Controller Failover for SDN Enterprise Networks

Vasily Pashkov, Alexander Shalimov, Ruslan Smeliansky
Lomonosov Moscow State University,
Applied Research Center for
Computer Networks
Moscow, Russia
pashkov@lvk.cs.msu.su, ashalimov@lvk.cs.msu.su, smel@cs.msu.su

***Abstract*—In SDN network based on OpenFlow a controller performs logically centralized control of enterprise network infrastructure, network policies, and data flows. At the same time the controller is a single point of failure which can cause a very serious problem (e.g. network outage) for network reliability and production use cases. To address this problem, we consider different active/standby strategies to provide a controller failover in case of controller failure. We propose a high-available controller (HAC) architecture, which allows to deploy a high-availability control plane for enterprise networks. We develop a HAC prototype to demonstrate the efficiency of our solution and also describe initial experimental results.**

Keywords—Software-Defined Networking; Control Plane Design; Controller Architecture; Fault-Tolerance; Redundancy.

I. INTRODUCTION

SDN is a new approach in networking, which significantly improves the programmability and flexibility of network management, simplifies the logic of network devices and reduces the cost of the network infrastructure deployment and the cost of its maintenance in comparison with traditional approaches [1, 2]. SDN separates the control plane and the data plane, which enables their independent deployment, scalability and maintenance. SDN involves centralized management of network infrastructure and data flows, but this approach can lead to network resilience and scalability problems.

The control plane can be deployed on one or several SDN controllers, which is running on dedicated servers [3]. The set of hardware and software components for providing of centralized network management in SDN is a control platform. The controller supports an actual global network view (GNV), which is stored in its network information base (NIB). Using network view controller applications control network devices states and data flows. That is why SDN network performance, reliability and scalability is defined by control platform characteristics.

In spite of the SDN advantages, one of the serious problems of SDN is that the controller is a critical point of failure and, therefore, the controller decreases overall network availability. A Controller failure can be caused by various reasons: failure of the server where a controller is running, the server operating system failure, power outage, abnormal termination of the controller process, network application failure, network attacks on the controller and many others.

In this paper we address to control plane for OpenFlow networks, as the one of the most promising implementations of SDN approach [4]. OpenFlow protocol is the open interface between the control plane and the data plane. The control plane in OpenFlow includes a controller (or NOS — network operating system) to monitor and control the state of OpenFlow switches, a set of network applications for network traffic and policy management, OpenFlow communication channels between the controller and switches and OpenFlow protocol for their interaction. OpenFlow controller can install rules in OpenFlow switches for data flows supporting predictive, reactive, and proactive or hybrid flow installation modes.

At the present time there are about 30 different OpenFlow controller implementations [5, 6]: NOX, POX, Beacon, MUL, Floodlight and the others. However most of them do not support the control plane restoration mechanisms in the case of controller failure. Only distributed control platforms Onix [7], Kandoo [8] and some proprietary controllers with OpenFlow 1.0 [4] support restoration procedure in case of a controller failure. Thus, a controller failure in the control plane of SDN/OpenFlow is a pressing issue.

An approach for improving the SDN control plane availability in case of a controller failure in the enterprise software-defined networks is presented in the paper.

In summary, in this paper the following points are presented:

- comparative analysis of the different active/standby strategies to provide a controller failover;
- a fault-tolerant control plane architecture for enterprise software-defined networks;
- a High-Available Controller (HAC) architecture that provides the network ability to fast recovery of the control plane;
- the control recovery procedure and the procedure for network view synchronization between active and standby controller instances;
- the HAC prototype implementation with supporting OpenFlow version 1.3.

II. BACKGROUND

A. Typical SDN controller architecture

A typical SDN/OpenFlow controller [6, 9, 10 and 11] includes:

- ***Controller core*** which handles and supports connectivity with switches and translates control protocol messages (e.g. OpenFlow) into internal controller events and vice versa.
- ***Controller network services*** which control, form and monitor network view, states of network devices, provide an interface (Northbound API) for controller applications. Usually network services include event dispatching, device managing, topology managing and the others.
- ***Controller network applications*** which configure network infrastructure and manage data flows to solve some business use cases.

