Failure Recovery and Routing Optimization in Software-Defined Networks

Adrian Alan Pol, Technical University of Denmark

Friday 4th September, 2015 **Status report for Week 5. This document frames down the problem of the master thesis, presents an initial design and a tentative time schedule with tasks and completion dates**

I. Problem description

Tolerating and recovering from link or switch failures (e.g. component or configuration) are fundamental requirements of the network. Software-Defined Networks (SDN) open possibilities for much simpler, more controllable and faster detection of failures.

OpenFlow [1] is an open source project for network management. It consists of two elements: centralized controller and switches that understand OpenFlow protocol. The controller makes decisions about network control while switches are minimized to forwarding engine only.

Thanks to OpenFlow and separation of the data and control plane, the routing management is shifted to controller. This split allows making forwarding plane simple and enables different, and possibly better handling of network failures. Controller can operate on whole network view unlike in traditional networks. Potentially better performance than in traditional networks can be disrupted by number of new type of failures.

There are three fault domains in SDN networks to look at. First at the data plane, where a switch or link fails. Second, the control plane, where the connection between the controller and switch fails. And third on controller itself, where the controlling machine fails. Those three can obstruct normal traffic forwarding even if there is an alternate path in the network. Predictable consequences are packet loss, performance degradation or even full network unavailability.

This project will focus primarily on data plane type of failures. There are several stages of recovery from this type, as first controller discovers about the failure, then can calculate the new path only if older path is remembered. In practise it is achieved in following manner.

OpenFlow uses LLDP (Link-Layer Discovery Protocol) to discover and maintain the network topology. Upon this information OpenFlow controller is able to generate flow tables. Flow table is an abstraction of Forwarding Information Base (FIB). All the packets received by the forwarders are compared with the flow table entries present in its memory first. If the packet matches any entry, forwarder will perform one of the specified actions. By default OpenFlow follows an on-demand approach. Hence, when no matching is found in flow table, switch sends a query to controller. Upon a change in a network link state, a distributed system builds consistent up-to-date map of the network and then runs Dijkstra's algorithm. The new flow table entries have to be wiped out and renew. Summing up, if forwarder discovers failure upon one of its links, it can only recover if new correct flow entry is added in its memory.

With such approach scalability limitations have to be taken into account, because the controller has to be involved in the processing of all the LLDP monitoring messages. This causes load on the controller in robust networks. Additionally running Dijkstra's on network with many nodes will end up with certainly breaking the tolerated fast recovery maximum. Fast recovery means milliseconds interval after the failure. As stated in [2] the desired interval is 50 ms.

Additionally, network controller has to perform other tasks as applying firewall or load balancing. This results in recovery process being even more complicated. This project will focus primarily on implementing recovery services targeting 50ms boundary. It will use features of OpenFlow to ensure backup paths. Additionally, on links showing high congestion, optimization will be applied. The rules must ensure homogeneity with firewall and load balancing rules.

II. Related work

Since the introduction of OpenFlow, an extensive research in the area of recovery was conducted. Some of the research work, as [3], focuses on the critical forwarder to controller connection. As it points out, those links are the most important part of the SDN network and proposes algorithms to specify the best location to place the controller.

Other work focused on data path recovery. Sharma et al. [4] showed that controller-initiated recovery needs

approximately 100ms to regain network connectivity. Authors have proposed a fast restoration mechanism in which forwarding rules are statically defined by the network administrator. However this higher the risk of forwarding loops as the possibility of misconfiguration.

Automatic Failure Recovery for OpenFlow (AFRO) [5] looked at the problem differently. As the backup logic ultimately lies in controller, the crucial requirement has to be portability. As the list of controller softwares is long (e.g. POX, NOX, Floodlight), the possibility to run recovery software on any controller is desireable. The proposal decouples failure recovery and controller logic. It is done by emulating shadow controller, running same software as the main controller, which sees part of the network without the failed element. Hence, it can hold information about backup paths upfront. And, as failure occures, instead of a controller replying to failure events, it is the shadow. Computation time of new path per each failure is decreased to zero. Additionally such approach reduces risk of forwarding loops in the network [6].

Another interesting setup is CORONET [7] which tackles the problem of scalability. It is using proactive approach with routing paths computed by VLAN growing algorithm, that creates multiple link disjoint shortest path using Dijkstras shortest path. However this work, as many others address the problem without taking into account the dynamincs of network traffic. On the contrary [8] considers variable costs of the links depending on the current utilization, estimated in real-time by controller exploiting OpenFlow counters. Traffic paths are computed accordingly, leading to flows optimized distribution over the entire network. It is using proactive approach and the authors successfully meet the 50ms requirement. This time can be decreased using fast failover groups, which were introduced in OpenFlow 1.1.

