Automated Bootstrapping of A Fault-Resilient In-Band Control Plane

Ermin Sakic
Mirza Avdic
Siemens AG
Technical University Munich

Amaury Van Bemten
Wolfgang Kellerer
Technical University Munich

ABSTRACT

Adoption of Software-defined Networking (SDN) in critical environments, such as factory automation, avionics and smart-grid networks, will require in-band control. In such networks, the out-of-band control model, prevalent in data center deployments, is inapplicable due to high wiring costs and installation efforts. Existing designs for seamlessly enabling in-band control plane cater only for single-controller operation, assume proprietary switch modifications, and/or require a high number of manual configuration steps, making them non-resilient to failures and hard to deploy.

To address these concerns, we design two *nearly* completely automated bootstrapping schemes for a multi-controller in-band network control plane resilient to link, switch, and controller failures. One assumes hybrid OpenFlow/legacy switches with (R)STP and the second uses an incremental approach that circumvents (R)STP. We implement both schemes as OpenDaylight extensions, and qualitatively evaluate their performance with respect to: the time required to converge the bootstrapping procedure; the time required to dynamically extend the network; and the resulting flow table occupancy. The proposed schemes enable fast bootstrapping of a robust, in-band managed network with support for seamless redundancy of control flows and network extensions, while ensuring interoperability with off-the-shelf switches. The presented schemes were demonstrated successfully in an operational industrial network with critical fail-safe requirements.

CCS CONCEPTS

• **Networks** → **Network management**; • **Computer systems organization** → *Dependable and fault-tolerant systems and networks*;

KEYWORDS

Software Defined Networking, network bootstrapping, distributed control plane

ACM Reference Format:
Ermin Sakic, Mirza Avdic, Amaury Van Bemten, and Wolfgang Kellerer. 2020. Automated Bootstrapping of A Fault-Resilient In-Band Control Plane. In *Symposium on SDN Research (SOSR '20), March 3, 2020, San Jose, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3373360.3380829

SOSR '20, March 3, 2020, San Jose, CA, USA

ACM ISBN 978-1-4503-7101-8/20/03…$15.00
https://doi.org/10.1145/3373360.3380829

1 INTRODUCTION

Adoption of SDN architecture in industrial scenarios, especially as the enabler of Time Sensitive Networking mechanisms, has recently started gaining traction [2, 11, 15, 18, 19, 19, 22, 28, 35, 45]. Initial SDN deployments (i.e., in data-centers) have relied on *out-of-band control* (OOBC) with which control traffic is exchanged between switches and SDN controller(s) using dedicated links and switch management ports. Bootstrapping with OOBC is trivial, due to controllers being capable of controlling the switches without requiring connectivity to any other switches. OOBC, however, is often undesirable in large-scale and critical industrial environments due to associated CAPEX and installation efforts [4, 18, 39]. For example, deploying an additional network in machine tool manufacturing or drive technology networks is immensely expensive, due to associated cabling costs for the shielding required for RF/EMI noise suppression. Similarly, avionics and automotive networks benefit greatly from decreased cabling weight [18].

With *in-band control* (IBC), control traffic is forwarded along the same links used for the data plane traffic. This makes IBC networks the preferable solution from the cost and operational perspectives. However, their implementation is challenging. For example, current most popular open-source controller platforms OpenDaylight (ODL) [24] and ONOS [5] provide no mechanisms to support or automate the IBC bootstrapping, nor do they allow for protection of control plane traffic against arbitrary link and node failures, both which are of critical importance in industrial networks [2]. Summarized, the bootstrapping of a *reliable* IBC softwarized network must address the following challenges:

(1) Establishing robust switch-controller and controller-controller connections using IBC and an initially unconfigured control plane;
(2) Preserving the control plane stability in case of multiple link, switch and/or controller failures;
(3) Support for addition of new nodes and links to the previously reliably bootstrapped IBC network;
(4) Minimization of manual per-device pre-configuration.

The few automated bootstrapping approaches to date address the above mentioned challenges partially or in isolation of each other. For instance, they neglect the single-controller single-point-of-failure issue [39–41], do not guarantee reliable control connections [37, 43], omit the practical constraint of controllers' state synchronization prior to switch reconfigurations [7, 36, 37], or assume functional extensions of switch components, e.g., the DHCP client or OpenFlow agent [4, 39, 41]. This limits their applicability

Design	Auto. Switch Management IP Provisioning	Auto. Controller List Provisioning	Resilience of Control Flows	Multi-Controller Support	(R)STP Not Required	No Proprietary Switch Extensions
Sharma et al. [39–41]	✓	✓	✓ / ✗ (reactive)	✗	✓	✗
Schiff et al. I [36–38]	✗	✗	✓ / ✗ (timeout)	✓	✓	✓
Schiff et al. II [6]	?	?	✓ (timeout/reactive)	✓	✓	✓
Heise et al. [18]	✗	✗	✓ (proactive replication)	✗	✗	✓
Canini et al. [7]	✗	✗	✓ (reactive)	✓	?	✓
Su et al. [43]	✗	✓	✓ (reactive)	✗	?	✓
Bentstuen et al. [4]	✓	✓	✗	✗	✓	✗
Hybrid Switch (HSW)	✓	✓	✓ (proactive replication)	✓	✗	✓
Hop-by-Hop (HHC)	✓	✓	✓ (proactive replication)	✓	✓	✓

Table 1: Comparison of HSW and HHC to existing schemes. Additional details are provided in Sec. 9.

in existing infrastructures. Most concerning, the reproducibility of state-of-the-art proposals in real environments is often limited, due to unclear pre-configuration assumptions or closed-source implementations. An overview of the relevant literature is presented in Table 1 with detailed comparisons covered in Sec. 9.

Our contributions. This paper tackles the challenge of bootstrapping a reliable IBC softwarized network while satisfying challenges (1)–(4). Succinctly, our contributions are:

- We compare the existing bootstrapping approaches w.r.t. their reliance on legacy protocols, the need for manual configuration and proprietary extensions, the support for control flows resilience and multi-controller configuration.
- We propose two novel automated bootstrapping schemes targeting reliable IBC networks with multi-controller support and solving the above-mentioned challenges.
- We evaluate the proposed designs in realistic industrial topologies with varying sizes and controller placements w.r.t. *(i)* the time required to converge the bootstrapping procedure, *(ii)* the time required to dynamically extend the network, and *(iii)* the flow table occupancy. We limit our evaluation to the two schemes proposed in this paper, due to the unavailability and/or ambiguity of closed-source state-of-the-art solutions.
- We discuss implementation aspects relevant for the successful introduction of the designs in networks equipped with off-the-shelf OpenFlow agents.

In order to ensure reproducibility, we have released the complete source code for both our approaches (a critically missing aspect of existing works) [1].

Organization. Sec. 2 provides the motivation for automated and reliable in-band bootstrapping. Sec. 3 presents the system model and requirements and summarizes the terminology. Sec. 4 and 5 describe the two novel bootstrapping schemes. Sec. 6 discusses selected design & evaluation aspects, Sec. 7 presents the evaluation methodology and Sec. 8 accordingly discusses the results. Sec. 9 compares the presented designs with prior works. Finally, Sec. 10 concludes the paper.