The interaction between Network, controller services and applications is based on the publish-subscribe model.

B. Active/Standby strategies analysis for control platform

Let us consider the basic redundancy approaches to improve control platform availability for enterprise software-defined networks. The controllers can be active or standby mode in the control plane. An active controller directly receives and processes OpenFlow messages from network devices. A standby controller duplicates the functionality of the active controller, but receives and processes OpenFlow messages from network devices only in case of active controller failure. The number of standby controllers may be increased to tolerate more than one failure at a time. The primary controller for a network segment is a controller which configures network devices of its segment and installs the rules for data flows in this segment.

There are active/standby strategies and active/active strategies for controller redundancy. In case of active/standby strategies control platform has only one active primary controller. In case of active/active strategies control platform can have multiple active controllers. But in this paper we consider only active/standby strategies with one active primary controller in the control plane.

In case of primary controller failure the standby controller automatically takes over network infrastructure control and data flows management. This procedure is called ***controller failover***. ***Controller failback*** is the reverse procedure to failover. This procedure is used when the primary controller is restored.

The active/standby strategies based on the operational status of standby controllers (switch on/off and loading on/off before the start of work) and failover transparency are:

- no standby;
- cold standby;
- warm standby;
- hot standby.

«No standby» strategy. The control plane has a single active primary controller without standby controllers connectivity. In case of primary controller failure the network administrator manually resets the controller or replaces it. Thus, the control plane recovery time is significant and unpredictable and depends on the efficiency of support service.

«Cold standby» strategy. The control plane has an additional unloaded server connected to a server of the primary controller. The «cold standby» strategy uses automatically failover procedure. The standby controller is stateless. In case of primary controller failure a standby server runs the standby controller and its services and applications (including topology discovery service to form network view) from scratch. This strategy is preferable to use for stateless services and applications. Recovery time is determined by the controller start and time to restore the actual standby controller state.

In the case of cold standby strategy a redundancy hardware component is often unloaded, that is why it can be used for any optional extra work: for testing, debugging, maintenance and other services (e.g. testing of the new versions of controller network services and applications).

«Warm standby» strategy includes periodically primary controller state replication to standby controllers and automatically failover procedure. The «warm standby» strategy is usually provided by hardware and software redundancy. In case of primary controller failure the standby controller replaces a failed controller and continues to operate on the basis of its previous state. Control plane services for network devices are interrupted and some state is getting lost. The lost part of the control plane state is those state changes which were between the last state synchronization procedure and the primary controller failure.

«Hot standby» strategy includes full state synchronization of the primary and standby controllers and automatically failover procedure. No loss of the controller state provides the minimum recovery time. State of the primary controller is replicated to the standby controller for any change in it. In case of primary controller failure the standby controller replaces a failed controller and continues basing on the current state. The «hot standby» strategy is implemented by software and hardware redundancy.

TABLE I. COMPARATIVE ANALYSIS OF THE ACTIVE/STANDBY STRATEGIES

Criterion	**Active/Standby strategies**			
	No	***Cold***	***Warm***	***Hot***
Redundancy	hardware	hardware	hardware and software	hardware and software
Active controllers	1	1	1	1
Failover procedure	manually	auto-matically	auto-matically	auto-matically
State loss	complete loss of the state	complete loss of the state	partial loss of the state	without loss of the state
State and	no	no	regularly	up-to-date

Criterion	Active/Standby strategies			
	No	***Cold***	***Warm***	***Hot***
data syncro-nization				(any change)
Redundancy rate	1	1+N	1+N	1+N
Failover time	unpre-dictable	from minutes to seconds	seconds	from seconds to mille-seconds
Cost	no cost to low cost	moderate	moderate to high	moderate to high
Network user impact	high	moderate	low	none

C. Key metrics

Key metrics which characterize a fault-tolerant control platform are the following:

Controller redundancy degree is a number of standby controller instances included in the control platform. It determines the cost of the control platform and the number of failures that can be avoided.