The authors of [9] made use of this mechanism. Fast failover group provides the means for local rerouting in the switches. When the switch detects failure, the group can be marked as unavailable and another group can be used for performing the match action. This way switches can react to failure locally. This paper focused on end-to-end tunnel protection and it proved to have average 28ms of protection time. However the proposal included alternative OpenFlow switch design introducing logical group ports.

Depending on network topology and the specific failure local detour paths may not be available or they may be inefficient form the resource allocation point of view. [10] introduces OpenState, that extend data plane abstraction of OpenFlow to include the possibility for switches to make states evolve. It extends the fast-failover paradigm to act globally. The motivation beside OpenState is that control tasks that require only switch-local knowledge are unnecessarily handled at the controller and thus can be offloaded to switches while maintaining centralized control for those tasks that require global kmnowledge.

As per detection of failure, Yang, et al. [11] proposed a solution at the early stage of OpenFlow which combined Routing Information Database with Flow Table. The proposition focused on transient link failure. They argued that the detection of failures will be much faster in such network as the exchange of heart-beat messages will have much higher frequency and the controller will much faster instruct forwarder what to do. However, although their packet loss verification was in much improvement, the 300-400ms link failure detection was not satisfactory. [12], which focused on detection as well, managed to achieve 3.3ms average recovery time. Authors introduced a failover scheme with per-link Bidirectional Forwarding Detection sessions, a protocol that detects failures by detecting packet loss in frequent streams of control messages.

III. Initial design

The aim of the project is to provide an application for OpenFlow networks which ensures fast recovery and link optimization. Execution of the application should not disrupt other network operations, in particular firewall and load balancing. Additionally scalability, portability should be taken into account. This section presents certain design choices to achieve this goal. The tool will provide tools for verification.

A. Assumptions

Due to time constrains the project has to assume several points for simplicity. First, the network operates on single control and data plane. Different abstraction layers are not considered. Furthermore, this project will not study the placement of the controller and it will assume no other failures then on the data plane. Lastly it will mark as sufficient the 50ms path switching time.

B. Backup Path Generation

For this project for each link, a recovery path will be specified (1+1 protection). Simmilarly to AFRO [5], the path generation algorithm choice will be left out to controller.

C. Detection

Current OpenFlow implementations are mostly based on Ethernet networks. Ethernet was not designed with high requirements on availability and failure detection. In Ethernet physical layer sends heartbeat messages every 16ms when no session is active. The link is assumed as disconnected when no response is returned for 50-150ms.

Bidirectional Forwarding Detection (BFD) [13] protocol implements a control and echo message mechaninsm between pre-configured paths. This project will use per-link BFD for fast detection of data plane failures.

D. Recovery

A reliable and scalable mechanism to provide protection against a link or node failure has more in context of OpenFlow. Not only minimize the load on the controller but also react event when controller is unreachable.

In order to achieve 50ms path switching time connectivity monitoring needs to run at approximately a 10ms frquency. This means that the controller must be able to emit, receive and process around 100 monitoring packets per second per link. Such way, for robust networks, the controller must process millions of monitoring packets just to maintain the state ot the network. This is unacceptable load.

There are two categories of approaches to improve network recovery: reactive and proactive routing [14]. The first approach, reactive, compute a new route on demand, after detecting the failure. Second computes backup paths upfront.

Reactive approches may result in insufficient bandwith as and discontinuity after failure, however they certainly reduce flow table entries, speeding up match action. Proactive approach react locally to link failures as soon as they are detected. the network use pre-computed route to forward traffic.

Up to the introduction of select and fast failover groups in OpenFlow 1.3, SDN was unlikely to achieve faster detection of failures with respect to traditional technologies. The fast failover group entry consists of two or more action buckets with well defined order. First action bucket describes what to do with the packet during normal operation, if this bucket is declared as unavialavble then the packets are trated according to the second action bucket. The status of the buckets can depend on the status of physical port or on other buckets.

To meet the requirement of 50ms recovery time, this project will use proactive approach with a use of fast failover groups.

E. Optimization

In addition to fast recovery, this project will focus on routing optimization. Whenever link is underperforming, due to e.g. traffic congestion it could be treated as partially faulty. Traffic enginering can save the network from packet loss or decrease delay. As shown in [8] much benefits could be achieved by simple load balancing. Since optimization is not as critical as recovery, it could be done preactively by controller, which sees whole network view. Depending on the current utilization, reputation of link will be estimated and a portion of traffic will be load balanced.

