Control-Plane based Failure Detection for Highly Available Software-Defined Network

Patrick Ngai [1], Tung-Yueh Lin [1], Yu-Sung Wu [1*], Chien-Hua Lee [2]
[1] Department of Computer Science, National Chiao Tung University, Taiwan
[2] Chunghwa Telecom Laboratories Chunghwa Telecommunication Co., Ltd., Taiwan
[1] clpngai.cs03g@nctu.edu.tw, tylin.cs03g@nctu.edu.tw, hankwu@g2.nctu.edu.tw
[2] leejenhwa@cht.com.tw

Abstract

To ensure high availability, redundancies such as active/standby controllers have been adopted on production SDN networks. However, the complex functionalities embodied by a SDN network make it difficult to ascertain the presence of failures. Not all failures will exhibit as component crashes to be captured by heartbeat-based failure detectors. We propose a novel failure detector for SDN network based on transparent monitoring of control-plane messages and real-time modeling of network state. By detecting deviation of the model, the proposed failure detector is capable of detecting a wide range of failures. The detector does not depend on specific SDN applications or controllers. We implemented a prototype for OpenFlow-based [3] SDN network and provide the preliminary results confirming the feasibility of a control-plane based failure detector.

Keywords: software defined network, failure detector, high availability, control plane

1. Introduction

The use of software defined network allows centralized micro-management of network devices and the overall network topology. The network management backend, which is commonly referred to as the SDN controller, inevitably becomes a single point of failure, and efforts have been made to ensure high-availably of the SDN controller through redundancy and persistency mechanisms.

Existing redundancy mechanisms for SDN controller backend commonly use heartbeats as the failure detection mechanism. For instance, HyperFlow [1] introduces the event-based control plane, which can coordinate multiple local controllers into a single logical controller. HyperFlow checks the health of the controller by listening for the advertisements on the

* Corresponding author: Yu-Sung Wu

controller's network control channel. Similarly, ONOS [2] supports the formation of controller cluster and uses heartbeats to monitor the status of controllers.

As the SDN controller embodies the operation logic of the whole network, it is inherently complex and potentially buggy. We expect that a majority of the network failures would not manifest as controller crashes to be captured by heartbeat-based failure detectors. We propose a novel failure detector for SDN network based on the reconstruction of network model from control-plane messages and consistency checks on the produced model. The detector' s coverage is as comprehensive as the controller's ability in managing the SDN network.

