ACSIJ Advances in Computer Science: an International Journal, Vol. 3, Issue 6, No.12 , November 2014
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

Load Sensitive Forwarding for Software Defined Networking – Openflow Based

Nor Masri Sahri[1] and Koji Okamura[2]

[1] Department of Advanced Information Technology,
Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan
normasri@kyudai.jp

[2] Research Institute for Information Technology, Kyushu University, Japan
oka@ec.kyushu-u.ac.jp

Abstract

Avoiding congestion is important role in recently proposed software defined network (SDN) since various new kind of overhead and delay introduced compared to traditional network. In this paper, we propose a load sensitive forwarding metric for the Openflow controller to make decision for path selection. Our metric assign weight based on the network traffic link load that put into account. This metric helps the forwarding protocol to load balance the network and improve the network capacity by avoiding the congested nodes to forward the traffic. Extensive performance results based on our simulation are presented to demonstrate the effectiveness of our proposed metric, with comparison to the SDN state of the art forwarding decision.

Keywords: *Software Defined Networking, path selection, openflow*

1. Introduction

In order to elaborate more on our proposed load aware forwarding metric, it is important to put the highlight on OpenFlow [1] which is a clean slate project introduced by Stanford University. OpenFlow was implemented as the first open standard interface for Software-Defined Network (SDN) architecture. SDN enable network administrators with a central programmable management interface, which is decoupled from the underlying network infrastructure for current layer 2 and layer 3 switches. In OpenFlow, the data path and the high level routing decision are made from two different devices, which is the OpenFlow-enabled switch and controller, respectively. The central controller provides the switches with the operational rules instructions, which is pushed by the controller to the switch as individual flow entries via a secured channel between them using OpenFlow protocol. The switches search the flow table corresponding entries and if there is any rules match, it will process the packets according to the pre-specified actions in the entries. The incoming packet in OpenFlow-enabled switches is matched against the flow table and the associated actions are taken into action: similar to the existing conventional switches. The main difference is that if the packet has not match any rule in the flow table, the packet would be drop or flooded in the network. OpenFlow-enabled switch include the packet header and encapsulate into the Openflow asynchronous message name *Packet_In*, and forwarded to the controller. The controller then use the flexibility of software to do analysis and further do the path selection decision based on shortest path algorithm. The controller then install a flow entry to the switch together with associated action via *Packet_Out*. The specification of the Openflow protocol message can be found in [2] for further clarification.

In other words, the Openflow controller has the helicopter view of a particular network and with this feature, the network management is expected to be much more “controllable” in single machine which is the controller itself. However, the SDN is also come with a various new type of overhead and cost. We identify a disadvantage of Openflow when compared to the native packet forwarding where all the unclassified incoming packet in a particular switch must be forwarded to the controller for further processing. Then the controller will decide the path for the packets and install the flow entry into the chosen Openflow switches flow table. The mechanism may introduce a significant delay and in order for the switches forward the incoming packet thru most optimal path, the controller also need to do the decision based on the current network state and condition which introduced another type of cost. The delay in a node with higher traffic load goes in and out through an interface, could be larger than a node with lower traffic load. If the controller include this

heavy nodes as the flow entry to forward packet, it may actually cause end-to-end delay even longer. Furthermore, if one of the heavy nodes is congested, it may lead to packet drops and retransmission problem on certain nodes.

Many effort has been put into SDN research about load balancing and most notably that discussed load balancing in SDN is [3] where the author present algorithms that exploit the wildcard feature in Openflow that target to achieve equal fair distribution of the traffic and automatically adjust to changes without disrupting existing connections in data center network. In [4], the author proposed SDN load balancing that simplify the network placement in the network. When a new server is installed, the load balancing service will take appropriate decision to seamlessly distribute the network traffic among the available server, putting the network load and available computing capacity into consideration. They demonstrate that it can simplify the network management and provide some flexibility to network operators. Other recent proposals [5][6][7] also shows the load balancing work improvement in SDN.

In contrast to other works, we focus on designing a metric using Openflow protocol for controller to make path selection decision for forwarding a packet through most optimal available path. In this paper, we design a forwarding metric considering the available link load utilization that aim to minimize the network end-to-end delay. The delay through a node, which has larger link utilization could be larger than through the one, which has less traffic passing through their available ports.