Controller delay in the worst case is the maximum delay for processing the network device flow installation request by the controller which is attained in the control recovery process.

Controller failover time is the time during which network device requests can be lost due to absence of the primary controller in the network, i.e. the time during which the network is not the primary controller.

Thus, the SDN control platform should have controller redundancy degree at least one, delay in the worst case no more than 150 milliseconds as recommended maximum time delay for services. Failover time should be as low and close to zero.

D. Fault-Tolerant control plane requirements

To support redundant controllers the control platform must meet the following requirements:

- there are to be at least two servers;
- identical hardware and software server configuration;
- internal network between servers to decouple control platform communications from OpenFlow communication channels and for accessing to data store;
- each server must have access to SDN network segment with independent links;
- identical controller instances (with identical versions of controller network services and applications).

These requirements are due to the following reasons:

- to avoid single point of failure;
- standby controller must have sufficient computing resources for network infrastructure and data flows management in case of primary controller failure;
- standby controller must provide the same set of functions as the primary controller.

III. Proposed Approach

A. Proposal

Since we have previously discussed the active/standby strategies it is very important to define controller state.

Controller state includes states of controller services and applications, event queue state, controller network view and controller data. The state of the controller service or application includes values of internal service/application significant variables.

Service/application snapshot is a service/application state at a particular time. Controller snapshot is controller services and applications snapshots and current network view.

For solving the controller failure issue using active/standby strategies we need to define the basic modes for control platform: an initial mode which describes the order of launch controller instances, an operational mode which describes controller instances synchronization procedure and a primary controller failure mode which describes failure detection and failover procedures.

Initial controller mode. Running the primary controller of the control plane:

- The controller starts in accordance with the configuration file.
- The controller launches a timer for connecting standby controllers.

Running the standby controller of control plane:

- The controller starts in accordance with the configuration file.
- Standby controller establishes a connection to the primary controller via the internal control network between controllers.
- Standby controller requests a list of network services and applications of the primary controller and launches a similar set of applications and services.
- Standby controller requests the current Network view and network interfaces list for control channels connections, current states of network services and applications.
- Standby controller launches primary controller state monitoring service.

Operational controller mode. In this mode primary controller processes OpenFlow messages from network devices and controls network data flows, the standby controllers monitor the primary controller state and synchronize with it.

Controllers state synchronization includes:

- network view synchronization;
- controller network services and applications states synchronization;
- controller data synchronization.

In this paper we use two strategies for controllers.

For network view redundancy and synchronization we use hot active/standby strategy. Primary controller pushes up each network view change to all standby controllers.

For controller network services and application redundancy and synchronization we use warm active/standby strategy. Primary controller periodically or conditionally pushes up snapshots of services and applications to all standby controllers.

For controller data synchronization we use reliable shared data storage between controllers.

Primary controller failure mode. The control plane recovery procedure consists of two stages:

- **Failure detection stage.** Primary controller failure detection mechanism is based on the heartbeat. The main parameters are: heartbeat interval — the time interval between heartbeat messages, and dead interval — the time interval through which standby controller fixes primary controller failure.
- **Recovery stage.** The recovery stage starts after primary controller failure detection. It includes the following steps:
    - Defining a new primary controller. The new primary controller is a standby controller with the highest ID (or IP).
    - The new primary controller informs the other controller about its status change.
    - Controller network services and application restoration.
    - Control network interfaces up.

B. *High-Available Controller Architecture*

High-available controller (HAC) architecture is based on adding of additional cluster middleware between the controller core and controller network services and applications (see Figure 1).

To provide fault-tolerance of the control platform the HAC cluster middleware includes the following managers and services:

- ***Controller Manager*** to coordinate start/restart/stop controller network services and applications and up and down control interface for network devices connections.
- ***Cluster Manager*** to control the operation of the controllers cluster and distribute responsibilities (primary or standby) in accordance with the cluster configuration file.
- ***Sync Manager*** to control controller network services and applications synchronization between controller instances in the cluster.
- ***Recovery Manager*** to coordinate the recovery process (failover and failback) in case of controller instance failure in the platform.
- ***Message Service*** to provide control message distribution to other controller instances in the controller cluster.
- ***Event Service*** to provide filtering, distribution and processing to or from other controller instances.
- ***Heartbeat Service*** to monitor the operational status of the controllers and detects controller failures in the controller cluster.

