IRIS: The Openflow-based Recursive SDN Controller

Byungjoon Lee, Sae Hyong Park, Jisoo Shin, and Sunhee Yang

Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea

{bjlee, labry, jshin, shyang}@etri.re.kr

Abstract— **SDN is a novel and promising networking technology that enables a software-based control over reactive packet-switching devices which query forwarding decisions for every flow: by controlling the answers to the queries, it is possible to 'program' the whole network according to a specific set of networking policies. That kind of programmability is backed by NOS (Network OS), which is usually called SDN controller. A Controller supports network-controlling applications to run on top of it by transparently dealing with the actual communication with the switching devices. However, as the controller is normally centralized, the scalability and availability issue is inevitable: a controller should provide reasonable performance to each of the switching device regardless of the network size, and guarantee the non-stop operation of NOS applications against system and software failures to prevent the whole network from halting on the control plane crashes. In this paper, we investigate the issues deeply, and introduce a comprehensive solution called IRIS (ETRI Recursive SDN controller platform).**

Keywords— **SDN, NOS, Controller, Scalability, Availability**

I. Introduction

The current network infrastructure known as the 'Internet' has settled for more than a decade. However, enterprises and carriers are starting to realize the limitations of current Internet with the rapidly evolving network technologies and the growing demands of the users; the current network is too inflexible to be applied to the dynamic environment, such as Cloud data centers, where the network configurations need to be created and modified dynamically according to the user request. To overcome this limitation, Software-Defined Network (SDN) was suggested. SDN is a new networking concept that decouples the control plane from data plane to centralize network intelligence in the controller, and to make data plane as simple as possible; as the data plane entities queries packet-forwarding decisions to the control plane, it is possible to simplify the architecture of the data plane entities. Openflow is the *de facto* standard protocol for the communication between the control and data plane.

One important aspect of SDN is that it is strongly required for the control plane to provide high availability against failures, and scalability for the large number of data plane entities because the control plane is centralized [1]. However, among the many SDN controller implementations, there is not any implementation that suggests a solution comprehensively solves the issues.

IRIS is an SDN controller platform that tries to solve the issues. To control the large number of Openflow-based data plane entities, IRIS incorporates a horizontally-scalable architecture that allows the dynamic addition of servers to the controller cluster, which would enhance the performance of the control plane on the fly. Besides, IRIS adopts a domain-based network abstraction method that allows hierarchical SDN network management which reduces network management cost, and transparent interoperation with the legacy IP networks which facilitates the carrier-grade SDN deployment on the real networks. In this paper, we cover the IRIS architecture, and explain the rationale behind the architecture.

This paper is organized as follows. In section II, we revisit the scalability and availability issues, and list several research & commercial outcomes suggesting solutions for the issues. In section III, we introduce the IRIS architecture, and present the principles that the architecture follows. In section IV, we cover the current development status of IRIS, and a few experimentation results. In section V, we conclude this paper.

II. Issues and Related Works

A. *Scalability*

In designing the SDN controller, the scalability issue is a critical concern. As more switching devices are added to a single SDN network, the controller performance for each switching device linearly degrades. It means that we need to put some limit on the number of devices that a controller handles to guarantee reasonable switching performance. If more switching devices are inserted to the network beyond the limitation, the controller responses to device queries would be delayed. Thus, it is strongly required for SDN controllers to provide horizontal scalability which enables performance enhancement by putting more servers in controller cluster.

B. *Availability*

As SDN controller is a centralized control plane of SDN network, the availability of SDN controller is critical. If SDN controller crashes, the whole SDN network would stop to forward packets unless hybrid switches are used: hybrid switches operates in the legacy L2 switching mode if the controller is not available. Thus, SDN controller should handle system and software failures transparently to guarantee non-stop network operation.

C. Related Works

ONOS [2] is an SDN controller which provides scale-out design by allowing the controller to run on multiple servers. Such design is more suitable to guarantee scalability and availability. ONOS is an open-source controller, but the detail is not yet available because it is still being developed.

