Computer Communications 116 (2018) 172–183

Contents lists available at ScienceDirect

Computer Communications

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

The show must go on: Fundamental data plane connectivity services for dependable SDNs

Michael Borokhovich[a], Clement Rault[b], Liron Schiff[c], Stefan Schmid[*,d,e]

[a] AT&T Labs - Research, USA
[b] TU Berlin, Department of Telecommunication Systems, Marchstrasse 23, D - 10587 Berlin, Germany
[c] GuardiCore Labs, Israel
[d] University of Vienna, Austria
[e] Aalborg University, Denmark

ARTICLE INFO

Keywords:
Local Fast Failover
Software-Defined Networking (SDN)
OpenFlow
Algorithms
Connectivity

ABSTRACT

Software-defined network (SDN) architectures raise the question of how to deal with situations where the indirection via the control plane is not fast enough or not possible. In order to provide a high availability, connectivity, and robustness, dependable SDNs must support basic functionality also in the data plane. In particular, SDNs should implement functionality for inband network traversals, e.g., to find failover paths in the presence link failures. This paper shows that robust inband network traversal schemes for dependable SDNs are feasible, and presents three fundamentally different mechanisms: simple stateless mechanisms, efficient mechanisms based on packet tagging, and mechanisms based on dynamic state at the switches. We show how these mechanisms can be implemented in today's SDNs and discuss different applications.

1. Introduction

1.1. Motivation

Software-Defined Network (SDN) architectures distinguish between the *data plane*, consisting of the forwarding switches, and the *control plane*, consisting of one or multiple software controllers. Out-sourcing and consolidating the control over the data plane elements to a software controller simplifies the network management, and introduces new flexibilities as well as optimization opportunities, for instance, in terms of traffic engineering [1,2].

However, indirections via the control plane can come at a cost, both in terms of communication overhead as well as latency. Indeed, the reaction time to data plane events in the control plane can be orders of magnitude slower compared to a direct reaction in the network [3]: especially for the recovery of failures, a slow reaction is problematic. Worse, the indirection via the control plane may not even be possible: a controller may be temporarily or permanently unreachable, e.g., due to a network partition, a computer crash, or even due to a malicious attack [4].

This is problematic today, as computer networks have become a critical infrastructure and should provide high availability. Over the last years, researchers and practitioners have put much effort into the design of more reliable and available SDN control planes. In these designs, redundant (and possibly also geographically distributed) controllers manage the network in a coordinated fashion [5–9].

Despite these efforts to improve the control plane performance, redundant controllers alone are not sufficient to ensure the availability of SDNs. First, the additional latency incurred by the redirection via the controller may still be too high, even if the controller is nearby. Moreover, if implemented inband, even with a distributed control plane, we face a bootstrap problem [10,11]: the communication channels between switches and controllers must be established and maintained via the data plane.

Accordingly, we in this paper argue that highly available and reliable Software-Defined Networks require basic connectivity services in the data plane. In particular, the data plane should offer functionality for *inband network traversals* or *fail-safe routing*: the ability to compute alternative paths after failures (a.k.a. failover). Moreover, it should support connectivity checks.

1.2. Challenges of inband mechanisms

We are not the first to observe the benefits of inband mechanisms [12–14]. Indeed, many modern computer networks already include primitives to support the implementation of *local fast failover*

* Corresponding author at: University of Vienna, Währinger Straße 29, 1090 Vienna, Austria.
E-mail address: stefan_schmid@univie.ac.at (S. Schmid).

https://doi.org/10.1016/j.comcom.2017.12.004
Received 3 December 2016; Received in revised form 15 July 2017; Accepted 11 December 2017
Available online 13 December 2017
0140-3664/

mechanisms: mechanisms to handle the failures in the data plane directly.

For instance, in datacenters, Equal-Cost Multi-Path (ECMP) routing is used to automatically failover to another shortest path; in wide-area networks, networks based on Multiprotocol Label Switching (MPLS) use Fast Reroute to deal with data plane failures [3]. In the SDN context, conditional rules whose forwarding behavior depends on the local state of the switch, have been introduced in recent OpenFlow versions [12,15]. Future OpenFlow versions are likely to include more functionality or even support maintaining dynamic network state, see for example the initiatives in the context of P4 [16] and OpenState [17].

However, implementing network traversals or computing failover paths is challenging, even with the possibility to define OpenFlow local fast failover rules. Mainly for two reasons:

1) The OpenFlow failover rules must be *pre-computed* and installed ahead of time, i.e., without knowledge of the actual failures.
2) Failover rules can only depend on the *local* state of the switch, i.e., the local link failures. A local rerouting decision may not be optimal, especially in the presence of additional failures occuring in other parts of the network.

