CSRS: A Cross-domain Source Routing Scheme for Multi-domain Software-defined Networks

Wen Zhang*, Peilin Hong*, Long Yao*, Jianfei Li†, Dan Ni*
* CAS Key Lab. Wireless-Optical Communications, University of Science and Technology of China,
Hefei, Anhui, 230027, P. R. China
† Huawei Technology Corporation Limited, Beijing, 100085, P. R. China
zw1991@mail.ustc.edu.cn, plhong@ustc.edu.cn, fre.lijianfei@huawei.com, {yaolong, nidan}@mail.ustc.edu.cn

***Abstract*—SDN is a novel network architecture that separates the control plane from the data plane. It usually utilizes a centralized controller to manage all the switches in the network. With the expanding of network scale, both the complexity of computing routes and the amount of messages exchanged between controller and switches are increasing sharply. The limited performance of a single controller will cause congestion in the control plane. Some schemes have been proposed to solve this problem by dividing the network into multiple domains. In this paper, we propose a cross-domain source routing scheme for multi-domain network combining the thinking of source routing with pre-routing. The cross-domain flow set-up requests are generated only in the source domain and the destination domain. Each controller just needs to bear the load that related to the local traffic. The simulation results show that the scheme can reduce the load of controllers comparing with other schemes.**

I. Introduction

SDN [1] is a novel network paradigm which separates control plane from data plane. Openflow [2] which consists of centralized controller and openflow switches(OFS) is a famous implementation of SDN. However, in current OpenFLow network, the centralized architecture with a single controller is not an effective scheme for the large-scale network. With the increased network size, the limited performance of a single controller will become the network bottleneck, and block the network scalability. In [3], Fernandez mentioned that the relationship between the amount of switches and control messages was linear. When the network load is out of the power of the controller, a large amount of flow-requests will be denied.

In view of the network scalability problem, recent research works have proposed a number of schemes to solve this problem. The proposed schemes can be classified into two categories: (1) The switches have part of control functions to offload the controller, and (2) distributing the control plane across multiple controllers. DevoFlow [4] and DIFINE [5] have been proposed to solve the scalability problem which falls into the first category. To reduce the number of flow-request messages in control plane, the switches have some intelligence, and they can complete a part of control functions without invoking controller. Both of these proposals not only require the switches to own data forwarding function but also need switches own certain intelligence.

On the other hand, Kandoo [6], Onix [7], and HyperFlow [8] have been made to physically distribute the control plane. They improved the scalability of the OpenFlow network through multiple controllers. Each of them distributes controllers states differently. Kandoo is a two hierarchy architecture consisting of a root controller and multiple local controllers. The local controllers have no knowledge about the global state, and they just maintain local network state. The root controller takes actions when the events based on the global network state happened. Onix tries to manage the global network by use of a distributed hash table while HyperFlow manages the whole network through a distributed file system.

In this paper, we present a cross-domain source routing scheme named CSRS based on the source routing and pre-routing to alleviate the controllers load. A large-scale network is partitioned into multiple domains. Each controller is in charge of a SDN domain, and executes pre-routing process to build the paths between the border switches. The aggregated network information will be shared between the adjacent domains, thus each controllers can get the global aggregated network state to calculate the cross-domain path. When a new cross-domain flow arrives, a source routing based cross-domain path will be calculated by the controller of source domain. By replacing the destination address into the address of next border switch in packet header, the flow will arrive at the destination domain. By saving intermediate domain routing requests, the scheme can alleviate the controllers load compared with other cross-domain routing scheme. Finally, we evaluate the performance by use of Mininet [9] and POX [10] controller.

The rest of this paper is organized as follows. First, Sec. II analyzes the OpenFlow architecture and related work. Then, Sec. III details the proposed scheme named CSRS. The different use cases we considered and their evaluation are tackled in Sec. IV. Finally, Sec. V concludes this paper.

II. Related Works

In this section, we will introduce OpenFlow architecture briefly, and related work. In OpenFlow network, OFS(Openflow Switch) just performs look-up and forwarding functions without any control functions. OFS are connected to the controller southbound via secure channel, as Fig.1 depicts. And controller is in charge of the whole network operation and scheduling.

