A Parsimonious Monitoring Approach for Link Bandwidth Estimation within SDN-based Networks

El-Fadel Bonfoh, Samir Medjiah, Christophe Chassot

A Parsimonious Monitoring Approach for Link Bandwidth Estimation within SDN-based Networks

El-Fadel Bonfoh
LAAS-CNRS
University of Toulouse, INSA
Toulouse, France
efbonfoh@laas.fr

Samir Medjiah
LAAS-CNRS
University of Toulouse, UPS
Toulouse, France
medjiah@laas.fr

Christophe Chassot
LAAS-CNRS
University of Toulouse, INSA
Toulouse, France
chassot@laas.fr

***Abstract*—Resource monitoring is a key task in network management. The concept of Software Defined Networking (SDN) allows taking benefits of the advantages of both active and passive monitoring techniques. However, this monitoring has a cost, hence the importance of the selection of the "key" switches to be interrogated and their polling frequency in order to reduce monitoring cost. This cost is expressed here in term of computing time. Monitoring of links can be used to determine the available bandwidth on each link, with the aim to meet the applicative QoS requirements based on appropriate routing. In this context, this paper first provides a formulation of the problem of choosing key switches as a vertex cover problem and proposes a heuristic method to solve the formulated problem. It then provides an implementation and a performance evaluation of the proposed algorithm within the Floodlight SDN controller. These performances are compared to those of the currently existing Floodlight monitoring module. Finally, we present one application of our proposed monitoring.**

Keywords — Network monitoring; Software Defined Networking; QoS; Vertex cover problem.

I. INTRODUCTION

Monitoring network resources is essential to perform network operations such as anomalies detection. It consists of measuring the activity of the different elements of a network (switch, router, link, ...). These measures are used, among others, to ensure proper data routing and better load distribution. Historically, there are two types of monitoring: *passive* monitoring and *active* monitoring [1].

Passive monitoring consists of measuring or deducing performance-oriented criteria of an entire network from the observation of the state of the routers and switches that compose this network. The "real" traffic is then little impacted by that kind of the monitoring. However, such a performance measurement approach is tricky to deploy; it also poses safety problems when observing the state of a device as it requires the installation of an agent as in SNMP [2].

With active monitoring, additional packets that perform measurements are injected into the network. For example, the famous *ping* [3] command sends ICMP [4] packets across the network to test the reachability of a machine [1]. If this technique has the advantage of being easily deployable, it has the disadvantage of increasing the traffic load.

The concept of *Software Defined Networking* (SDN) [5] allows taking benefits of the advantages of both passive and active monitoring techniques thanks to a centralized controller. By having a global view of the network, this controller is able, without installing agents, to monitor the state of the switches and to retrieve information on active flows.

However, monitoring an SDN network has a cost that can be expressed in calculation time and/or in a number of messages exchanged between the switches and the SDN controller. Indeed, retrieving an information on a switch requires a request/response exchange, hence the importance of the choice of switches to be polled and their polling times. In this paper, we mainly focus on the former problem.

The selection of the "key" switches is essentially guided by the aim of the monitoring. *FlowCover* [6] proposes for instance to query only the switches that allow covering all network flows. *OpenTM* [7] develops several algorithms for selecting the switches to be interrogated ranging from a random selection to a selection of the least loaded switches.

To the best of our knowledge, there is currently no approach allowing to choose the key switches in order to cover all the links. However, link monitoring may be useful to determine the available bandwidth on each link, with the interest (for example) to meet the QoS requirements of an application based on appropriate routing.

In this context, the technical contributions of this paper are as follows:

- the problem of choosing the key switches is first formulated as a vertex cover problem [8],
- a heuristic method is then proposed to solve the problem,
- the corresponding algorithm is finally implemented as a *Floodlight* [9] controller module; its performances are evaluated and compared to that of

an existing *Floodlight* module, which, monitors all links, naively queries all switches in the network.

Based on a case study, Section II first explains the purpose of a link monitoring. The issue of choosing the key switches is then formulated as a vertex cover problem. Finally, an algorithm is proposed and its complexity is discussed. In Section III, we evaluate the performances of our proposal, implemented as a *Floodlight* module, and we compare it to an existing *Floodlight* solution. We end section III by a presentation of a link monitoring application. Section IV concludes this paper.

II. Proposed Approach

A. Problem formulation

The aimed goal is to minimize the number of switches to be polled to continuously monitor all links of the network.

Let us consider the network of Figure 1. It clearly appears more beneficial, in term of computing time and/or a number of request/response messages, to interrogate only switches S3 and S4 rather than (for instance) interrogating all switches of the network. Indeed, S3 and S4 cover the 6 links of the network. S3 and S4 are here the key switches of the Figure 1 network.

