Load Balancing in a Campus Network using Software Defined Networking

Ashkan Ghaffarinejad and Violet R. Syrotiuk
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ 85287-8809
e-mail: {ashkan,syrotiuk}@asu.edu

***Abstract*—Today, commercial load balancers are often in use, including in the production network at Arizona State University (ASU). One of the main issues such load balancers face is that they use a static scheme for load distribution. However, at particular times of the academic year, such as during course registration, the network exhibits significant variations in both temporal and spatial traffic characteristics. At these times, students experience much greater latency and become frustrated with the network service. To address this problem, our aim is to develop an SDN-based approach to load balancing to better cope with the traffic variation.**

I. Introduction

Software Defined Networking (SDN) has the potential to enable innovation in and evolution of computer networks. It is based on the principle of separating the control and data planes. The OpenFlow specification describes the information exchange between the two planes [3]. In this architecture, an OpenFlow switch contains a *flow table* consisting of flow entries. A *flow entry* is made up of fields on which incoming packets are matched, and actions to be applied upon a match. If there is no match, the packet is forwarded to a *controller*, which runs a program to handle the packet, and decides whether to insert, delete, or update flow entries in the flow table for subsequent packets matching the same fields. Statistics are collected on packet matches which may be used to make decisions.

Load balancing in enterprise networks is one potential application of OpenFlow. A binary tree is used to represent the space of all possible IP addresses in [7]. The ith level in the binary tree corresponds to the ith most significant bits of the IP address. The nodes in a subtree correspond to a prefix match on the path from the root to that subtree. Under the assumption that each IP address supplies equal load on the network, a tree representation is effective since at each level the load is distributed equally between the two subtrees. An algorithm is provided to minimize the expensive TCAM entries used to represent the tree.

Three OpenFlow algorithms are implemented and benchmarked in [6] to investigate whether a OpenFlow based load balancer can compete with commercial load balancers. The *random* policy sends a request to a random server. The *round robin* policy uses a circular queue to decide where to send a request. The *load-based* policy sends a request to the server with the lowest load, where load is defined as the number of pending requests. The results indicate that as the processing time per packet at the server is increased, the load-based policy achieves the best performance.

In Plug'n Serve, load balancing is studied in an unstructured network, i.e., where the client session is shared among all server instances [5]. The question addressed is whether adding more servers can improve the overall performance. Aster*x advances Plug'n Serve by performing load balancing on a larger scale network, in particular, over a WAN [4]. In addition, it supports client diversity. Aster*x has used GENI [2] to evaluate its proposed solution.

II. Proposed OpenFlow Load Balancer

Our interest is to develop an SDN-based load balancing solution for the ASU campus network. Fig. 1 shows ASU's current campus network architecture. The core of the network is two Cisco 6500s located in diverse campus locations to provide redundant 10 Gbps layer 3 connections to nine regional Cisco 6500 routers. Links from campus buildings connect to the regional routers at 10 Gbps or 1 Gbps depending on usage. Building switches consist of a mix of Cisco 6500s, 3750s, and 2960s, and provide 1 Gbps connectivity to end users.

Fig. 2 illustrates the architecture of our SDN testbed. It contains two Arista 7050 Series OpenFlow switches and numerous hosts. The hosts can be reconfigured as the research demands, e.g., as OpenFlow software switches or controllers, or to emulate web servers or compute nodes. A tap of the traffic flowing into the NAT boxes in ASU's production network is fed into this testbed, with appropriate filtering.

Our proposed solution shares elements with the distributed control plane of Kandoo [8]. It introduces a two level hierarchy of controllers consisting of a

