Computer Networks 92 (2015) 330–340

Contents lists available at ScienceDirect

Computer Networks

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

Scalable and fair forwarding of elephant and mice traffic in software defined networks

Saumya Hegde*, Shashidhar G. Koolagudi, Swapan Bhattacharya

Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, India

ARTICLE INFO

Article history:
Received 31 October 2014
Revised 20 May 2015
Accepted 15 September 2015
Available online 25 September 2015

Keywords:
Software defined networking
Data center networks
Source routing
Mice and elephant traffic

ABSTRACT

A software defined network decouples the control and data planes of the networking devices and places the control plane of all the switches in a central server. These flow based networks do not scale well because of the increased number of switch to controller communications, limited size of flow tables and increased size of flow table entries in the switches. In our work we use labels to convey control information of path and policy in the packet. This makes the core of the network simple and all routing and policy decisions are taken at the edge. The routing algorithm splits the elephant traffic into mice and distributes them across multiple paths, thus ensuring latency sensitive mice traffic is not adversely affected by elephant traffic. We observed that label based forwarding and traffic splitting work well together to enable scalable and fair forwarding. Our approach is topology independent. We present here a few preliminary simulation results obtained by running our routing algorithm on random network topologies.

1. Introduction

Data center networks must evolve to better adapt to the challenges brought about by virtualization and cloud computing. Traditional networking switches are not capable of handling these without degrading the performance. These switches are vertically integrated where the application specific hardware chips, the hardware and the entire software stack are singly sourced. This tight coupling makes it hard to program the switches. It also makes it difficult for the switches to evolve fast enough to keep up with the demands of virtualization, cloud computing and new applications developed.

Hence it is desirable to separate the control path from the data path, i.e. decouple the network control (software, protocol and state) from the network hardware. This separation is what software defined networking (SDN) aims to achieve [2]. With SDN the control path of the switches resides in a central controller on a server and the data path continues to reside on the switch as shown in Fig. 1.

SDN architecture is characterized by three important features: separation of control plane from the data plan; centralized control and centralized view of the network; programmability of the network by external applications using the controller. Data centers have a single administrative domain whose control is centralized, thereby making it suitable to use SDN in data centers.

In SDN architecture, the ingress switch encapsulates the first packet of a new flow and sends it to the SDN controller. The controller runs the required module to identify a path for the flow. The forwarding rules are then installed in the flow tables (FT), on all the switches along the path of the flow. Only the flows matching these flow entries in the switches are acted on as per the controller's instructions. Packets which fail to match the flow entries are discarded or sent to the controller. This communication between the controller and the switches is shown in Fig. 2 and is done

* Corresponding author. Tel.: +91 9482513958.
E-mail addresses: hegdesaumya@gmail.com (S. Hegde), koolagudi@yahoo.com (S.G. Koolagudi), bswapan2000@yahoo.co.in (S. Bhattacharya).

http://dx.doi.org/10.1016/j.comnet.2015.09.014
1389-1286/

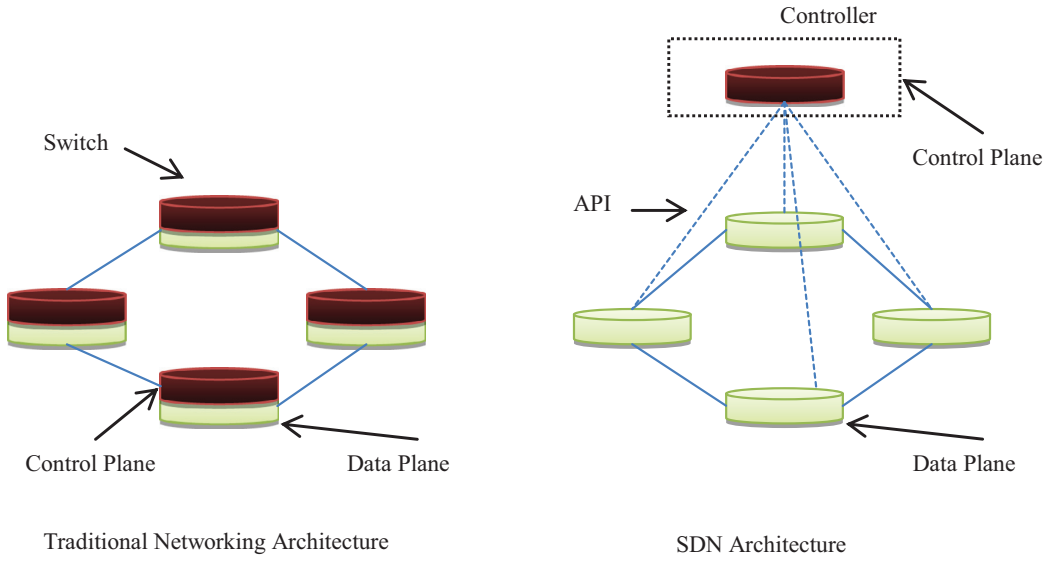

Fig. 1. Traditional network architecture vs. SDN architecture.