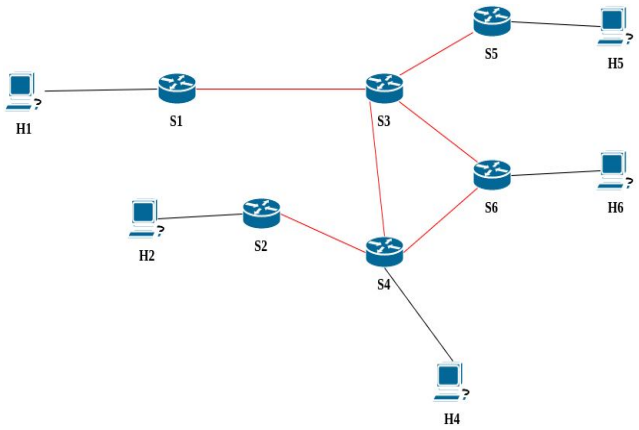

Fig. 1: The case study
(NB: only the links between the switches are considered)

The network is modeled by an undirected graph $N = (S, L)$ where:

- $S = \{s_1, s_2, ..., s_n\}$ is the set of switches,
- $n = |S|$ is the number of switches in the network,
- $L = \{l_{ij}\}_{i,j \in [1, n]}$ is the set of links of the network.

Let x_{s_i}, $i \in [1, n]$, having value 1 if the switch is chosen and 0 otherwise. Let denote $deg(s_i)$ the degree of the switch expressed as the number of incident links (i.e. to which s_i is connected). We then search the set C of switches that cover all the links. The set C is called the *vertex cover* of the graph N.

Formulated as an integer linear program, the problem may be expressed as follows:

$$minimize \sum_{s_i \in S} x_{s_i}$$
$$subject\ to: \ x_{s_i} + x_{s_j} \geq 1 \ \forall \ l_{ij} \in L$$
$$with \ x_{s_i} \in \{0, 1\} \ \forall \ s_i \in S$$
$$and \ i, j \in [1, n]$$

This formulation is the integer linear program of a vertex cover problem known to be NP-hard [10]. Let call this formulation, problem (1).

B. The resolution algorithm

To solve efficiently the NP-hard problem (1), we propose a heuristic method inspired from Clarkson [11] consisting in:

- *selecting a switch of maximum degree,*
- *removing from the graph all the links that are incident to the selected switch,*
- *iterating until the graph no longer has links.*

This so-called *greedy algorithm* is described in Algorithm 1. With $n = |S|$, the main loop of the algorithm runs in $O(n)$ time units and it takes $O(log(n))$ time units to find a switch of a maximum degree in a sorted list of size n by binary search [12]. Therefore, the complexity of the algorithm is $O(nlog(n))$.

Algorithm 1: Key Switches selection

```
Input: N = (S, L).
Output: C.
00: begin function
01:     C ← ∅
02:     Lc ← L
03:     while (Lc ≠ ∅)
04:         Find s_i such that deg(s_i) is maximum
05:         C ← C ∪ {s_i}
06:         Lc ← Lc \ {l_ij}_{j ≤ n}
07:
08:     end while
09:     return C
10: end function
```

III. Experiments and Application

A. Implementation

Our solution, whose architecture is shown in Figure 2, has been implemented as a *Floodlight* [9] module in Java language. Floodlight is an open-source SDN Controller written in Java and containing many modules on which several SDN Applications can be built. In this paper, we mainly use four Floodlight modules: *Switch Services*, *Thread Pool*, *Timer* and *Link Discovery*.

The polling time is determined in two ways: either periodically or on an event (such as a link adding, a link deleting, etc.). The *Link Discovery* module periodically sends LLDP [13] packets across the underlying network. These LLDP packets make it possible to notify the *Enhanced Statistics Collector* module (Figure 2). Each time a notification occurs or following a fixed period, the determination of the key switches is re-performed.

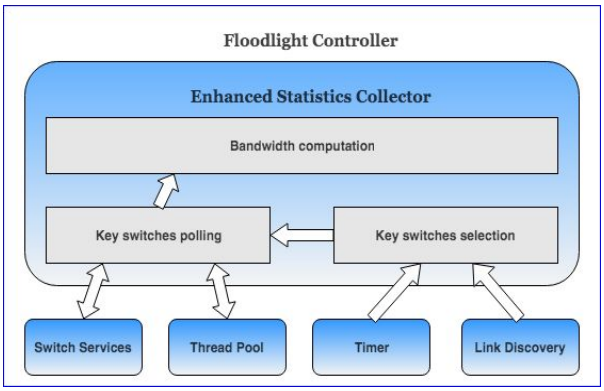

Fig. 2: Internal structure of our solution

The polling of the switches is carried out through the modules named *Switch Service* and *Thread Pool*. The *Switch Service*s module provides the necessary services (notification, identification, status, etc.) for the management of the switches managed by the controller. The *Thread Pool* module executes several processes in parallel and allows aggregation of the retrieved statistics.

