Dynamic Path Discovery for In-band Control Plane Communication in a Tactical SDN Network

Anders Fongen
Norwegian Cyber Defence Academy (FHS/CIS)
Lillehammer, Norway
email:anders@fongen.no

***Abstract*—Software Defined Networking (SDN) offers promising improvements in operational control of tactical networks, in terms of traffic prioritization, topology management, infrastructure protection, resource monitoring, configuration and deployment. Military networks are characterized by relatively slow radio links which are vulnerable to detection and intrusion, so novel technologies like SDN are highly relevant and actively researched for these purposes. However, the SDN architectural blueprint needs several modifications to meet the typical requirements of a tactical mobile network. This paper addresses the need for reliable in-band control plane traffic across the southbound interface, and suggests two different algorithms to obtain adaptive forwarding decisions for southbound protocol traffic across the data plane links. Among the challenges related to the implementation of adaptive forwarding mechanisms in SDN equipment is the limited expressiveness in the OpenFlow language. Based on expiry mechanisms in flow rules, the SDN switches were able to choose alternative forwarding ports in case of link or switch failure in the grid. The conclusion of the study is that it is possible to make adaptive mechanisms with recovery times comparable to the Spanning Tree Protocol (STP), but with better utilization of link resources since link loops are allowed to exist for load distribution (contrary to the STP protocol).**

***Keywords*—software defined networks; tactical networks; adaptive forwarding; resilience**

I. INTRODUCTION

In a tactical military network, the links are the resource of greatest scarcity and need to be utilized as efficiently as possible. Also, the links are exposed to a range of threats to security, integrity and availability. The SDN blueprint, developed with centralized data centers in mind, assumes a separate set of links for the control plane, which is not an affordable luxury in a tactical network [1]. The use of so-called *in-band control plane communication* has been pursued in this paper for this particular reason.

The term *In-band control plane* describes an SDN configuration were the links between the Network Elements or switches (NEs) carry both user data and southbound protocol traffic between the NEs and the SDN controller (SDNC). In-band control plane have been widely discussed and is implemented in a basic manner in OpenVswitch [2]. However, the OpenVswitch implementation does not offer adaptive paths for the southbound traffic in case of link or NE failure.

The need for reliable control plane connectivity must be combined with the desire to control the links with regard to traffic and security policies. The Spanning Tree protocol (STP, IEEE 802.1D) offers a form for adaptive paths since it may reconstruct the forwarding path in case of link or node failure, but the STP protocol does not employ redundant links for load-balancing purposes and does not offer the flexible traffic policing for which the SDN is popular.

Redundancy management, both for the data plane and control plane traffic, should employ all available link resources both for resilience and load balancing purposes. The presence of link loops requires that received frames cannot be broadcast in order to avoid traffic loops. This restriction affects both discovery protocols like Address Resolution Protocol (ARP), etc., and the bridge link-address learning process.

The construction of the distributed logic necessary for the implementation must take into account the primitive nature of the NE; They are not programmable in the ordinary sense, only through rule-oriented protocols like OpenFlow. The lack of a transaction context in OpenFlow processing generates race conditions and inconsistent intermediate states [3], and failure detection must be handled through the flow expiry mechanism [4].

A. Related Research

The application of SDN in tactical military networks has been discussed from perspectives of robust flow separation [1], security [5] and in-band control plane [1]. Still there are few proposals on how to implement a robust and resilient control plane tunneled through the data plane links. Schiff et al. [3] and later Canini et al. [6] provides good analysis of the problem space and a solution outline, but no detailed and tested proof of concept. Both papers propose the same "hard-timeout"-based mechanism for failure detection, but do not provide a convincing fail-over mechanism.

Sharma et al. [7] offers a comprehensive solution to the connectivity problem as well as congestion in the control plane. They base their solution on modification of the OVS switch code, which is avoided by the work presented in this paper, since the solution should be able to run on commercial and unmodified OF-switches. A fully implemented and tested algorithm for dynamic path discovery in an in-band control plane has, to the author's best knowledge, not been presented before.

B. Contribution of the paper

The research question being pursued in the presented paper can be expressed as *is it possible to implement adaptive routing in the control plane using OpenFlow protocol mechanisms?* The challenges related to this question is that link failure in the control plane will isolate NEs from the SDN controller and NEs will have to detect and recover from link failure autonomously.