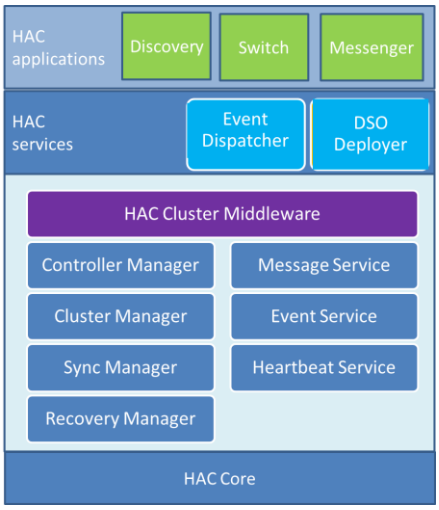

Fig. 1. High-Available Controller architecture

C. *Control Plane Design with HAC*

In order to avoid single point of failures in the control platform we propose the following design of the control platform (see Figure 2).

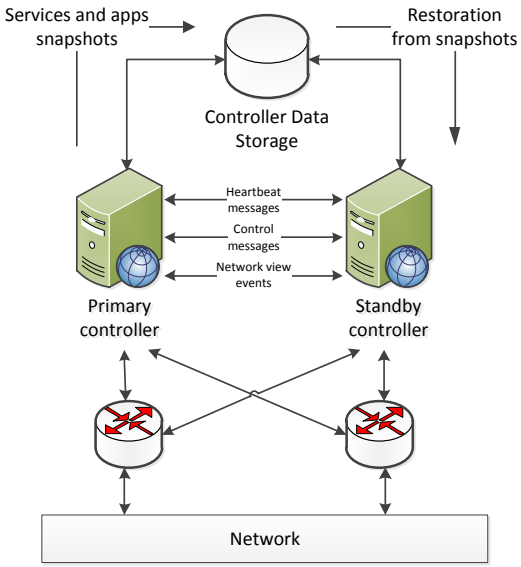

Fig. 2. Fault-tolerant control plane design with HAC controller

D. *HAC Imlementation*

Based on the review of modern open-source SDN control platforms [5] as a base controller for HAC controller we choose NOX13oflib from the laboratory CPqD [12]. This controller supports OpenFlow control protocol version 1.3.0 [13].

All HAC cluster middleware managers and services have been implemented in C++ with using Qt 4.8.1 and Boost libraries. Applications and services snapshots are formed using boost serialization mechanism. Interaction between the middleware services and managers provides through the Qt signal-slot mechanism to ensure the independence from the base controller.

IV. Evaluation

The HAC controller prototype implementation has been deployed on a Linux virtual machine for functional and performance testing. Our experimental evaluation includes two parts: synchronization overhead evaluation and controller failover time evaluation. In the first part we evaluate performance overhead connected with primary and standby HAC controller synchronization. Using cbench we evaluate throughput of NOX13oflib and throughput of two-node fault-tolerant HAC cluster.

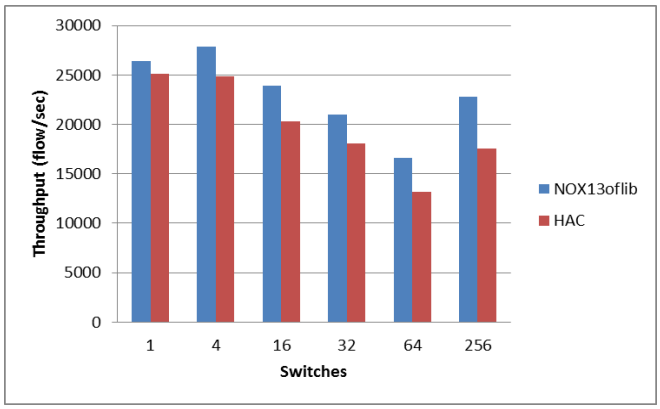

Fig. 3. HAC controller synchronization performance overhead

Synchronization overhead range is from 5 to 23 percent of the nox13oflib controller throughput (see Figure 3).