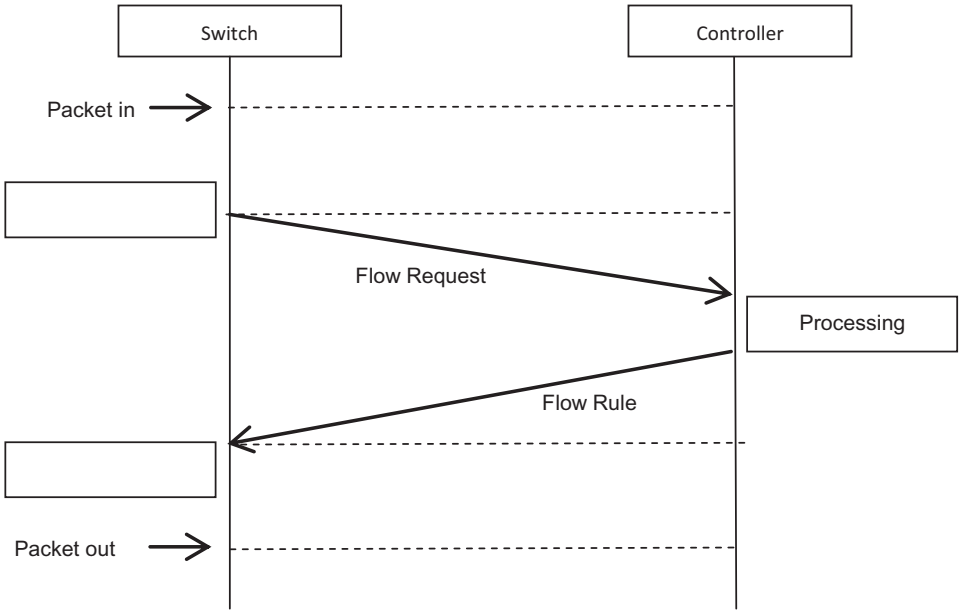

Fig. 2. Routing in SDN.

thorough various APIs, that allow the controller to address a wide variety of operating requirements.

Since SDN provides a global view of the network, it is now possible to take centralized decisions about routing and react to changes in the network state dynamically and redirect flows. This is different from the distributed routing approach of traditional networks, where decisions about routing, redirection etc. had to be taken locally. Routing which was essentially a distributed function can now be done centrally by the controller. Path computation can be done proactively before the flow begins, using the complete view of the network available at the controller. Network policy enforcement which had to be done on a switch by switch basis can now be implemented programmatically using the controller.

Although SDN provides a centralized view and allows for programmability of the network, it has some inherent scalability issues. Fig. 3 shows an example network where a packet is forwarded from source host s to destination host d. Five communications between the controller and switch are required, in order to forward the packets of this flow. Also the flow tables of switches S_1, S_2, S_3 and S_4 have to store the forwarding rules. The scalability issues here are twofold: (i) communication between the switch and the controller, (ii) memory required on the TCAM switches to store the forwarding rules. Source routing alleviates this problem since

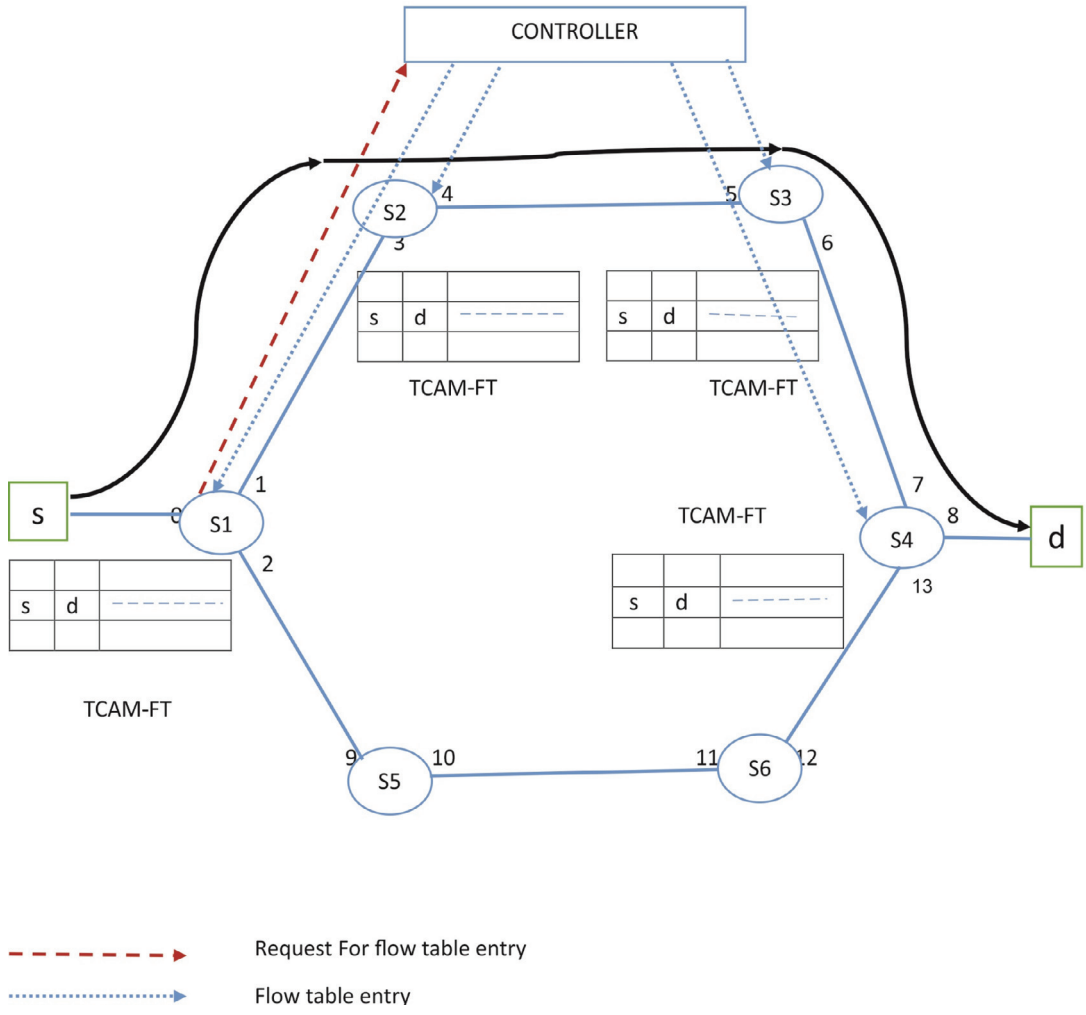

Fig. 3. Scalability issues in SDN.

it does not require the forwarding rules to be stored on intermediate switches and hence does not require the controller to contact all the switches along the path.

