A Method for Load Balancing based on Software-Defined Network

Yuanhao Zhou¹, Li Ruan¹, Limin Xiao¹, Rui Liu¹

¹. State Key Laboratory of Software Development Environment, School of Computer Science and Engineering
Beihang University, Beijing, China
Email: zhouyuanhao@gmail.com, {ruanli, xiaolm}@buaa.edu.cn, longruiliu@gmail.com

Abstract. The control plane is decoupled from data plane in the Software-Defined Networking (SDN) architecture, under which network intelligence and state are logically centralized and the underlying network infrastructure is abstracted from the applications. Load balancing is a significant technology and it can help save power and improve resource utilization in network. Since SDN makes it possible to manage the whole network with a centralized controller, with which we can distribute traffic more efficiently and easily. In this paper, the research background of OpenFlow-based SDN technology is summarized firstly. Then we will analysis current methods to achieve load balancing, and propose taking advantage of SDN to solve the problem.

Keywords: OpenFlow; SDN; event-driven controller; load balancing

1 Introduction

OpenFlow[1] is first posed to separate the data plane and control plane from each other among traditional network equipment. By OpenFlow, researchers could use programmable features to design and experiment with new network technologies and architectures in real network[2]. Referred to the principle of operating system, we can abstract a layer of Network Operating System. It provides unified administrate view and programming interfaces for upper applications. Therefore, users can define the logical network topology with software method, without concern for underlying network structure[3]. SDN represents a paradigm shift from traditional network to future Internet. Load balancing is a hotspot in network. By distributing traffic efficiently so that individual servers are not overwhelmed by sudden fluctuations, it can save power and improve resource utilization.

In this paper, we study the current load balancing method, and propose a load balancing method based on SDN. The structure of this paper is as follows. In Sect. 2 we will discuss related work. In Sect. 3 we present the design and implementation of our load balancing method based on SDN. We carry out the test results in Sect. 4 and we conclude the paper in Sect. 5, where we also indicate future research lines.
2 Related Work

2.1 SDN Architecture

OpenFlow provides an open protocol to program the flow table in different switches. A logic view of the SDN architecture[3] is shown in Figure 1. There are three layers in SDN architecture: (1) Layer of infrastructure; (2) Layer of controlling; (3) Layer of applications. The layer of infrastructure represents the underlying and forwarding devices such as switches. The layer of controlling provides both northern interface and southern interface. Controllers can maintain the state of network via southern interface. Users can make use of the northern interfaces to program applications in order to configure, manage and optimize the network resources flexibly.

![Software-Defined Network Architecture](image)

Fig. 1. Software-Defined Network Architecture

2.2 Load Balancing

Load balancing can be achieved not only by specialized software or protocol, but also by installing specialized gateway or load balancers. We will talk about load balancing from three aspects: network, client and server[4].

It’s common that DNS(Domain Name Server) maps the host name to an IP address in network. In order to ameliorate the balance, most servers adopt RR-DNS(Round-Robin DNS) scheduling algorithm which maps the host name to a group of IP addresses. Different users visit the same host name and will be directed to different addresses. This method will cause a problem of poor reliability and maintenance. Once one of the servers is down, it will take a long time to recover the failure server and all the requests sent to this server won’t be answered in time.
The solution of load balancing based on clients is that each client application will apply for balance information from servers and send requests to the lightweight server according to statistical information. Even though this measure can realize effective load balancing, it will bring additional flow expenses for collecting information from servers. In addition, this method is opaque.

Another solution is making use of splitter to organize and coordinate servers in network[5]. The splitter, as a proxy, receives all requests and distributes requests to the backend servers. The splitter can choose different forwarding level to achieve load balancing: (1)L4/3[5]; (2)L4/2[5]; (3)L7[6]; using transport layer or application layer to exchange packets and using network layer or data link layer to forward packets.

3 Design and Implementation

We design and implement load balancing according to PyResonance controller. PyResonance is Resonance implemented with Pyretic. Resonance is an SDN control platform that advocates event-driven network control[7]. It preserves a Finite State Machine(FSM) model to define a network policy. States represent different actions or behavior depending on network condition and transitions between states represent reactions to dynamic network events(e.g., intrusion detection, authentication of hosts, etc.). Based on different network events, operators can enforce different policies to the network using FSM model.

3.1 Finite State Machine and Policy

Each host will have a FSM and the states are mapped. Each state is related to a policy which decides the rule of flows’ forwarding in network. If the host’s state changes, the relevant policy will be forwarding to underlying devices.

Suppose that the topology of the SDN network is as shown in Figure 2. There are two policies in SDN network. And the host has two states A and B. Actually the controller maps the host to the state it is being in to achieve maintaining the FSM. If the host is in the state of A, the rule of flow forwarding is Policy 1 and the host will send requests to Server 1. On the contrary, when the state of the host turns from A to B, Policy 2 will be taken. In this way, the controller can control the flows in network and balance traffic among servers in network.
3.2 Load Balancing Model based on PyResonance

Based on the principle of Section 3.1, we now explain how to design the load balancing model using PyResonance. The flow chart is shown in Figure 3. There are mainly four modules to achieve balancing load in network:

- Load Balancing Module is responsible for monitoring the traffic in network. If a sudden fluctuation is in active, this module will send a JSON message which produces an event and contains the policy to the Event Handler Module to redistribute traffic in the network.

- Event Handler Module is used to receive state as JSON messages from Load Balancing Module and trigger the change of host state and the policy which are monitored by the function of catching the transition signal.

- Finite State Machine Module implements the Finite State Machine. It performs the state transition and assigns host to one of the states that are already defined with mapping. In addition, this module provides the methods used to get the list of hosts assigned to a given state.

- Policy Module includes the default policy and the corresponding policy of each state. The policy defines the rule of forwarding flows in network.

Fig. 2. Example Topology of SDN network
4 Experiment and Evaluation

In this section, we validate and test the load balancing method based on PyResonance controller. In our experiment environment, we use Mininet[8] to simulate the infrastructure of network. We use Mininet to simulate the hosts, servers and OpenFlow switches in our experiment. The topology we simulate is just like Figure 2 mentioned above. The operating system is Ubuntu 12.10 and we access external PyResonance controller to control our simulative network.

Initially, all links are set delay time is 0ms. The default policy is Policy 1. So Server 1 offers service to the host. We check the delay time with order “ping”, which uses ICMP packets. Then we set the delay time of link between Server 1 and OpenSwitch to 100ms. We continue to check the ICMP packets. Then the overload will trigger the change of host’s state. The controller changes the policy to Policy 2 and Server 2 will offer service to the host. The delay time of ICMP packets is shown in Figure 4. Notice that in each situation we just take 9 packets for examples and the first three are used for establishing connection.
Conclusion and Future Work

This paper presents a method for load balancing based on SDN. We have described the method and achieve every part of the load balancing module. Meanwhile, we validate and test the SDN method and give the result of distributing traffic in overload environment. However, our module is just a prototype. Actually, in real SDN network environment, it’s more complex for balancing traffic. In the future, we should consummate the load balancing module in controller with adding more functions such as authentication, IDS and so on.

References