 ISBN: 978-1-61208-740-5

The contribution of this paper is the investigation of discovery and fail-over mechanisms in the control plane. It will report on algorithms, design patterns and the experimental evaluation. This paper does not address the obvious security problems related to SDN in a tactical environment, since those problems have been addressed in other recent publications [5].

The remainder of the paper is organized as follows: Section II discusses the benefits of in-band control plane and two alternative design methods. Section III presents the chosen technology components and Section IV shows the configuration of the experimental network used in the experiment. Section V discusses how the underlying SDN protocols can support the operations necessary for the purpose of the experiment. Sections VI and VII provide details of the two proposed algorithm for redundancy management and contains the main contributions of the paper. Furthermore, Section VIII describes the mechanisms necessary for operation of a network without using broadcast frames. Section X summarizes the paper with a few conclusive remarks.

II. IN-BAND CONTROL PLANE

The initial SDN architecture presumes the existence of a separate link infrastructure where every NE is directly connected to the SDNC. Through this infrastructure (the control plane) the SDNC will manage the topology control and traffic policing in the data plane. The control plane is silently expected to be reliable, and any fail-over mechanisms are kept outside the SDN scope.

> *"An SDN controller may use an SDN data plane for some or all of its internal or external interfaces, as long as the SDN controller does not rely for its connectivity on the operability of the data plane that it controls; otherwise, the SDN controller may find itself stranded or irrecoverably fragmented."* - Sect 6.4 of [8]

In a military network, the resource of greatest scarcity is the set of communication links, and a separate control plane infrastructure is an unaffordable luxury. Intuitively, there are resource benefits by merging the two planes into a shared set of communication links. Multiplexing and switching units external to the NE can allow the two types of traffic to be logically separated yet sharing a link. This solution is expensive and cumbersome in terms of hardware resources though, and offers no fail-over mechanisms.

In a more straightforward manner, the OpenVswitch [2] offers a mode whereby it at startup time allows Dynamic Host Configuation Protocol (DHCP), ARP and TCP traffic to and from the SDNC to pass through its switching fabric. The trick is to assign the control plane IP address of the NE to the switch pseudo interface, not to a physical port. OpenVswitch will then use its switching fabric as a MAC learning switch to find a path to the SDNC, using the OpenFlow `NORMAL` output port.

Despite a low cost solution to the in-band control plane problem, it still offers no fail-over mechanism. Besides, the presence of redundant links in the data plane will result in traffic loops.

OpenVswitch offers redundant link management through the STP. However, STP will simply disable links that result in loops and prune the link structure into a tree. Nor will STP support the tight traffic policing and priority mechanisms that are the hallmark of the SDN architecture.

Therefore, there is a need for a mechanism to allow an in-band control plane to be dynamically overlaid on the data plane link structure, without reserving valuable links for fail-over purposes only. The load-sharing capabilities of the redundancy must be utilized and the traffic policing enforced by the SDNC must not be hindered. This paper investigates two approaches to meet these requirements:

1) A reactive method, where the individual NEs discover the path to the SDNC and connects to it.
2) A proactive method, where the SDNC keeps a catalogue of the switches and the links, and actively constructs a tree structure and connects the switches accordingly.

Both methods allow the NEs to be indirectly connected to the SDNC, and both allow for fail-over operations to take place if a link or an NE falls out of service. However, they differ in their programming structure and how well they fit into the software environment.

III. TECHNOLOGY PLATFORM

In this section, the choice of technology components will be described. The components are all software, including operating system, hypervisor and system-level components.

The study of a medium sized networks with more than 10 nodes is best conducted in a virtualized environment. The hypervisor of choice is Oracle's VirtualBox, which is free, easily configured, and offers the right degree of scalability. The limit of four ports per VM is the most limiting factor during the experiment.

For the NEs, complete instances of Linux were chosen. The reason for this choice is that the experimental network is used for testing several services and protocols auxiliary to the OpenFlow protocol, and a general computing platform offers the necessary flexibility and software availability, contrary to Mininet. The Linux instances do not need a GUI and was installed with a text console interface only for the sake of saving memory.