In this paper we propose to make routing in SDN scalable and also make it fair for both mice and elephant traffic. In Section 2 we survey the recent work on source routing in SDN. In Section 3 we present the challenges and opportunities faced by SDN, with respect to scalability and fairness in the data plane. In Section 4 we describe our architecture. We present our analytical results in Section 5. We discuss our simulation results in Section 6. We conclude with our findings in Section 7.

2. Related work

Destination based routing in an SDN environment faces scalability problems as discussed in the previous section. One of the solutions to this problem is the use of source routing. Source routing had not been popular till now because of the infeasibility of a central location where the decisions could be taken and the lack of knowledge of the entire networks status. SDN on the other hand provides both these and source routing is once again gaining popularity. In this section we discuss some of the previous work on the data plane scalability problems of SDN.

Although [8] was not employed in an SDN environment it is one of the earliest papers to implement port switching based source routing in a data center. The length of the encoded path is less when output port numbers are used instead of switch identifications since the number of ports is smaller than the number of switches in the network. They show that their solution is scalable by maintaining all the path computation work on the servers and keeping the switches stateless. Source routing in an SDN environment is discussed in [9]. The authors propose to embed the output port numbers, along the path the packet is to traverse, in the packet header. They also show how reverse paths can be calculated as the packet is forwarded at each switch, by storing the input port number on which the packet arrives, in place of the output port number which is already embedded in the packet. The paper also discusses how to handle link failure locally, wherein the switch chooses an alternate link based on the recovery information stored in the switch. Source routing tries to reduce the burden on the controller

by reducing the number of path installation required on the switches. In [10] the authors have concentrated on developing resilient source routing. The solution comprises four main components, the source routing, building reverse paths by storing the input port number at each switch, the control plane for path computation and the Plinko forwarding function. The control plane for path computation computes the primary path and an alternate path if the primary path fails. The Plinko forwarding function installs this new alternate route in the switches, to protect the paths built in source routing against failure. In [11] the authors have done extensive work on source routing. They have developed a resilient source routing algorithm. Here they compute a primary route and then find alternate paths to route packets if each of the links along the primary path failed. They ensure this does not cause loops. In encoding this back up path they ensure that the length of encoding is minimum based on the observation that the average number of neighbors of a router in a real world network is smaller than the total number of routes in the network, allowing for labels that are only locally unique. The topology under consideration here was the fat tree for data centers. They further improved on this work in [12] to support arbitrary topologies.

In [13] the authors address the data plane scalability problem by implementing stateful forwarding in the data plane with the help of a co-processing unit called the Forwarding Processor in the switches. Work carried out in [14] shows that latency can be reduced significantly by removing fields in packet headers that are unnecessary in the SDN environment. Although this paper does not employ source routing, it suggests how source routing can be implemented by using the unused header bits of packet.

Breaking of packets and spraying them across multiple paths is discussed in [15] for symmetric data center network topologies. They show that it provides better load balancing and also show that packet reordering is not a major concern since topologies are symmetric.

All of the above papers have motivated us to pursue work on source routing. We have tried to take a holistic approach to routing. Our initial path computation module breaks the elephant flows into mice flow so that long elephant flows do not affect latency sensitive mice flow.

3. Challenges and opportunities

Data centers are singly owned with centralized control making it ideal to implement SDN here. The data center network traffic characteristics differ from the internet traffic characteristics. Less than 10% of all flows are elephant flows and the rest are mice flows. But elephant flows account for more than 80% of the entire traffic volume i.e. majority of the flows in the data center are short but the majority of the packets belong to a few long lived flows [3,4]. Care must be taken that the routing algorithm designed for data center must cater to both latency sensitive mice flow and throughput sensitive elephant flows. Hence any routing algorithm must take into account the unique characteristics of data center network topology and traffic patterns. The following Sections 3.1 through 3.3 discuss some of the challenges we address. We present our contributions in Section 3.4.

3.1. Scalability: destination based routing vs. source routing in SDN challenges

When the control and data planes are separated and destination based forwarding is used, the controller has to update the data plane every time a switch encounters a packet for which an entry is not installed in the flow table. This happens for the first packet of every flow and flow table entry must be done on every switch along the path that the packet must take. Such a scenario has the following scalability issues [1].

R1: Flow table size: TCAM flow tables inherently impose size restrictions.
R2: Flow entry size: The size of flow entries are themselves larger with up to 15 field tuples, requiring 356 bits, to define a flow [14].
R3: Controller scalability: The controller is busy periodically collecting statistics about the network in order to maintain a centralized view. It is now also burdened with having to install the flow rules on *all* the switches along the path the packet has to take.

Opportunities:

A routing algorithm which stores very little state information in the intermediate switches and with less communication between the switch and controller is desirable. These issues can be addressed by using labels for routing. Since the entire path from source to destination is computed in the controller, this path information can be encoded in the packet header itself and need not be stored as forwarding rules in the switches along the path. This basically allows for routing decisions to be carried out at the edge keeping the core network simple. Also the controller has to pass the path information only to the ingress switch as opposed to all the switches along the path. This essentially addresses the above scalability issues *R1* and *R2* as no information is stored in the switches and *R3* since the controller does not have to insert flow rules in intermediate switches.

Traditionally source routing allows for path selections that optimize metrics like bandwidth, latency, hop count etc. Source routing however faces several challenges. These are

- The overhead in the packet due to the path information embedded in it.
- The difficulty in having a single server which can calculate the path.

These challenges are overcome in an SDN enabled data center because the overhead of path information in a packet is limited, since the average number of hops in a data center is 6–8 [3] and because the central controller is a convenient location to compute the paths. Source routing also speeds up the packet forwarding process as no lookup of the forwarding table in switches is needed.

3.2. Fairness: elephant and mice flows challenges

Elephant flows in a data center are generated because of operations like VM migration, VM cloning, backup, large file transfer etc. These are throughput sensitive. Mice flows on the other hand in a data center are due to applications

that use distribute/aggregate functions in the computations. These are bursty and latency sensitive. Elephant flows which are long lived tend to fill the network buffers and introduce queuing delay to latency sensitive mice flows that share these buffers even in the presence of multiple paths.

