A critical review based on Fault Tolerance in Software Defined Networks

Shailly[a]

[A]
Assistant Professor, Chandigarh University, Punjab.

Article History: Received: 11 January 2021; Accepted: 27 February 2021; Published online: 5 April 2021

Abstract: SDN (Software-Defined Networks) is an incipient architecture of decoupling control plane and data plane involved in dynamic management of network. SDN is being installed in production based networks which ultimately lead to the need of secure and fault tolerant SDN. In the present investigation, we are discussing about the kind of failures with label happen in SDN. A critical survey based on the recently proposed mechanisms for handling failures in SDN. Initially, we discussed with the help of tabular data involving mechanism of data plane failure. We also discussed the various mechanisms for handling misconfiguration of drift able of switches and control plane failure handling mechanisms. We also epitomize issues with both data and control plane mechanism that are discussed earlier. In the end, we are stating that there is need of build much efficient and secure mechanism for SDN networks.

Keywords: Software Defined Network (SDN), Open Flow, VeriFlow

1. Introduction

The whole activities are connected by internet or networking. The internet is composite of the large set of devices e.g. switches, routers, servers and host systems. Traditionally a network consists of router and switches. A network consists of the set of protocols which are statically defined in routers and switches. Current network architecture only partly meets today's requirement for enterprises carriers and end users. Control and forwarding rules are defined statically in routers. The current network architecture is not suited for changing patterns, applications and user demand. SDN is an evolving approach aiming to transform the current network architecture. Control plane, a controller manages the flow flowing through the data plane [1] [2]. Open Flow [3] is a communication protocol that network switch over the network or forwarding plane of a router. Open Flow defines standard of all event processing between control plane controller and data plane switch/router.

Data plane moves the traffic. It grips the forwarding table, which specifies the interface where the packet will be forwarded. Data plane actions are decided by control plane. SDN network can be centralized or decentralized. In centralizes only one controller manage the flow table of all switches, In contrast, In decentralized multiple controller are unscrewed , distributed at multiple sites to manipulate flow table as Hyper flow [4].

SDN is depended on controller function if the controller fails it will stop everything except the traffic flowing through switches as per already install rules in flow table before controller fails. Controller is like brain of SDN architecture and its performance has most impact on throughput.

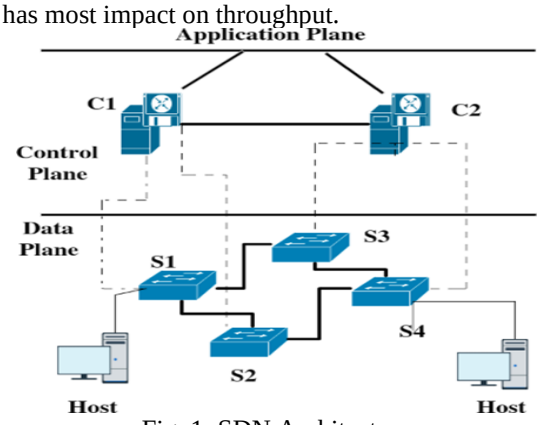

Fig. 1: SDN Architecture

There is basically two type of mechanism for securing data pane. The first regression [5] [6], recovery path can be dynamically allocated. Additional resources neither are nor reserved. Signal- ing is required to inform about failure. Second protection [5] [6], the paths are reserved before a link fails. No signaling is required in this method. Control plane can be secured by protection. Multiple controllers reserved for backup. When a controller fails, a backup controller replaces the failed controller. To increase reliability more instances of controllers are required, it is important to find how many backup controllers required and where to install these controllers.

I. CLASSIFICATION OF FAILURE

A. Data Plane

Data Plane is a connected network of all switches. A fault in data plane is defined as if a switch fails or a clinch between two switches fails.

Switch fails: Whenever switch stops working due to hardware failure e.g. power failure.