978-1-4673-6305-1/15/$31.00 ©2015 IEEE

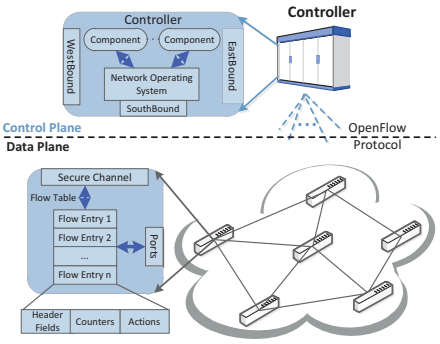


Fig. 1. OpenFlow Architecture

Each switch maintains one or multiple flow tables which are the basis of the data forwarding. Flow Table is installed by controller or configured manually. Each flow table contains multiple flow table entries which are used to match flow and take corresponding actions. Each flow table entry consists of three parts: Header fields which defines the incoming flow, counters for statistical objective, actions applied to the matching flows. When the switch receives a packet, it will look up the flow table to find matching flow table entry. If there exit a matching entry, it will forward the packet according to the actions in the entry. Otherwise, it will send a packet-in message which contains the packet information to the controller. When a controller receives a packet-in message, it will parse the packet and determine the path for this flow. Then it will install entries in the corresponding switches.

Due to the limited performance of the single controller, one controller can't undertake the network load when network scale reaches a certain size. Thus, the network scalability will be constrained by the controller performance. In order to improve the network scalability, several schemes that distribute the control plane across multiple controllers have been proposed. Distributed architecture includes two typical paradigms: (1) deploying multiple controllers managed by a root controller, and (2) full distributed architecture that multiple controllers cooperate with each other. A hierarchical architecture RMOF [11] proposed by Xuan Thien Phan et al. and Kandoo are belonged to the first paradigm. In RMOF architecture, network is partitioned into multiple domains, and each domain is managed by a centralized controller. All the local controllers are also managed by a root controller which is used to determine the global routing. Local controller should send the information about the cost and weight between any cross-domain switches to the root controller. By means of this method, root controller can calculate the optimal global routing. Hierarchical architecture can respond to the network events quickly. However, the root controller will become network bottleneck, and the network reliability can't be guaranteed in hierarchical architecture. The scheme that Hilmi E. Egilmez et al. proposed [12] is a full distributed architecture. In this scheme, network consists of multiple domains which are managed by a local controller. Controllers exchange aggregated network information with each other so that each controller can take forwarding decisions. The scheme named DCPP proposed by the Md. Faizul Bari et al. [13] based on the full distributed architecture is similar to our scheme. In [12] and [13], cross-domain flows will rely on the controllers in each domain they will pass through. Thus, the cross-domain flows will burden the controller load. It isn't beneficial for the network scalability.

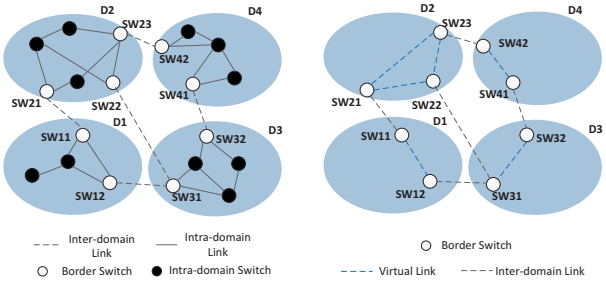
Based on the above analysis, the aforementioned multi-domain proposals just focus on the cross-domain routing realization. However, they ignore the controllers load that cross-domain flows generated in the intermediate domains. Faced with that problem, we propose a cross-domain source routing scheme for multi-domain software-defined network (CSRS) based on the thinking of source routing and pre-routing. The cross domain packets carrying the necessary address information will be transmitted via the paths that built in the pre-routing process. Thus, the cross-domain flows won't generate flow-setup requests in the intermediate domains. The controllers load in intermediate domains can be reduced.

III. Cross-domain Source Routing Scheme

A. Overview of the cross-domain source routing

For large-scale network, the whole network is partitioned into multiple domains as Fig.2(a) depicts in terms of the algorithm based on the controller capacity proposed in [14]. Each domain is managed by a centralized controller. Based on the information shared with adjacent domains, each controller can get the global aggregated topology as Fig.2(b) shows as well as IP distribution. On the basis of local detail topology and global aggregated topology, local controller in the source domain can determine the optimal routing from the source switch to the destination domain. However, the existing cross-domain routing schemes such as [12] and [13] will involve the controllers in each domain they pass through. Thus, the cross-domain flows would increase the controllers load and are not beneficial for the network scalability.