Adaptive routing techniques are not effective with mice traffic because they are bursty and not long enough to be acted upon by these adaptive routing decisions. Therefore routing in data centers often use stateless, hash based multi-pathing such as equal cost multi-pathing (ECMP) which randomly splits flows across the available equal cost paths. However using the same approach for elephant flows can cause some links to be overloaded, because of hashing several elephant flows on to the same link when another link is free.

Opportunities: The above observations lead us to design a routing algorithm that caters to both types of flows by essentially splitting up packets of elephant flow into several mice flows similar to packet spraying [16]. This allows for packet level load balancing which ensures fairness to both elephant and mice traffic.

3.3. Policy enforcement

The data center traffic has to follow strict service level agreement with its tenants. This may involve latency constraints, bandwidth constraints, security constraints like avoiding certain paths or application level constraints like certain high priority applications should take high bandwidth paths etc. Network policies can be broken into two categories: *end point policies* and *routing policies* [5]. This categorization is very helpful as the endpoint policies can be easily converted into flow entries for the edge switches which can be easily achieved. However, the routing policies are to be encoded in the packet header, since there is no more control information exchange between intermediate switches and the controller. The intermediate switches are to be made intelligent enough to understand this encoded information to achieve network policy implementation.

Examples of the policies are priority level of the packet. This is used by the intermediate switches to decide which packets to drop when a switch buffer is filled. The packets marked as low priority are dropped. Hence, we find that most of the routing policies can be handled at the edge, but some policies have to be implemented at the switch and this policy information must therefore be embedded in the packet itself.

3.4. Our contribution toward scalable and fair routing

Source routing ensures scalability and flow splitting ensures fair forwarding of both elephant and mice traffic but together they provide an elegant solution for scalable and fair routing.

When flow splitting is used with destination based routing, packets of a flow take multiple paths, requiring forwarding rules to be installed on all switches along the multiple paths, for each flow. This causes a serious scalability concern. On the other hand if flow splitting is used with source routing, no forwarding rules entries are required on intermediate switches and the overhead remains the same irrespective of the path chosen. Flow splitting causes the problem of packet reordering at end nodes but as shown in [16] it is not a major concern if symmetric topologies are used. We intend to look at this problem as part of our future work. Table 1 shows how flow splitting and source routing work together.

Table 1
Source routing with flow splitting.

	Without flow splitting	With flow splitting
Without source routing	Not scalable Not fair	Not scalable Fair
With source routing	Scalable Not fair	Scalable Fair

4. Architecture

In this section we discuss our framework that addresses the data plane scalability and fairness challenges of SDN. We describe the various components in our architecture.

Policy data base: Network policies can be sub-divided into *endpoint policy* and *routing policy*. Endpoint policies can be handled by the edge switches. Routing policies on the other hand concern intermediate switches. Hence the routing policy information has to be encoded in the packet headers.

Proactive path computation: All source to destination paths are proactively computed and these paths will be compliant with the user defined end point policies that are stored in the policy database. The paths are stored as a sequence of output ports of the switches along the path. Output port numbers are a better choice than the next switch identification because the number of ports in the switches is much less than the number of switches in the network.

Obtaining the path from controller: When an edge switch encounters the first packet of a new flow, it checks its local cache to see if path information is stored for this source destination pair, if not the packet is sent to the controller. The controller replies by sending the k equal cost, policy compliant paths to the edge switch. The edge switch caches the routes. The cache hit is likely to be high because in a data center most flows are destined to a few destinations.

For packets of this flow the edge switch embeds the packet with one of the k paths, chosen in a round robin manner. This approach is essentially turning the elephant flows into mice flows and performing per-packet forwarding. By splitting the flows across available k paths, it ensures that long flows do not fill the network buffers along a single path and adversely affect the latency sensitive short flow, while also ensuring a high throughput. Here we rewrite the Mac header, thereby avoiding having to introduce a separate header for path information. This solves the packet overhead issue discussed in the previous section.

Fig. 4a shows the initial architecture where flow table based forwarding is used. Flows are hashed onto different paths, with additional overhead of having to store extra forwarding rules for the multiple paths. This approach is neither scalable nor fair, as two elephant flows are hashed onto a single path. Fig. 4b shows our proposed architecture which uses source routing with flow splitting. Source routing does not require update in flow tables at intermediate switches and