Flow table overloaded: a number of flows needed in flow table are more than the memory of flow table. When a number of hosts connected to switch become higher than threshold than flow table overloaded. In this case, overall throughput decreases.

A. Data Plane

III. EXISTING MECHANISMS

Switch overload: higher rate of traffic flowing through switch can decrease switch performance. In this case, outgoing traffic is not equal to incoming traffic.

Link-down between switches: connection between two switches goes down.

Reliability: a sent packet drops before receiving at the next switch.

DoS Attack: Switch forwarding table is flooded with new entries.

Misconfiguration: Multiple controller application compete each other for modifying flow tables. It may arise contradiction. This direct to invalid or conflict rules in flow table.

Hardware faults in switches are easily traceable and other faults are hard to detect e.g. power-off of switch and link- down are easy to detect. Reliability is depended on load and bandwidth of link assuming there is no attack on the clinch (man in the middle attack).

B. Control Plane

Connected network of controllers is the control plane. It is logically centralized network which manage all flow of switches. Anything which interrupts the normal processing of control plane is classified as control plane fault. Some of control plane faults as follows:

Crash failures: A controller shut down due to hardware fault or any other fault.

- Link-down between switch and controller.
- Link-down between controller and controller.

The controller's software is inevitability vulnerably susceptible, because of open programmability and complexity of its functionality, and this can be exploited for maleficent attacks.

Byzantine faults: These are arbitrary failures. The controller behaves arbitrary in nature. It can give in- appropriate commands, send duplicate messages.

Omission failure: A controller fails to respond to incoming requests. A loaded controller may fails to receive events from switch or send commands to a switch.

Response failure: A controller may send wrong con- figuration to other controller or switches.

Event un-ordering: Command sent to switch may be received irregularly. This may change behavior of switch or inappropriate flow table.

DoS attacks: A controller is flooded by multiple switches events. Whenever a switch received an in- cipient flow packet. A malevolent host can flood the switch-controller link with incipient flow packets.

CORONET [7], Controller based Robust Network; provide the recovery from the failure in switch or data plane. It runs an efficient algorithm on global network view of present network. CORONET has five properties, fast recovery, scalable, multi path routing, independent of topology, only one single controller. It is reactive mechanism.

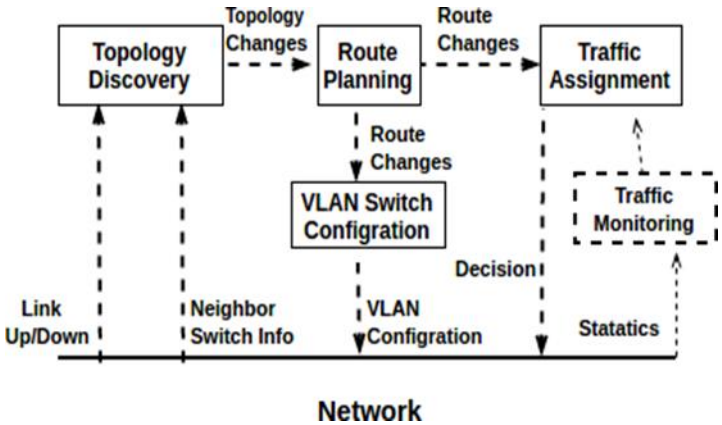

Fig. 2: CORONET Modules.

Topology discovery module will collect information periodically and whenever switch fails it receive asynchronous events. Routing planning module will provide alternate paths, input gets from topology module. Algorithm used by this module is VLAN growing algorithm.

TCAM-aware local re-routing [9], a proactive mechanism in which the forwarding rules were stored in TCAM switch. Normally switches could hold up to 1500 Open Flow rules. Now TCAM would have only one entry for

one path, backup path can be stored in a testing interface. It used two algorithms for computing backup paths: FLR algorithm and BLR algorithm. A controller could hold both algorithms for computation. FLR selected a minimum number of switches that flows traffic from point of switch failure. BLR was simpler than FLR. BLR forward traffic backed to the source from point of failure and re-routed traffic to a clinch-disjoint backup path.