1.3. The case for robust inband traversals

A local fast failover mechanism must essentially be able to perform a *network traversal* for failsafe routing: it should find a route from source to destination, despite failures. Such robust inband network traversals may also be a useful data plane service, e.g., to check network connectivity.

Ideally, a network traveral provides a *maximal robustness*, in the sense that any packet originating at s and destined to d will reach its destination independently of the location and number of link failures, as long as s and d belong to the same physically connected component.

Little is known today about the feasibility and efficiency of implementing robust inband network travesals in software-defined networks, the topic addressed in this paper. In particular, robust routing algorithms known from other types of networks such as MPLS, are sometimes impossible to implement without additional functionality at the switch, or inefficient (e.g., require large packet headers), and hence do not scale to large networks.

1.4. Our contributions

This paper studies the feasibility and efficiency of inband network traversals, a fundamental building block for more advanced data plane services of dependable SDNs, such as robust routing and connectivity testing.

We present a comprehensive approach, exploring conceptually different solutions which provide different tradeoffs in terms of overhead and performance:

1) *Stateless mechanisms:* We show that it is feasible to implement simple yet very robust data plane traversals using today's OpenFlow protocol. In particular, we present a simple stateless scheme which is maximally robust: a route from source to destination is found, whenever this is possible. The disadvantage of this scheme are the potentially high and unpredictable route lengths.
2) *Tagging mechanisms:* We present more efficient robust traversals in a more advanced network model, using *packet tagging*, as it is also supported in OpenFlow. Our OpenFlow model and approach may be of independent interest, as it introduces an interesting new graph exploration problem.
3) *Stateful mechanisms:* Given the benefits of maintaining state in the packets, we further explore means to introduce state in an OpenFlow network. We show that, maybe suprisingly, using packet tagging is not the only way state can be introduced in OpenFlow traversals. In fact, it is possible to implement simple state machines on the switches, using the standard OpenFlow protocol, and we will refer to the corresponding state as inband registers. Moreover, we present an interesting and novel mechanism to store state in the hosts attached to the network, in a completely transparent manner, using MAC addresses to encode paths.

Finally, we discuss applications for a robust inband network traversal, including robust routing and efficient connectivity checks.

1.5. Organization

The remainder of this paper is organized as follows. Section 2 introduces the necessary background on SDN and OpenFlow. In Section 3, we present and discuss different robust traversal algorithms, using packet tagging, and in Section 5 we show how to introduce state in switches and hosts. In Section 6, we discuss applications. After reviewing related work in Section 7, we conclude with a discussion in Section 8.

2. Background and model

2.1. SDN and OpenFlow

Our work is motivated by the Software-Defined Networking paradigm, and especially OpenFlow, the predominant SDN protocol today. This section provides the necessary background on OpenFlow (focusing on the commonly used version 1.3). OpenFlow is based on a match-action concept: OpenFlow switches store rules (installed by the controller) consisting of a match and an action part. For example, an action can define a port to which the matched packet should be forwarded or change a header field (e.g., add or change a *tag*).

A new flow entry can either be installed *proactively* or *reactively*. In the reactive case, when a packet of a flow arrives at a switch and there is no matching rule, the *table miss* entry will be used. By default, upon a table miss, a packet is forwarded to the controller. Given such a *packet-in* event, the controller will create a new rule and push the new flow entry to this switch. The switch will then apply this rule to the packet. In the proactive case, flow entries are pushed to the switches ahead of time.

Each OpenFlow switch stores one or multiple *flow tables*, each of which contains a set of rules (a.k.a. flow entries). Flow tables form a *pipeline*, and flow entries are ordered according to *priorities*: A packet arriving at a switch is first checked by the rule of the *highest priority* in *table 0*: the fields of the data packet are compared with the match fields of that rule, and if they fit, some instructions (the actions) are executed; subsequently, lower priority rules are checked. Depending on the outcome of the table 0 processing, the packet may be sent to additional flow tables in the pipeline. Concretely, instructions can be used to define additional tables to be visited (*goto* instruction), to *modify* the set of to-be-applied actions (either by appending, deleting, or modifying actions), or *immediately apply* some actions to the packet. A meta-data field can be used to exchange information between tables. Part of the header can be inserted or removed from a packet via pushing and popping of labels and tags, e.g., of MPLS and Virtual Local Area Network (VLAN) fields.

In general, a packet can be matched against any of its header fields, and fields can be *wildcarded* and sometimes *bitmasked* (e.g., the meta-data field is maskable). If no rule matches, the packet is dropped. The use of multiple flow tables (compared to a single one) can simplify management and also improve performance.

Our robust traversal algorithms make use of *Group Tables*, and especially the Fast Failover (*FF*) concept introduced in OpenFlow 1.3. The group table consists of group entries, and each group entry contains one or more *action buckets*. For the group entries of the fast failover type, each bucket is associated with a specific port (or group), and only