NEC ProgrammableFlow controller [3] achieves high availability by placing redundant stand-by server beside the master server. However, ProgrammableFlow architecture does not provide any solution for controller scalability.

McNettle [4] is an extensible SDN controller system whose event processing throughput scales with the number of system CPU cores. It has been shown that McNettle performance is linearly enhanced by the number of cores. However, McNettle does not suggest any design for achieving the horizontal scalability, which is important to enhance the controller performance beyond the maximum that a single server is able to provide.

Onix [5] introduces a middleware that handles connectivity to SDN devices, state distribution among Onix servers, and load-balancing between servers. However, the Onix architecture is not thoroughly analyzed in the horizontal-scalability context.

HyperFlow [6] uses event-driven publish-subscribe messaging system to synchronize network state among all controllers by sharing network events. This enables each controller to make local decisions safely without the cooperation with other controllers.

D. Levin et al [7] emphasized that state distribution method between controller servers can impact the overall performance of a controller cluster.

ElastiCon [8] suggest a very effective and practical solution to the scalability and availability problems by making the controller pool dynamically grow or shrink according to traffic conditions and by making the load dynamically shifted across controllers using a novel switch migration protocol.

D. Lessons Learned

From the related works, we have learned following implementation-related lessons in achieving both high availability and horizontal scalability:

1) Choosing a correct state-sharing method among servers in the same controller cluster is important in achieving the horizontal scalability*:* if not carefully chosen, the method can be a performance bottleneck [7].

2) Switches should be able to be freely migrated among servers in the same controller cluster, or should not be migrated at all*:* of course the automatic migration strategy is a good option [8], but we think it's better to avoid the migration whenever if possible against various causes including server crashes and traffic load balancing.

3) Cooperation among controllers are required if a network spans to multiple remote geological locations*:* further, it might require cooperation among different network service providers, which should be handled by network OS (that is, controller). But this issue has been barely addressed.

III. IRIS ARCHITECTURE

IRIS (ETRI Recursive SDN Controller Platform) tries to solve aforementioned issues in the carrier-grade network context. Because carrier-grade network are composed of multiple networks with different transport technologies, one of our major concern is the seamless interoperation between the network edges where the difference should be resolved.

Further, a carrier-grade network is normally too large to be controlled by a single controller. That means we need to split the network into controllable units by following the inherent hierarchy within the network. Therefore, we tackle the scalability and availability issues differently from the contributions mentioned in Section II.

Firstly, we split the network into multiple 'unit SDN networks.' Each unit is easily interoperable with different transport mechanisms, easily controllable by a single controller cluster, and governed by a single network service provider.

Secondly, we tackle the scalability and availability issues within a unit SDN network by providing a horizontally-scalable middleware written on top of Openflow protocol.

A. Recursive network abstraction to reduce overall network complexity to manageable level

Considering the interoperability with other transport network, especially the legacy IP network, we have defined a unit SDN network with the following structure (Figure 1).