Software fast failure detection and path layout planning and [10], secure survivability contrary to single link failure. It took three things into account bandwidth consumption, survivability, and fast failure recovery. It transformed the topology into a graph where a node with k neighbors transform into 2k + 2 nodes. It used a shortest path algorithm for assigning paths. It took an advantage interface specific forwarding (ISF) for path preplanning. For detecting link failure monitoring paths were used. A monitoring path was circuit including controller. A packet (Path Alive Monitoring Packet) circulated around path. Whenever controller did not get packet, result there was a link failure. For detecting link failure FLI (Failure Location Identification) packets were used.

Mechanism	Number of messages after failure	Extra flow entries	Detection algorithm	Single or multi link failure	Controller Load Consideration	Timing	Restoration al- gorithm	Controller job	Bandwidth consideration
CORONET	$2f_s$ (average)	No	Link Layer Discovery Pro-tocol(LLDP) [8]	Both	No	High	VLAN grow- ing algorithm	Update flow table with VLAN configuration	No
TCAM aware local routing	0 (Use backup path, ideal)	Yes	NA	Both	No	Average	NA	Computes backup paths	No
Group table resilience	0 (Use next rule in group entry)	Yes	BFD	Both	No	Very Less	NA	Computes backup paths	No
Path layout plan- ning and software fast failure detec-tion	n_s (worst case)	No	Monitoring	Single	No	High	Shortest path algorithm	Detecting link failure and in- stall new path	Yes

TABLE I: Comparision

Every Switch in path would pass FLI packet to next switch and controller. So a switch next to failed link would not get FLI packet. Total number of monitoring path professed the load of controller and overall time.

Group Table Concept discussed in [11] with BFD [12], each flow table was associated with group entry. A group entry had a bucket consisting of multiple rules. Each rule in bucket had alive status. So whenever a link fails, it alive status goes down. So next rule in bucket was used as action. BFD is used to promulgate the status in bucker.

Multiple OpenFlow application running over controller might modify insertion and update flow table. This led to misconfiguration in the flow table.

Flow Checker [13], identified any misconfiguration within single flow table. It implemented a slice policy between multiple controller applications and the OpenFlow. It partitioned flow table and checked with a partition belong to particular controller application. State machine representation and binary decision diagrams (BBDs) checked reach ability verification and security properties written in CTL (computation tree logic).

Fort FOX[14], a kernel ,a extension, role base authentication and security enforcement for NOX OpenFlow controller. SDN application could complete each other and might write individual flow rules to switch. It instigated all flow rule insertions, checked for rule conflicts. It might accept or reject a particular rule based on its role (priority). It gave role based authentication.

Human administrator applications got higher priority. Security applications had higher priority than no security applications. A digital signature scheme was used to identify particular role. It converted all rules into alias reduced rules, performs conflict analysis. The result of conflict resolution was high priority overrides lower priority rule in the flow table. State table manager managed the state of all active rules. Security directive translator mediated directives into flow table for conflicts.

2. Summaries:

Table I summarized the comparison between various fault handling methods for data plane. Here, fs is the number of faulty switches, ns is the total number of switches. None of the above methods considered the controller load. Multiple switch failures can increase more load over controller which can lead to low performance. No method provideed both protection and mending except group table resilience but group table used BFD that only provide that path is normal or not, it did not give information about the location. It also required more time. No

single method could be fully protective because of limitation of TCAM memory and no method could be full restoration based as it took more time. Restoration was more flexible than protective method. The protective method has limitation of switch memory. Protective methods were also considered as static because backup path calculated before the failure on basis of distance or bandwidth; it did not consider real-time traffic into consideration at the time of failure. All methods focused on the single point of failures, multiple link failure scenarios were not inferred. So scalability was also an issue while considering these methods. A mixture of both restoration and protection based method for multiple link failure required for high reliability and scalability.