The chosen OpenFlow switch (the NE) implementation is OpenVswitch [2], which is easy to install, relatively easy to configure, and offers the necessary inspection and logging mechanisms for testing and debugging purposes.

As the network controller (SDNC), the Ryu framework was used [9]. Ryu is very popular as an experimental platform with a relatively low abstraction level: OpenFlow statements are generally not automatically generated, but individually constructed through Python programming code. For the experimentation at hand, Ryu performs well and with good stability, although the API and the required design patterns takes some time to learn.

For all the chosen technology components, an important property is the community support offered. Most problems are easily solved through these support resources.

 ISBN: 978-1-61208-740-5

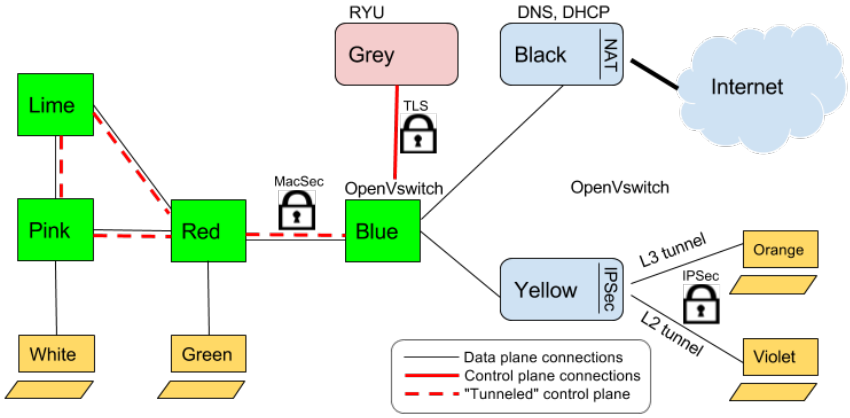

Fig. 1: Current SDN laboratory configuration

IV. Experimental network

The network used in the experiment is shown in Figure 1. The network consists of a number of green switching nodes (NEs), a number of yellow general clients and a number of blue nodes for serving IPSec, OpenVPN, DHCP, DNS, VXLAN etc.

The links between the NEs are somewhat redundant and consequently form loops. The redundant links are essential for the study of fail-over mechanisms and load balancing services. The experimental network was used also for investigating security mechanisms in an SDN based environment [5], thus the presence of IPSec, MacSec and TLS is indicated in the figure.

V. Software design patterns

Adaptive forwarding would intuitively need general programmable logic in the switches, in order to test environmental conditions and make decisions accordingly. This is particularly likely when the data plane and the control plane use the same links, and an NE would need to find an alternative forwarding path in a situation where it is isolated from the SDNC.

Another matter is that the NE does not have any direct mechanism to reveal the physical port it uses for the control plane traffic. The controller needs that information in order to install flows that avoid traffic loops (the use of the OpenFlow `NORMAL` output port must be avoided for the same reason).

The OpenFlow mechanisms that were used for building the necessary distributed logic were:

1) The output port `CONTROLLER` which allows traffic that matches a flow to be handled over to the SDNC, with information about the ingress port of the switch. This mechanism is part of the Port Discovery procedure which will be discussed in Section VI-A.
2) The use of flows associated with expiry mechanisms and high priority in combination with low priority flows without expiration. In case the high priority flows fail to be renewed, they will disappear and the low priority flow will be set in effect. The necessary fail-over mechanisms are built on this design pattern, which is also proposed by Canini et al. in [6].
3) Timestamps on certain connection requests. Where there are several paths from an NE to the SDNC, the first node common to these paths will experience connection requests from the same node over a short period of time. The timestamps will serve to recognize the first request and discard the others.
4) Proxy operation of broadcasts. To avoid traffic loops, NEs cannot broadcast received data, only data which is locally originated. Broadcast packets are passed on to the SDNC which may locally resolve the situation or pass them on to each NEs with the instruction to flood them.
5) Data plane forwarding based on MPLS labels. Flooding as seen in MAC-learning bridges cannot be used, but the SDNC keeps track of all client MAC addresses and their associated NE. The SDNC will install flows (on-demand) to attach MPLS labels to frames signifying the egress NE in the path to the destination. Forwarding in the data plane is based on MPLS labels only.