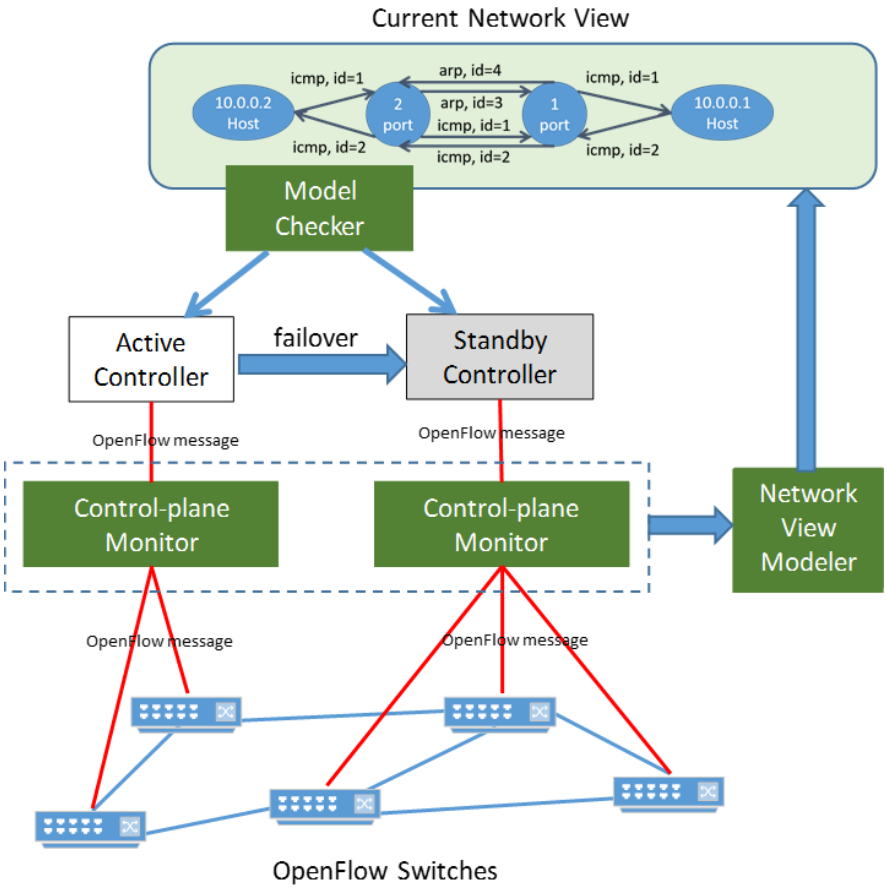

Figure 1. Control-plane based failure detection

2. Control-plane based failure detection

The architecture of the proposed failure detection system is shown in **Figure 1**. It consists of three components: control plane monitor, network view modeler, and model checker.

Control-plane monitors are responsible to forward controller/switch ‘s OpenFlow traffic to the modeler. On receipt of the messages, network view modeler creates network view to describe current network state. Model checker will constantly compare current network view against the expected network view. If after some time since the creation of the network, the active controller fails, the current network view thus deviates from the expected network view. If this deviation exceeds certain pre-specified threshold, the model checker will initiate a controller failover so that the standby controller will take control of the network.

2.1 Control-plane Monitor

In SDN network, the switches store flow rules in their flow tables. The rules represent the network states. The network view is inferred from the OpenFlow messages shown in **Table 1**.

Table 1. A brief description of OpenFlow messages meaningful to network view

OpenFlow messages	**Descriptions**
Feature Reply (FeatureRes)	It is used by the switch to tell controller its abilities.
Flow Modification (FlowMod)	It is used by the controller to modify OpenFlow switch’s state.
Flow Remove (FlowRemoved)	It is used by the switch to notify controller that a specific entry in its flow table is removed.

Feature reply is the message return by switch in response to a Feature request from controller. This often happens at the feature discovery phase of the handshake and is required to complete the handshake. Network view attempt to discover switches from this message. Network view can identify switches from the datapath-id field in the message which is a 64-bit value analogous to MAC address serving as a unique identifier for the switch.

FlowMod allows the controller to modify state of the OpenFlow switch. Nodes and edges can be inferred from this message. Network view will parse the match field to determine source, destination and action of the message.

FlowRemoved message lets us identify which nodes and edges need to be removed in network view. The cookie is used to identify the flow rule.

The use of the above is enough to construct the network view. Although it is possible to discover hosts from PacketIn and PacketOut messages. But Since flow rules will have source and destination field and its actions for match packet. Looking at the rule will have the same effect and avoid paring multiple packets which will be much more efficient.

```
1.  Loop
2.      Read TCP stream from message snooper socket
3.      If received TCP stream is a Feature reply message
4.          Extract dpid, datapath MAC and ports information
5.          Update network view with dpid, datapath MAC and ports
6.      If received TCP stream is a Flowmod message
7.          Extract dpid, source, destination, in_port, out_port
8.          Update network view with dpid, source and destination and
            link(source, in_port), link(in_port, out_port), link(out_port,
            destination)
9.      If received TCP stream is a Flow Remove message
10.         Extract dpid, and link represent by the message
11.         Delete the links with correct type in network view
```

Figure 2. Pseudocode of conversion

Figure 2 shows the conversion algorithm. For Feature Reply, we need the dpid, datapath MAC and ports. Dpid is used to identify ports which belongs to the same switch. Datapath MAC is a unique identifier for the MAC of the port. For Flow Modification, we need the source, destination, in port and out port. We need the in port and out port to identify which PORT node the HOST node should attach to. For Flow Remove, we just need the link information encapsulated in the message which are the source and destination nodes and link type and then delete the corresponding link in the graph.