B. Control Plane

Smart Light [15], considered an architecture, network was managed by a single controller. Extra controllers used as backup controllers. Whenever primary controller fails, one backup controller will take place as the primary controller. Controller stored all the network and application related data (Network and application state) in data store called Replicated State Machine (RSM). It used cache for better performance. A switch was connected multiple controller. Two algorithms were used one for fault detection and other for leader election.

Controllers run coordination module to ensure safety and likeness. Every controller would execute acquire Lease (id,L) [15] periodically on data store where it was unique identifier of controller and L was lease time. If controller will get its id then it will become primary controller for L lease time

CPRecovery was a primary backup controller mechanism. A primary controller might fail due to the unwanted reason for example heavy load, hardware fails, controller-switch link failure. Messenger component on each controller managed the events between each controller.

If a switch did not get a message from the primary controller for a threshold time, switch sent inactivity probe to primary controller and did the handshake with the secondary controller, made secondary controller new primary.

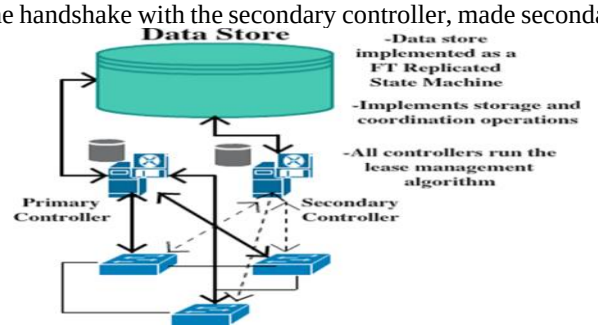

Fig. 3: Smart Light architecture [19].

Some load balancing mechanism discussed in [16] [17] [4] distributed SDNs also followed recovery-replication mechanism, preventing controller from failing because of load balancing. In controller placement problem [18] stated how a backup controller location can affect overall performance.

Ravana[19], a two stage protocol , first stage ensured that the very received event was reliably replicated to the log and second stage ensured that the event transaction was completed. Ravana protocol was ensuring the switch state is consistent. It considered three basic problems that can happen: 1. Inconsistent event ordering: replicas might get different order of events due to network latencies 2. Unreliable event delivery: new master did not know about a link-down if master fails 3. Commands : new master could send different rule for same event.

Protocol overview: 1. A switch can forward packet next hop if rule is there in table 2. If it trigger event than send an event to master and buffers it 3. Master stores it into log 4. Master acknowledge the switch, switch deletes it from buffer

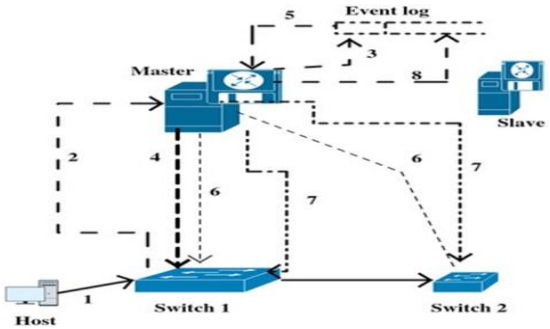

Fig. 4: Steps for processing packet in Ravana [19].

5. Processing of event, issues commands 6. Send command to switches 7. Acknowledge back to master 8. Store to log that event is complete. Whenever master fails leader election module running on slaves will elect new master.

Byzantine-Resilient Secure SDN [20], was developed to defend against Byzantine failures, in which components of a system fell in arbitrary ways. A switch was connected to multiple controllers. The number of controllers depended upon how many controller can fail e.g. there was f controller can tolerate, the number of the controller would be n 2f + 1 [MInZyzzyva] [21]. Consider for f = 1 there were three controllers if one controller

issued faulty commands other two controllers would verify it and guaranteed correct flow table update. Services were replicated to each replica and executed independently. Step 1 the client sent the request to master controller 2. The master sent the request to other slave replicas 3. Slave replicas executed the request and sent the reply to client 4. The client waited for f + 1 replica for the same result. It used CAFTS algorithm to the assigning controller to switches.