In the following sections the two different algorithms will be described in more details. The two alternatives are designated as reactive and proactive, respectively.

VI. Reactive adaptive forwarding algorithm

This algorithm does not presume any knowledge about the collection of links and switches. The knowledge is built through a discovery process. The switches can be in one of three states: *Disconnected, Connected* and *Operating*:

Disconnected: The switch is not under control of the SDNC and contains only flows installed during the bootstrap process. These flows floods TCP and ARP packets originated from the switch (in-port: `LOCAL`) destined to the SDNC, and sends ARP and TCP traffic from the SDNC to the `LOCAL` port for processing by the MAC-learning switch fabric.

Connected: The switch is under control of the SDNC through the OpenFlow protocol, but is unable to bridge connections from lower-tier switches. In this mode, it still floods locally generated packets to the SDNC. The SDNC will initiate

 ISBN: 978-1-61208-740-5

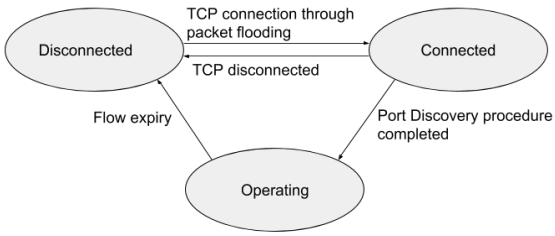

Fig. 2: State diagram of a switch/NE

the *Port Discovery procedure* when the switch enters the connected mode.

Operating: The Port Discovery procedure is completed, and the switch now communicates unicast (called "unicast flows") with the SDNC through its *command-port* (c-port). The switch is able to receive connection requests (TCP SYN) from lower-tier switches and pass them on towards the SDNC, which happens later as explained in Section VI-B.

The state transitions are shown in Figure 2. Note that the transition from Operating to Disconnected happens as the installed flows expires due to lost communication from the SDNC. The SDNC has to refresh the "unicast flows" on a regular basis to avoid them from expiring. If the communication is lost, the flows that floods packets to the SDNC will again come into effect and let the NE look for other paths to the SDNC. *This is the fail-over mechanism available in OpenFlow when the NE becomes disconnected from the SDNC.*

A. The Port Discovery procedure

The port on a NE currently used for the southbound interface is called the *command port* (c-port). The c-port value is important to the SDNC, since it need to set up flows in the NE for unicast communication with the SDNC. Unicast communication is also necessary for forwarding of TCP connections from lower-tier NEs without creating traffic loops.

The procedure to identify the c-port and for subsequent flow installation is called the *Port Discovery procedure* and is shown in Figure 3. It consists of the following steps:

1) The SDNC sends a UDP packet to the NEs IP address. It will be trapped by a flow (installed during the Connected state) and sent back to the SDNC as an OpenFlow PacketIn message, revealing the ingress port of the UDP packet. This port will be used also for the southbound traffic, called the c-port.
2) Flows will be installed for the NE-SDNC communication to take place over the (c-port,`LOCAL`) pair of ports.
3) Flows will be installed on-demand to handle connection from NEs in the lower tiers of the link tree. The individual flows will be installed as the lower-tier NEs floods their TCP SYN packets, since their MAC address and the connected NE port is not known until then. The detail of lower-tier connections will be presented in Section VI-B.

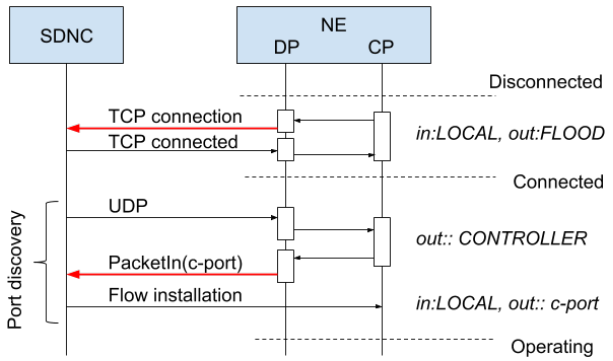

Fig. 3: Protocol elements of the Port Discovery procedure. Red arrows indicate flooded traffic. DP=data plane, CP=control plane

