Packet Loss Prevention Systems for Failover Incident on Network Infrastructure

by Samuel Samuel

Submission date: 03-Mar-2023 02:10PM (UTC+0700)
Submission ID: 2027800278
File name: Prosiding_Packet_Loss_Prevention_Systems_1.pdf (286.09K)
Word count: 3355
Character count: 16855

Packet Loss Prevention Systems for Failover Incident on Network Infrastructure

Richard Dharmawan
Department of Computer Engineering
Universitas Multimedia Nusantara
Tangerang, Indonesia
richard.dharmawan@student.umn.ac.id

Samuel, M.T.I.
Department of Computer Engineering
Universitas Multimedia Nusantara
Tangerang, Indonesia
samuel.hutagalung@umn.ac.id

***Abstract*—There are many external factors that can cause a link failure between network appliances. Thermal, electrical/magnetic field or any harmful physical object is a common example. On traditional layer 2 network appliances, the failover mechanism decided internally by Spanning Tree Protocol (STP) calculation. Unfortunately, in every occurrence of failover, the communication will suffer from packet losses. When detecting a link failure, layer 2 appliances will calculate STP to decide which backup link to be selected and in that time period all of the packets suppose to be forwarded in the link will lose. On the other hand, OpenFlow capable network appliances (Software Defined Networks) can be orchestrated with a sensor to detect earlier any possible harmful external factors before a link failure happen. This initiation research only focuses to overcome an extreme thermal condition that can cause a link failure. OpenFlow Switch connected to a thermal sensor with a Raspberry Pi as a bridge. When the sensor detects temperature beyond its limit, it will tell the OpenFlow Switch to use a backup link immediately before the link failure occurs. With this mechanism packet loss can be prevented.**

Keywords*—*failover, OpenFlow , packet loss, STP, thermal censor

I. Introduction

In a redundant local network system (a system with many path/link between endpoints), switches use STP to ensure the logical topology has no loop which can trigger packet flooding by disabling and enabling certain ports based on protocol parameters. A system with an active STP will automatically set the logical network topology to a tree-like structure. Currently, most of the switches around the world use Rapid STP (RSTP) to enable faster network convergence and topology change time. If there is a downed link, the system will use another link to transfer packets, also called as failover. In order to do failover, the switches need to detect the failing link and then do the failover. The downside of this method is there is a time when this link cannot be used anymore to transfer packets, but the switches have not detected this failure and still sending a packet through that port, therefore will cause packets to disappear on its way to the other switch port. Temperature increase can be one of many causes to link failure. Upon writing this paper, we detect that when an ethernet cable (Cat5) has reached a high enough temperature, it will cause the link between two devices to be down, then use another link to communicate. Some packet loss will arise if there are any data transfer ongoing (the system is using RSTP).

In modern networking, SDN has the ability to manage network-based on many conditions that can be done programmatically. This concept gives a separation of network devices into a data plane (network router, switch) that handle packet forwarding in a simplified manner and a control plane (controller) which handle the decision making [1]. This concept of SDN can help developers to manage IP networks. It can also reduce jitter and provide consistent a UDP packet receive order achieved by programming a controller [2]. On the other hand, with IoT, we can connect any electronic devices which can produce unlimited implementation combinations starting from house automation, smart resource management, to environmental condition monitoring and control [3].

To provide failover improvements by traditional networks, this paper consists of packet loss prevention system design using SDN and in addition, a comparison number of packet loss between this system and traditional network system. In this system, there will be a collaboration of programmatically managed network by SDN and sensor data gathering by IoT which results in faster and dynamic failover based on environmental conditions.

II. Packet Loss Prevention System

Not like traditional network failover concept, this system can detect if there are any dangerous conditions (in this case fire) around available sensors and do failover immediately after the sensor detects them. This system is to *predict* incoming dangerous environmental conditions and do failover based on that information. By predicting and doing failover before link failure occurs, packet loss caused by link failure detection can be eliminated. To speed up the temperature rise, we burn a spot of the cable near a temperature sensor.