2 MOTIVATION

(1) Establishing robust switch- and controller-controller communication using IBC and initially unconfigured control plane:

In industrial networks, OOBC imposes a requirement for ruggedized wiring immune to electromagnetic interference and heavy electrical surges typical for electric utility substations, factory floors and traffic control cabinets, further raising the costs. Alternatively, IBC is the preferable way of controlling SDN switches. It involves sharing the same physical network for data plane and control plane traffic forwarding. However, in contrast to OOBC, IBC requires a more complex initial setup, due to: i) unpredictable data plane traffic that could impact the control plane responsiveness; ii) data plane failures which may impact the availability of the control plane; iii) establishing deterministic control plane flows in a non-configured data plane is non-trivial, and typically requires manual configuration.

(2) Preserving the control plane stability in case of arbitrary number of link, switch and/or controller failures:

To protect against controller's single-point-of-failure, state-of-the-art network controllers deploy and replicate their state across multiple replicas in strong and/or eventually consistent manner [21, 29, 34]. ODL [24] and ONOS [5] leverage the strong consistency model relying on RAFT consensus [20]. RAFT guarantees linearized reads and writes at all times, including network partition and controller failure scenarios. In the context of a distributed SDN control plane, each controller instance is a RAFT node with a data-store that stores information about the network topology and switches. Prior to allowing for data-store modifications, RAFT elects a leader replica, responsible for committing state ordering updates into the distributed data-store and their replication to followers. If a leader fails, a new leader is elected, resulting in a temporary downtime [34, 47]. Only a replica with an up-to-date data-store may become a new leader [20, 27].

Ensuring consistent topology and switch state views relies on controller-to-controller communication, thus making it crucial for bootstrapping procedure to enable robust control channels by design. The physical dislocation of controllers additionally imposes transmission of the synchronization traffic using IBC flows. ODL's southbound module (*OpenFlowPlugin*) requires a prior controller-to-controller channel establishment and a successful consensus round even for a "trivial" population of OpenFlow rules during bootstrapping. Similarly, the per-switch controller's role (i.e., "backup" or "master") must be decided using consensus among the contending controllers. Hence the configuration of switches for multi-controller support and the support for controller state synchronization must be targeted in tandem.

Enabling a resilient distributed control plane thus requires the bootstrapping processes to ensure resilience in case of following failures:

- Failures of F out of $2F + 1$ [14, 34] controllers: No managed switches may be left unmanaged due to individual replica failures;

[1] Automated Bootstrapping of In-Band Controlled Software Defined Networks [GitHub] - https://github.com/ermin-sakic/sdn-automated-bootstrapping

- Assuming $k + 1$ disjoint paths between each two controllers, failure of k switches / links must not result in control plane partitioning [14, 47].

(3) Support for node and link extensions of the previously reliably bootstrapped IBC network:

Dynamic network topology changes, i.e., attachment of new end-devices (host discovery) and packet forwarding devices to existing managed networks should be natively supported by the proposed designs. In particular, dynamic plugging and unplugging of machine networks (i.e., network cells) in industrial backbones, reconfiguration of modular machine network assemblies and addition/removal of production line networks are typical Industry 4.0 use cases [11, 28].

(4) Minimization of manual per-device pre-configuration:

Involved configuration effort should be minimized - Configuration of controller lists (comprising IP address / port number pairs) and switch identities should be automated so to enable agile device deployment / network extensions (e.g., for VNF on-boarding). To this end, controllers or external DHCP servers should be charged with providing switches with management IPs and thus automated identity binding.

3 SYSTEM MODEL AND TERMINOLOGY

3.1 General Model

We assume the deployment of $2F + 1$ controller replicas in order to cater for F non-Byzantine [32] controller failures. A *failed* controller replica is an inactive, unreachable replica. Apart from individualized identity configuration, each replica is an exact clone and thus implements the OpenFlow southbound logic and is capable of executing the bootstrapping procedure. Updates to a replica's distributed data-store are synchronized among all reachable and active replicas using RAFT.

Switches are operated in OpenFlow Master-Slave mode [42], where any controller replica may become the Master of a switch. Forwarding table modification messages initiated by a Slave controller are proxied through the switch's Master. All switches are controlled in IBC manner, using OpenFlow v1.3+ [42]. To enable the (optional) automated identity (IP address) and controller list assignment, each switch deploys a DHCP client and an SSH server. The controllers accordingly execute DHCP server instances for automated IP address roll-out. For purpose of automated authentication, prior to bootstrapping step, switches and controllers may have their certificates or symmetric keys on-boarded so to enable PKI-, or MAC-based authentication [10, 12], respectively. Depending on the target scheme, the switches either all support (R)STP or they do not (i.e., they have it disabled). Moreover, each switch is capable of forwarding traffic via traditional MAC learning (using `NORMAL`-mode forwarding). We assume an IPv4-enabled infrastructure.

A k link / switch fault resilience model requires an adequate underlying topology, where $k + 1$ disjoint paths can be found for any controller-switch / controller-controller pair. For simplification, we relax this condition and model a link failure between an SDN controller and the switch directly attached to it as a controller failure. We duplicate traffic on disjoint paths for each communication pair. Higher layer protocols are then tasked with duplicate elimination, i.e., we rely on TCP at transport layer. This is feasible, since both OpenFlow and controller-to-controller synchronization in OpenDaylight transmit TCP traffic. A generalized solution should instead deploy robust duplicate frame elimination measures (e.g., as per TSN 802.1CB [30]).

3.2 Secure and Standalone Modes

In *Standalone* mode, a switch forwards packets autonomously - i.e., it behaves like a non-managed MAC learning forwarder. Namely, it contains one *generic* OpenFlow rule that matches all traffic and forwards it to the reserved `NORMAL` (ref. [23]) port. After establishing a connection to a controller, the `NORMAL` rule is automatically removed. In *Secure* mode, it is non-existent. Following an established controller connection in Secure mode, the flow table remains unmodified, and the forwarding behavior continues according to controller-added rules. In case of an empty flow table and non-configured controller lists, the switch in Secure mode drops all arriving traffic. Depending on the scheme, switches must support either *Secure and Standalone* mode or *Secure* mode only.

3.3 In-band Mode

With *In-band* mode, a switch is initialized with *generic* rules that ensure controller-destined traffic is forwarded according to the MAC learning table (i.e., using the `NORMAL` port).

In switches basing their OpenFlow implementation on the Open vSwitch (OVS) agent, the In-band rules have a priority higher than any flow rules that may be configured by the controller. There, In-band mode automatically pre-configures the rules for forwarding: i) ARP requests and DHCP Discover messages generated by the switch; and ii) ARP replies and DHCP Offer messages destined for the switch. For each controller, rules matching following traffic are added: i) ARP replies destined for controller's MAC address; ii) ARP requests generated with controller's MAC address as source; iii) ARP replies containing controller's IP address as target; iv) ARP requests containing controller's IP address as source; v) traffic destined for controller's IP / TCP port pair; and vi) traffic with controller's IP and TCP port as source.

4 HYBRID SWITCH (HSW) SCHEME

We first introduce the Hybrid Switch (HSW) bootstrapping scheme that heavily relies on existence of (R)STP, Standalone, In-band modes, and `NORMAL` forwarding. HSW leverages (R)STP to establish an acyclic graph and thus remedy the potential storm issues stemming from traffic broadcasts (e.g., from ARP and DHCP requests). Fig. 1 summarizes the workflow of HSW. In `Phase 0`, controllers establish the controller-to-controller communication over the spanning tree computed by the network. In `Phase 1`, switches are provided dedicated management IP addresses and controllers' IP/port pairs. In `Phase 2a`, controllers establish the control over the connected switches, install a set of initial rules and eventually disable (R)STP in switches so to enable blocked ports. Explicit resilient flow rules are embedded in `Phase 2b`, so to fulfill the fault-tolerance requirements. To support network extensions at runtime (`Phase 3`, not depicted), a virtual spanning tree is maintained in controllers and enforced for discovery traffic, allowing for safe forwarding

of (broadcast) traffic initiated by newly attached nodes, even after disabling (R)STP.