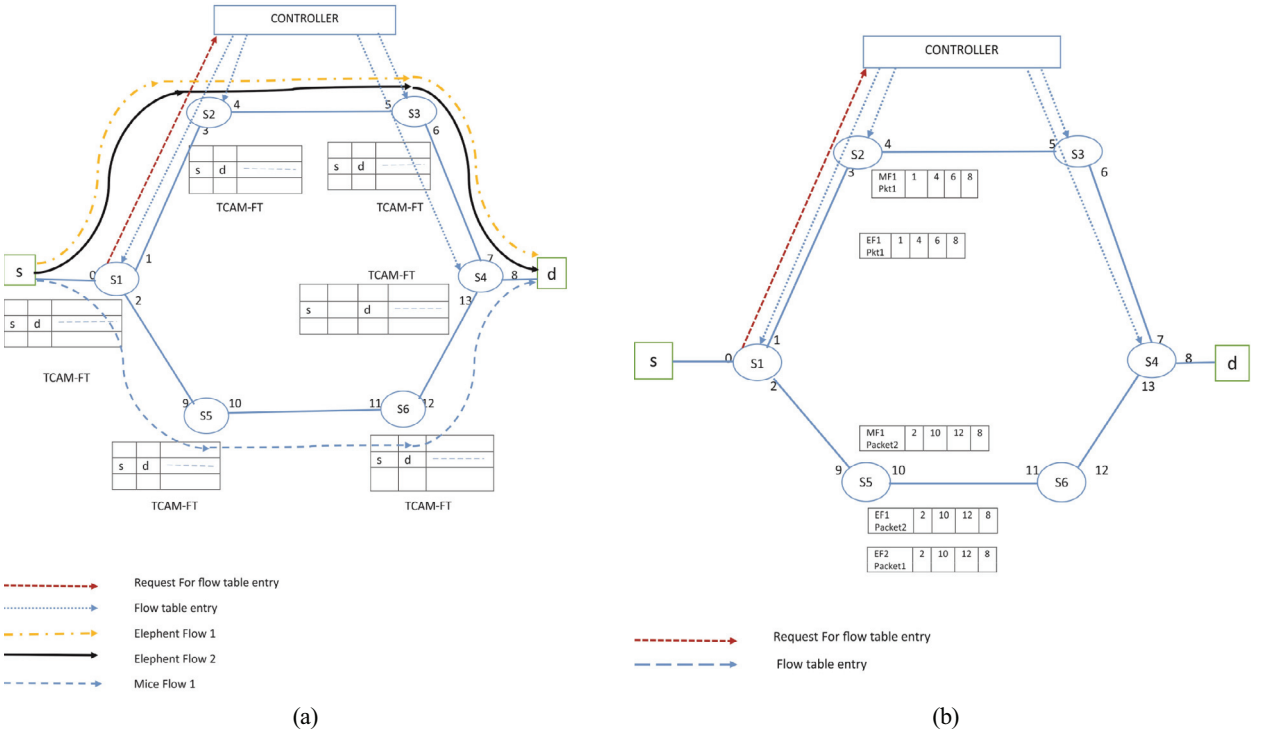

Fig. 4. (a). Initial architecture using flow table. (b) Our proposed architecture with source routing and flow splitting.

flow splitting ensures fair bandwidth utilization of multiple paths with no additional overhead.

Forwarding by intermediate switches: The intermediate switches now forward the packet by looking at only the forwarding port number embedded in the packet. This approach solves the scalability problem with destination based forwarding because no information is stored in the switches and the controller does not have to push forwarding information to the intermediate switches. This keeps the core switches simple and they can operate fast. Here we do not compute the reverse path at each switch as the packet is being forwarded, as is the case in [9], for two reasons. One being that the all-to-all paths have already been proactively computed and second being that this slows down the forwarding process. We would like to keep the core switches as simple as possible. Once the packet reaches the egress switch it is over written with the original header and sent to the host.

5. Analysis

In this section we analyze source routing with flow splitting in SDN environments with regard to flow table memory requirements, processing time on intermediate switches and number of communications between controller and switches. We compare it with traditional routing and source routing.

Let n be the number of switches in the network, p be the path length as hop count between a source s and destination d, f be the size of forwarding rules, k the number of equal cost multiple paths, t the time required to lookup the forwarding table, r the time required to forward a packet at a switch and pl the packet header look up time.

5.1. Flow table memory requirement

Traditional routing requires the flow rules to be stored on all the switches along the path of the flow. Therefore memory required on the flow tables for a flow between source s and destination d is given as

$$M\_FT_{(s,d)} = p * f \tag{1}$$

For multipath path traditional routing the memory requirement is

$$M\_FT_{(s,d)} = p * f * k \tag{2}$$

Source routing does not have flow table memory requirement since it does not require flow rules to be stored.

5.2. Total processing time on intermediate switches

In traditional routing, in each switch, the packet header has to be looked up, followed by flow table look up for matching flow rule, and followed by packet forwarding. This is repeated on all the intermediate switches. Hence the processing time on intermediate switches for a flow from source s to destination d is given as

$$PT_{(s,d)} = pl * t * r * p \tag{3}$$

On the other hand source routing and source routing with flow splitting do not require a flow table lookup, since the

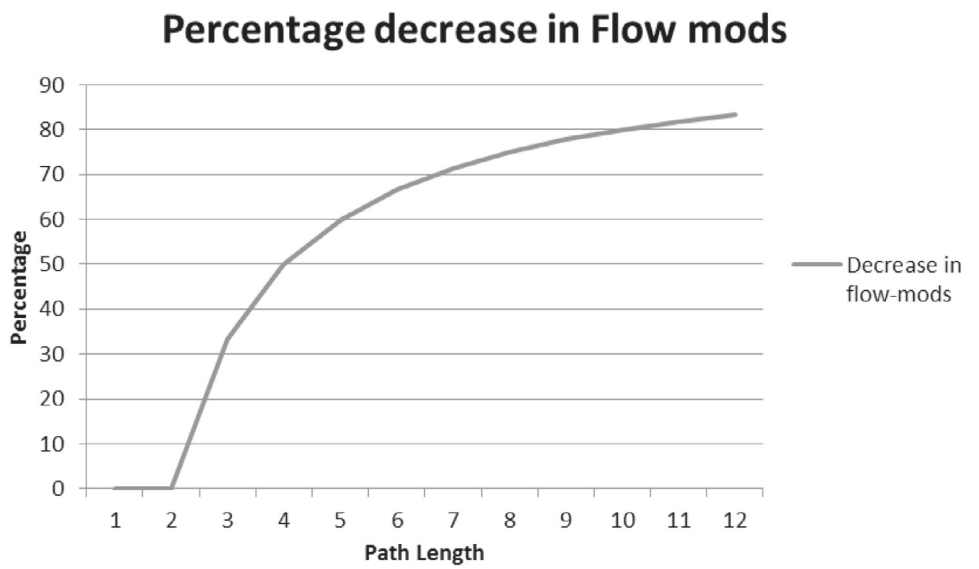

Fig. 5. Percentage reduction in flow mods.