The rest of the paper is organized as follows. **Section 2** provides the details of our proposed load sensitive metric algorithm for SDN using Openflow protocol and in **Section 3**, we discuss some of our proposed metric implementation issues. In **Section 4**, we present the forwarding protocol design followed by the performance evaluation results in **Section 5** and finally, **Section 6** concludes this paper.

2. Load Sensitive Metric

We define network load over a path as the *average rate of bytes* that goes through the link and correctly received by the other node that attached to it. Therefore, the link load utilization is expressed as the current link usage over the maximum link capacity and is a number from zero to one. When the utilization is zero it means the link is not used and when it reaches one it means the link is fully saturated [8]. We use Load Sensitive for Software Defined Networking (LSSDN) terminology to denote the proposed load sensitive metric at each link.

To get the link load utilization, the formula are as follows

$$\left(\frac{throughput}{datarate}\right) * 100 \qquad (1)$$

The *throughput* is measured by the sum of incoming and outgoing bytes. In Openflow, the needed information (*byte count*) is available from Openflow switch port counter [2]. To measure the link load utilization, the Openflow controller needs to monitor all switches port traffics, cache all of the number of incoming and outgoing bytes through their interface for further calculation.

Assuming that L links are available between two nodes. The bandwidth of i^{th} link between $node_m$ and $node_n$ is $B^i_{m\to n}$ (i=1,…L). If the total number of bytes going through an interface denote as T, then we can calculate the traffic utilization as follows. We define the link load utilization of a node interface, U over $link_i$ from $node_m$ to $node_n$ as the *current total of throughput* through the interface at certain time over the *link transmission rate*. The formula can be simplified as below.

$$U_i = \left(\frac{\sum(T^i)_{m\to n}}{B_{m\to n}}\right) * 100 \qquad (2)$$

The path weight of the LSSDN metric is defined as (consider end to end path including H hops),

$$\omega U_i = \sum_{i=1}^{H} U_i \qquad (3)$$

Note that the LSSDN metric given in (3) is under the assumption that all the packets can continuously go through all the path hop-by-hop without any node or link failure.

Let vector $[B^i_{m\to n}, T^i_{m\to n}, L]$ is characteristic of link between $node_m$ and $node_n$. $B^i_{m\to n}$ denote the bandwidth of i^{th} channel between $node_m$ and $node_n$, L is the number of available links and $T^i_{m\to n}$ is the total throughput of i^{th} link between $node_m$ and $node_n$. For a given network of *G (V, E)*, and the source node N_s and destination N_d, the LSSDN algorithm include the following steps:

Step 1: Calculate the bandwidth capacity ($B^i_{m\to n}$) for every link in network *G (V, E)*.

Step 2: Calculate the total throughput (in *bytes*) over a link $T^i_{m\to n}$ for every link in network *G (V, E)*.

Step 3: Calculate the weight ωU_i for every link in *G (V, E)* according to Eq. (3)

Step 4: Use Dijsktra Algorithm to find the smallest sum of weight in the paths of *G (V,E)* from node N_s to node N_d. The details of LSSDN are given in Algorithm 1.

Algorithm 1: Smallest Link Utilization Path Selection Algorithm

Input: $B^i_{m\to n}$, $T^i_{m\to n}$, *(i=1,...L);*
$V=\{v_1,v_2,\ldots,v_n\}$ *:The set of nodes;*
$N_s \in S$*: source node;*
$N_d \in V$*: destination node;*
for j=1 to N do
  for k=1 to N do
    for =1 to M do
      find $B^i_{m\to n}$ *;*
      find $T^i_{m\to n}$*;*
      calculate ωU_i*;*
    end for
  end for
end for
S: The current set of nodes (from N_s *to* N_d*) which has smallest load path*
$T(V_i)$*: The current sum of link load of the links on the smallest weight path from* N_s *to* N_d*.*
for =1 to N do
$T(V_i) = \infty$;
$T(N_s) = 0$;
$S = \emptyset$;
end for
while $N_d \in S$
{
u=v; //$V_i \in S$ *and ( )* $T(V_i)$ *is smallest load in all nodes in V→S*
$S = S \cup \{u\}$;
for all $v_k \in V \to S$
  $if\ \left(T(v_i) + \omega U_{m\to n} \leq T(V_k)\right)$
$T(V_k) = T(V_i) + T(V_k) + \omega U_{m\to n}$;
}