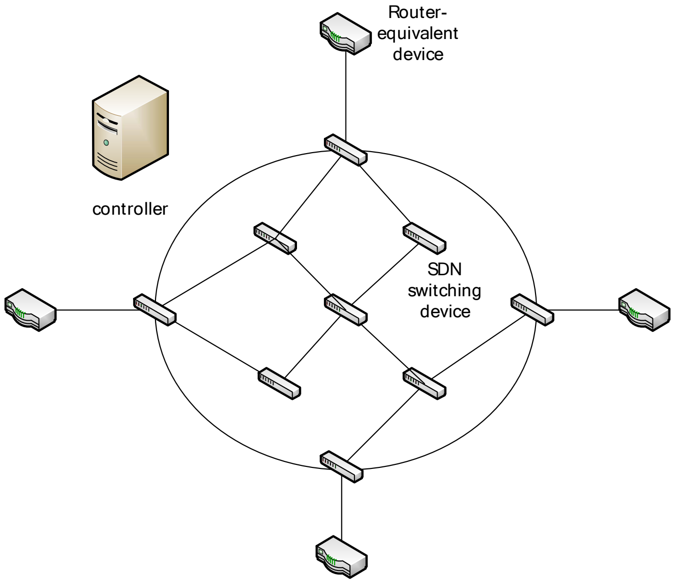

Figure 1. A Unit SDN Network

In the figure, the route-equivalent device is a device that is able to IP routing & forwarding. The device is introduced to facilitate the interoperation with IP networks. Therefore, a unit SDN network can have multi-homing relationships with multiple IP network. That means a packet from the host within the network might be directed across one of the router-equivalent devices. Thus, the controller should be able to decide a next-hop router to forward cross-network packets.

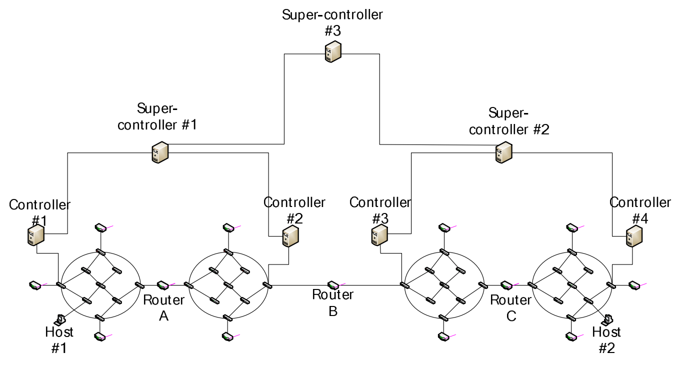

Figure 2. Interoperation among Unit SDN Networks

For the IP packet forwarding is correctly done at the router-equivalent devices, the IP address range assigned to the hosts within the same unit network is given to the devices. Using the information as a basic input to their routing protocol, the devices are able to exchange routing protocol packets (for example, OSPF packets) with each other. To support the exchange, each unit network is configured to pass routing protocol packets to all router-equivalent devices.

With the definition of unit SDN network, it is easy to define a large network which is composed of multiple unit SDN networks, where each unit is managed by a separate controller. Figure 2 is an example of such network. As depicted, controllers are placed hierarchically. The controllers cooperate with others to deliver packets to their destinations.

For example, let the host #1 try to send a packet to host #2. We assume the IP address of host #2 is already available to #1. Before sending IP packets, host #1 sends ARP first to find out destination MAC address, and the switch that receives the packet queries controller #1 how to handle the packet (the query includes the ARP packet within). As the IP address within the ARP is not in the same network, the controller should determine the next-hop IP router using the IP address as destination. However, controller #1 does not know any network information outside the same unit network. Therefore, it queries the super-controller #1 which router it should choose as a next-hop router. However, the IP address of the host #2 is also beyond the reach of it. Thus it also queries the super-controller #3 which router it should choose as a next-hop router. As the super-controller #3 knows the overall topology (we will explain how it acquires the topological information later), it returns the address of the router B. Using the address, the super-controller #1 also calculates the next-hop router A, and replies its address to the controller #1. Using the information, the controller #1 creates a reply for the ARP and push to the host #1. Using the information, the host #1 starts to send packets to their destination mac address which is assigned to the router A. As the packet forwarding mechanism within the same unit network is not a new technology, we do not cover the detail here. Above procedure is repeatedly applied whenever a packet cross the unit SDN network boundaries.

The topological information that we have assumed in the above explanation can be acquired in various ways. For example, for the unit SDN networks, the IP address range of the hosts attached to the network can be manually input to the designated controllers. The 'leaf' controllers pass the information to the super-controllers, and the super controllers build the topological database using the information. Based on the database, each controller is able to decide if it can make a local decision, or it should query to a super-controller.

As we placed the router-equivalent devices at the borders of the unit SDN networks, the interoperation with the legacy IP network is transparently handled because the legacy IP networks are always placed between two router-equivalent devices. It greatly reduces the complexity required for the interoperation. Further, because of the router-equivalent devices, we don't need to write complex applications that translate the cross-border L2 packets to the other unit networks.