HSW requires the following assumptions to hold at startup: i) Controllers are aware of IP addresses of other controllers, or are capable of resolving these using standardized DNS queries; ii) Switches are initialized in Standalone mode with (R)STP and In-band mode enabled; iii) Switches are provisioned with controllers' public certificates or symmetric MAC keys [10, 12].

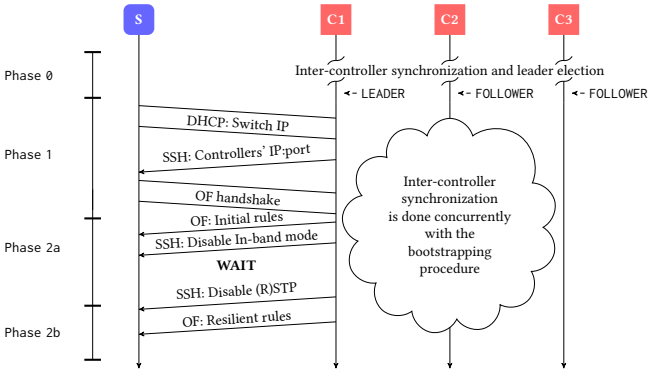


Figure 1: HSW - Message sequence diagram of the bootstrapping procedure as described in Sec. 4.

4.1 Phase 0 - Network Startup

4.1.1 Phase 0a - *In-Band Flow Rules.* Switches boot with In-band flow rules set up, enabled to bidirectionally forward traffic between switches and controllers. The traffic matching In-band flow rules is forwarded using the reserved NORMAL OpenFlow port. After a switch establishes a connection with the controller, due to enabled Standalone mode, the switch automatically deletes any rules which are of a priority lower than the In-band rules. Without enabling the In-band mode, this behavior would lead to switch flow tables becoming empty, and thus traffic drops, including the controller-initiated OpenFlow traffic. Instead, In-band mode rules (ref. Sec. 3.3) ensure the incoming traffic is forwarded even after a successful initial controller-switch connection.

4.1.2 Phase 0b - *Controller Synchronization.* The switches initially boot with Standalone mode and (R)STP enabled. Prior to taking the switch ownership, the controllers elect a leader and establish an active RAFT cluster using the established spanning tree thus enabling synchronization.

4.2 Phase 1 - Distribution of Switch and Controller Connection Identifiers

As mentioned before, a fully-automated address distribution to the switches' management interfaces is an optional feature of our schemes. To this end, a DHCP server instance, e.g., realized as a replicated application of the controller, distributes the IPv4 addresses to switches on an FCFS basis. To omit potential address conflicts, the RAFT leader replica is charged with address leasing using an automatically derived list of free addresses given the reserved controller IPs. Follower replicas on the other hand, ignore DHCP requests, as long as the current leader is active. In case of duplicate DHCP requests, DHCP server replies only to the first request [39].

The leader then establishes management session with provisioned switches (using SSH / OVSDB). Since in this phase all switches forward control traffic according to the NORMAL data-path (Standalone mode) the connection establishment succeeds. The leader next proceeds to provision the switches with redundant controllers' IP addresses and TCP ports.

4.3 Phase 2 - Enabling a Functional and Resilient Control Plane

The goal of Phase 2 is to provide the discovered switches with rules necessary to enable a resilient control plane. Phase 2 is divided into two sub-steps: Phase 2a - initial control plane rules are installed to allow communication with adjacent switches; and Phase 2b - resilient control plane rules are installed. To avoid broadcast storms, (R)STP remains functional until all initially booted switches are discovered. (R)STP is disabled in Phase 2b and, after all links were discovered, RAFT leader computes and embeds resilient paths.

4.3.1 Phase 2a - *Initial OpenFlow Flow Rules.* For each connected switch the leader controller installs the rules depicted in Table 2. The matching semantics correspond to OpenFlow fields defined in [42]. The leader next disables the In-band mode flow rules using the SSH / OVSDB management channel. This is done in order to enable full control over control plane traffic forwarding based on initial rules. Otherwise, e.g., in OVS, the switches would continue to forward OpenFlow, ARP and DHCP traffic according to In-band mode imposed rules, since they have a highest available priority.

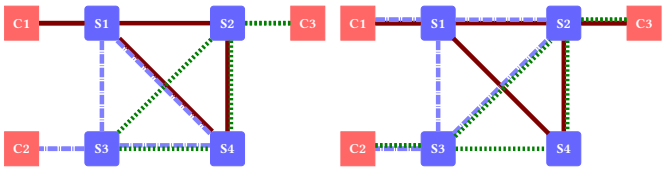
On rule semantics (Table 2): DHCP and SSH rules enable a continued successful execution of Phase 1 for switches adjacent to the configured switch. ARP rules prevent the ARP table cache timeouts. LLDP rules allow the controllers to discover topology updates. An additional per-controller ARP rule is required for controller's placement discovery. To this end, controllers periodically generate probe packets.

4.3.2 Phase 2b - *Enabling Control Plane Resilience.* After the initial rules embedding, the leader controller disables (R)STP on each switch in order to fully discover the underlying topology. To ensure the topology is entirely discovered, the leader controller waits for a predefined period (ref. Sec. 6.2) and eventually computes and installs the resilient rules. Resilient control flows are installed for each switch-controller and controller-controller pair as per Sec. 4.5. Finally, the controller removes the initial Phase 2a rules (except for LLDP and ARP).

After finalizing Phase 2b, the control plane is resilient according to Sec. 2 requirements. Data plane failures are covered using the disjoint paths. Leader controller failures are covered by deploying multiple controller backup instances.

Purpose	Packet Type	Matching	Action
Dynamic switch IP address configuration	DHCP	udp, udp_src=68 udp, udp_src=67	Send to NORMAL
Remote switch configuration	SSH	tcp, tcp_src=22 tcp, tcp_src=22	Send to NORMAL
Controller-Switch OpenFlow interaction	OpenFlow	tcp, tcp_src=6653 tcp, tcp_dst=6653	Send to NORMAL
Switch IP resolution	ARP	arp, arp_tpa=control plane IP prefix arp, arp_spa=control plane IP prefix	Send to NORMAL
Topology discovery	LLDP	eth_type=0x88cc	Send to CONTROLLER
Controller IP resolution	ARP	arp, arp_tpa=c1 IP arp, arp_spa=c1 IP	Send to NORMAL
		... arp, arp_tpa=cN IP arp, arp_spa=cN IP	Send to NORMAL
Controller self- discovery	ARP	arp, arp_tpa=arbitrary IP	Send to CONTROLLER
Inter-controller synchronization	TCP	ip, ip_src=c1 IP, ip_dst=c2 IP ip, ip_src=c2 IP, ip_dst=c1 IP	Send to NORMAL
		... ip, ip_src=cX IP, ip_dst=cY IP ip, ip_src=cY IP, ip_dst=cX IP	Send to NORMAL
NEXT: Dynamic switch IP address configuration	DHCP	in_port=TREE port, udp, udp_src=68 in_port=TREE port, udp, udp_src=67	Send to other TREE ports
NEXT: Remote switch configuration	SSH	in_port=TREE port, tcp, tcp_src=22 in_port=TREE port, tcp, tcp_dst=22	Send to other TREE ports
NEXT: Controller-Switch OpenFlow interaction	OpenFlow	in_port=TREE port, tcp, tcp_dst=6633 in_port=TREE port, tcp, tcp_src=6633	Send to other TREE ports
NEXT: Any ARP traffic	ARP	in_port=TREE port, arp	Send to other TREE ports
NEXT: Network extension discovery	DHCP	in_port=INACTIVE port, udp, udp_src=68	Send to CONTROLLER

Table 2: HSW - Initial and Network Extension (NEXT) flow rules installed on switches throughout Phase 2.