In view of the traditional cross-domain routing scheme, we proposed a scheme based on the thinking of source routing and pre-routing for decreasing controllers load. In our routing scheme, we perform pre-routing process timely according to the link load to establish paths between border switches in each domain. For the cross-domain flows, the controller determines the path consisting of the full route in the source domain and the inter-domain routing which consist of the border switches in other domains. The addresses of border switches that cross-domain flows would pass through will be written into the IP header option. When the packet arrive at the border switch in the intermediate domain, the destination address will be replaced by the next border switch address recorded in the IP header option. The paths that built in the pre-routing process will be leveraged to transmit cross-domain flows. When the data arrive at the destination domain, the data will be sent to the controller to get the final path.



(a) Multi-domain network (b) Global aggregated network

Fig. 2. Partition and aggregation for the whole network

B. Cross-domain source routing scheme description

In the multi-domain network, switch is classified into boarder switch denoted filled dots and intra-domain switch denoted unfilled dots as Fig.2(a) depicts. There are two types of link which are inter-domain, and intra-domain links. Adjacent domains are connected by the inter-domain links. In order to support our scheme, we firstly modify the border switches. The modified switch can recompose the packet header fields.

The entire routing process consists of three parts including the information exchanging, pre-routing process and routing data flows.

1) Information Exchanging: The information shared among adjacent domains is the aggregated topology and IP prefix information. The global aggregated topology includes the local aggregated topology and aggregated topology received from neighbor domains. And it is represented by the inter-domain links between adjacent domains and the virtual links between the border switches in the same domain as Fig.2(b) shows. The weight of virtual links can be gotten by executing existing routing algorithm such as shortest algorithm in each domain. Combined interactive information with local network information, the local controller can get the global network state information. Based on the global network state information, controller can handle cross-domain flow set-up requests. Table I is one entry that represents the inter-domain link or virtual link. The aggregated topology information consists of multiple entries.

Table I is an illustration of the global network information that controller maintained and exchanged.

TABLE I
INTER-DOMAIN LINK AND VIRTUAL LINK ENTRY

Seq	Domain_ID1	SW1_ID	Domain_ID2	SW2_ID	Weight

Each table including the following fields:

- Seq: The tag that mark the information to avoid the repetitive and invalid link information;
- Domain_ID1: The ID of the domain which contains switch SW1;
- SW1_ID: ID of the border switch SW1;
- Domain_ID2: The ID of the domain which contains switch SW2;
- SW2_ID: ID of the border switch SW2;
- Weight: The weight of path between the border switches SW1 and SW2.

In each entry, if Domain_ID1 doesn't equal to Domain_ID2, this entry represents the inter-domain link. Otherwise it indicates the virtual link.

TABLE II
IP PREFIX INFORMATION ENTRY

Seq	Domain_ID	IP_Prefix	...	IP_Prefix

The Table II indicates that IP distribution in one domain. The IP prefix information is composed by these entries. These entries represent the IP address distribution in the whole network.

Each entry includes these fields:

- Seq: The tag that mark the information to avoid the repetitive and invalid IP information;
- Domain_ID: ID of domain which contains these IP address;
- IP_prefix: IP distribution in this domain.

The controller can obtain the IP distribution in the whole network through the interactive information, and locate the packet destination. When the controller gets the position of the destination, it can make optimal routing from source switch to the destination domain based on the aggregated topology.

2) Pre-routing: In this section, we will explain the detailed pre-routing process. During the pre-routing process, we establish optimal paths between any border switches in each domain and install the corresponding entries in the border switches at the end of inter-domain link. For simplicity, we use count as metric. The optimal routing paths between the border switches are the shortest paths.