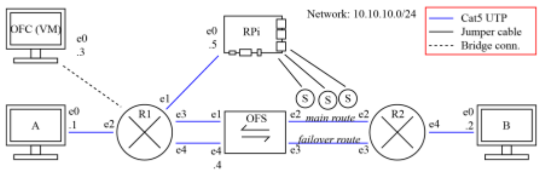

Fig. 1. SDN topology

To be properly functioning, the system consists of three main components, which is the topology itself as shown in Fig. 1, the client application running in RPi, and server application running in VM (also known as OFC, OpenFlow

Controller, controller, or server). The client application is a Python program with a 1-Wire library and the server application is also a Python program running Ryu Framework [4] for OpenFlow communication. In addition, we also made a corresponding traditional topology referring to SDN topology. This topology only takes the bottom part of SDN topology without OFS in it.

A. SDN Topology

This topology runs on the data-link layer of OSI Model with two PC (PC A and PC B), a VM as an OpenFlow Controller (OFC) running inside PC A, two routers (R1 and R2, both are MikroTik hAP Lite [5]) configured as an L2 switch, an Zodiac FX [6] (OFS), a Raspberry Pi 3B [7] (RPi), and three different sensors (temperature, passive infrared, vibration) placed near the main route cable. Between R1 and OFS, the upper cable is used for main data transfer and the lower cable is used for OpenFlow Protocol (OFP) communication. Notice that there is one port with a 10.10.10.4 IP address on OFS *e4* to specifically communicate OFP with OFC. The connection between OFS and R2 will be used to simulate a redundant link as this connection can become the main route or backup/failover route. The upper cable is used as the main route for data transfer and the lower cable as a failover route. The failover route will be used only if the main route is not currently available. This paper will focus on using DS18B20 temperature sensor which can provide temperature value every 0.7 seconds and has a maximum temperature reading of 125 degrees Celcius [8, p. 20], but as we test the component, it can endure up to 128 degrees Celcius before the output went totally wrong. The other sensor can be observed as a reference that we can read multiple sensor data at one time and can be modified as the user wish.

B. Client Application

The main job of client application inside an RPi as seen in Fig. 2, is to collect temperature data from sensor, filter, and send it encrypted to the server application. There is also some important variable in this application to indicate whether the system is in *danger* or *safe* state. This variable will be used to determine when to send sensor data in addition to dangerous sensor data. The server application will further process these data and make a decision whether to do or not to do failover. First, the application will read all sensor data every second (0.7 seconds from temperature sensor minimum frequency + 0.3 seconds to make it round). This interval is intended to be associated with IPERF which provide output per second. The 0.3-second delay can be modified later. When all sensor data has been read and converted to bitmap, there is a filtering phase where only a data meeting the certain requirement will be sent to the server. The requirements are one of the following;

- The bitmap contains at least one of 1-bit value, indicating if there are any dangerous activity near the cable, in this case, fire (temperature rise). For the temperature part, the bit will have a *1* value if temperature goes above 55 degree Celcius, five-degree lower than a typical maximum operating temperature [9, p. 5], [10] for safety reasons.
- The system is currently in *danger* state and has proceeded through *a safe* state for five seconds. In this application, *safe* means that the bitmap doesn't contain any *1* bit value in it.
- The system is currently in *a safe* state for five seconds. If the system is always on *safe* state, then it is guaranteed that the application will send sensor data every five seconds.

Every data to be sent to the server will be encrypted first. This application use RSA (can also be modified later) while communicating with server and will always send sensor data on an encrypted form via socket connection.

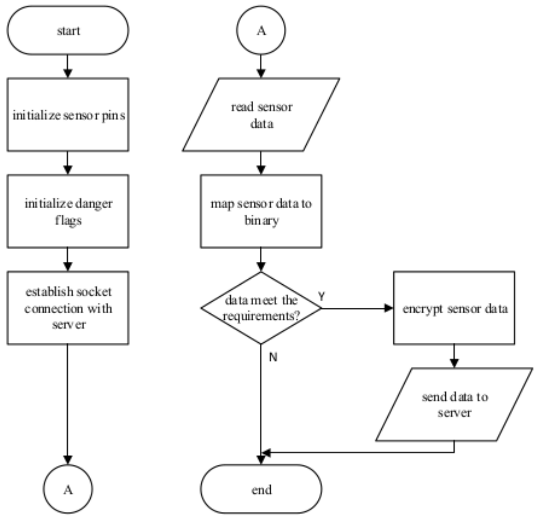

Fig. 2. Flowchart of client application

C. Server Application

The server application is responsible for OFS behavior management and shown in Fig. 3. This includes installing default and basic flow, responding to packets sent from OFS, and choose which route to use by the OFS necessarily. This application also manages a list of the connected client(s) and manage incoming client request.