The Control-plane Monitor's main job is to capture Openflow Messages from controllers and switches and forward those messages to Network View manager. The design goal of the Control-plane Monitor is to be extremely reliable to run 24/7 so if any controllers fail the monitor could still operate and feed information to the Network View manager to maintain the most accurate representation of the current view of the network. Much of the resiliency is handled by the framework implemented in Erlang [2] which is renowned in managing application level availability. The supervisor framework is used for monitoring child processes. When the Control-plane Monitor starts, the Erlang runtime will spawn a process called supervisor. The supervisor then starts the Control-plane Monitor manager process. This means that supervisor will monitor the Control-plane Monitor manager and any processes spawn by Control-plane Monitor manager process as shown in **Figure 3**. Each proxy process will handle a single connection between switch and controller as shown in **Figure 4**. If

connection at either end fails, the process will be killed and the Erlang supervisor restarts the process.

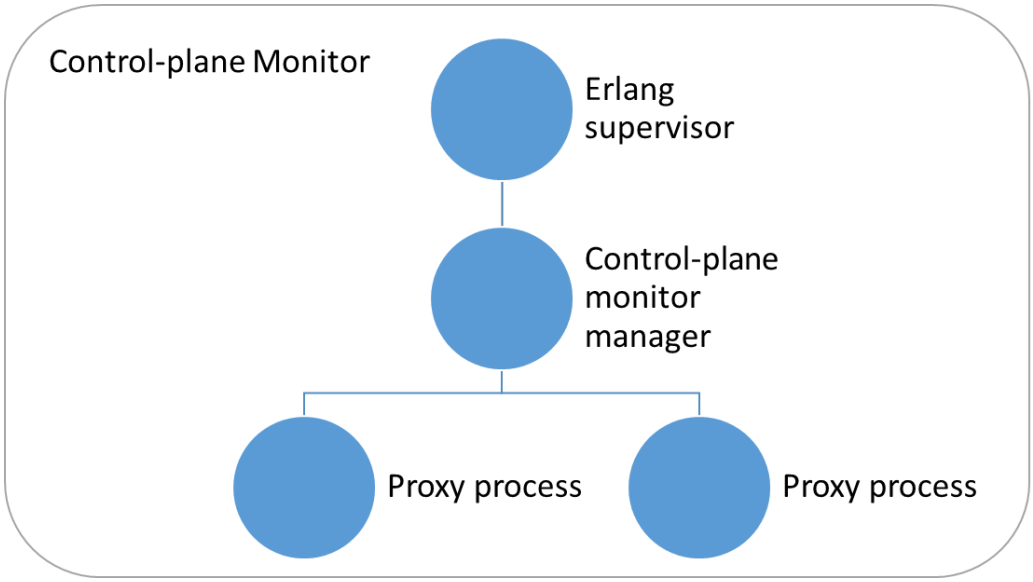

Figure 3. Process tree in Control-plane Monitor

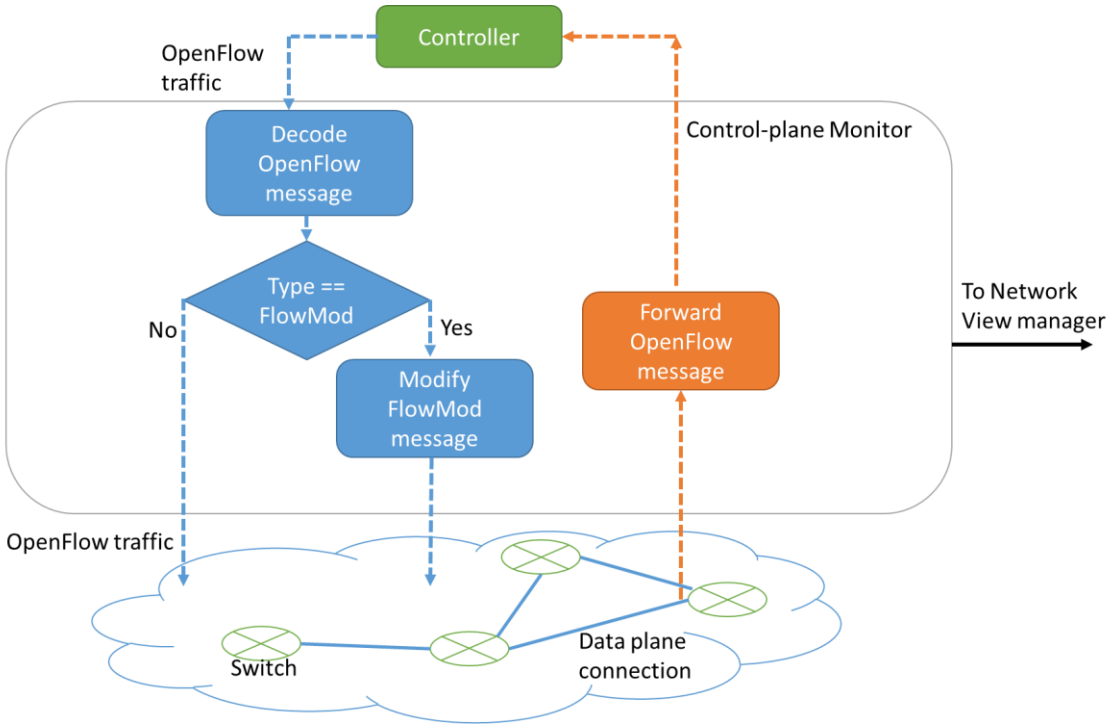

Figure 4. Control-plane monitor process flow

The Control-plane Monitor acts as a transparent layer between switches and its corresponding controllers. It creates an illusion such that the controller thinks it is

communicating with its switches and vice versa. Apart from maintaining controller/switch communication the Control-plane Monitor has to "eavesdrop" and modify OpenFlow messages.

Data packets send from controllers and switches vary in size and depends on the protocol version they use. One of the problem is handling multipart messages. Due to limitations such as MTU, long flows need to be segmented such that it can be fit into multiple messages. This typically occurs when the controller/switch tries to send a large amount of flows to the other side. The use of this is to improve utilization of network bandwidth. In order to parse these with maximal efficiency, when the monitor receive data from one side the data is concatenated with previous left overs. Then multiple attempts to decode it is made. It is then send to the network view manager if successful. Otherwise, any left overs are preserved for next iteration.