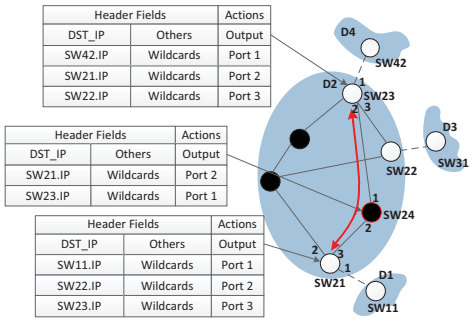


Fig. 3. Entry Format of Pre-routing in D2

In the multi-domain network, the pre-routing process will be executed in each domain. Such as in the domain D2, the controller will build optimal paths between any border switches in this domain. As Fig.3 shows, we build an optimal path between border switch SW21 and SW23, all the switches located in this path have the entries which destination field is SW23 and SW21, and other fields are wildcards. In Fig.3, we list the entries format in some nodes. For example, the

switch SW23 have three entries, and the destination in each entry is SW42, SW22, SW21, and other fields are wildcard. The switch SW24 on the path between SW21 and SW23 has two entries, and the destination in each entry is the address of SW23 and SW21. Entries in these nodes consist of the pre-routing paths. After completing the pre-routing process, we establish the paths between any border switches in each domain. The data flows that pass through domain D2 can utilize the paths built by pre-routing process, so the flows do not need to interact with controller.

The controller must perform pre-routing process according to the network load. Once some paths are failure or congested, the controller must rebuild new paths to replace these invalid paths. After new paths are established, the controller must advertise the routing information.

Our scheme doesn't focus on the specific path selection algorithm, thus we can adapt an existing routing algorithm to build paths between border switches. Based on these paths, we propose a mechanism to improve the SDN scalability by easing controllers load.

3) Routing data flows: Faced with the problem that we analyse above, we can handle the problem by means of the thinking of the loose source routing. Loose source routing protocol(LSR) defined an option in IP header called Loose Source and Record Route Option. This option listed a set of intermediate address that flows must visit before reaching the destination.

Routing process:

After interaction between any neighbor domains and pre-routing process, the controller obtains IP distribution and a global aggregated network which consists of local detailed topology and aggregated topology about other domains.

When the switch receives a packet, it will look up flow table. If there is no matching entry, it will send a packet-in message to the controller. Otherwise, it will take actions according to the entry defined. When the controller receives a flow set-up request, the controller will adopt intra-domain routing if the packet source domain and destination domain are the same. Otherwise, the controller will adopt cross-domain source routing scheme.

As Fig.4 depicts, the source host in the domain D1 sends data to the destination host in the domain D4. Owing to the non-match entries in the source switch, the source switch sends the packet-in message to the controller1. After packet is parsed, the IP prefix of the packet destination is matched in the domain D4. The controller will determine the optimal routing from the source switch to the destination domain. Then the controller installs corresponding entries from the source switch to the border switch which will route the data out of the source domain. The entries in this border switch will be different from other switches. It will take actions as following: Modify the destination, Load the border switch addresses and destination address in the LSR field, and Forwarding.

In Fig.4, the data from source to the destination domain D4 will pass the border switches SW11, SW21, SW23, SW42 illustrated as red unfilled dots in Fig.4. The whole routing process as following:

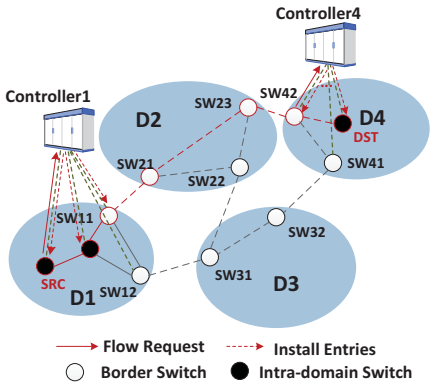


Fig. 4. Cross-Domain Data Transmission

1) The controller installs corresponding entries to route the data to the border switch SW11. When the data matches the entry in SW11, the switch takes actions as following: Modify the destination address with the next hop border switch address SW21, and all the left border switch address and destination address will be reloaded in the LSR field. For example, SW23, SW42, and destination DST address will be recorded in LSR field.
2) When the data arrives at SW21, the switch finds that packet destination address is itself address, then it will read the next border switch address from the LSR field and replace the IP destination into this address. The data will be transmitted according to the matched entries which has been installed during the pre-routing process. When the data is routed to the SW23, SW23 takes the same actions as SW21.
3) When the data arrives at the border switch SW42 in the destination domain, the switch replaces the IP destination into the destination host address, and sends packet-in message to the controller due to lack of corresponding entry. The controller verifies that flow is cross-domain flow in terms of the port that packet comes from which connects to another switch in the neighbor domain. The controller calculates the path for this cross-domain flow from this switch to the destination host and installs corresponding entries.
4) The data will be routed to the destination host along the path in the destination domain. Then the entire global routing is accomplished.