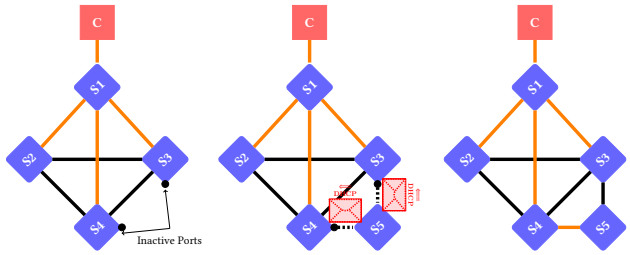
(a) S-C resilient paths **(b) C-C resilient paths**

Figure 2: Exemplary resilient path output for $k = 1$ for: (a) switch-controller paths between S4 and all controllers; and (b) all controller-controller pairs.

4.4 Phase 3 - Dynamic Network Extensions

To enable dynamic network extensions at runtime, the managed switches which are direct neighbors of the newly booted switches must forward the control plane traffic between the newly connected switches and controllers. In particular, DHCP, SSH, OpenFlow and ARP traffic forwarding must be enabled so that newly added switches can be bootstrapped. Since the HSW disables (R)STP prior to executing Phase 2b, the rules that should match above mentioned traffic may not rely on MAC-learning based NORMAL forwarding, due to potential broadcast storms. Instead, the leader controller maintains a virtual tree topology, used to compute and install the network extension (NEXT) rules that broadcast the control plane traffic to and from newly attached switches on the tree links only. Due to generic match semantics, NEXT rules are installed with a lower priority than Phase 2b rules.

A special discovery rule matches packets arriving on inactive ports (i.e., the last rule of Table 2). An inactive port is a port without an active neighbor, i.e., initially unconnected to another switch. If a new switch is connected to an already bootstrapped network, the discovery rule will forward its DHCP Discover message to controllers, encapsulated as a PACKET-IN message. The PACKET-IN contains the information about the ingress port the message arrived on, i.e., corresponding to the previously inactive port. The leader replicas processes the PACKET-IN by extracting the newly attached switch's MAC address, and adding the previously inactive port to the existing tree. If the new switch is connected to the existing network with multiple links, only the first activated port is used. Fig. 3 illustrates this scenario.



(a) Initial virtual spanning tree. **(b) Addition of a switch with 2 links.** **(c) Resulting virtual spanning tree.**

Figure 3: Exemplary network extension with one switch and two links. c) depicts the resulting tree.

4.5 Control/Data Plane Failure Handling

Impact of Data Plane Failures on Control Plane Flows: To tolerate switch node/link failures, Phase 2b identifies $k + 1$ resilient paths for each controller-controller and controller-switch connection pair, necessary to tolerate k arbitrary data plane failures. We implement a simple algorithm to find such paths. The leader controller executes $k + 1$ subsequent Dijkstra runs per connection pair. After each run, the link metrics of the found path are multiplied by a factor of 1000. Typical industrials networks having diameters of at most dozens of hops [19, 46], this ensures previously used links are reused only if no other path is available. Thus, through this iterative metric adaptation, $k + 1$ maximally disjoint paths are effectively computed (as in 45.3.3 of 802.1Q-2018 [1]). The leader controller then maps and installs these paths. Exemplary resilient paths for $k = 1$ are depicted in Fig. 2.

Resilient flow rules are installed for ARP, OpenFlow and SSH flows. OpenFlow and SSH use TCP at transport layer, that takes care of duplicate packet elimination. Duplicate ARP requests and replies, on the other hand, are delivered duplicated to sink nodes, which does not negatively impact the correctness. Thus, with the "seamless" replication mechanism, individual data plane failures never lead to packet loss, as long as disjoint alternative paths exist. In [25, 26, 33], the resulting global inter-controller traffic generated by topology state exchange was shown to grow quadratically with the number of controllers and linearly with the number of network elements. For moderate control plane sizes, the imposed control plane load for a routing application was determined close to negligible - ~ 6.7 Mbps of per-replica load in a 5-replica cluster [33]. Hence, we propose a form of conditional traffic policing for misbehaving redundant switch-controller and controller-controller flows but do not investigate this issue further in this paper.

Impact of Data Plane Failures on Network Extension Tree: Failures of individual data plane elements must result in adaptation of the

network extension tree. The approach we followed in Sec. 4.4 assumes initial tree computation and embedding, following by the controllers computing an alternative tree for each possible data plane failure in the previously embedded tree. When a link in the underlying topology fails that is mapped to the currently active tree, the leader controller refreshes the rules with those of an alternative tree, proactively computed for that particular link / node change. During the transition period, no new switches can be admitted, but since data plane failures are rare events, we consider the additional delay in network extension, incurred by reactive tree re-embedding, a negligible disadvantage.

Handling Control Plane Failures: If the leader replica fails, the remaining controllers elect a new leader using the distributed leader election procedure, thus incurring a short interruption period where no client requests relying on consensus can be handled by the controllers [34]. If a follower replica fails, current leader and the operation of the system remain unaffected. The resilient control plane remains operational as long as the controller majority remains active.

Note: Tolerating transitional faults during Phases 0-2 is currently unsupported and will be investigated in future work.

5 HOP-BY-HOP (HHC) SCHEME

Our Hop-by-Hop scheme (HHC) realizes an iterative approach to switch discovery. In contrast to HSW, it alleviates the need for (R)STP and thus the (R)STP expiration timer (ref. Sec. 6.2). As before, a number of minimum constraints must hold at network startup: i) Controllers are aware of IP addresses of other participants, or are capable of discovering them using standardized DNS queries; ii) Switches are initialized in *Secure* mode *without* (R)STP and with *disabled* In-band mode; iii) Switches are provisioned with controllers' public certificates (PKI) or symmetric MAC keys [10, 12].

Fig. 4 depicts the abstract sequence diagram of HHC. In Secure mode, a switch relies on the initial *generic* (non-customized) flow table rules, available at boot time. By leveraging these, in `Phase 0`, the controllers establish bilateral connections. In `Phase 1`, switches are assigned management IP addresses and controller lists. Using the *generic* rules, the switches adjacent to controller establish their OpenFlow sessions. Appropriately in `Phase 2a`, the leader rolls out the control plane flow rules to these switches. The provisioned rules realize the spanning tree forwarding functionality, used for iterative propagation of next hop switch's control traffic to the controller. With each newly discovered network element, HHC iteratively updates the tree. In `Phase 2b`, the leader computes and installs resilient paths for all control plane flows, whenever such paths become feasible. The attachment of new switches to an already bootstrapped network is possible by gradually expanding the spanning tree.

