Computer Networks 122 (2017) 43–55

Contents lists available at ScienceDirect

Computer Networks

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

Partial flow statistics collection for load-balanced routing in software defined networks

Hongli Xu*, Xiang-Yang Li, Liusheng Huang, Yang Du, Zichun Liu

School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, 230027, China

ARTICLE INFO

Article history:
Received 19 September 2016
Revised 19 January 2017
Accepted 7 April 2017
Available online 18 April 2017

Keywords:
Software defined networks
Partial flow statistics collection
Approximation
Primal-dual
Load balancing

ABSTRACT

In a software defined network (SDN), it is usually required to frequently collect state/statistics of all the flows, which may result in large overhead on control links. To reduce the flow re-routing overhead, we perform load-balanced routing using the traffic knowledge by carefully taking flow statistics collection. A key challenge for achieving effective almost-optimal load-balanced routing with less overhead relies on the quality of flow statistics collection. To address this challenge, we propose a *partial flow statistics collection (PFSC)* problem, in which we need to inquire statistics of flows from a subset of switches such that the flow recall ratio on *every* switch is at least a given value $\beta \in (0, 1]$ while minimizing the number of queried switches. We prove that the PFSC problem is NP-Hard and present an algorithm based on primal-dual with an approximation factor $\frac{f}{\beta}$ in most situations, where f is the maximum number of switches visited by each flow. To further reduce the overhead, we design an *adaptive* flow statistics collection mechanism, as a complementary scheme for PFSC, based on link load similarity measurement. We implement our partial flow statistics collection algorithm and a load-balanced routing method on a testbed platform. Our extensive experimental and simulation results show that our methods can reduce the overhead by 56% compared with the previous collection method while preserving a similar routing performance (with peak-load ratio increased by ~3%).

1. Introduction

The software defined network (SDN) [17] is an emerging networking paradigm that separates the control and data (or forwarding) planes on the independent devices. In general, an SDN comprises two main components: SDN controllers and SDN switches. More specifically, one or several controllers constitute the control plane of an SDN, and determine the forwarding path of each flow. A set of switches constitute the data plane of an SDN, and response for packet forwarding and traffic measurement of each flow. Since the controller is able to provide centralized route control for all flows, an SDN helps to improve the network resource utilization and alleviate link congestion compared with traditional (or non-SDN) networks [20].

Many previous works, *e.g.*, [18,21], have shown that the efficient route configuration depends on the current workload (or the traffic intensity of each flow) in a network. Without accurate traffic knowledge or only with outdated/inaccurate traffic knowledge, the controller just blindly performs routing for incoming flows, which may result in significantly suboptimal networking performance, such as load imbalance. We should note that, even though the controller does not know the traffic intensity of a new-arrival flow beforehand, the controller can adjust the route of this flow after the flow traffic intensity is measured on a switch. Therefore, a global view of flow intensity is instrumental to effective routing of flows.

In an SDN, switches are able to measure different per-flow traffic statistics, including packets, bytes or duration, so that accurate flow intensity can also be derived. Thus, to provide efficient routing for each flow, it is necessary to obtain the flow traffic statistics from switches. OpenFlow [5] specifies two different approaches, *push-based* and *pull-based*, for flow traffic statistics collection. We will explain these two methods in Section 2.1. In many practical scenarios, the traffic intensity of some flows may vary dynamically [8]. To provide smarter route selection, the pull-based collection mechanism requires that the flow statistics should be collected frequently enough for efficient flow scheduling [12]. Accordingly, this will generate massive traffic overhead through control links between switches and the controller. For example, it takes 88 bytes for traffic statistics of each flow entry using the HP ProCurve 5406zl switch [1]. Then, reading the statistics for 16K exact-matched rules supported on the 5406zl would return 1.4 MB; doing this twice per second would require about the

* Corresponding author.
E-mail address: xuhongli@ustc.edu.cn (H. Xu).

http://dx.doi.org/10.1016/j.comnet.2017.04.028
1389-1286/

22.5 Mbps bandwidth on a control link. The massive statistics traffic brings some disadvantages for an SDN. *First*, the massive traffic on control links may increase the delay and loss ratio of control commands. More seriously, the loss of control packets will result in running error or network paralysis in an SDN. For example, if one route update command is lost, some paths may be disrupted [18]. *Second*, when a switch reports the new-arrival message to the controller, this message may be processed until the statistics traffic has been processed, which will increase the blocking delay for the new-arrival flows and make the user experience worse. Thus, we expect to provide the efficient route in an SDN with minimum flow traffic statistics collection overhead.

Since one switch will measure the traffic of each forwarded flow in a network, a natural way for reducing the statistics collection overhead is to select a minimum subset of switches so that the traffic statistics information of *all* the flows in a network can be gathered by the controller [30,36]. This is essentially a minimum set cover problem. The previous works [32] also addressed the issue of switch selection for collecting the traffic statistics information of *all* the flows. As shown by our experimental results in Section 2.2, *the statistics information of partial flows also helps to achieve almost-optimal load balancing as that of all the flows*. That is, it is not necessary to collect the full flow statistics for efficient routing.