2.1 Impact of traffic load utilization

Besides the update frequency, the number of transmitted and received bytes information affects the estimation of link utilization for the LSSDN metric. The number of throughput changes instantaneously. If we use the value directly for link utilization calculation, frequent rerouting might occurred. To avoid the problem, we maintain a weighted average link utilization in the controller, denoted as $\bar{U}$ and controller use this weighted average value as the backlog information instead of instantaneous sample value for the LSSDN computation. Specifically, the controller samples the instantaneous throughput according to a schedule, and let U_n denote the n^{th} sample. The average link load utilization, $\bar{U}$ by incorporating the instantaneous link utilization U_n, according to the exponential weighted moving average scheme [10], is

$$\bar{U} = (1-\alpha)U_{old} + \alpha * U_n \qquad (4)$$

$where\ 0 \leq \alpha \leq 1$ //$U_n = n^{th}$ sample

To show the need to include link utilization as a metric in SDN forwarding decision, we demonstrate the relation between link utilization and the input load using simple simulation. In the simulation, we tried to vary the aggregate input load traffic and measured the usage of the bandwidth. To simplify the network, the packet size was set to 1000 bytes and the link data rate was set to 1Mbps. The chart as illustrated in **Fig. 1** is to emphasize the impact of the link load utilization on forwarding packet to their destination. From the figure, it clearly shows that the link load utilization is proportional to the traffic that sent through the link. It is observed that the bandwidth utilization is increase linearly with the input load and then it get saturated as the input load reach approximately 800 Kbps. When the link is saturated, the link load utilization is almost constant even though the input load increases and it is happened because the available link bandwidth is almost fully utilized. The higher the value of link load utilization, the lesser traffic can be send over the link.

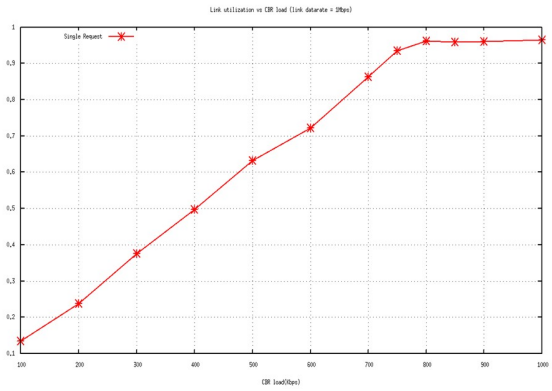

Fig. 1 Link utilization as a function of input traffic

We also carried another simulation to study the capability of a link for to tolerate more traffic at different link load utilization. We simulate 5 pair of nodes exchanging data with another and we can derive the relation between link load utilization and

delay of packet. As illustrated in **Fig. 2**, we observed that the packet delay time increase dramatically when the link utilization start to climb at 90 percent of link utilized. This shows that the need to consider the link load is vitally important at high link utilization but its effect may also be ignored when the link load in under-utilized.

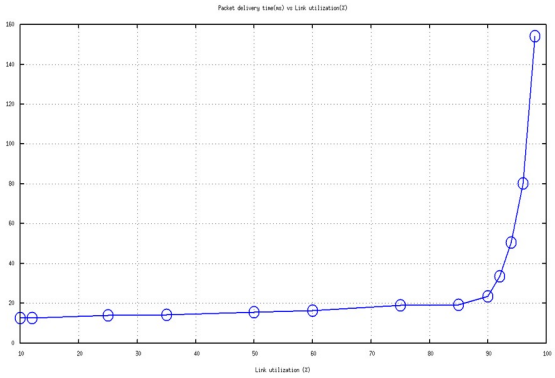

Fig. 2 Packet delay time as function of link utilization

3. IMPLEMENTATION DESIGN ISSUE

3.1 Update frequency

Our proposed LSSDN forwarding metric can be viewed as a load sensitive metric as it is heavily depend on the switch port information. Similar to other load sensitive metrics, the Openflow controller is require to perform recalculation by updating the traffic status to avoid usage of the congested link in the network. To balance the tradeoff between performance and the overhead, the route update frequency is a critical factor. More frequent updates of network state will introduce unnecessary overhead. On the other hand, large gap of update frequency will prevent the route from timely tracing the network status, and the network performance may dropped. We adopt the multipart message provided in Openflow feature which use to encode request or replies to or from the controller to switches. We simulate the feature to get port statistic for all of the switches to calculate our proposed link load metric in the controller. In our proposed algorithm, the message collect *byte count* information to measure the packets going in and going out through a particular port. In our simulations, we set the time for the Openflow switches to update the controller with the needed information automatically every *5 seconds*. After the controller receive the new information it will perform recalculation based on our proposed metric to define the most optimal next forwarding path. In Openflow, the controller able to modify the existing flow entries *action* field that installed in the Openflow switches via flow table modification message that modify all flow that match. In our case, the controller will modify the existing flow entries in the switch with the newly most optimal next forwarding path.