To be able to monitor the status of the network, certain kind of response is needed from switch to determine whether the flow rule is still intact or evicted due to timeout etc. The definitions of timeout in the OpenFlow protocol make this issue much more complicate. There are two kinds of timeout in the protocol, hard and idle timeout. Idle timeout is a value which determines how long the rules stays if no traffic is matched. Hard timeout dictates maximum time a flow can stay in the switch. Any assumption from controller alone will not reflect accurate state of the network. The Control-plane Monitor needs to modify FlowMod message to make sure the OFPFF_SEND_FLOW_REM [5] flag is enabled as shown in **Figure 3**. The presence of OFPFF_SEND_FLOW_REM flag will cause the switch send a FlowRemoved message to the controller when an explicit request is received or flow expiry. So the Control-plane Monitor can inform network view manager to update the network view.

2.2 Network View Modeling

We use graph to represent the network state, so there are nodes and links in the network view. Nodes may represent a host or a network port. A node may carry attributes such as the associated MAC address and IP address. Links represent the network connections among nodes. A link may carry attributes such as datalink type, network protocol type, timestamp and action type.

Figure 5 gives an example of network view from corresponding OpenFlow messages. First, we determine the nodes from the OpenFlow messages, which include port 1, port 2, host 10.0.0.1 and host 10.0.0.2. Second, we find the attributes in the OpenFlow rules, where we have protocol types ICMP and ARP. Third, we create a link for each OpenFlow rule. For

instance, for the first OpenFlow rule, we build the links along the path from host 10.0.0.2 to host 10.0.0.1 with the link attribute ICMP.