For the whole routing process, the data flows just need to interact with controllers in the source domain and destination domain. In the intermediate domain, the data flows can be routed along the paths established during the pre-routing process, so the data flows do not burden the controllers in the intermediate domains. Through above processes, we can relieve the load in control plane to improve the network scalability.

C. Discussion

Our scheme is mainly designed for the multi-domain SDN network. We adopt the traditional routing algorithm for intra-domain routing. For the inter-domain routing, we alse use the traditional algorithm to select path. Based on this path, we leverage the pre-routing method and the address substitution strategy to avoid the interaction between controllers and cross-domain flows in the intermediate domains.

In our scheme, the cross-domain flows just need to get the controller service in the source and destination domain. The source domain controller determines the global routing, and destination domain controller mainly calculates the path between last border switch and destination host. The controllers in the intermediate domain don't need to afford any service for the cross-domain flows. Cross-domain flows are routed by the pre-established paths in the intermediate domains.

IV. SIMULATION AND RESULTS

A. Simulation Setup

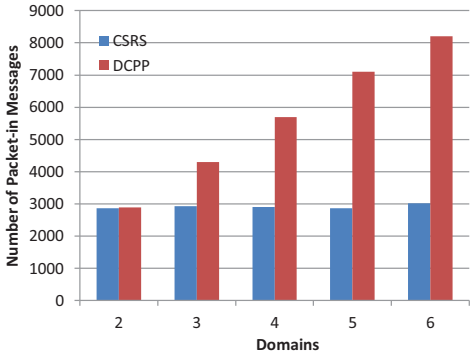
To verify the performance of our scheme, we simulated our scheme by use of Mininet and POX controller. Mininet is a network simulator for simulating SDN scenario in the Linux environment. The components we developed in the Mininet can be migrated to the real network without modified. It is an effective tool to make simulation for SDN.

In our simulation, we write a script based on Python to generate flows that we need. We randomly select the node pairs as source and destination, and transmit data between source and destination. In our simulation, the network consists of multiple domains, and each domain contains 15-25 nodes. The topology is generated by the GT-ITM tool. We import the topology by use of the API provided by Mininet. Then controller starts pre-routing process. A large flows will be generated between different domains and the total number of packet-in messages will be counted. We set two different scenarios, and make comparison between our scheme and correlation scheme DCPP.

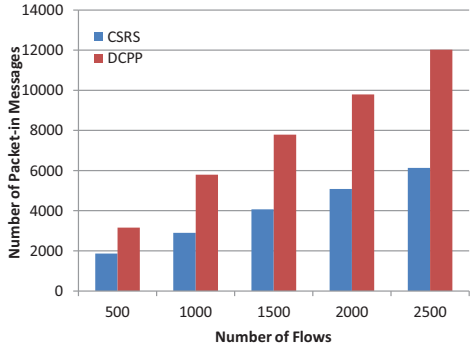
B. Result

In the simulation, we use the number of packet-in message that controller received to represent the controllers load. The smaller number of the packet-in messages means the lighter load in the control plane, and the better scalable of the multi-domain network. Thus, in our simulation, we will focus on the total number of the packet-in messages that cross-domain flows generated.

To verify CSRS performance in terms of the number of packet-in messages, we simulated our scheme in two different scenarios. In the first scenario, the network is connected by the different number of domains with given load. In each multi-domain network, we generate total of 1000 flows randomly. We count the total number of packet-in messages that switches send to the controller under the scheme of CSRS and DCPP, and make comparison. As shown in Fig.5(a), the number of packet-in messages that generated by cross-domain flows by use of CSRS is far less than DCPP. In CSRS, the controllers