4. FORWARDING PROTOCOL DESIGN

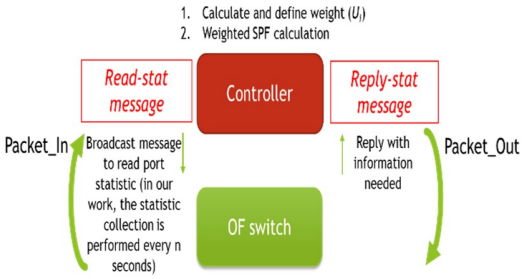

Fig. 3 Packet delay time as function of link utilization

4.1 Route Discovery

We design a forwarding protocol for SDN IP network which aim to create congestion free flow entry by making use of information gathered from Openflow switches MAC layer. Now we describe the route discovery process in our proposed method. As illustrated in **Fig. 3**, any source node wishing to transmit data to a given destination will be process by the Openflow switch first. In Openflow, the switches contain exact matching tables for the forwarding databases. The incoming traffic will be check by the switches whether the packet has any matching in the flow table by performing a table lookup process. In our works, the matching field is based on the incoming source and destination IP address. When there is no match, the unmatched packet header will be extracted by the switch to be include in a special Openflow protocol message called *Packet_In*. This packet is use by the Openflow to transfer the control of any unmatched packet to the controller. We assume that all of the Openflow switches support the internal buffering to keep the unmatched packet in the switch buffer. Any unmatched packet will be buffered in the switch waiting to be forwarded to the next hop. *Packet_In* contain the buffer ID to represent the unmatched packet that stored and also some fraction of the packet header to be used by a controller when it is ready for the switch to forward the packet.

Any forwarding metrics require the real-time traffic information. When the controller receive the *Packet_In* message, it will send specially crafted type of message by Openflow to collect all of the switches port related statistic information via broadcast technique in order to decide the most optimal next hop to forward the packet to their destination. The message will request the number of transmitted and received bytes that go through all of the available ports of a node which is a vitally needed information for our proposed load sensitive metric. In Openflow, various kind of statistic information can be requested such as number of incoming and outgoing bytes or packets, the number of drop packets and also the time duration for how long the port has been alive in seconds.

4.2 Route Reply

After the switches receive the request message from the controller, it will include the needed current transmit and received bytes information from the ports in the port statistic reply message. The switches will send back the reply message together with the information back to the controller. Once the controller received the requested statistic, it will perform the path selection calculation to decide the most optimal next path for the switch to forward the packet using our proposed metric as presented in Section III. After the decision, it will install the flow entry into switches flow table via *Packet-Out* messages that contain the buffer ID referencing a packet that previously stored in the switch. The message also contain the action field that decide the next path for the switch to forward the buffered packet to their next destination. When the switch receive the Packet_Out message, it will install the flow entry in the current flow table and match the buffered packet with the identical buffer ID and continue to forward the traffic to the next path.

5. PERFORMANCE EVALUATION

The goal of our evaluation is to show the effectiveness of proposed metric to be adopted in SDN. We consider random topology which is based on Power Law model [9]. In this model, which is often use to represent the actual Internet, most of the nodes has lower number of links while a small number of nodes have a larger number of links. In our simulation, we set the number of nodes to 20 and the node degree is set to two which means that each nodes has at least four connected links on average. Five nodes were randomly chosen to generate UDP traffic across the network, with the packet size of 512 Bytes respectively.

Fig. 3 and **Fig. 4** present the results of our simulation that shows the performance of our proposed methodology with the comparison of the native Openflow forwarding mechanism. The total of network throughput and end-to-end delay versus the various flow rate transmission is chosen as the performance metric to be evaluated. The queue size is pre-defined to 20 packet in each switches and the simulation time is set to 600 seconds. From the two figures, it is explicitly demonstrated that our proposed metric result in much better performance for end to end delay than the native forwarding in Openflow under the random topology.