K^+ (public key) and K^- (private key) is essential when it comes to sensor data communication. When there is a client requesting a socket connection, the application will add it to a local list of the connected client. After a connection is established, client requests for K^+ and the server replied with its K^+. Sensor data transfer can be done afterward. At this time, server is ready to receive more connection from different client.

The first thing to do when this application starts is initialize its socket, generate the K^+ and K^-, then listen for these events;

- Receive sensor data. The application receives encrypted sensor data from the client. This will trigger failover logic.
- Packet in from OFS. The OFS does not know where to send the corresponding packet, thus sent to the controller. This is meant to make OFS as a functional learning L2 switch.

- OFS port status changes. Either one of OFS ports is going up or down.

The application will respond to received sensor data by decrypting it first, then determine whether to do failover or not. If the data is considered *dangerous* (contains a 1-bit value on it) the application will send a failover instruction in the form of flow mod to OFS, commanding it to use failover route as a communication route. Otherwise, if the data is considered safe the application will send a flow mod to OFS, commanding it to use the main route as a communication route. When the application receives a packet in from OFS, it will decapsulate the packet, collect sender and receiver information from it, save the information locally, and send a "basic" flow mod to OFS along with packet out so it can forward next packet with the same source and destination directly without sending the packet to controller first as packet in. This is almost the same as filling MAC address table in a traditional network system. To respond to OFS port status change, the application will read incoming OFP port status packet and access the information regarding which port status changed. If the OFS has its port-2 down (port used as the main route), then the application will give the same failover instruction as described above and vice versa with port-3 down. Note that before changing route, the application will always check the current status of OFS port. For example, if a failover action is about to be executed, then the application will check port-3 of the OFS. If it is currently down then the failover action will not be executed. Similar logic also applies when it comes to doing multiple subsequent same actions. For example, if the system is currently in failover state and receives an action to do failover (which will result to the same state), then the application will not do the second failover action nor sending any flow Modo to OFS. These logics are implemented in order to save processing power and network bandwidth.

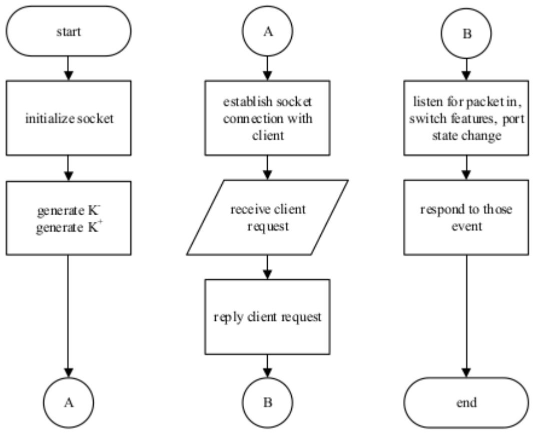

Fig. 3. Flowchart of server application

III. Improvement over Traditonal Network

To test both traditional and SDN topology, we use IPERF [11] on both PCs. PC A is running as an IPERF client and PC B as an IPERF server. Every single trial is done by sending 150,000 UDP packets with the size of 8KiB per packet over 100Mbps bandwidth configuration and completed in 105.3 seconds. All packets are sent from IPERF client to server. All output attached in this paper is taken from IPERF server. For each topology, there are two scenario to be done. The first one is a benchmark, intended to provide an initial number of packet loss. The second is a link burning scenario where we start to burn a point of ethernet cable near the temperature sensor from the tenth seconds until the end of the trial.

A. System State

Under normal condition, the system uses the main route as a communication route. In this state, all packets sent from IPERF client will be forwarded through OFS *e2*. When using the failover route, all packets sent from IPERF client will be forwarded through OFS *e3*. The system will enter failover state immediately after it detects any dangerous environmental condition.