F. Composition

Networks perform many tasks, monolithic application can monitor, route, firwall and load balance traffic. However it is genreally not a good practise since such applications are hard to program, test, debug, reuse and port. A solution is to take a modular approach. However getting modules to work together is tricky as they are not deadling with differnet portion of the traffic but the same. Those modules have to be combined to correctly perform desired tasks.

There are two approches to gain desired result: parallel composition (multiple operations simultaneously) and sequentional composition (perform one opertaion after another, e.g. apply firewall rules before routing).

This project will demonstrate possibility of sequentional composition by applying firewall and load balancing.

G. Portability

One of the requirements for SDN modules should always be portability. The solution should not be controller specific and controller code modifications should be reduced to minimum. Using Northbound API of the controller should give controller flexibility. For this project it is pointed out the importance of portablity (or standarization of Northbound APIs) is high, however due to time constrains such implementation will be possible only for one controller (has not been chosen yet).

H. Verification

Verification and monitoring are everyday task for network administrators. Unfortunately, most of the time it is just plain text logs. OpenFlow and counter field in flow tables introduce new possibility to use open source software that are proved to be successful in other domains, such as web development. This project will store the statistics on Elasticsearch [15] cluster, a

search engine rapidly gaining in popularity in software engineering community. The aim of the creators is to have a full stack monitoring tool. For this purpose, they introduced Kibana extension, which is producing a user-defined chart dashboard, which for this project will be deployed to see changes in network parameters.

I. Testing

To eveluate the performance of the implemented traffic recovery strategies, two performance metrics should be measured. First is time to detect, which is the time interval between the failure occurrence and its detection by the node. The second is time to repair, which is an interval between failure detaection and restore path activation. For positive result the sum of those should stay below 50ms. Apart from the delay, packet loss should be noted down. Different scenarios should be taken into account (e.g. with BFD intervals).

Conformity with firewall and load balancing may be checked after backup path generation stage with classical unittests. Routing optimization should show sighnificant improvment in link load distribution in Kibana view (delay and throughput).

To prove the effectivness of the proposed solution, Mininet tool will be used. However the reliable results could be generated only if tests cover different network topologies. Mininet emulations may be considered inaccurate in terms of timing due to the high level of virtualization [16]. Hence, for some time sensible tests (e.g. BFD recovery) Open vSwitch should be used. Both BFD and Group Table functionality are available in the recent version of Open vSwitch code [17].

IV. Development

This sections present time organization of the project. Figure 1 shows task split and complition dates. Figure 2 presents this information in a form of Gantt diagram.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks,"

[2] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, and S. Ueno, "Requirements of an mpls transport profil,"

[3] N. Beheshti and Y. Zhang, "Fast failover for control traffic in software-defined networks,"

[4] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks,"

[5] M. Kuzniar, P. Pereni, N. Vasic, M. Canini, and D. Kostic, "Automatic failure recovery for software-defined networks,"

[6] P. Pereni, M. Kuzniar, N. Vasic, M. Canini, and D. Kostic, "Consistent packet processing for openflow,"

[7] H. Kim, J. R. Santos, Turner, M. Schlansker, J. Tourrilhes, and N. Feamster, "Coronet: Fault tolerance for software defined networks,"

[8] D. Adami, S. Giordano, M. Pagano, and N. Santinelli, "Class-based traffic recovery with load balancing in softwaredefined networks,"

[9] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, and A. Takacs, "Scalable fault management for openflow,"

[10] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, "Detour planning for fast and reliable failure recovery in sdn with openstate,"

[11] Y. Yu, C. Shanzhi, L. Xin, and W. Yan, "A framework of using openflow to handle transient link failure,"

[12] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast recovery in software-defined networks,"

[13] R. Aggarwal, K. Kompella, T. Nadeau, and G. Swallow, "Bidirectional forwarding detection (bfd) for mpls label switched paths (lsps),"

[14] S. Lee, Y. Yu, S. Nelakuditi, Z. li Zhang, and C. nee Chuah, "Proactive vs reactive approaches to failure resilient routing,"

[15] https://www.elastic.co/

[16] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks,"