The calculation of the available bandwidth is done between two times t_1 and t_2. At each polling time, we note the number of bytes consumed on the link. This value is made available through bytes counters [14]. The available bandwidth, whose formula is given in (2), is the difference between the bandwidth allocated to the link when it has been created (i.e. its initial capacity supposed to be static) and the quotient of the number of bytes exchanged on the link on the time elapsed between t_1 and t_2.

$$BW = Link\,BW - \frac{Bytes\ counter\ value\ at\ t_2 - Bytes\ counter\ value\ at\ t_1}{t_2 - t_1} \quad (2)$$

The difference $t_2 - t_1$ has an impact on the accuracy of the bandwidth estimation. The shorter it is, the better the calculated bandwidth is up to date, with however an increase in the monitoring traffic. It is, therefore, necessary to find a compromise between the precision of computed bandwidth and the traffic load introduced by the monitoring.

B. Evaluation

For the experiments, we built a testbed based on the *Mininet* [15] network emulator. The machine used for the test has a 3.20GHz Intel Core i5-65000 processor having 4 Gbytes of RAM.

The *Statistics Collector* module provided by *Floodlight* estimates link bandwidth by a simple polling of all switches. The computation time of this module has been compared to our *Enhanced Statistics Collector* module. The comparative graph of Figure 3 shows the evolution of this computation time as a function of the number of switches.

We can see that the computation time of our approach (in green on Figure 3) remains lower than the computation time (in red) of the basic module *Statistics Collector* of Floodlight. Moreover, we can notice a rapid growth of the *Statistics Collector* calculation time with the number of switches while the calculation time of our approach is not subject to the number of switches. Reducing compute time is crucial for real-time applications like bandwidth modification who is time-sensitive [16]. In the next section, we will show how useful can be to minimize monitoring computing time.

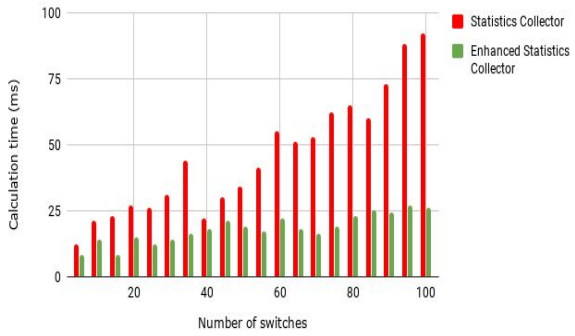

Fig. 3: Computation time as a function of the number of switches

C. Application: Bandwidth on Demand Service

Monitoring is prior to traffic engineering tasks like Bandwidth on Demand (BoD) Service. BoD allows users (private, enterprise, etc.) to request to change bandwidth [17]. BoD enables network operators to optimize per bit revenues thanks to an adaptation of the cost to the demand [16]. On the other hand, users have the opportunities to adapt their consumption to their needs and so to keep control of the cost.

Typically, user request a flow going from (Src, Dst, BW_{cur}) to (Src, Dst, BW_{req}), where Src and Dst are the end-hosts implied in communication, BW_{cur} is the current bandwidth and BW_{req} is the new bandwidth required by the user. The decision of accepting or rejecting the request for bandwidth allocation is made by the admission control module which bases his decision on the information made available by monitoring module (see Figure 4 algorithm).

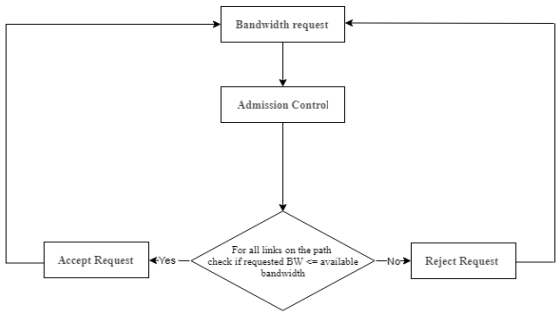

Fig. 4: Bandwidth allocation algorithm

During a duration time T, we simulate a real traffic with several users requesting bandwidth allocation. Users requests denoted by random variable X is exponentially distributed [18] with rate parameter λ fixed to ½. Flow removal denoted by the random variable Y has a triangular distribution [19] with lower limit $\alpha = 1$, upper limit $\beta = 2$, and mode $\gamma = 3$. The simulation results are shown in the Figure 5a and Figure 5b. Through this simulation, we can observe that reducing the monitoring computing time is helpful to improve the accepted connection hit ratio and therefore increase the occupation rate of links.