B. Iperf Monitoring

During IPERF data transfer, the tool provides detailed information of total packet transferred, bandwidth, jitter, and a number of a lost packet every second. The average of a lost packet per second is taken and displayed as a time series in Fig. 4.

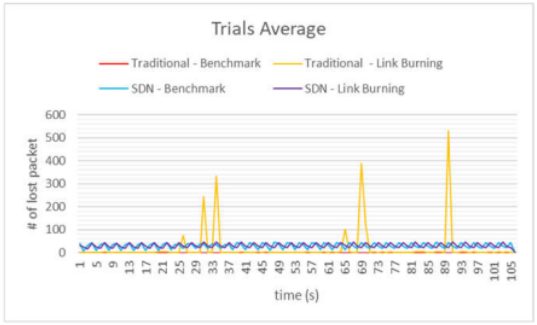

Fig. 4. Average iperf output on every trials

In traditional topology benchmark, the graph (red line) is seemed to be flat all the time. This shows us that in normal condition, the traditional topology system tends to rise a minimal amount of packet loss. This is the best condition that can be provided by traditional topology. When it comes to link burning, there is some spike in the graph (yellow line). These spikes happen when the main link between R1 and R2 fails and system begin to change switch connection to failover link. The time when a link failure occurs in our experiment can vary depending on how much fire burns the cable at the time.

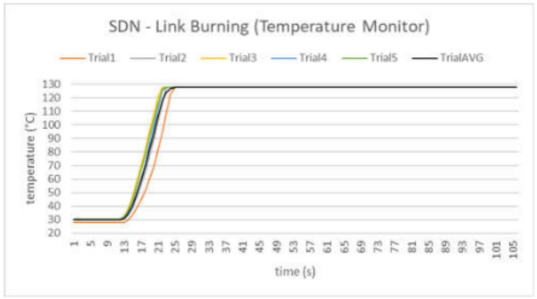

Fig. 5. Temperature monitor of SDN topology link burning scenario

Notice that in both SDN topology graph (blue and purple line), there are a small number (an average of two percent per second) of constant packet loss all the time indicated by a slightly wavy graph compared to a flat one in traditional topology. This is due to the OFS capability of forwarding packet with the addition of communicating OpenFlow with OFC. This number though, will not actually cause any significant impact on real network performance. As seen in the graph, link burning scenario of SDN topology has a relatively same pattern as in the benchmark scenario of the same topology. Fig. 5 shows us that the system should do failover approximately at the eighteenth seconds, but as we can see in Fig. 4 there is no noticeable rise of packet loss at those specific ticks. Thus, we can assume that temperature rise does not affect system performance at all.

C. *Packet Loss Reduction*

Table 1 shows the number of packet loss caused by failover (link burning). Numbers that are included in calculations are a number of packet loss when the system is doing failover based from Fig. 4, e.g. the spiking part of traditional link burning graph. In every trial we took one second before and after the failover is executed to get the exact number before the system goes stable again after the failover. In the table, the packet loss percentage is calculated by dividing the total of packet loss in each trial by 4,348 and convert it to percent form (multiply by 100%). The value 4,348 is an average number of packet received by IPERF server for three seconds. While mean is an average value of packet loss percentage in each topology.

TABLE I. PACKET LOSS CAUSED BY FAILOVER

Topo.	**Traditional**					**SDN**				
Trial	***1***	***2***	***3***	***4***	***5***	***1***	***2***	***3***	***4***	***5***
1st second	3	3	0	0	0	44	3	4	42	3
2nd second	1218	369	2650	1123	1654	2	43	45	44	44
3rd second	0	0	0	0	0	42	42	43	4	41
% Loss	28.08	8.56	60.95	25.83	38.04	2.00	2.00	2.09	2.02	2.02
Mean	32.29070837					2.028518859				

Packet loss reduction from traditional topology to SDN topology is shown in Table 2. All values are displayed in percent form. For the benchmark scenario, the values are calculated by dividing the total of the lost packet with a total number of actual packet received by IPERF server which is 153,599 and multiplies it by 100%. For link burning scenario, the number is taken from average in the previous table. From these data, we can calculate the increase of packet loss when it comes to link burning by subtracting a number of packet loss on link burning scenario with its initial condition (benchmark). Finally, we can calculate the number of packet loss reduction that can be achieved by SDN topology over traditional topology from the difference of packet loss increase in both topology.