5.1 `Phase 0` - Network Startup

5.1.1 `Phase 0a` - *Pre-Configured Flow Rules.* HHC assumes a set of initially preconfigured generic OpenFlow rules, necessary: i) to allow an initial connection with the controllers while in Secure mode; ii) to prevent broadcast storms in non-bootstrapped parts of a network.

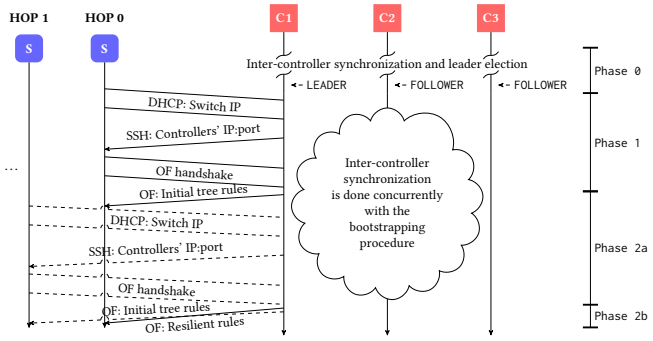


Figure 4: HHC - Message sequence diagram of the bootstrapping procedure as described in Sec. 5.

In contrast to HSW, which bootstraps individual switches concurrently in FCFS manner, HHC bootstraps the network iteratively hop-by-hop, starting from switches adjacent to the leader controller. The non-customized *generic* rules (ref. Table 3) allow receiving traffic addressed to the switch itself, i.e., dropping any other traffic, except for the traffic generated by the switch itself. Traffic initiated by a switch is flooded on all its ports. This traffic should only be allowed to reach the leader controller, and is therefore, dropped by any neighboring switches. These rules thus prevent the occurrence of broadcast storms (special case being the inter-controller flow rules, ref. Sec. 5.1.2). Furthermore, they allow the controller to configure the switches connected directly to it.

Purpose	Packet Type	Matching	Action
Dynamic switch IP address configuration	DHCP	in_port=LOCAL, eth_src=switch_mac, udp, udp_src=68 udp, udp_src=67	Send to ALL Send to LOCAL
Remote switch configuration	SSH	in_port=LOCAL, eth_src=switch_mac, tcp, tcp_src=22 eth_dst=switch_mac, tcp, tcp_dst=22	Send to ALL Send to LOCAL
Controller-Switch OpenFlow interaction	OpenFlow	in_port=LOCAL, eth_src=switch_mac, tcp, tcp_dst=6633 eth_dst=switch_mac, tcp, tcp_src=6633	Send to ALL Send to LOCAL
Switch-Controller IP Resolution	ARP	in_port=LOCAL, eth_src=switch_mac, arp, arp_op=1 eth_dst=switch_mac, arp, arp_op=2	Send to ALL Send to LOCAL
Controller-Switch IP Resolution	ARP	in_port=LOCAL, eth_src=switch_mac, arp, arp_op=2	Send to ALL
Controller-Switch / Controller IP Resolution	ARP	arp, arp_op=1 arp, arp_op=2	Send to ALL
Inter-controller Synchronization	TCP	tcp, tcp_src=2550 tcp, tcp_dst=2550	Send to NORMAL

Table 3: HHC - Pre-configured OpenFlow rules

5.1.2 `Phase 0b` - *Controller Synchronization.* Excluding (R)STP implies we should avoid forwarding controller synchronization traffic using `NORMAL` port so to avoid broadcast storms. However, Secure mode implies that this traffic must be handled with additional initial preconfigured rules that match and forward this traffic type (ref. last four rules of Table 3), prior to establishing OpenFlow connection with controller. Thus, it is impossible to come up with the generic set of preconfigured flow rules that do not leverage the `ALL` or `NORMAL` ports. Using either, however, initially results in broadcast storms. Therefore, apart from the preconfigured flow rules, we deploy a mechanism to cope with broadcast storms for controller-to-controller traffic (ref. Sec. 6.5).

5.2 Phase 1 - Distribution of Switch and Controller Connection Identifiers

The leader controller assigns the IP addresses initially only to its direct neighbor switches, and subsequently provisions them with controller lists. In order for the switches located two hops away from leader to receive their IP addresses, the generic preconfigured rules in the direct neighbor switches must first be extended with a new set of rules in Phase 2a.

5.3 Phase 2 - Enabling a Functional and Resilient Control Plane

In contrast to HSW, which computes the resilient control plane flows after disabling (R)STP, HHC tries to compute and deploy resilient flow rules whenever feasible. Namely, it installs the resilient flow rules as soon as there exist $k+1$ disjoint paths for a single switch-controller communication pair. If the current discovered topology does not allow for identifying all required paths, flow rules are provisioned for a single path only. Whenever there is a change in topology, the leader retries computing the remaining disjoint paths.

5.3.1 Phase 2a - *Initial OpenFlow Flow Rules.* In this sub-step, leader provides the direct neighbor switches with rules that allow for the next-hop switches to communicate with all controllers (ref. Table 4). These rules have a lower priority than the resilient rules computed in Phase 2b. Since (R)STP is unavailable, broadcast storms must be avoided. Thus, in addition to the base topology discovered by LLDP- and ARP-probing, HHC maintains a virtual spanning tree.

The tree topology is updated and enforced upon switches on every topology change using Table 4 rules. Packets used in topology and controller discovery are sent directly to CONTROLLER port as PACKET-INs. In OVS, packets forwarded to CONTROLLER port leverage the NORMAL data-path, which initially may seem problematic. However, due to not relying on Standalone operation, MAC-learning tables are empty and every packet is flooded instead. The flooded traffic cannot create broadcast storms as it can only reach two types of switches: i) those with TREE rules installed and, ii) those with generic preconfigured rules, which drop all traffic except their own. The discovery traffic is hence broadcasted only in the tree, since the PACKET-INs match the OpenFlow type.

Purpose	Packet Type	Matching	Action
(NEXT) Dynamic switch IP address configuration	DHCP	in_port=TREE port, udp, udp_src=67 in_port=TREE port, udp, udp_src=68	Send to other TREE ports
(NEXT) Remote switch configuration	SSH	in_port=TREE port, tcp, tcp_dst=22 in_port=TREE port, tcp, tcp_src=22	Send to other TREE ports
(NEXT) Controller-Switch OpenFlow interaction	OpenFlow	in_port=TREE port, tcp, tcp_src=6633 in_port=TREE port, tcp, tcp_dst=6633	Send to other TREE ports
(NEXT) Any ARP traffic	ARP	in_port=TREE port, arp	Send to other TREE ports
Topology Discovery	LLDP	eth_type=0x88cc	Send to CONTROLLER
Controller self-discovery	ARP	arp, arp_tpa=arbitrary IP	Send to CONTROLLER
NEXT: Network extension discovery	DHCP	in_port=INACTIVE port, udp, udp_src=68	Send to CONTROLLER

Table 4: HHC - Initial and Network Extension (NEXT) Flow Rules Installed on Switches in Phase 2a.

5.3.2 Phase 2b - *Enabling Control Plane Resilience.* To compute resilient paths, same as in Sec. 4.3.2, we deploy Dijkstra's algorithm. HHC does not assume visibility of entire topology to compute per-switch resilient paths. Instead, resilient paths are installed iteratively, whenever disjoint paths become available. This results in a quicker control plane resilience, as confirmed by experimental results in Sec. 8.