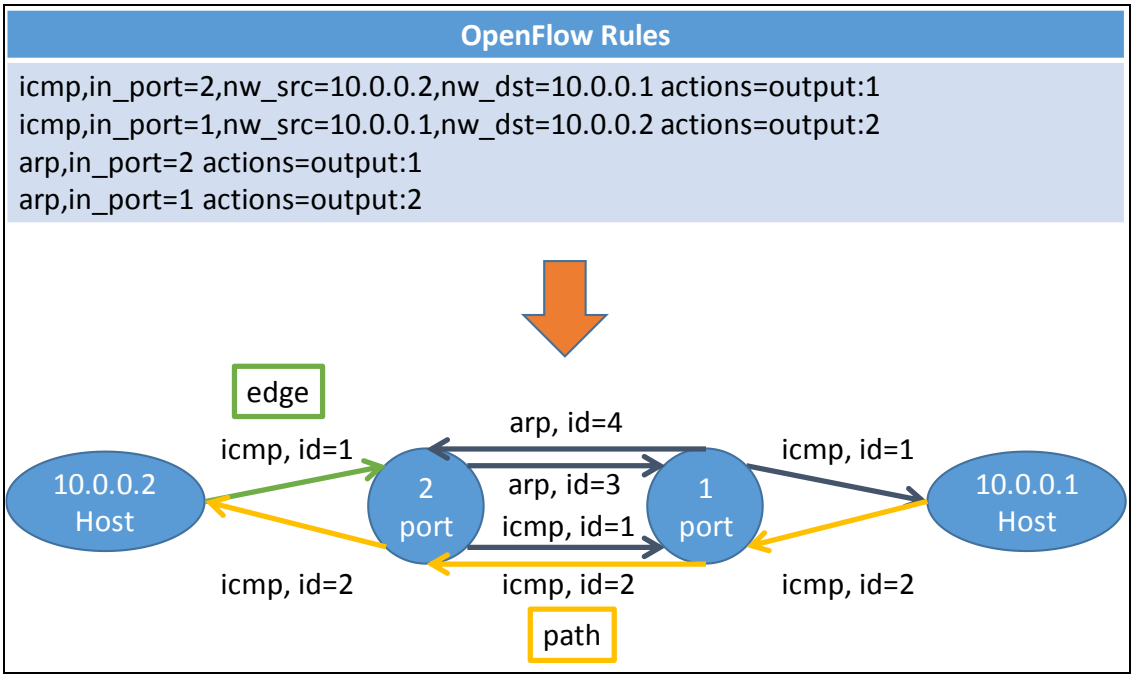

Figure 5. Network view model

3. Experiment Result

3.1 Control-plane monitor throughput and latency

We use Cbench [7] enhanced by Kulcloud to measure the throughput and latency of the Control-plane Monitor. This is because cbench is no longer being actively updated to support newer version of the OpenFlow protocol. Cbench simulates OpenFlow switches and have the switches sending Packet In messages to the controller. The packet In messages cause the controller to send back matching FlowMod to the Cbench's simulated switches. Then Cbench records the FlowMod returned by the controller. In the test, Cbench is configured to run 5 tests against 1) Ryu and 2) Ryu + Control-plane Monitor, each test last 10 seconds, with one simulated switch and 1000 unique MAC addresses. The results are averaged at the end. The tests are to evaluate how much degradation when the control-plane is added.

The controller and control-plane monitor runs on a machine with an Intel Core i7-2600 processor and 16GB of RAM. Cbench runs on a machine with an Intel Core i7-3770 processor and 30GB of RAM.