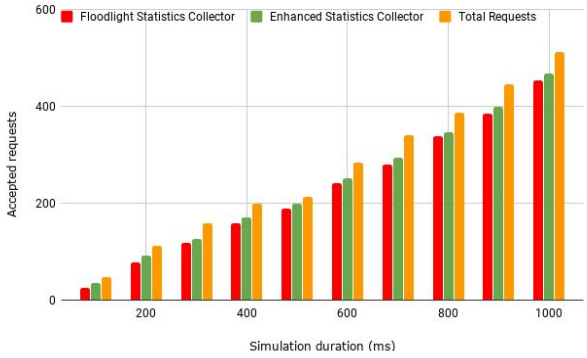

Fig. 5a: The number of accepted request as function of simulation time

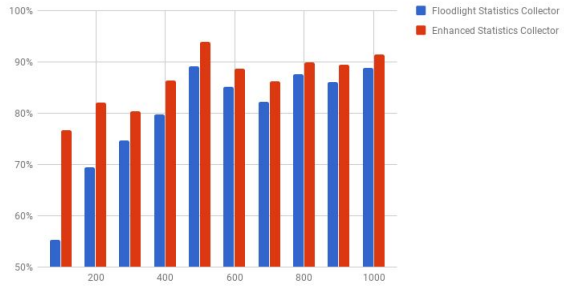

Fig. 5b. The hit ratio for Enhanced Statistics Collector and for Floodlight Statistics Collector

III. Conclusion

In this paper, we proposed a parsimonious monitoring aimed at reducing the number of switches to be interrogated to cover all the links within SDN networks. The selected switches, called "key" switches, have been obtained by solving a vertex cover problem. The heuristic method used to solve this problem allowed us to reduce the computational time needed to estimate the available bandwidth on a link. We finally showed how our proposed monitoring can leverage to optimize link utilization.

References

[1] N. L. M. Van Adrichem, C. Doerr, F. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks", *IEEE NOMS*, 2014.

[2] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1157 (Historic), Internet Engineering Task Force, May 1990.

[3] M. Muuss, "The Story of the PING Program", U.S. Army Research Laboratory, December 1983.

[4] J. Postel, "Internet Control Message Protocol", RFC 792, Internet Engineering Task Force, September 1981.

[5] B. A. A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present and Future of Programmable Networks", IEEE Communications Surveys and Tutorials, Vol. 16, No 3, 2014, pp. 1617-1634.

[6] Z. Su, T. Wang, Y. Xia, M. Hamdi, "FlowCover: Low-cost Flow Monitoring Scheme in Software Defined Networks", IEEE Global Communications Conference (GlobeCom 2014).

[7] A. Tootoonchian, M. Ghobadi, Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks", PAM 2010.

[8] D. Avis, T. Imamura, "A list heuristic for vertex cover", Operations Res. Lett., 35 (2007), pp. 201-204.

[9] "Project Floodlight", http://www.projectfloodlight.org/, accessed 2017-12-17.

[10] V. V. Vazirani, *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

[11] K. Clarkson, "A modification to the greedy algorithm for the vertex cover problem" IPL, vol 16:23-25, (1983).

[12] D. Knuth, *Sorting and Searching.* The Art of Computer Programming. 1998.

[13] T. Jeffree, "Station and Media Access Control Connectivity Discovery", IEEE Standard 802.1AB, 2009.

[14] "OpenFlow switch specification 1.3.0", https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf, accessed 2017-12-17.

[15] "Mininet", http://mininet.org/, accessed 2017-12-17.

[16] Open Networking Foundation, "Operator Network Monetization Through OpenFlow-Enabled SDN", https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/03/sb-network-monetization.pdf, accessed 2018-01-03.

[17] C. Yin, T.C. Kuo, T.Y. Li, M.C. Chang, B.H. Liao, "Mediating Between OpenFlow and Legacy Transport Networks for Bandwidth On-Demand Services", Asia-Pacific Network Operation and Management Symposium (APNOMS) 2014.

[18] A. Elfessi and D. M. Reineke, "A Bayesian Look at Classical Estimation: The Exponential Distribution", *Journal of Statistics Education* Volume 9, Number 1 (2001).

[19] C. Kokonendji, T. S. Kiesse, S. S. Zocchi, “Discrete triangular distributions and non-parametric estimation for probability mass function”, Journal of Nonparametric Statistics 19 (2007) 241–254.