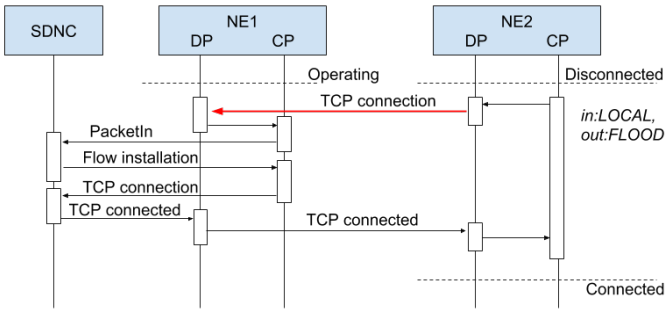

Fig. 4: Protocol elements for lower-tier NEs connecting to the SDN via a higher-tier NE

B. Connection of lower-tier NEs

NEs do not know if they are directly or indirectly connected to the SDNC, and they do not know which port that leads in that direction. The state diagram and the Port Discovery procedure just described are valid for all NEs regardless their position in the link tree.

The following paragraphs describe the process whereby an NE establishes an indirect connection to the SDNC. The protocol elements of this process is shown in Figure 4.

ARP and TCP SYN packets flooded from an NE in the Disconnected state will be received by one or more of its link neighbors. Each of them will, provided that they are in Operating state, pass the frame towards the SDNC, which will respond by an installation of the flows for passing southbound traffic between the ingress port and the c-port, i.e., connecting the lower-tier NE to the control plane link tree so that it will enter the Connected state.

Three details should be mentioned: (1) This process will be repeated for every tier of NEs towards the SDNC, since they all need these flows to be installed to serve the new connecting NE, (2) the flows are given an `idle_timeout` expiration time, to sanitize stale flows if forwarding paths change, (3) several of the NEs neighbors may try to create a forwarding path to SDNC, and a timestamp based mechanism will reject a connection attempt if the same lower-tier NE has connected through the same higher-tier NE recently (the last 10 seconds), applying the heuristic that the first received connection request has chosen the best path. This suppression mechanism will

 ISBN: 978-1-61208-740-5

have to keep rejecting until the lower-tier NE has completed the Port Discovery procedure and started to use c-port (rather than flooding) for communication with the SDNC, i.e., it will have entered the Operating state.

VII. Proactive adaptive forwarding algorithm

Another algorithm for dynamic path discovery and management was investigated, based on a proactive design principle. The NEs, the clients, the communication links and the ports they are connected to are known in advance in the form of a *catalogue*, and the spanning tree calculation can be done by the controller and the resulting structure be imposed on the NE structure proactively.

Having a detailed catalogue of the communication resources and the connected clients may sound cumbersome, but it is frequently seen in military application that detailed information about the system configuration is a prerequisite for the planning of an operation. For this reason, the presence of a catalogue is not regarded as an unreasonable requirement. The following sections will present the essential characteristics of the proactive algorithm and implementation details.

A. SDNC-initiated connections

The Ryu framework does not offer any blocking operations for connecting to NEs. The TCP connection always originates from the NE and results in an event in the controller code. An unconnected NE will continuously make connection attempts, so once the SDNC has created a path between it and the NE, a connection is expected to take place within a few seconds. The failure of a connection is therefore indicated in the SDNC code by a timeout event, while a successful connection is indicated by a connection event.

Once connected, the liveness of the connection is monitored by the SDNC through heartbeat messages. Connection loss is signalled in the SDNC by a *dpset* event.

B. No port discovery needed

Port details are recorded in the catalogue so the Port Discovery procedure described in Section VI-A is no longer necessary. The c-port is now used differently than in the reactive algorithm; the c-port is used only for transit traffic to and from lower-tier NEs. The control plane traffic originated in the NE (ARP and TCP port 6633 from the `LOCAL` port) is flooded to all ports, like in the Connected state in Section VI. Similar incoming control plane traffic from any port is forwarded to the `LOCAL` port. This choice was made to simplify the procedure whereby the SDNC will operate an NE through a different port, as a part of a fail-over procedure. The flooded traffic will not be forwarded by the neighbor NEs except for the parent node in the spanning tree, which is given explicit flow instructions for that purpose.