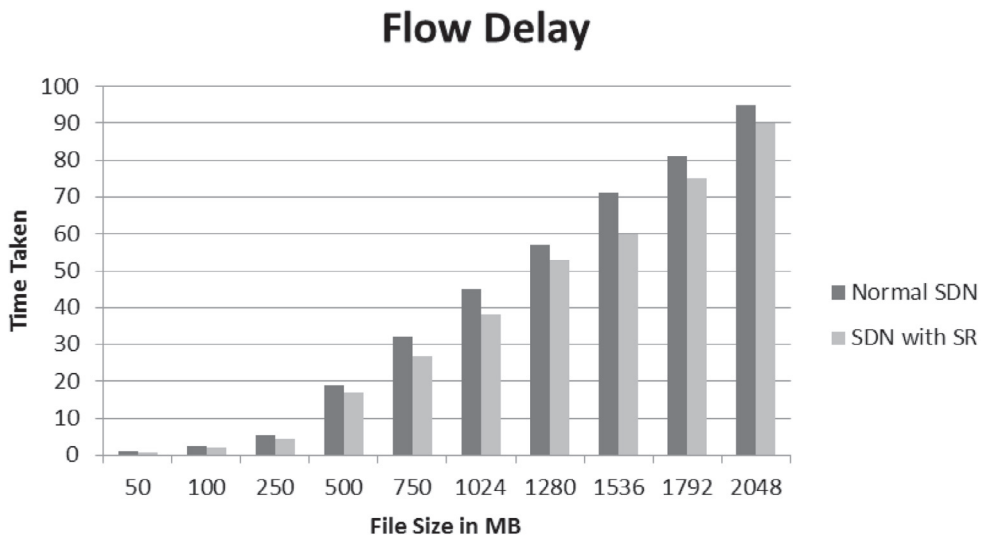

Fig. 6. Decrease in flow delay.

path information is stored in the packet itself. Hence processing time on intermediate switches is given as

$$PT_{(s,d)} = pl * r * p \tag{4}$$

5.3. Number of communications between controller and switch

In traditional routing, the controller has to place the forwarding rule on all the switches along the path of the packet. Hence number of controller to switch communications is

$$C_{(s,d)} = p \tag{5}$$

With source routing it is a constant of 2 communications between the ingress and egress switches. In source routing with flow splitting across multiple paths the number of updates is 2 for every path, hence

$$C_{(s,d)} = k * 2 \tag{6}$$

6. Implementation and results

In this section we evaluate the performance of our proposed routing algorithm with respect to traditional SDN routing. The SDN scenario has been emulated using Mininet [6] emulator and Floodlight [7] as the SDN controller. In order to evaluate the scalability of our algorithm parameters like reduction in flow mods, flow delay, flow setup delay, edge switch performance, controller performance, and overhead experienced are considered. We have considered a random

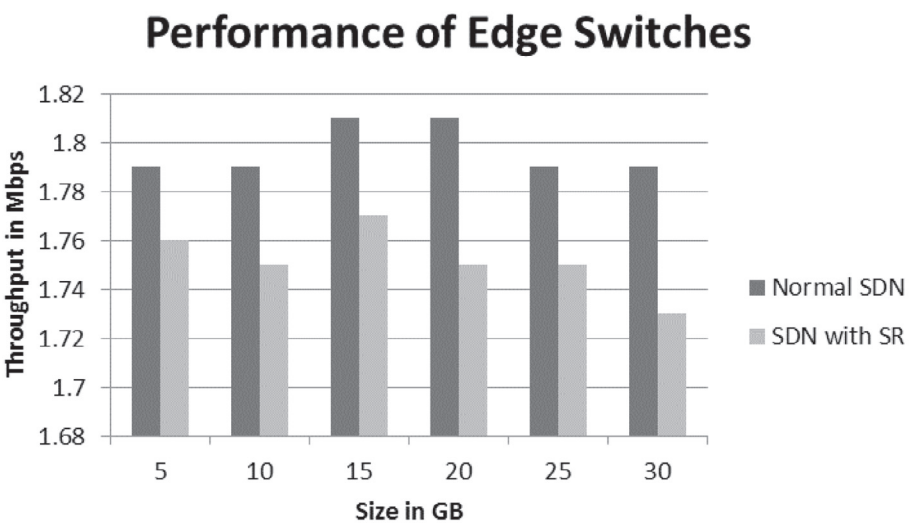

Fig. 7. Performance of edge switches.

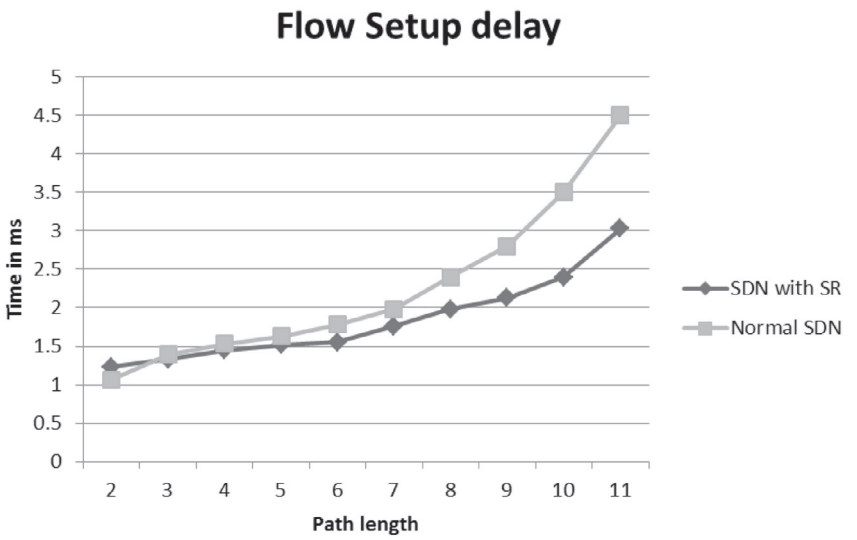

Fig. 8. Flow set up delay.

topology with 256 nodes and 30 hosts and generate the traffic using *iperf* [16].