5.4 Phase 3 - Dynamic Network Extensions

To support dynamic network extensions, HHC adopts the same idea as HSW. Main difference to HSW is that HHC enforces the virtual tree topology together with remaining initial flows, i.e., the spanning tree is (re-)enforced iteratively during the bootstrapping procedure itself. Compared to rules installed in Phase 2, a single discovery rule per inactive switch port must be additionally installed, in order for new switches to successfully register with controllers (ref. last rule of Table 4). Whenever an element of the tree fails, HHC refreshes the rules with the alternative tree (ref. Sec. 4.5).

6 DESIGN & EVALUATION ASPECTS

We next discuss selected design & evaluation aspects of the two automated bootstrapping schemes.

6.1 Flow Table Occupancy

Both bootstrapping schemes enforce a non-negligible number of forwarding rules. The exact flow table occupancy (FTO) can be predetermined only for loop-free topologies. In non-loop-free topologies, the FTO varies depending on the outputs of the used tree computation and routing algorithm. However, the lower and the upper bound FTO can always be calculated from derived expressions. For brevity, we next provide only the upper FTO bounds for both schemes.

6.1.1 HSW. An HSW-bootstrapped switch has its FTO upper-bounded by F_{HSW} rules:

$$F_{HSW} \leq n*4 + (5 + 3*|C|) + i*7 + m*j*6 + k$$
$$n \in [0, \binom{|C|}{2}]; i \in [1, D_{Tree}]; m \in [0, |C|]; j \in [0, |\mathcal{S}|-1]; k \in \mathbb{N}$$

where $|C|$ denotes the number of deployed controllers, D_{Tree} is the maximum node degree of the computed spanning tree, and $|\mathcal{S}|$ is the number of switches.

The 4 fixed rule types of Table 2 are TCP and ARP rules that allow for controller synchronization. The 5 fixed rules are composed of: ARP, SSH, OpenFlow rules for forwarding incoming traffic from controllers to the LOCAL port; and 2 discovery rules (LLDP, ARP). 3 flow rules (ARP, SSH, OpenFlow) are embedded per controller so to forward local traffic toward the respective controller (ref. Table 2). Index i denotes the degree of a switch in the virtual spanning tree used for network extensions. The 7 fixed rules are the NEXT discovery rules. Index j denotes how many resilient paths that start/end in a particular controller traverse a switch. Index m denotes the number of controller replicas. The 6 fixed rules are the resilient flow rules (ARP, SSH, OpenFlow) used for traffic relaying, in directions to/from other switches. k denotes the no. of inactive ports, imposing a discovery rule per port.

6.1.2 HHC. HHC's FTO is upper-bounded by F_{HHC}:

$$F_{HHC} \leq 13 + n * 4 + (5 + 3 * |C|) + i * 7 + m * j * 6 + k$$

HHC's flow table has the same composition as HSW, except for the additional 13 preconfigured rules (ref. Table 3).

6.2 (R)STP Timer Parametrization

HSW assumes that in `Phase 2a` all available switches are discovered and are provided with necessary flow rules. Hence, in `Phase 2b` the leader controller proceeds to disable (R)STP on all switches, so to enable the blocked ports and populate the network topology view. Safe expiration timer T_{RSTP} after which to disable (R)STP can be estimated as:

$$T_{RSTP} \geq T^{R}_{DHCP} + T^{HS}_{DHCP} + T_{C_{list}} + T^{HS}_{OF} + T^{FO}_{RSTP}$$

where T^{R}_{DHCP} represents the transmission interval of DHCP Discover packets by the switches' DHCP clients; T^{HS}_{DHCP} is the delay imposed by a successful DHCP handshake; $T_{C_{list}}$ is the time required to deliver the controllers' IP address list; T^{HS}_{OF} comprises the OpenFlow session establishment time and time required to install and confirm the initial flows (ref. `Phase 2a`); and T^{FO}_{RSTP} is the time required for (R)STP to recover from potential network failures during bootstrapping. Worst-case time necessary to recover the spanning tree after switch / link failures with STP may reach up to 50s [9]. In corner cases, the recovery time for RSTP may increase up to 120s (ref. Count-to-Infinity problem [13]). In experiments conducted in [13], however, the worst-case RSTP recovery time in a 16-switch topology peaked at 50s. If a new switch sends a DHCP Discover message before the timer expiration, HSW preempts the timer. On a successful expiration, HSW disables (R)STP on the switches as per `Phase 2b`. Parameters used in evaluation were deduced empirically (ref. Table 5).

Parameter	T^{R}_{DHCP}	T^{HS}_{DHCP}	$T_{C_{list}}$	T^{HS}_{OF}	T^{FO}_{RSTP}
Value	1s	1s	1s	2s	50s [13]

Table 5: Parametrization of the (R)STP timer in HSW.

6.3 Network Topology Discovery

ODL's OpenFlowPlugin relies on LLDP flow rules to learn the network topology. The controllers periodically output OpenFlow `PACKET-OUT` messages with encapsulated LLDP data units (LLDPDUs) on each switch's port. Neighbor switches then receive and forward these packets back to their Master. Thus, controllers learn about the topology adjacency. By default, OpenFlowPlugin transmits the LLDPDUs each 5s. If no probes are received for 3 consecutive periods, the link originating in that port is considered unavailable and is removed from the data-store. To facilitate faster discovery of switches, we increase the rate of LLDPDU transmissions to 1s. Since the additional control plane load is generated only on unused ports, this optimization imposes no overhead.

6.4 Overhead of Controller Clustering

On a newly discovered switch, the *OpenFlowPlugin* module of an ODL replica attempts to take its ownership by initiating a role request to become its Master. Another controller replica may, however, be elected as the RAFT leader of the node inventory data-store used to serialize (i.e., reach consensus on and order) switches' state modifications. The election of the OpenFlow Master is thus independent of the inventory data-store ownership and the leader of the bootstrapping procedure. According to the OpenFlow specification, only the Master of a switch may directly modify its flow table. For this reason, multiple controllers may exchange data in order to apply changes to a switch's flow table. Fig. 5 illustrates the worst case relevant for our evaluation.

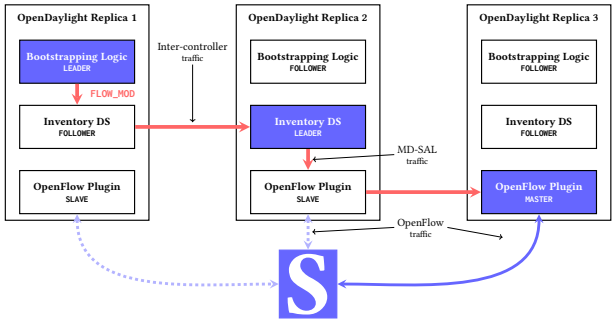


Figure 5: An exemplary data flow of a `FLOW_MOD` RPC with each entity's leader on a different controller.

6.5 Coping with broadcast storms in HHC

To solve the issues related to `Phase 2b` where particular flows may initially cause broadcast storms, we rely on rate limiting mechanisms provided by the data plane (e.g., OpenFlow's Metering [23] or Linux's Traffic Control). Namely, we police the following inter-controller flows (ref. Table 3): i) controller-initiated ARP traffic; ii) `TCP SYN`s destined for the inter-controller TCP port; ii) `TCP SYN ACK`s with inter-controller TCP port as source. The rate limit for policers may be configured to a very low value, e.g., we used 1.5$kbps$ for metering both ARP requests and replies (~ 12 ARP pps). Similarly, a low maximum rate can be chosen for `TCP SYN` and `TCP SYN ACK` packets. It suffices to match and rate-limit only `TCP SYN` and `TCP SYN ACK` traffic (as only these packets may generate broadcasts) and not the complete TCP flow.