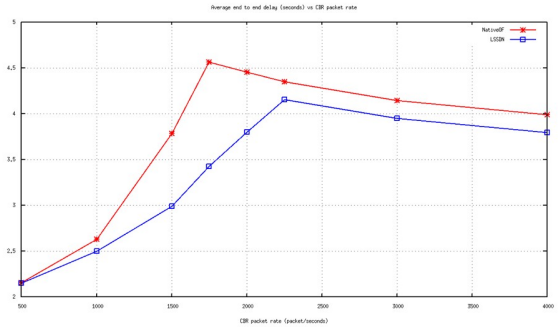

Fig. 4. Average end to end delay versus CBR rate

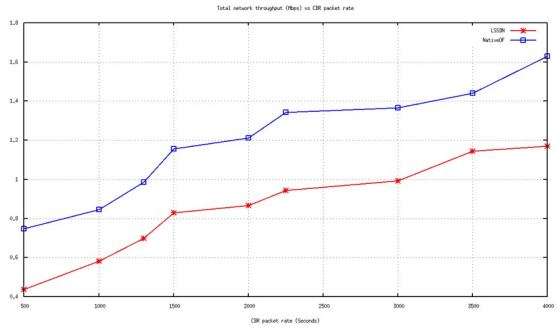

Fig. 5. Throughput versus CBR rate

6. CONCLUSION AND FUTURE WORKS

In this paper, we evaluated the applicability of native OF data path selection for forwarding action in SDN environments. Our study shows that when the SDN network link is almost saturated, the packet delivery time is also increased hence the need to propose

forwarding path selection algorithm is proposed. We also demonstrate that by using our proposed metric, it can lead to path selection with minimum end-to-end delay and higher network throughput is also achieved. Since Openflow is a clean slate techonology, various type of delay and network overhead is introduced. We plan to study those limitation and try to identify other types of possible metric for path selection problem in SDN network.

References

[1] N. McKeown, T. Andershnan, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM Computer Communication Review 38 (2) (2008) 69–74. New York, USA.

[2] Openflow specification version 1.1.0, http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[3] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In Hot-ICE, 2011

[4] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using OpenFlow," In ACM SIGCOMM Demo, August 2009.

[5] C. Macapuna, C. Rothenberg, and M. Magalhaes, "In-packet bloom filter based data center networking with distributed OpenFlow con- trollers," in GLOBECOM Workshops (GC Wkshps), 2010 IEEE, 2010, pp. 584–588.

[6] N. Handigol, S. Seetharaman, M. Flajslik, A. Gember, N. McKeown, G. Parulkar, A. Akella, N. Feamster, R. Clark, A. Krishnamurthy, V. Brajkovic, and T. A. and, "Aster*x: Load-Balancing Web Traffic over Wide-Area Networks," 2009.

[7] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," in Proceedings of the 2010 internet network management conference on Research on enterprise networking, ser. INM/WREN'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–1.

[8] P. Chimento and J. Ishac. Defining Network Capacity. RFC 5136 (Informational), February 2008

[9] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An ap- proach to universal topology generation," Proc. IEEE MASCOTS, pp.346–353, Aug. 2001

[10] J. M. Lucas and M. S. Saccucci, "Exponentially weighted moving average control schemes: properties and enhancements," Technometrics, vol. 32, no. 1, pp. 1-12, Feb. 1990

Nor Masri Sahri received his first Bachelor Degree (B. of Information Technology) from Northern University of Malaysia on 2001 and obtained his Master Degree (MSc. of Information Technology) from University of Technology MARA on 2006. He is currently a first year Ph.D. student and belongs to the department of Advanced Information Technology, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan.

Koji Okamura is a Professor at Department of Advanced Information Technology and also at Computer Center Kyushu University, Japan. He received B.S. and M.S. Degree in Computer Science and Communication Engineering and Ph.D. in Graduate School of Information Science and Electrical Engineering from Kyushu University, Japan in 1988, 1990 and 1998, respectively. He has been a researcher of MITSUBISHI Electronics Corporation Japan for several years and has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan and Computer Center, Kobe University, Japan. He's area of interest is Future Internet and Next Generation Internet, Multimedia Communication and Processing, Multicast/IPV6/QoS, Human Communication over Internet and Active Network. He is a member of WIDE, ITRC, GENKAI, HIJK project and Key person of Core University Program on Next Generation Internet between Korea and Japan sponsored by JSPS/KOSEF.