Summaries: Table II summarized the comparison between various fault handling methods for the control plane. DCS, DCC was delay between controller-switch and controller-controller respectively. NC, NF C, NS was total controller, total faulty controller and total switches respectively. Most of the methods were using controller replication for dealing with failures in control plane. Many were proposed for centralized SDN. Ravana could be scalable to distributed architecture. NCP was total number of primary controllers.

There were still a major concern that was an optimal number of backup controller and optimal placement of backup controllers.

Mechanism	Total messages between control and data plane (Considering one switch in data plane)	Total delay depending on	Minimum No of Backup Controller Required	Event Or-dering	Exactly Once Event	Single or Multi Controller	No of Data Store or In Mem- ory Log
Smart Light	$2.N_C$	Delay between controller and data store , Electing primary controller, D_{CS}	≥ 1 (Less)	Yes	No	Single	1 Data store (less)
CPRecovery	$2 + 3$ (before + after failure)	D_{CC} , D_{CS}	1 (Less)	Yes	Yes	Single	0 (Ideal)
Ravana	$4 + 6$ (before + after failure)	Delay between controller and event log, Electing primary controller , D_{CS}	$\geq N_C$ (Average)	Yes	Yes	Multiple	N_{CP}
Byzantline-Resilient Secure SDN	$1+2.N_C$ (before + after failure)	D_{CC} , D_{CS}	$> 2N_{FC} N_S + 1$ (Higher)	Yes	Yes	Multiple	0 (Ideal)

TABLE II: Comparison

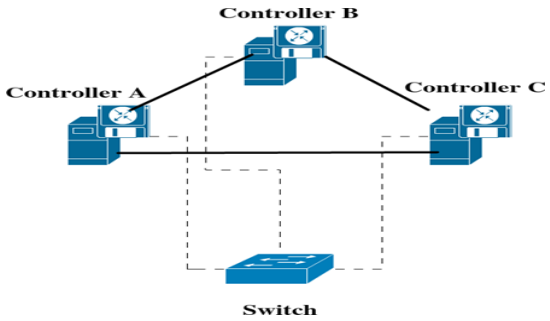

Fig. 5: Byzantine-Resilient Secure SDN model.

A number of a backup controller increased the cost of the whole network. Ravana stated consistency while controller failure but it took more time. There was the need for fast, consistent mechanism. Elasticity was also the major concern. A controller failure could result in high load over a new controller. The load should be balanced properly between controllers.

3. Conclusion

In this paper, we discussed both protective and restoration based on failure recovery. For data plane, many solutions proposed for single link failure, but there were still issue with multiple failures and time. It was still a problem of developing a mechanism which is fast and recovered from multiple data plane failures. Better utilization of switch memory for back paths was also a major challenge. Considering control plane, many backup controllers were placed for recovery, but an optimal number of the backup controller was still a major research area.

Many backup controllers were idle while there was no failure. The controller placement problem was also a major challenge. Load balancing while failure was also an important point. Most of the methods were for centralized SDN, developing highly reliable and distributed fault tolerant SDN was the major challenge.

References

1. Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ra- mana Kompella. Towards an Elastic Distributed SDN Controller. In HotSDN13. ACM, Aug 2013.
2. Amin Tootoonchian and Yashar Ganjali. HyperFlow: A Distributed Control Plane for Openflow. In INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking. USENIX Association, April 2010.