Based on collected flow statistics, we can perform load-balanced routing, so as to achieve the trade-off optimization between flow statistics collection overhead and route performance. A key challenge for achieving effective and almost-optimal load-balanced routing relies on the quality of flow statistics collection. To address this challenge, we propose a *partial flow statistics collection (PFSC)* problem, in which we need to inquire intensities of *all* flows from a subset of switches such that the flow recall ratio on *every* switch is at least a given value $\beta \in (0, 1]$ while minimizing the number of queried switches. Note that, to reduce the flow statistics collection overhead, one may say that another way is to collect statistics information of β-fraction of all flows in a network. As all flows distribute across a network, if we only require traffic statistics of fraction of all flows, the flow recall ratio on some switches may be smaller (even close to zero under the worst case). As a result, the dynamic traffic change through these switches may result in local load-imbalance or congestion. We also show that our flow statistics collection scheme helps to improve the route performance compared with only having fraction of all the flows in a network in Section 4. The main contributions of this paper are:

1. We prove that the PFSC problem is NP-hard and present an approximation algorithm, called SCPD, based on a primal-dual method. The SCPD algorithm achieves the approximation factor of $\frac{f}{\beta}$ in many situations, where f is the maximum number of switches visited by each flow in a network, and β is the required flow recall ratio on each switch.
2. To provide the route efficiency and reduce the flow statistics collection overhead, we then describe an adaptive flow statistics collection mechanism, based on (switch) port statistics knowledge, for efficient routing by exploiting the similarity among link loads.
3. We implement the proposed partial flow statistics collection method, and an efficient flow re-routing mechanism based on the collected flow statistics information on an SDN platform. The testing results and the extensive simulation results show that our proposed method helps to reduce the collection overhead by 56% compared with the previous pull-based method while preserving the similar routing performance where the peak-load ratio is increased by only 3 – 5%.

The rest of this paper is organized as follows. Section 2 describes the challenges in the PFSC problem. We propose an approximation algorithm based on primal-dual, design a re-routing method, and describe an adaptive collection mechanism in Section 3. We implement our proposed flow statistics collection algorithm on the SDN testbed, and report our extensive experiment/simulation results in Section 4. We review related work in Section 5 and conclude the paper in Section 6.

2. Preliminaries

We first introduce the network and flow models in an SDN. Then, motivated by the experimental observation, we define the partial flow statistics collection problem.

2.1. Network and flow models

An SDN typically consists of two device sets: a controller, and a switch set, $V = \{v_1, \ldots, v_n\}$, with $n = |V|$. These switches comprise the forwarding (or data) plane of an SDN. Thus, the network topology from a view of the date plane can be modeled by $G = (V, E)$, where E is a set of links connecting switches. Besides these links in the data plane, there are a set of control links connecting switches and the controller. We assume that each flow is unsplittable for the following reason. Though the splittable flow scheme [18] can improve the route performance compared with the unsplittable flow scheme, it needs additional management mechanisms, such as traffic amount division and multi-path packet order maintenance, etc. These operations will incur additional costs and accordingly decrease the resource utilization in an SDN.

Under the SDN architecture, flow tables play an important role to provide different functions, such as packet forwarding and traffic statistics, *etc.*. A flow table consists of many flow entries (*e.g.*, 16K on the 5406zl switch [1]), and a standard flow entry (*e.g.*, on the H3C S5120-28SC-HI switch) is illustrated in Fig. 1. The match fields and priority together identify a unique entry in the flow table. For simplicity, if the priority field is the same for all the entries, each flow entry can be identified only by match fields. When a flow arrives at a switch, the header packet will be matched with all the flow entries. There are two cases of the matching result. If one flow entry is matched, this switch directly takes the action specified by the instruction field in the matched entry. For example, one flow may be forwarded to the designated port. Otherwise, the switch reports the header packet of this flow to the controller. The controller shall determine the route for this flow, and set up a new entry on this switch. After that, the switch will count the flow traffic intensity through the counters field. The controller can send a Read-State message to retrieve the statistics of all the flows through a switch. To reduce the control overhead, the efficient selection of switches for flow statistics collection is the main focus of this paper.

OpenFlow [5] specifies two different approaches for flow traffic statistics collection. One is the *push-based* mechanism. The controller learns the start of a flow whenever it is setting up an entry in the flow table of a switch. When the switch detects the significant intensity change of a flow, the switch will report it to the controller. Moreover, OpenFlow allows the controller to request an asynchronous notification when a flow entry is removed from a switch, as the result of a controller-specified per-flow timeout [12]. However, several factors limit its application in practice. *First*, different from OpenFlow's specification, most current commodity switches do not inform the controller about the behavior of a flow before the flow entry times out. *Second*, it needs some additional requirements on both hardware and software to support the push-based flow statistics collection. For example, it requires