7 EVALUATION OF THE SCHEMES

7.1 Evaluation Environment

We have implemented HSW and HHC as extensions of ODL's `Boron-SR3` release and have evaluated their performance against emulated industrial topologies [8, 17, 19, 46]. We focus on the impact of topology type on the bootstrapping efficiency and not on the impact of network size. This said, we did successfully validate our approaches in topologies comprising 50 switches in a single Layer-2 domain, which are larger than the average industrial topologies with requirements on strict QoS guarantees [19, 46]. Our network emulator generates the input topology by isolating OVS instances in

Docker containers and interconnecting them as per target topology. In the single controller scenario, the controller and the topology were hosted on PC equipped with a recent Intel Core i7 CPU. Multi-controller scenarios were executed on a dual-CPU Intel Xeon E5 server.

7.2 Evaluated Metrics

The schemes are compared along with the following KPIs:

Global bootstrapping convergence time (GBCT): Defined as the difference between i) the time instant at which all switches are provisioned with resilient flow rules; and ii) the instant when the first switch was observed by any controller.

Time required to extend the network (TEXT): For each added switch, we measure and average the difference between the instant i) when a switch is provided with the control flow rules; ii) when the switch was first observed by a controller.

Flow table occupancy (FTO): No. of active flow entries in a flow table after successful bootstrapping procedure.

Industrial topologies, in particular those often found in process industry and factory automation, were selected due to their requirement on redundant communication and reliable and dynamic network adaptation [2]. Fig. 6 depicts the evaluated topologies, with corresponding suffixes denoting the topology size. *line-N* and *star-N* topologies were evaluated with a single controller only, while the remaining topologies deploy up to 3 controllers. Note that while *line-N*, *star-N* and *1-ring-N* topologies do not satisfy the conditions for path disjointness, we also evaluate these topologies to gain additional insights. In single-controller scenarios, the replica was always placed adjacent to S_1. For scenarios involving 3 controllers, controllers were placed according to Table 6. Indeed, in general, the controller placement influences the observed measured KPIs in both single- and multi-controller scenarios. For brevity, we however limit our discussion to the impact of the RAFT leader placement.

Topology	Controller placement	Topology	Controller placement
ring-4	S_1, S_2, S_3	1-ring-5	S_1, S_4, S_5
ring-8	S_1, S_4, S_6	1-ring-7	S_1, S_3, S_7
ring-16	S_1, S_6, S_{11}	1-ring-10	S_1, S_3, S_{10}
grid-4	S_1, S_2, S_3	2-ring-6	S_1, S_3, S_6
grid-9	S_1, S_5, S_9	2-ring-11	S_1, S_7, S_{11}

Table 6: Controllers' placement with 3 controllers.

8 EVALUATION RESULTS

8.1 Bootstrapping Convergence Time

8.1.1 Single-Controller Setup. Global Bootstrapping Convergence Times (GBCTs) are depicted in Fig. 7. GBCT of HSW is not impacted by the type of the underlying topology. Instead, the time to embed resilient paths increases with the overall topology size. This is due to the fact that HSW's Phases 1 and 2 do not execute in a concurrent manner. HHC's GBCT, on the other hand, is dependent on the design of the underlying topology. In particular, its convergence time scales with the number of additional hops the control traffic must traverse to reach a certain switch in the network.

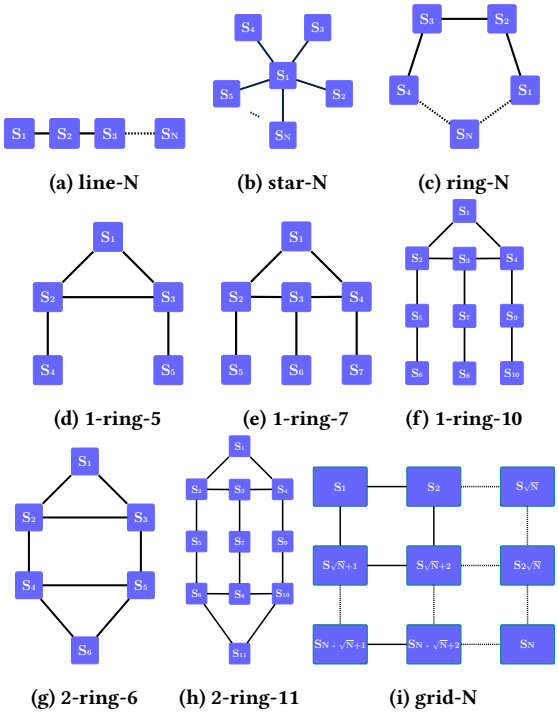


Figure 6: Considered industrial topologies [8, 17].

HHC bootstraps the switches with same hop distance from the controller concurrently. For example, in grid topologies, the max. no. of hops between a switch and controller does not increase with the same rate as the number of switches. Thus, going from *grid-4* to *grid-9*, the absolute number of switches increases by 5, but the hops required for HHC's controller to reach most distant switches increases only by 2, leading to an increased performance gap over HSW. Additionally, in contrast to HSW, HHC does not suffer from an artificially introduced lower bound (due to its non-reliance on (R)STP).

8.1.2 Multi-Controller Setup. HHC relies heavily on the controllers' data store during its operation - each read / write operation requires proxying the request through the current RAFT leader. Additionally, controllers frequently block and synchronize their data-stores, thus adding additional processing latency in sequential code execution. In contrast to HHC, HSW does not rely on the data-store as much, providing for better scaling with increasing control plane size, hence the drop in performance gap for the 3-controller scenario.

With HSW, the RAFT leader's placement does not impact the resulting performance, since the switches are bootstrapped in the FCFS manner. Thus, HSW's values in Fig. 7b are generally less spread than for HHC. For example, with HHC *grid-9* may be bootstrapped as quick as *grid-4*. If the leader for *grid-9* is elected adjacent to S_5 (ref. Fig. 6), the leader will require 2 hops to reach any switch, i.e., the same number of hops as required by *grid-4* for a leader placed on any of the 4 switches. Note, however, that such comparison does not hold in ring topologies, since the leader placement there does not influence the max. hop distance to leader.