[17] http://git.openvswitch.org/

Fig. 1. A tentative time schedule with tasks and completion dates

Task	Start	Duration
Choose an OpenFlow controller for development of proof-of-concept	07-09-2015	12
Setting environment: Mininet and Open vSwitch configuration	19-09-2015	6
Testbed preparation: different topology emulations	25-09-2015	6
Implementation of per-link Bidirectional Forward Detection	01-10-2015	14
Validation of results (BFD vs LLDP)	15-10-2015	2
Pre-computation of backup paths (shadow controller)	17-10-2015	6
Validation of results (proactive vs reactive)	23-10-2015	2
Implementation of fast failover group	25-10-2015	14
Validation of results (with vs without fast failover)	08-11-2015	2
Applying sequentional conformity with firewall module	10-11-2015	6
Applying sequentional conformity with load balancing module	16-11-2015	6
Applying monitoring with elasticsearch server and Kibana	22-11-2015	4
Customizing Kibana dashboard	26-11-2015	8
Implement multipath routing optimization	04-12-2015	4
Configure multipath selection as automatic	08-12-2015	8
Validation of results (with vs without optimization)	16-12-2015	2
Project documentation	18-12-2015	2
Report polishing	20-12-2015	14

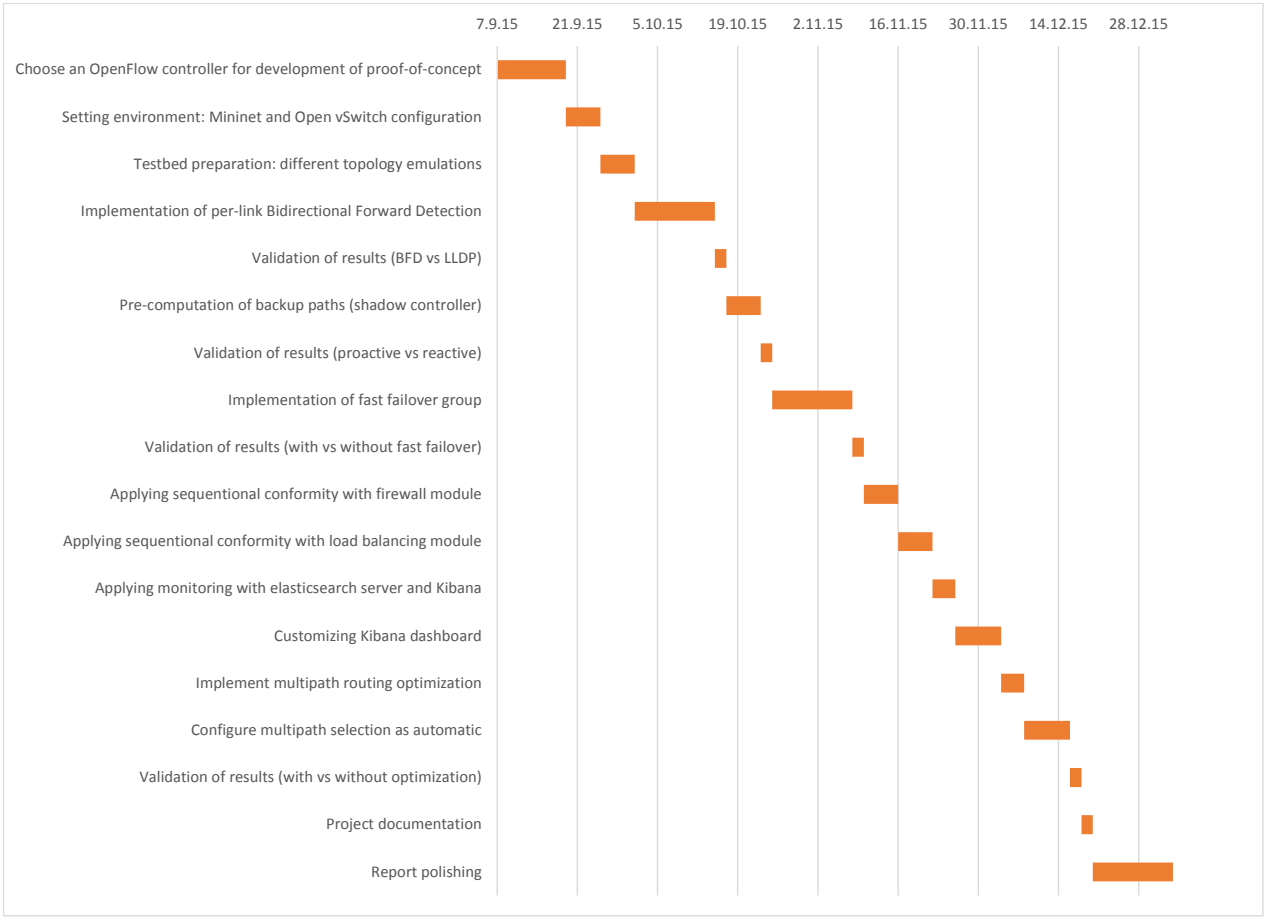

Fig. 2. Gantt diagram