The biggest advantage of this networking scheme is that the super-controllers that manage interoperation between unit networks see only the abstracted view of each unit network. As this abstraction is *recursively* done, the complexity of each child SDN network is hidden to super-controllers. They are able to focus on the cross-border forwarding policies only. Thus, super-controllers have much simpler architecture which is desirable to achieve high performance required to controllers for the bigger networks.

B. IRIS middleware for high availability and scalability

In designing a middleware which supports high availability and scalability for unit SDN network controllers, we have followed several principles as design guidelines.

1) A controller cluster is a big black box which gives the illusion of a system that never crashes: Thus, every server within the controller cluster should be easily added and removed without the reconfiguration that requires the other cluster components to be halted.

2) Both switches and applications should be able to use the box using only a single public IP address: Crashes within the controller cluster should be so transparent that every external entity that communicates with the cluster can use only one IP address to configure the communication channel.

3) The amount of code that should be newly introduced or fixed to integrate the middleware should be kept minimal: Contrary to that most of the middleware requires the users to rewrite the existing software using the middleware API, the IRIS middleware should not require the existing controller or application code to be modified to benefit from the higher scalability and availability.

Based on the principles, we have decided to implement the IRIS middleware, called HiSA (High Scalability and Availability), externally to the IRIS controller software. As most of the functionalities required to provide scalability and availability are about handling network and system event such as loss of connection to servers or processes, we have concluded that those are able to be implemented not touching the existing software which is written for one-server environment. The key technology that we have used to support the principles is Openflow.

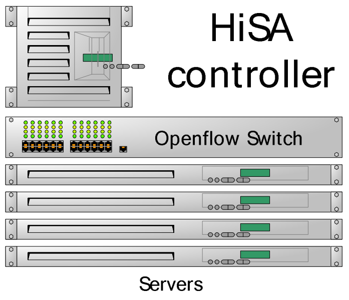

Figure 3. HiSA Middeware and IRIS controller cluster

Simply put, HiSA is a specialized IRIS controller that handles Openflow packets to IRIS controller cluster. It manages load balancing and crashes among physical IRIS servers belong to the cluster. Thus, the cluster is composed of HiSA controller, multiple physical IRIS servers, and an Openflow switch (HiSA switch) which connects the HiSA controller, the IRIS servers, and all the data-plane Openflow switches.

***1) Load balancing**:* When a new connection request packet comes in to the HiSA switch, the switch queries the HiSA controller to which IRIS server the new connection should be assigned. According to the decision, the new connection is forwarded to one of the servers. The connections from the IRIS applications from the outside of the cluster are handled in the same way. If a switch support Openflow protocol 1.2 or higher, the switch is allowed to try two connections to IRIS server using two public IP addresses assigned to the cluster. The first connection becomes a master connection, and the second one becomes a slave.

2) Detecting IRIS failures: As mentioned briefly, there is one IRIS controller in each physical IRIS server. The controller instance is a process running on top of a very small runtime called HI. HI uses two types of messages to let the HiSA controller learn the state of IRIS controller that HI hosts. On boot, HI sends a message 'RUNNING' that delivers the MAC address of the server to the Openflow switch. The packet is tossed to the HiSA controller, letting it to register the new controller into an active server list. The packet is periodically re-sent to the switch to allow the HiSA controller to detect IRIS physical server failures. If no packet is from a server for a specific amount of time, the HiSA controller removes the server from the active server list and engages server failover procedure. Besides, on detecting the controller software failure, HI sends another special message 'STOPPED' to the Openflow switch to remove the server immediately from the active server list and start the server failover procedure right away. Those two types of packets do not require any special communication channel between IRIS and HiSA controllers. Besides, HI switches the status of the controller from master to slave or vice versa on receiving a message 'SWITCH' from the HiSA controller. Further, HI cuts down the connections between a specific data-plane switch and the controller instance when a 'SHUTDOWN' message is received from the HiSA controller.