Figure 5 shows how connections are being made by NEs in successively lower tiers in the spanning tree. The tree structure is shown on the right side, and the interaction diagram shows the details of the protocol. The spanning tree is traversed in-order and connections are accepted from successively lower tiers of the tree. Dotted lines are links that are not part of the spanning tree but still employed by the data plane. The grey boxes on the interaction diagram on the left indicates operations that are given a deadline and watched by a timer mechanism. Transactions shown with a red arrow are flooded to all output ports of the NE.

C. Fail-over procedure

The failure of a NE or a link, indicated by the loss of a TCP connection or failure to create one, will cause a fail-over procedure to be executed. The loss of connection to a node can be caused by failure in any link or node along the path from the SDNC to the NE, and a link failure will cause loss in all nodes connected along that link, but the resulting software events may be generated in any order. It is therefore a complicated task to identify the exact failed link, and a heuristic has been chosen to simplify the algorithm: Communication loss to a node is taken as an indication of a failure in the link closest to it, i.e., the link connecting to the nearest parent node. This simplification may generate unnecessary fail-over actions, but the spanning tree will still be operating correctly and the optimized tree structure will be constructed later as a result of other procedures soon to be described.

If a disconnected node has not alternative paths to it, the parent node will keep its existing flow, expecting the node to resume operation later. This rule alleviates the consequences of a mistaken fault detection, as discussed in the previous paragraph.

Figure 6 shows state transitions on a spanning tree during a link failure. Since node E is first reported as disconnected, the fail-over action in State 2 is taken. Later, when node C also is reported as lost, the fail-over action shown in State 3 is taken. All nodes are now connected, although node E is one step deeper than necessary in the tree. On a later instance, the SDNC can use the Peer Discovery information (cf. Section VIII-B) to learn that the link C-E is in operation, and optimize the spanning tree accordingly.

VIII. Broadcast-free operation

The suggested network design is an L2 switched network with topology loops, and must therefore refrain from ordinary broadcast-operation in order to avoid endless traffic loops. The switches cannot broadcast received frames as a part of the switches' MAC-learning process. For this purpose four services have been implemented:

A. Proxy ARP

ARP requests from a client (connected to an NE) are trapped by the NE and passed to the SDNC for resolution. For this purpose, the SDNC maintains an ARP table (MACaddr,IPaddr) which learns from all packets delivered to the SDNC, not only ARP replies. In case of a table miss, the SDNC will instruct every individual NE to flood the request, and the resulting ARP reply is trapped in the first switch and sent to the SDNC, which will update its ARP table. Since NEs do not forward the flooded request, traffic loops will not be formed.

Actually, the ARP table also contains columns for the identifier of the NE connecting to the client, and the port number of the NE used for the connection. This information is

