different load margins for the two proposed LVS-based schemes. Table 8 and Fig. 12, respectively, show the average number of synchronizations per minute for LSVS/LDVS and the load-balancing performance of LSVS/LDVS using different load margins under Case 2. For both LSVS and LDVS, as $\Delta U$ increases, the number of synchronizations per minute decreases, and the load-balancing performance degrades. For the same load margin, LSVS outperforms LDVS.

Table 6

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$S_{\text{thd}}$</th>
<th>$U^2_{\text{thd}}$</th>
<th>$U^6_{\text{thd}}$</th>
<th>$U^{10}_{\text{thd}}$</th>
<th>$U^{20}_{\text{thd}}$</th>
<th>$U^{50}_{\text{thd}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSVS</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>18</td>
<td>14</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 7

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$D_{\text{thd}}$</th>
<th>$D^2_{\text{thd}}$</th>
<th>$D^3_{\text{thd}}$</th>
<th>$D^4_{\text{thd}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDVS</td>
<td>18</td>
<td>18</td>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
LDVS in the metric of server RMSE, but it requires more synchronizations. The selection of $\Delta U$ depends on the trade-off between the expected load-balancing performance and the synchronization overhead of controllers.

6. Discussion

In this section, we discuss a practical issue in the LVS-based schemes implementation.

6.1. Forwarding loop in the procedure of synchronization

LVS-based schemes ensure that all controllers have consistent information about the least loaded server/domain during the interval between two consecutive synchronizations. However, in the procedure of synchronization, some undesirable situations could occur. For example, during the procedure of synchronization, the latest network status is propagated to all controllers in the SDN network. Due to the network congestion and controllers’ different locations, different controllers could update their NIBs at different times. Thus, some controllers use the new NIB to make forwarding decisions for new flows, while others still use the old NIB to decide where to forward new flows. The inconsistent network status of the controllers could cause the forwarding loop. This forwarding-loop problem can be solved by applying per-packet consistency [17]. In per-packet consistency, the old NIB and new NIB are separately stored in two tables. Each table is specified with a unique configuration version number. At ingress switches, packets are stamped with their configuration version numbers. When a packet enters a domain, the version number of this packet is tested to select which table to use. The per-packet consistency guarantees that each packet flowing through the network is processed according to a single network status - either the old NIB before to the update, or the new NIB after the update, but not a mixture of the old and new NIBs.

![Figure 11](https://www.tarjomano.com/order)

**Fig. 11.** Load-balancing performance of LVS schemes using different $S_{\text{thd}}/D_{\text{thd}}$ in the threshold under Case 2.

![Figure 12](https://www.tarjomano.com/order)

**Fig. 12.** Load-balancing performance of LVS schemes using different load margins in the threshold under Case 2.

### Table 8
The average number of synchronizations per minute for LSVS/LDVS using different load margins in the threshold under Case 2.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>$\Delta U$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>LSVS</td>
<td>22</td>
</tr>
<tr>
<td>LDVS</td>
<td>18</td>
</tr>
</tbody>
</table>

Thus, per-packet consistency prevents the forwarding loop.

7. Conclusion

In this paper, we investigate the controller state synchronization issue under the application of load balancing in the SDN network. We propose two LVS-based schemes to solve the forwarding-loop problem and reduce the synchronization overhead of controllers. LVS effects the controller state synchronizations only when the load of a specific server or domain exceeds a certain threshold. Simulation results show our proposed two specific LVS-based schemes, LSVS and LDVS, achieve loop-free forwarding and good load-balancing performance with much less synchronization overhead of controllers, as compared with existing PS-based schemes. In our future work, we plan to evaluate two proposed LVS-based schemes in a real testbed. We also plan to extend the LVS-based schemes to further reduce the synchronization overhead of controllers while maintaining reasonable load-balancing performance.

References


Zehua Guo is currently a Ph.D. candidate at School of Electronics and Information of Northwestern Polytechnical University. He has been worked as a Visiting Scholar in Department of Electrical and Computer Engineering at New York University Polytechnic School of Engineering since 2011. He received his M.S. from Xidian University, in 2010, and his B.S. from Northwestern Polytechnical University, in 2007. His research interests include Data Center Network, Software Defined Networking, Network Resilience, and Signal Analysis.

Mu Su is Software Developer at Cisco Systems. He was a Graduate Assistant at High Speed Network Lab of New York University Polytechnic School of Engineering. His research interests include SDN, network security, and network debugging. He received his Master of Science degree at Department of Electrical and Computer Engineering of New York University Polytechnic School of Engineering in 2013 and his Bachelor of Science degree at Beijing University of Technology in 2010.

Yang Xu (IEEE Member, 2005) is a Research Assistant Professor in the Department of Electrical and Computer Engineering of New York University Polytechnic School of Engineering, where his research interests include Data Center Network, Network on Chip, and High Speed Network Security. From 2007–2008, he was a Visiting Assistant Professor in NYU-Poly. Prior to that, he completed a Ph.D. in Computer Science and Technology from Tsinghua University, China in 2007. He received the Master of Science degree in Computer Science and Technology from Tsinghua University in 2003 and Bachelor of Engineering degree from Beijing University of Posts and Telecommunications in 2001.
Zhemin Duan is a Professor at School of Electronics and Information of Northwestern Polytechnical University. His research interests include Signal Processing, IC Analysis and Design, Networking, Data Center Network, and Software-Defined Networking. He has published more than 80 journal and conference papers. He received National Teaching Master Award in 2011. He received his M.S. from Xi’an Jiaotong University and B.S. from Northwestern Polytechnical University.

Luo Wang is Software Developer at Barracuda Networks. He was a Graduate Assistant at High Speed Network Lab of New York University Polytechnic School of Engineering. His research interests include SDNs, network security, and network debugging. He received his Master of Science degree at Department of Electrical and Computer Engineering of New York University Polytechnic School of Engineering in 2013 and his Bachelor of Science degree at Beijing University of Posts and Telecommunications in 2011.

Shufeng Hui is a Graduate Assistant at High Speed Network Lab of New York University Polytechnic School of Engineering. His research interests include SDNs and big data management. He is now pursuing his Master of Science degree at Department of Computer Science and Engineering of Polytechnic Institute of New York University. He received his Bachelor of Engineering degree from Northeastern University Neusoft Institute of Information, China.

H. Jonathan Chao is Department Head and Professor of Electrical and Computer Engineering at New York University Polytechnic School of Engineering. He is a Fellow of the IEEE for his contributions to the architecture and application of VLSI circuits in high-speed packet networks. He holds 46 patents with 11 pending and has published more than 200 journal and conference papers. He has also served as a consultant for various companies, such as Huawei, Lucent, NEC, and Telcordia, and an expert witness for several patent litigation cases. His research interests include high-speed networking, data center network designs, terabit switches/routers, network security, network on chip, and medical devices. He received his B.S. and M.S. degrees in electrical engineering from National Chiao Tung University, Taiwan, and his PhD degree in electrical engineering from Ohio State University.