3. B. Duhan and N. Dhankhar, "Hybrid Approach of SVM and Feature Selection Based Optimization Algorithm for Big Data Security," Lect. Notes Electr. Eng., vol. 605, pp. 694–706, 2020.
4. Brandon Heller, Rob Sherwood, and Nick McKeown. The Controller Placement Problem. In HotSDN12. ACM, Aug 2012.
5. D. Katz and D. Ward. Bidirectional Forwarding Detection.https://tools.ietf.org/html/rfc5880.
6. Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Verissimo, Chris- tian Esteve Rothenberg, Siamak Azodolmolky, Senior Member, and Steve Uhlig. Software-Defined Networking: A Comprehensive Survey. Proceedings of the IEEE, 103(1):14 – 76, Jan 2015.
7. Ehab Al-Shaer and Saeed Al-Ha. Flowchecker: Configuration Analysis and Verification of Federated Openflow Infrastructures. In SafeConfig '10 Proceedings of the 3rd ACM workshop on Assurable and usable security configuration, pages 37 – 44. ACM, Oct 2010.
8. Fbio Botelho, Alysson Bessani, Fernando M. V. Ramos, and Paulo Ferreira. SMaRtLight: A Practical Fault-Tolerant Sdn Controller. In https://arxiv.org/abs/1407.6062, June 2014.
9. He Li, Peng Li, Song Guo, and Amiya Nayak. Research Challenges for Traffic Engineering in Software Defined Networks. IEEE Transactions, 2(4):436 – 447, Dec 2014.
10. Hillip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A Security Enforcement Kernel for Openflow Networks. In HotSDN12. ACM, Aug 2012.
11. Hyojoon Kim, Mike Schlansker, Jose Renato Santos, Jean Tourrilhes, Yoshio Turner, and Nick Feamster. Coronet: Fault Tolerance for Software Defined Networks. In Network Protocols (ICNP),IEEE, Nov 2012.
12. Ian F.Akyildiz, Ahyoung Le, Min Luo, and Wu Chou. Research Challenges for Traffic Engineering in Software Defined Networks. IEEE Network, 30(3):52 – 58, May 2016.
13. JThomas D. Nadeau and Ken Gray. SDN: Software Defined Networks. O'Reilly Media, First edition, 2013.
14. Jue Chen, Jinbang Chen, Fei Xu , Min Yin, and Wei Zhang. When Software Defined Networks Meet Fault Tolerance: A Survey. Algo- rithms and Architectures for Parallel Processing, 9530:351 – 368, Dec 2015.
15. Naga Katta, Haoyu Zhang, Michael Freedman, and Jennifer Rexford. Ravana: Controller Fault-Tolerance in Software-Defined Networking. ACM Newyork, June 2015.
16. Openflow Switch Specification. https://www.opennetworking.org/sdn- resources/openflow.
17. Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian OConnor, Pavlin Ra- doslavov, William Snow, and Guru Parulkar. ONOS: Towards an Open, Distributed SDN OS. In HotSDN14. ACM, Aug 2014.
18. Paul Congdon. Link Layer Discovery Protocol.
19. Purnima Murali Mohan, Tram Truong-Huu, and Mohan Gurusamy. TCAM-Aware Local Rerouting for Fast and Efficient Failure Recovery in Software Defined Networks. In Global Communications Conference (GLOBECOM). IEEE, Dec 2015.
20. Ramakrishna Kotla, Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin. Zyzzyva: Speculative Byzantine Fault Tolerance: speculative Byzantine fault tolerance. pages 45–58, Oct 2007.
21. Sachin Sharma, Dimitri Staessensa, Didier Collea, Mario Pickaveta, and Piet Demeestera. OpenFlow: Meeting Carrier-Grade Requirements. In Computer Communications, volume 36, pages 656–665. Elsevier, March 2013.
22. Steven S. W. Lee, Kuang-Yi Li, Kwan-Yee Chan, Guan-Hao Lai, and Yao-Chuan Chung. Path Layout Planning and Software based Fast Failure Detection in Survivable Openflow Networks. In Design of Reliable Communication Networks (DRCN). IEEE, April 2014.