3) Handling IRIS failures: On detecting the server failure, the HiSA controller starts to move the data-plane switches which were originally connected to the failed server. If the switches supports Openflow 1.0, the lost connection is re-established by retries from the switches. However, if the lost connections are master connections for switches which supports Openflow 1.2 or higher, the HiSA controller failover the connections by switching the slave connections for the switches into masters using the SWITCH messages. The lost connections (which were originally the master connections) are re-established as slave connections.

4) Switch migration: On detecting a new IRIS server is added to the cluster, the HiSA controller migrates some of the connections to the new server. For connections to switches which support Openflow 1.0, the HiSA controller migrates them by sending SHUTDOWN messages to cut down the connections. The lost connections will be re-establishes to the new server by the connection retries from the switches. For master connections to switches which support Openflow 1.2 or higher, the HiSA controller first moves the slave connections associated with the master connections to the new server by cutting down the connections using SHUTDOWN messages. The lost slave connections will be re-established to the new server by switch retires. After that, the HiSA controller changes the moved slave connections into master connections using SWITCH messages: the existing master connections will be automatically changed into slave connections by the switches. For slave connections for switches which support Openflow 1.2 or higher, the HiSA controller migrates them by simply cutting down the connections using SHUTDOWN messages. The connections will be re-established by switch retries to the new server.

5) State-sharing method: For the state sharing between the IRIS servers, we use MongoDB [10]. It automatically replicates data across the server to provide high availability. Further, it scales horizontally without compromising functionalities. It is easy to query, and updates are done in-place providing contention-free performance. Thus, important information that should be shared between servers to allow each IRIS server to make local decisions is kept in the MongoDB.

One key advantage of this design is that it does not require switches to have all IP address of the physical servers in the cluster. The state-of-the-art switch migration protocol covered in [8] requires Openflow switches to configure IP addresses of all Openflow controllers to achieve scalability: to grow the size of the cluster, the configurations of all switches should be modified. Our suggestion only requires switches to have two representative IP addresses of the cluster. The two IP address of the cluster is broadcasted by ARP packets generated by the HiSA controller. Thus, the scalability of the IRIS cluster is not restricted by whatever reasons such as the number of public IP addresses that a cluster is able to export to data-plane switches, the number of Openflow sessions that a switch is able to maintain with Openflow controllers.

IV. IRIS Development Status and Some Basic Experiment Result

IRIS is currently in the incubating stage, but the prototype that supports Openflow specification 1.0 is ready and released as an open-source project, OpenIRIS [11]. OpenIRIS is implemented by Java, and NIO-based event processing engine which is similar to that of Beacon [12]. OpenIRIS provides a set of northbound API which is fully compliant to that of Floodlight, the most popular Java-based open-source Openflow Controller [13]. Thus, most of the Floodlight REST applications can be safely ported to OpenIRIS with few modifications. OpenIRIS equips almost the same set of core modules with Floodlight, but it guarantees much better performance. Currently, OpenIRIS is being extended to support Openflow specification 1.3.2. To support easy integration with other versions of Openflow, we use protocol stack generator which is very similar to Loxigen [14]. Though Loxigen only supports C and Python, the IRIS Openflow stack generator produces Java code which is very similar to OpenflowJ [15]. The Architectural decisions covered in the previous section will be fully integrated to OpenIRIS at the end of 2014.