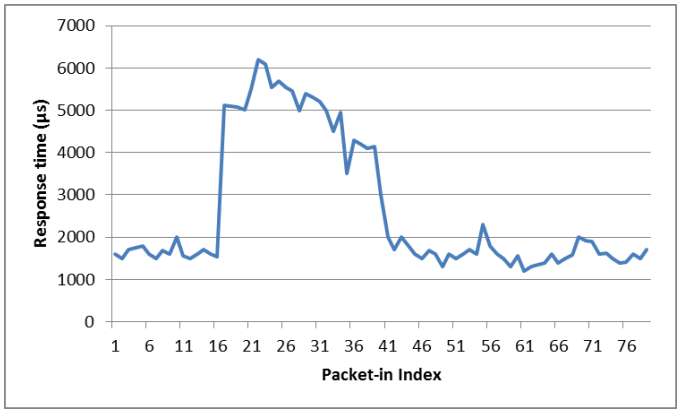

Fig. 4. Response time during HAC failover

Figure 4 shows the change of response time depending on the packet-in message index during primary controller failure and controller failover procedure. Initial experimental results showed that average failover time for two-node HAC cluster are from 40 to 50 ms, which is less than the maximum delay for services and that is why network services for end users will not be interrupted during controller failover.

V. Conclusion and Future Work

Controller is a critical component of enterprise software-defined networks. In this paper, we showed the relevance and significance of the control plane availability problem for SDN in case of controller failure.

We considered and carried out a comparative analysis of active/standby strategies for their applicability to the control plane. We formulated a set of necessary requirements for controller redundancy.

Moreover, we presented control plane design, HAC controller architecture and tools for controller network applications and services synchronization and network view synchronization between controllers in the control plane. We implemented the HAC cluster middleware that can be easily adapted to other more productive basic controller implementation. We showed that our initial evaluation results are quite encouraging.

Thus, in this paper we proposed approach to solve controller failover problem for SDN control platform, we proposed middleware implementation which provides opportunities for active/active strategies studies and distributed controller development.

We are continuing the implementation of the HAC cluster middleware with focus on developing controller state synchronization algorithms and adding of load balancing mechanisms between controller instances. We plan to extend the list of failures that the control platform can prevent.

References

[1] N. McKeown et al., “OpenFlow: Enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, i 2, April 2008, pp. 69-74.

[2] R.L. Smeliansky, “Software-Defined Networks,” Open Systems, N.9, 2012. [in Russian]

[3] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks,” ONF White Paper, 2012.

[4] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01),” 2009.

[5] A. Shalimov, D.Zuikov, D.Zimarina, V. Pashkov, R. Smeliansky, “Advanced Study of SDN/OpenFlow controllers,” Proceedings of the CEE-SECR13: Central and Eastern European Software Engineering Conference in Russia, ACM SIGCOFT, October 23-25, 2013, Moscow.

[6] A. Shalimov et al. “Analysis of SoftwareDefined Networks Performance and Functionality,” editor-in-chief R. Smelianskiy. – M.:MAKS Press, 2014. – 148 p. [in Russian]

[7] T. Koponen et al., “Onix: A Distributed Control Platform for Large-scale Production Networks,” in OSDI, 2010.

[8] S. H. Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in HotSDN, 2012.

[9] D. Erickson, “The Beacon OpenFlow controller,” in Proc. HotSDN, Aug. 2013.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Mckeown, and S. Shenker, “NOX: Towards an Operating System for Networks,” in SIGCOMM CCR, 2008.

[11] Floodlight OpenFlow Controller. http://floodlight.openflowhub.org/.

[12] NOX 1.3 Oflib, https://github.com/CPqD/nox13oflib.

[13] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04),” 2012.