In order to show that our algorithm reduces the number of communications between the controller and the switches we measure the reduction in the number or forwarding rules or flow mods distributed by the controller as the flow path lengths vary. In order to do this we generate flows of different path lengths. In traditional SDN when a new flow is encountered by the controller, all the switches along the flow path have to be notified with the flow entries by the controller. However, with our implementation of source routing in SDN, it can be seen that only the edge switches are required to be notified. Hence the required number of flow mods remains at two irrespective of the path length. Fig. 5 gives the percentage decrease in flow mods required.

We show that the forwarding of packets on the switches is faster with source routing since flow table look ups are not required, instead forwarding action on the packet can be taken by referring the header entries in the data path itself. Here we consider HTTP file transfer of varying size, between two hosts with a path length of eleven between them. Fig. 6 shows the impact of source routing on the delay experienced by the flow. Initially due to TCP bursts, the small files are transferred very fast and not much difference can be observed in those cases. However, after a certain point in time, change in delay can be clearly seen and is maintained throughout.

Although decrease in delay is observed by the flows in the presence of source routing, the edge switches see a decrease in the performance due to increase in the number of actions to be carried out per flow at these switches. Fig. 7 shows the

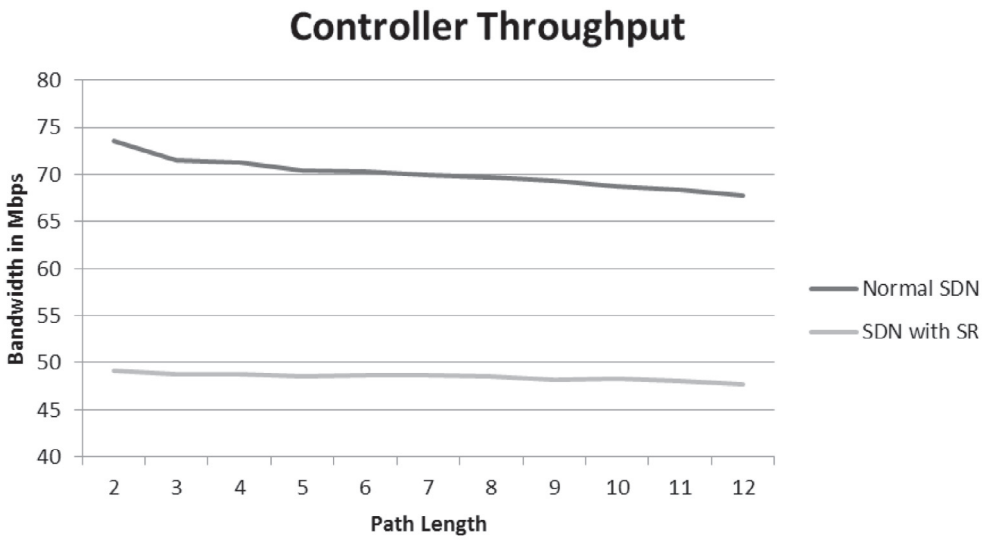

Fig. 9. Controller throughput.

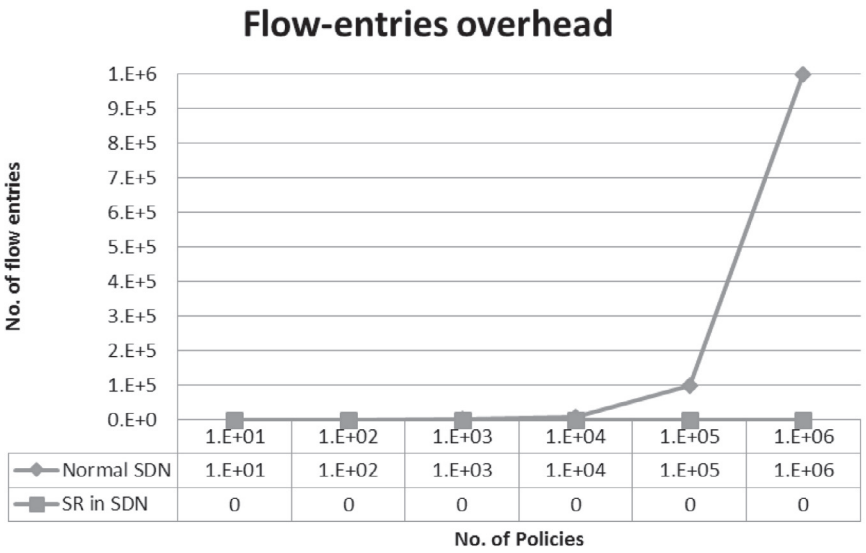

Fig. 10. Flow-mod overhead for network.

decrease in throughput at the edge switches when source routing is used.

In order to evaluate the efficiency of our algorithm, we measure the time taken by the controller to set up a flow under constant minimum workload. The time taken in the destination based SDN to push all the flow mods and set up the flow will increase as the number of flow mods to be pushed increases with path length. On the other hand in source routing SDN, the length of the path to be encoded in the packet increases with the increase in flow path length. However, since the encoding work is pure CPU based and I/O work remains the same, the percentage increase of flow setup time with path length is comparatively lower as can be seen in Fig. 8. Only in the case of the path length being two, the time taken is more than the previous case since more number of actions is to be inserted in the edge switch flow mods.