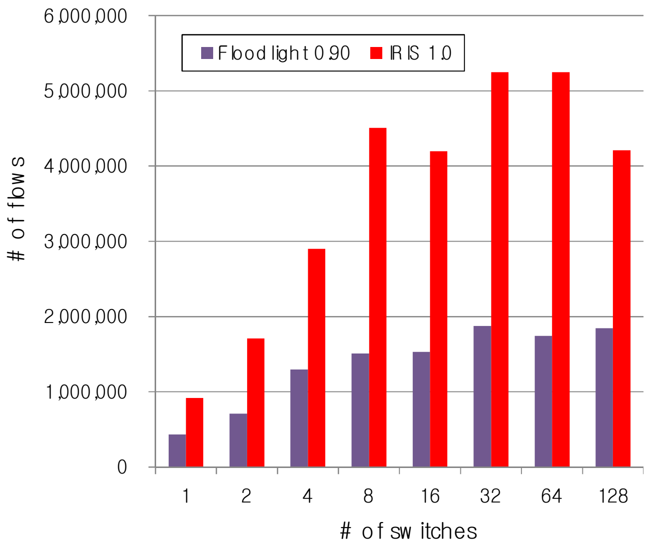

Figure 4. OpenIRIS Flow Processing Performance vs. Floodlight 0.90

Figure 4 shows the flow-processing performance of OpenIRIS 1.0.0 vs. Floodlight 0.90. As depicted, OpenIRIS 2.5 times outperform Floodlight due to its NIO-based IO event processing engine. The experiment is done using cbench with 100,000 hosts on top of Intel server with Xeon X5690 processor (3.47 GHz and 6 physical cores) and 64G RAM. We have used Ubuntu 12.04 LTS 64bit as OS. To our best knowledge, OpenIRIS is the fastest Java-based Openflow controller platform which is fully compliant with Floodlight.

V. Conclusion

IRIS is another Openflow-based SDN controller platform whose goal is to provide high availability and scalability suitable to control carrier-grade large networks. To achieve the goal, we take two different strategies: (1) to abstract network recursively to reduce overall network complexity to manageable level, and (2) to implement an Openflow-based middleware that provides high availability and scalability. In this paper, we explain the two strategies and the architectural decisions in depth. The extensive experiment result that shows the feasibility of our design will be given in the next version of this paper.

Currently, IRIS is being actively developed. IRIS which supports Openflow specification 1.0 is now open source (OpenIRIS) and publicly available [11]. A complete IRIS platform that incorporates all the features that covered in this paper will be available at the end of 2014.

We expect OpenIRIS to draw a lot of attention from research communities looking for an alternative to Floodlight. As widely known, there is no explicit plan by BigSwitch to release Floodlight which supports Openflow 1.3.1. To support the research activities ongoing on Floodlight and waiting for support of newer Openflow specifications, the next version of OpenIRIS will support Openflow specification 1.3.1 by the end of 2013.

Acknowledgment

This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013.

References

[1] http://highscalability.com/blog/2012/6/4/openflowsdn-is-not-a-silver-bullet-for-network-scalability.html, *OpenFlow/SDN Is Not A Silver Bullet For Network Scalability*.
[2] http://onlab.us/tools.html#os, *ONOS* by OnLab.
[3] http://www.nec.com/en/global/prod/pflow/controller.html, *ProgrammableFlow Controller* by NEC.
[4] A. Voellmy, and J. Wang, *Scalable Software Defined Network Controllers*, SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, Onix: A Distributed Control Platform for Large-scale Production Networks, OSDI'10, October 4-6, 2010, Vancouber, BC, Canada.
[6] A. Tootoonchian, and Y. Ganjali, *HyperFlow: A Distributed Control Plane for Openflow*, INM/WREN'10, April 27, 2010, San Jose, CA.
[7] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, *Logically Centralized? State Distribution Trade-offs in Software Defined Networks*, HotSDN'12.
[8] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, *Towards an Elastic Distributed SDN Controller, HotSDN'13, August 16, 2013, Hong Kong, China.*
[9] Open Networking Foundation, *Openflow specification 1.3.1,* September 6, 2012.
[10] http://www.mongodb.org/, MongoDB.
[11] http://openiris.etri.re.kr/, Project OpenIRIS.
[12] https://openflow.stanford.edu/display/Beacon/Home, Project Beacon.
[13] http://www.projectfloodlight.org/floodlight/, Project Floodlight.
[14] https://github.com/floodlight/loxigen, Loxigen: the Openflow stack generator.
[15] https://openflow.stanford.edu/bugs/browse/OFJ, OpenflowJ