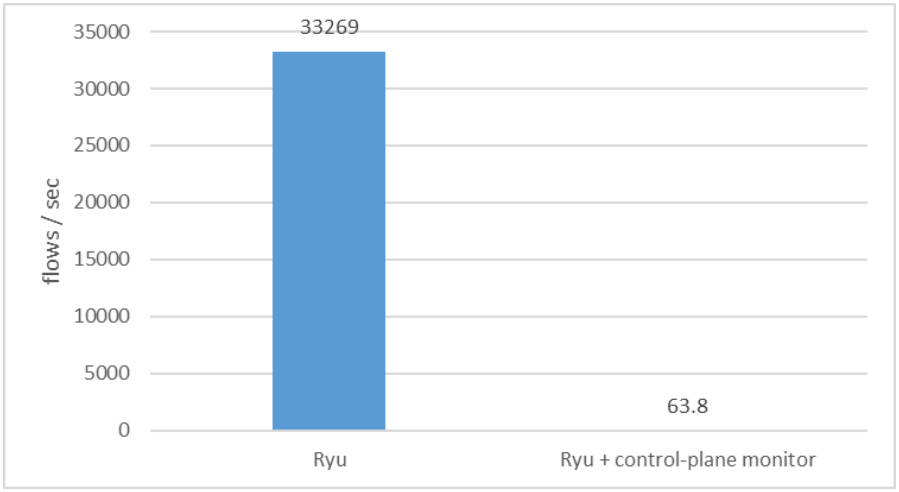

Figure 6. Control-plane Monitor throughput

Figure 6 shows the results of Cbench in throughput mode. We observed that with control-plane monitor attached, the throughput degraded from 33269 flows per second to 63.8 flows per second. There are a number of factors contributed to this observation. First, the Control-plane Monitor is implemented in pure Erlang using including the OpenFlow Protocol library. We believe that replacing the library with a C based library should improve the situation. Second, the way we decode the multipart messages add overhead. This is because we need to sequentially decode the message to see if it is a FlowMod message. Third, each proxy process in the Control-plane Monitor handle bi-directional message flow. So if we have southbound and northbound messages interleaved, then Control-plane Monitor will handle each one by one thus slowing down the throughput.

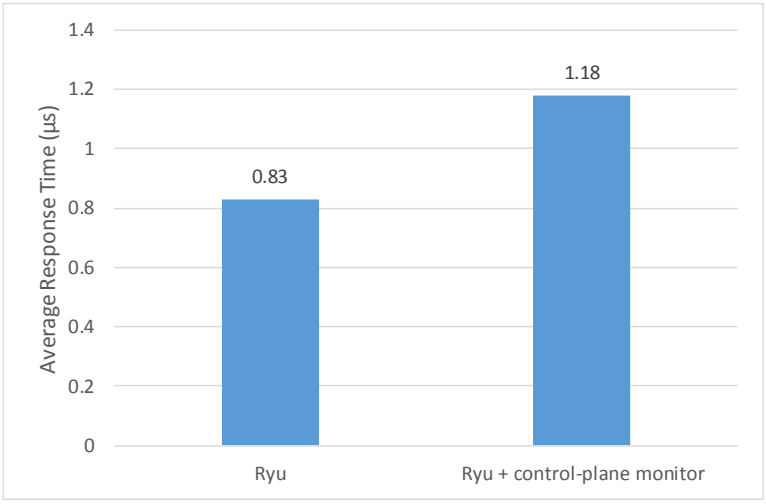

Figure 7. Control-plane Monitor Latency

Figure 8 shows Cbench latency mode results. We see that adding Control-plane Monitor adds an extra 0.34 microseconds. This is attributed the simplicity of tacks needed to perform in the Control-plane Monitor.

3.2 Network view modeling

We set up a testbed consisting of two Floodlight 1.1 controllers [8] and an Open vSwitch 2.5.0 [6]. We use Mininet 2.2.1 [4] to emulate a network of four host machines. The two controllers and the four host machines are connected to the Open vSwitch.