Next we compare the controller throughput as we vary the path lengths of the flows. From Fig. 8 it is seen that the controller takes more time to set up flows as the path length increases. A similar effect can be seen on the controller throughput as well. Fig. 9 shows the decrease in controller throughput as the path length decreases. This is because under traditional SDN routing the intermediate switches are to be programmed by the controller for each and every control decision to be implemented. In order to evaluate the overhead incurred, we measure the number of flow entries required and also the number of bytes required to encode the path and policy information. From Fig. 10 it can be seen that to implement various routing policies in traditional routing,

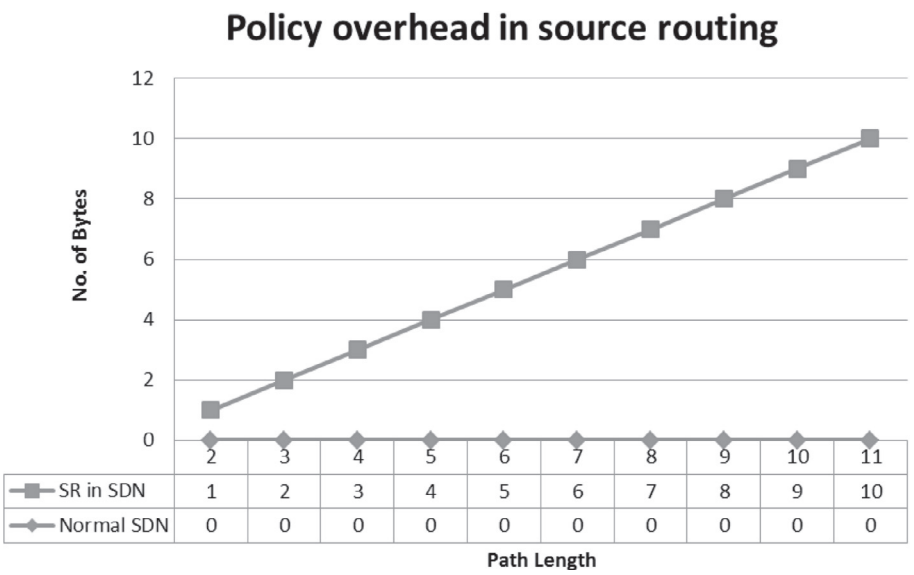

Fig. 11. Byte overhead for network policies.

equal number of flow entries are to be pushed to each intermediate switch. However, with source routing no flow entries are used.

Although source routing shows a decrease in the flow entries required to implement the policies, it consumes a few bytes in the packet header to achieve the same. From Fig. 11 it can be seen that in the worst case the number of bytes required to implement a singly policy increases linearly with path length.

7. Conclusion

SDN that uses destination based forwarding faces scalability issues because of limited TCAM flow table size, increased number of communications between the controller and the switches. It will also encounter major design issues if intermediate switches have to identify and handle mice and elephant flows differently. When the routing algorithm uses labels, to encode path and policy information, performs flow splitting and routing decisions at the edge, it ensures scalability and fairness. It also follows the end to end principle in general. As future work, we plan to further study how our routing method performs with respect to packet reordering. Fault tolerance is an important aspect with source routing and we plan to investigate it further.

References

[1] S.H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Commun. Mag. 51 (2) (2013) 136–141.
[2] N. McKeown, T. Anderson, H. Balakrishnan, G.M. Parulkar, L.L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, Comput. Commun. Rev. 38 (2) (2008) 69–74.
[3] T. Benson, A. Anand, A. Akella, M. Zhang, Understanding data center traffic characteristics, SIGCOMM Comput. Commun. Rev. 40 (1) (2010) 92–99.
[4] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, New York, NY, USA, ACM, 2010, pp. 267–280.
[5] N. Kang, J. Reich, J. Rexford, D. Walker, Policy transformation in software defined networks, in: Proceedings of the Second ACM SIGCOMM, 2012, pp. 309–310.
[6] Mininet. http://mininet.org/ Accessed 2015-05-20.
[7] Floodlight Controller. http://www.projectfloodlight.org/floodlight Accessed: 2015-05-20.
[8] C. Guo, G. Lu, H.J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, Y. Zhang, SecondNet: a data center network virtualization architecture with bandwidth guarantees, in Jaudelice Cavalcante de Oliveira, in: Maximilian Ott, Timothy G. Griffin, Muriel Médard (Eds.), CoNEXT, ACM, 2010, p. 15.
[9] M. Soliman, B. Nandy, P. Smith, Source routed forwarding with software defined control, considerations and implications, CoNEXT Student'12, ACM, December 10, 2012, pp. 43–44.
[10] B. Stephens, A.L. Cox, S. Rixner, Plinko: building provably resilient forwarding tables, HotNets, 2013, pp. 26:1–26:7.
[11] R.M. Ramos, M. Martinello, C.E. Rothenberg, Data center fault-tolerant routing and forwarding: an approach based on encoded paths, LADC, IEEE, 2013, pp. 104–113.
[12] G.T.K. Nguyen, R. Agarwal, J. Liu, M. Caesar, P.B. Godfrey, S. Shenker, Slick packets, ACM Sigmetrics, 2011, pp. 205–216.
[13] S. Zhu, J. Bi, C. Sun, SFA: stateful forwarding abstraction in SDN data plane, in: Open Networking Sumit—Research Track, Usenix, USA, 2014.
[14] K. Kannan, S. Banerjee, Scissors: dealing with header redundancies in data centers through SDN, in: CNSM, IEEE, 2012, pp. 295–301.
[15] A.A. Dixit, P. Prakash, Y.C. Hu, R.R. Kompella, On the impact of packet spraying in data center networks, in: INFOCOM, IEEE, 2013, pp. 2130–2138.
[16] Tirumala, A., et al. Iperf: the TCP/UDP bandwidth measurement tool http://dast.nlanr.net/Projects (2005) Accessed 2015-05-20.

Saumya Hegde: She completed her M.Tech. in Computer Engineering from NITK and is currently pursuing her Ph.D. from the same institute. Her research work is on scalability and resiliency issues in SDN.

Shashidhar Koolagudi: He is currently working as an assistant professor in the Dept. of Computer Science and Engineering at NITK. He completed his Ph.D. from IIT Karagpur. His research area is speech and signal processing and Networks.

Swapan Bhattacharya: He is currently working as professor in the Dept. of computer science and engineering at NITK. He completed his Ph.D. in computer science and engineering from University of Calcutta. His research area is software testing and design of algorithms.