As we can see from the table, the number of packet loss in traditional topology caused by the failover in traditional topology succeeded in reaching 32.29%. This is a significant increase in value and can affect network performance. On the other hand, SDN topology results in 2.02% packet loss, which is almost the same as a benchmark value. This value strengthens our assumptions that packet loss caused by failover in case of temperature rise does not give any significant effect on our system. Thus, we got a 32.24% packet loss reduction, which means almost all packet loss produced by traditional topology system can be eliminated.

TABLE II. PACKET LOSS REDUCTION

Scenario		**Loss (%)**	**Loss Increase (%)**	**Loss Reduction (%)**
Benchmark	***Trad.***	0.010416734	-	-
	SDN	2.087383381	-	
Link Burning	***Trad.***	32.29070837	32.28029164	32.24
	SDN	2.028518859	-0.058864522	

IV. CONCLUSION

In this paper, we introduce a new concept of executing failover by approaching both technologies owned by SDN and IoT. In this concept, failover action not only can be executed faster but also provide a lossless packet transmission when it comes to dangerous environmental conditions. The observed data show us that any packet loss caused by RSTP failover in traditional network system can be eliminated in SDN system, thus improving system reliability and performance on the specified condition.

REFERENCES

[1] K. S. Sahoo, B. Sahoo, and A. Panda, "A secured SDN framework for IoT," in *2015 International Conference on Man and Machine Interfacing (MAMI)*, 2015, pp. 1–4.

[2] S. Samuel and C. Eko Samudera, "Rancang Bangun Mekanisme Load Sharing Pada Link Aggregation Menggunakan Software Defined Networking," *J. ULTIMA Comput.*, vol. 9, Jun. 2017.

[3] P. Suresh, J. V. Daniel, V. Parthasarathy, and R. H. Aswathy, "A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment," pp. 1–8, Nov. 2014.

[4] Ryu SDN Framework Community, "Ryu SDN Framework." [Online]. Available: https://osrg.github.io/ryu/. [Accessed: 27-May-2019].

[5] MikroTik, "MikroTik Routers and Wireless - Products: hAP lite." [Online]. Available: https://mikrotik.com/product/RB941-2nD. [Accessed: 14-Jul-2019].

[6] Northbound Networks, "Zodiac FX OpenFlow Switch Hardware." [Online]. Available: https://northboundnetworks.com/products/zodiac-fx. [Accessed: 14-Jul-2019].

[7] Raspberry Pi Foundation, "Buy a Raspberry Pi 3 Model B." [Online]. Available: https://www.raspberrypi.org. [Accessed: 14-Jul-2019].

[8] Maxim Integrated, "DS18B20 - Programmable Resolution 1-Wire Digital Thermometer." Sep-2018.

[9] Farnell, "Cat5 Cable." 04-Jul-2011.

[10] Draka, "SuperCat 5 24 Cat.e." 16-Mar-2012.

[11] iperf, "iPerf - The TCP, UDP and SCTP network bandwidth measurement tool." [Online]. Available: https://iperf.fr/. [Accessed: 27-May-2019].

Packet Loss Prevention Systems for Failover Incident on Network Infrastructure

ORIGINALITY REPORT

4%	4%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.semanticscholar.org Internet Source	3%
2	Vania, Kanisius Karyono, I. Hargyo Tri Nugroho. "Smart dog feeder design using wireless communication, MQTT and Android client", 2016 International Conference on Computer, Control, Informatics and its Applications (IC3INA), 2016 Publication	<1%
3	www.coursehero.com Internet Source	<1%
4	Submitted to Academic Library Consortium Student Paper	<1%
5	F. Bendaida. "A Nonlocal Model for Reconstructing Images Corrupted by Cauchy Noise", Moroccan Journal of Pure and Applied Analysis, 2023 Publication	<1%
6	publikationen.uni-tuebingen.de Internet Source	<1%

Exclude quotes	On	Exclude matches	< 7 words
Exclude bibliography	On		