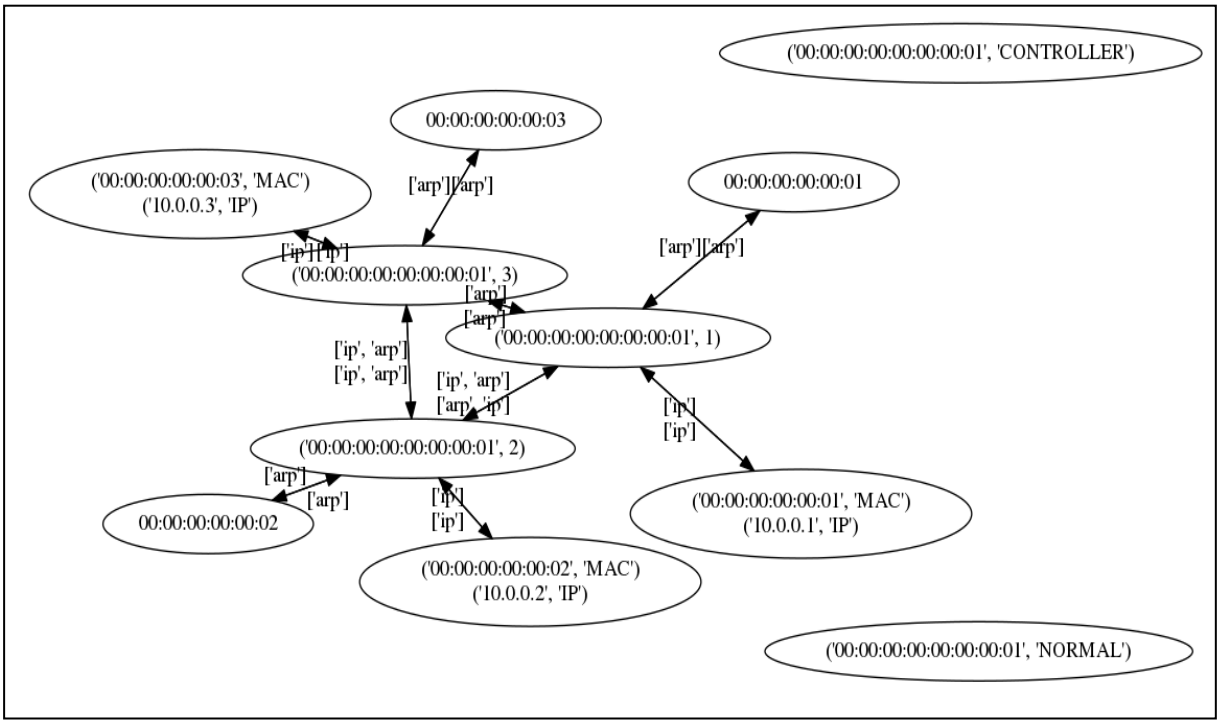

Figure 8. Network view for testbed

We also set up a firewall application on the controller to restrict the connectivity among the hosts according to **Table 2**. We then began to inject traffic to the network to trigger the installation of OpenFlow rules on the switch. The control-plane monitor then began to forward the OpenFlow messages to the network view modeler. The resulting network view is presented in **Figure 8**.

Table 2 Connection relation between hosts (V: has connection)

	Host1	Host2	Host3
Host1		V	
Host2	V		V
Host3		V	

4. Related Work

The idea of using proxy-based techniques to gather network information is not new and use of it mainly focus in network virtualization in multi-tenant networks. Some of the influential works are FlowVisor [9], OpenVirteX [1].

Flowvisor [9] is one of early attempts to use proxy to slice physical network to provide network virtualization. It examines messages pass through proxy to ensure policy enforced by corresponding controller map to and only affects certain part of the physical network. It ensures that policy created and issued by one tenant will not interfere with other tenants. This allows multiple virtual networks overlay on top of the same physical infrastructure. However, its design is unable to provide enough flexibility to let user to specify custom topology.

OpenVirteX is a work that builds on FlowVisor. It differs from FlowVisor in that OpenVirteX provides tenants the ability to virtualize a physical network and utilize every network resource available. To be able to spawn virtual networks without limitation, OpenVirteX maintains an internal virtual-to-physical mapping for resources like switch, address, port, link and network. This allows features where simply slicing components cannot provide such as topology customization, address scheme customization.

5. Conclusion

We propose a failure detector for software defined network based on real-time modeling of SDN network topology from the control plane messages. The detector can be applied to detect component failures, inconsistent policy implementations, and potential security attacks on the network control plane. We have built a prototype system, which includes a collection of control-plane monitors and a network view modeler. The control-plane monitors intercept the control plane messages between and SDN controllers and SDN switches. The network view modeler maintains the network view of the SDN network according to the intercepted control plane messages. Failure detection is conducted by detecting deviation in the network view. The experiment results have confirmed both feasibility and flexibility of the proposed solution.

Acknowledgment

The work is supported by ChungHwa Telecom under the project of SDN-enabled Cloud-based Wireless/Broadband Network Technologies & Services. The work is also supported in part by Ministry of Science and Technology of Republic of China under grant 104-2221-E-009 -104 -MY3.

References

[1] A. Al-Shabibi, M. D. Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: make your virtual SDNs programmable," in *Proceedings of The Third Workshop on Hot Topics in Software Defined Networking*, Chicago, Illinois, USA, 2014, pp. 25-30.

[2] Ericsson. *Erlang Programming Language*. Available: https://www.erlang.org/

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, pp. 69-74, 2008.

[4] Mininet. Available: http://mininet.org/

[5] Open Networking Foundation. OpenFlow Switch Specification. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

[6] Open vSwitch. Available: http://openvswitch.org/

[7] Openflow.org. *Oflops*. Available: http://archive.openflow.org/wk/index.php/Oflops

[8] Project Floodlight. Available: http://www.projectfloodlight.org/

[9] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, pp. 1-13, 2009.