 ISBN: 978-1-61208-740-5

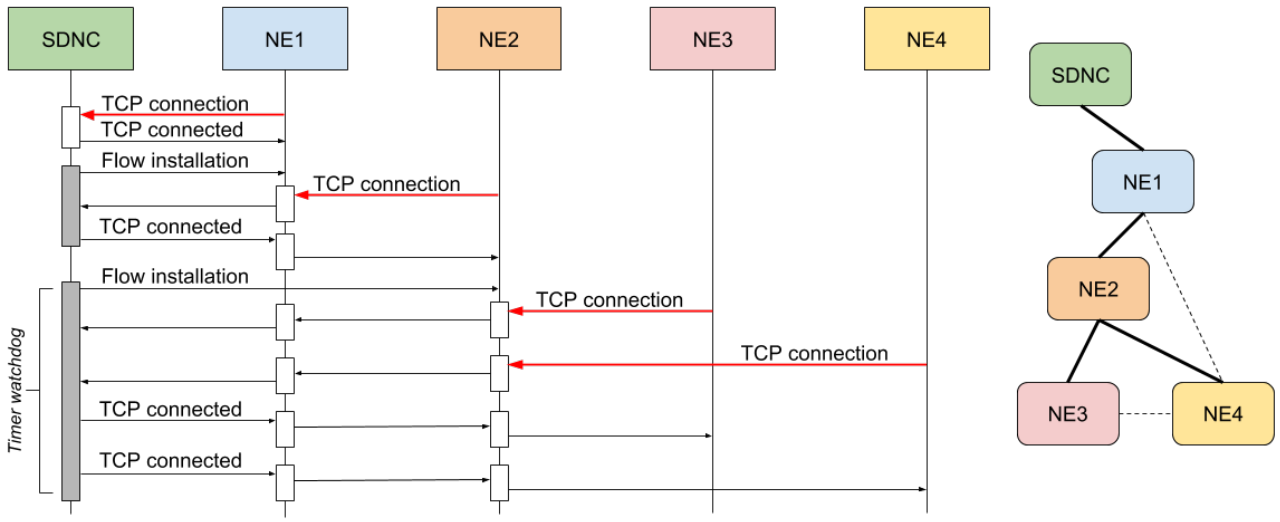

Fig. 5: Connection creation in a spanning tree of NEs using the proactive algorithm. See the text for detailed explanations.

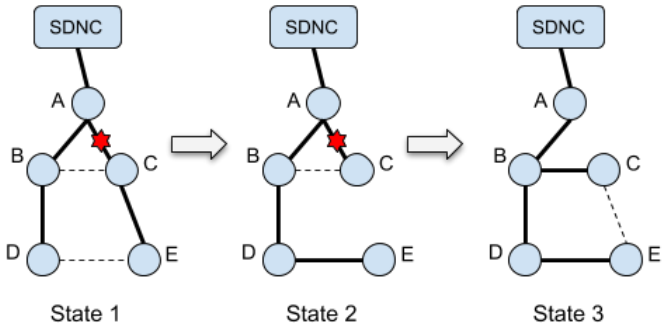

Fig. 6: A possible set of state transitions in the spanning tree as a result of superficial fault detection. See text for full details.

used for MPLS based forwarding, explained in section VIII-C. The bond between L2-mechanisms like MPLS to an IPv4 protocol like ARP is not a good design, and Section VIII-D will describe a cleaner alternative to broadcast operation.

B. Peer Discovery

To identify all links in the data plane, and to detect link failure, regular messages are sent from SDNC to every NE with instructions to flood the data through all ports. At the other end of the link, the received frame is passed to the SDNC which adds a timestamp and updates its link database. This method is quite similar to what is found in OpenFlow Discovery Protocol [10] and Bidirectional Forward Detection (RFC 5880).

In the case of the proactive algorithm (cf. Section VII), explicit messages for this purpose may not be necessary, since every NE will reply to heartbeat polling from the SDNC with a flooded frame, and these frames can be used for peer discovery. This protocol variant has not been tested.

C. MPLS based forwarding

In the NEs, traditional MAC-learning is not used, since that involves broadcast operations. Instead, every NE is associated with an MPLS label value, and every frame sent from a client will be given an MPLS label indicating the egress NE on the path to the destination. The egress NE will strip off the MPLS label and deliver the frame to the destination host.

This process relies on a number of information sources: The ingress switch needs a flow to attach the MPLS label, and the egress switch will need a flow to strip off the MPLS label and deliver the frame through the correct port. Intermediate NEs will need flows to associate MPLS labels with an output port. This forwarding information is derived from the peer discovery protocol link database (using a shortest path calculation) and installed as flows in NEs as needed.

D. Multicast trees

The ARP proxy described in Section VIII-A binds L2-mechanisms to the IPv4 protocol, which is not a desirable design, since the infrastructure should not make any assumptions with regard to the network-layer protocol in use. Therefore, at a later iteration of the design, the peer discovery link database was used to build multicast trees for every NE, so that broadcast frames initiated from one NE will propagate to every NE in a loop-free manner, and the NE will again deliver the frame to every connected client. MPLS labels are used to identify the originating NE of a broadcast frame, so that intermediate NEs may make the correct forward decisions.

This arrangement permits the use of, e.g., IPv6 Neighbor Discovery Protocol (NDP), so that IPv6 and IPv4 traffic can use the same mechanisms for broadcast frames. However, it is not established if the ARP proxy described in Section VIII-A is still more effective in terms of link usage.