978-1-4799-5120-8/14 $31.00 © 2014 IEEE
DOI 10.1109/GREE.2014.9

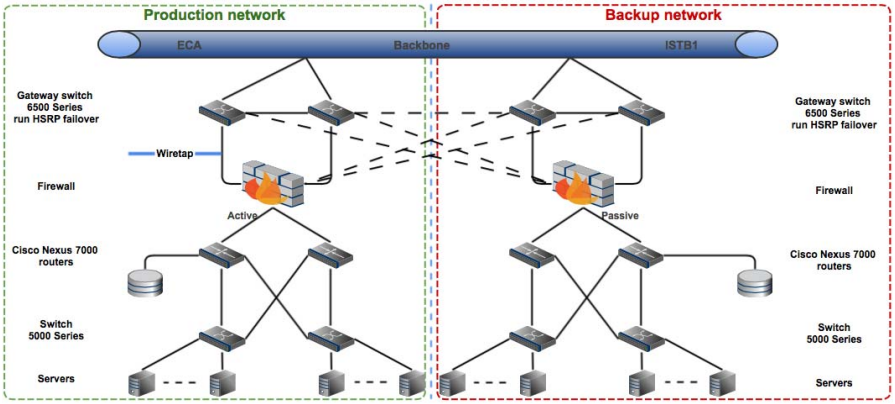

Fig. 1. ASU's current network infrastructure.

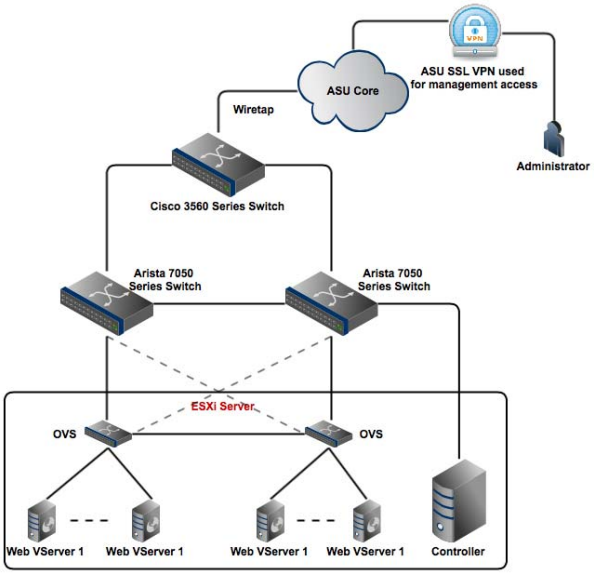

Fig. 2. ASU's SDN testbed.

root controller and multiple local controllers. Local controllers process events that do not require network-wide state information. Whenever an event requiring global information is encountered, it is forwarded to the root controller which maintains such state, and can best decide how to process the event.

A limitation of most distributed control plane designs is that the mapping between a switch and a controller is configured statically, making it difficult to adapt to traffic load variation. We propose to introduce an application that runs on each local controller. If the load on a switch exceeds an upper threshold, the application requests the root controller to migrate the flow to another local controller, similar to Elasticon [1]. This may require the root to spawn another local controller in order to support and rebalance the load. If the load on a switch drops below a lower threshold, flows may be migrated to another controller, followed by termination of a local controller. This design allows the number of local controllers to grow and shrink in response to the traffic characteristics. An study of how to optimize the local thresholds, taking into account the cost of creating and deleting local controllers, and the cost of migration, on the latency under a variety of traffic conditions is required. The use of GENI to evaluate this proposed solution is planned.

III. Summary

We propose to design and implement an OpenFlow load balancer for ASU's campus network. This is a work in progress; our aim is to show that such a solution may be less expensive, and more adaptive, than the commercial load balancers in use today.

References

[1] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella. Towards an elastic distributed SDN controller. In *HotSDN*, August 2013.

[2] Global Environment for Network Innovations (GENI). http://groups.geni.net/geni/wiki/GeniNewcomersWelcome.

[3] Open Networking Foundation. OpenFlow switch specifications. https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[4] N. Handigol, M. Flajslik, S. Seetharaman, N. McKeown, and R. Johari. Aster*x: Load-balancing as a network primitive. In *Architectural Concerns in Large Datacenters (ACLD'10)*, 2010.

[5] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-serve: Load-balancing web traffic using OpenFlow. SIGCOMM Demonstration, 2009.

[6] H. Uppal and D. Brandon. OpenFlow based load balancing. http://www.cs.umass.edu/~hardeep/cse561_openflow_project_report.pdf.

[7] R. Wang, D. Butnariu, and J. Rexford. OpenFlow-based server load balancing gone wild. In *USENIX*, 2011.

[8] S. H. Yeganeh and Y. Ganjali. Kandoo: A framework for efficient and scalable offloading of control applications. In *HotSDN*, August 2013.