(a) Comparison on different number of domains with 1000 flows



(b) Comparison on different number of flows with 4 domains

Fig. 5. Comparison of the controllers load between CSRS and DCPP

load is stable with the larger scale of the network. From this aspect, we can verify that the load in the control palne is none business with the number of network domains under the fixed-number of flows in our scheme. Thus, our scheme CSRS is more applicable than DCPP in a large-scale network. Fig.5(b) shows the simulating results of the second scenario. In this scenario, the number of flows varies from 500 to 2500 with the same scale of network domains. In this scenario, the network consists of four domains, and these flows are configured to pass through multiple domains. In this environment, we perform CSRS and DCPP, and make comparison between the total load of the controllers that generated under CSRS and DCPP. The result shows that the total controllers load is increased with the increased number flows, but the controllers load by use of CSRS is far less than DCPP. The growth rate of the controllers load is also less than DCPP with the increased number of flows. Thus, CSRS is more effective when there exist amount of cross-domain flows.

As shown in the simulation result, CSRS can effectively decrease controllers load compared with DCPP, and the performance is more significant with the growing number of domains and the increased load of the cross-domain data. The most important is that the controllers don't need to carry the load which has no relationship with local business. Cross-domain data won't worsen the congestion in control plane. Based on the simulation, we verified CSRS effective in the large-scale SDN network.

V. Conclusion

In this paper, we proposed a cross-domain source routing scheme for alleviating the controllers load to improve the scalability of SDN. The scheme is based on the full distributed structure. Through the pre-routing process and address substitution strategy, the cross-domain flow set-up requests are avoided in the intermediate domains. Each controller just undertakes the load that is related to the local domain. The controller load won't be improved with the growing expansion scale of SDN and the increased load in other domains. The simulation result shows that our scheme can greatly improve the performance of SDN network.

Acknowledgment

This work is sponsored by Huawei Innovation Research Program and the Fundamental Research Funds for the Central Universities.

References

[1] McKeown N. "Software-defined Networking," Keynote Talk at IEEE INFOCOM 2009. Available at http://tiny-tera.stanford.edu/nickm/talks/.
[2] "OpenFlow switch specification Version 1.3.0," [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.
[3] M. P. Fernandez. "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive," in IEEE AINA13, 25-28 Mar 2013, pp. 1009-1016.
[4] A. R. Curtis, J. C. Mogul, J. Tourrilhes. "DevoFlow: scaling flow management for high-performance networks," In Proceedings of SIGCOMM 2011, pages 254-265.
[5] M. Yu, J. Rexford, M. J. Freedman. "Scalable flow-based networking with DIFANE," In Proceedings of SIGCOMM 2010, pages 351-362.
[6] S. Hassas Yeganeh and Y. Ganjali. "Kandoo: a framework for effcient and scalable offoading of control applications," In Proceedings of HotSDN 2012, pages 19-24.
[7] T. Koponen, M. Casado, N. Gude. "Onix: a distributed control platform for large-scale production networks," In Proceedings of OSDI 2010, pages 1-6.
[8] A. Tootoonchian and Y. Ganjali. "HyperFlow: a distributed control plane for OpenFlow," In Proceedings of NM/WREN 2010, pages 3-3.
[9] B. Lantz, B. Heller and N. McKeown. "A network in a laptop: rapid prototyping for software-defined networks," in Proc. ACM SIGCOMM Workshop on Hot Topics in Networks, Oct. 2010.
[10] "POX Python-based OpenFlow Controller," [Online]. Available:http://www.noxrepo.org/pox/about-pox/.
[11] X. T. Phan, N. Thoai and P. Kuonen. "A collaborative model for routing in multi-domains OpenFlow networks," in IEEE ComManTel13, 21-24 Jun. 2013, pp. 278-283.
[12] Egilmez, E. Hilmi, S. Civanlar, and A. Murat Tekalp. "A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks," in IEEE ICIP13, Sep. 30-Oct. 03, 2012, pp. 2237-2240.
[13] M. F. Bari, A. R. Roy, S. R. Chowdhury. "Dynamic Controller Provisioning in Software Defined Networks," in 9th International Conference on Network and Service Management 2013 (CNSM 2013), Oct 2013, pp. 18-25.
[14] G. Yao, J. Bi, Y. Li. "On the Capacitated Controller Placement Problem in Software Defined Networks," In IEEE Communication Letters, VOL. 18, NO. 8, Aug 2014.