IX. Performance and Scalability

The performance and the scalability of the proactive and reactive algorithms will be discussed in this section. The lab design shown in Figure 1 was used to develop the algorithms and test the functional correctness of the reactive algorithm (the proactive design has not been tested). However, this design

 ISBN: 978-1-61208-740-5

was not sufficient for scalability experiments or comprehensive performance measurements.

The chosen baseline for the performance discussion is the Spanning Tree Protocol in the default configuration. A number of measurements indicated that STP recovers the network in **30 seconds after** a link failure. The reactive algorithm, with our chosen parameters, exhibits an average recovery time of **14 seconds**.

A link discovery mechanism based on polling (in the form of renewed flows) will introduce a trade-off between failure detection time and generated traffic volume. A halved detection time will require the doubled number of liveness control messages. Besides, once a link connection has been established, an unconnected switch will make connection attempt with regular intervals and thus introduce a mean delay after a path to the SDNC has been established.

With regards to scalability, both the reactive and proactive algorithms need to build up a path between NE and SDN step by step. As the number N of NEs grow, the average number of links D between an NE and the SDN is expected to be growing like the depth of a tree:

$$D = O(logN) \tag{1}$$

The establishment of a single step in the path from an NE to the SDN will generate a constant number of messages, so the total volume M of messages associated with path discovery from every NE is expected to be

$$M = O(NlogN) \tag{2}$$

The required time T for re-connection is assumed to grow with the number of links in the path and with the same order as D:

$$T = O(logN) \tag{3}$$

During the experiments, it was observed that a connection from an NE before the SDNC had closed the previous SSL connection resulted in a "duplicate connection attempt". A re-connection attempt from an NE should not happen earlier than this timeout value in the SDNC, which is a configurable parameter value in Ryu.

X. Conclusion and future research

The contribution of this paper is a detailed analysis and a proof-of-concept implementation of an in-band control plane with failure detection and dynamic path discovery. These properties allow for a control plane that survives link and node failure, since it will employ alternative paths to connect the NEs to SDNC. An important contribution is that these mechanisms are offered in the presence of link loops. Link loops represent redundant communication resources which should be employed for load balancing and traffic separation, not only for resilience, which is why the Spanning Tree Protocol was abandoned.

Future research and development on this topic will include better testing of the separation between L2 and L3 protocols, so that the ARP proxies are replaced by mechanisms to distribute broadcast frames along multicast trees. We also are in the process to include multi-tenancy separation in the L2 forwarding mechanisms based on the clients' X.509 certificate information. IPv6 support will also be added to the prototype.

References

[1] J. Spencer and T. J. Willink, "SDN in coalition tactical networks," in *2016 IEEE Military Communications Conference, MILCOM 2016, Baltimore, MD, USA, November 1-3, 2016*, pp. 1053–1058, 2016.

[2] "Open vSwitch." http://openvswitch.org. Online, Accessed Aug 2019.

[3] L. Schiff, S. Schmid, and M. Canini, "Ground control to major faults: Towards a fault tolerant and adaptive SDN control network," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2016, Toulouse, France, June 28 - July 1, 2016*, pp. 90–96, 2016.

[4] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed sdn control planes," *SIGCOMM Comput. Commun. Rev.*, vol. 46, pp. 37–43, Jan. 2016.

[5] A. Fongen and G. Køien, "Trust management in tactical coalition software defined networks," in *2018 International Conference on Military Communications and Information Systems, ICMCIS 2018*, pp. 1–8, Institute of Electrical and Electronics Engineers Inc., 5 2018.

[6] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A self-organizing distributed and in-band SDN control plane," in *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, pp. 2656–2657, 2017.

[7] Y.-L. Su, I.-C. Wang, Y.-T. Hsu, and C. H.-P. Wen, "FASIC: A fast-recovery, adaptively spanning in-band control plane in software-defined network," in *IEEE GLOBECOM 2017*, pp. 1–6, Institute of Electrical and Electronics Engineers Inc., 12 2017.

[8] O. N. Foundation, "SDN architecture, issue 1.0." https://www.opennetworking.org/, 2014. Online, Accessed July 2019.

[9] "Ryu SDN Framework." https://osrg.github.io/ryu/. Online, Accessed Aug 2019.

[10] "OpenFlow Discovery Protocol." http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol. Online, Accessed Aug 2019.

 ISBN: 978-1-61208-740-5