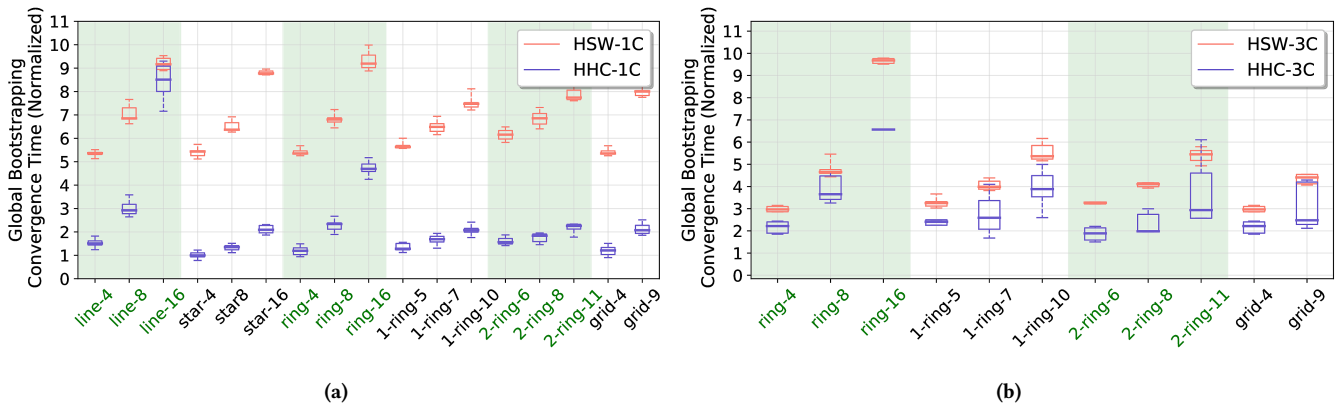


Figure 7: Observed GBCT for single- and 3-controller scenarios. Measured time is normalized with respect to the minimum observed GBCT means (13.5*s* and 33.9*s* for and 1 and 3 controllers, respectively). HHC outperforms HSW for all evaluated topologies, mostly due to HSW being lower bounded by the (R)STP timer (ref. Sec. 6.2).

8.1.3 Discussion. In general, HHC outperforms HSW for all evaluated topologies. This is due to the lower bound on the GBCT in HSW as imposed by the (R)STP timer (set to 55*s*, ref. Sec. 6.2). In HSW, GBCT increases linearly with the topology size, contrary to HHC, where GBCT exhibits a non-linear relation with the maximal hop distance between the leader controller and the switch. The larger the distance, the larger the increase in necessary bootstrapping time. Thus, the performance gap between HSW and HHC depends on the placement of the RAFT leader and the overall topology size. In 3-controller scenarios, HHC's extensive reliance on the distributed data-store reduces the performance gap.

8.2 Network Extension Time

8.2.1 Single-Controller Setup. In single-controller scenarios, HHC requires a nearly constant Time to Extend (TEXT) a topology, i.e., deploy a new switch, independent of the existing topology and no. of new switches (ref. Fig. 8). The slight increase for larger topology sizes relates to the accumulated CPU load from having to consider additional switches, e.g., additional LLDP packets to process when refreshing the topology and spanning tree computation overhead. For HSW, just like with GBCT, TEXT grows linearly with the topology size. This is due to the waiting period related to disabling the (R)STP timer and contention in sequential rule installation. An optimistic case is the single-switch extension where no contention impacts rule installation order (not depicted).

8.2.2 Multi-Controller Setup. A newly added switch in a multi-controller setup must on average wait longer on its control flow rules. The inter-controller synchronization results in a higher TEXT degradation for HHC than for HSW. Additionally, compared to the single-controller case, where TEXT remains mostly constant, in scenarios with 3 controllers, HHC's TEXT grows linearly with topology size, due to a larger resulting controller-controller separation.

8.2.3 Discussion. HHC outperforms HSW both in single- and 3-controller scenarios. However, due to the distributed synchronization overhead, and the fact that control flow rules can be provided only when a controller is discovered, the performance gap between HSW and HHC is reduced.

8.3 Flow Table Occupancy

Fig. 9 portrays the substantial growth in the Flow Table Occupancy (FTO) when deploying 3 controllers instead of a single one. This is is due to switches being provided with resilient flow rules for connections to all controllers. Additionally, some switches contain rules used to forward the inter-controller traffic. However, the change in FTO is influenced not only by number of controllers, but also by the topology size, degree of connectivity, *and* the controller placement. The ratios of FTOs are summarized in Table 7.

In HSW, the placement of the leader does not influence the FTO. This is due to resilient and tree rules being installed only *after* a successful discovery of the entire network. Thus, the FTO depends only on the output of the routing and tree computation algorithm. On the contrary, in HHC the placement of the RAFT leader controller influences the output of the iteratively-built spanning tree, producing fluctuating FTOs for repeated executions. Notably, in *grid-N*, *x-ring-N* topologies, the leader controller placement influences the FTO fluctuations, while in *ring-N* topologies, different leader placements have no effect on FTO (visual results omitted due to space considerations). In all evaluated scenarios, HSW results in lower minimal, average, and maximal FTOs. The average difference in FTO between HSW and HHC equals approximately the number of preconfigured rules in HHC scheme. Indeed, both bootstrapping schemes enforce a non-negligible number of forwarding rules. Investigation of methods for shrinking the number of active flow rules leveraged by the schemes, i.e., by means of flow table compression [3, 31] should be considered in future studies.

Topology	HSW	HHC
ring-{4, 8, 16}	{2.44, 2.62, 2.77}	{2.2, 2.43, 2.43}
grid-{4, 9}	{2.44, 2.22}	{2.2, 2.07}
1-ring-{5, 7, 10}	{2.19, 2.31, 2.36}	{2.0, 2.12, 2.16}
2-ring-{6, 8, 11}	{2.29, 2.27, 2.44}	{2.08, 2.07, 2.34}

Table 7: Ratios of observed average per-switch FTOs. Values are normalized respective to the FTO in 1-controller case for the same scheme and topology.

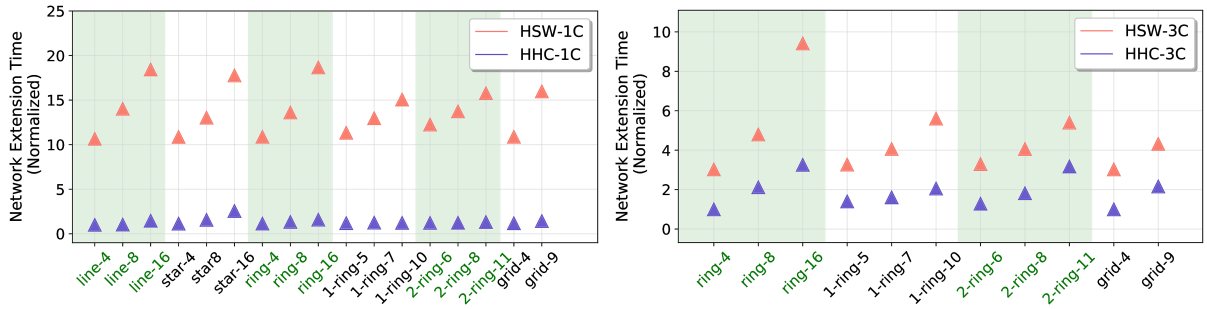


Figure 8: TEXT values of the two schemes for configurations deploying 1 to 3 controllers. Y-axis depicts the (per-topology) normalized TEXT, relative to the lowest obtained mean TEXT, i.e., 6.5*s* and 33.5*s* for 1- and 3-controllers.

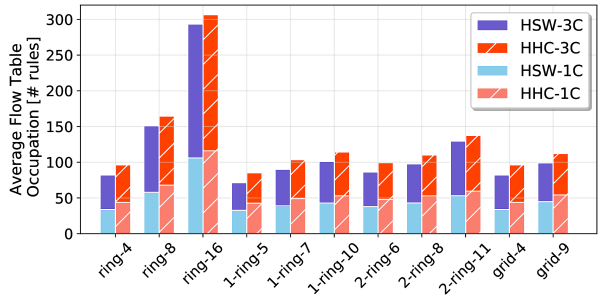


Figure 9: Bar charts comparing the average FTO for HSW and HHC bootstrapping schemes.

9 RELATED WORK

Sharma et al. [39] were first to propose an automatic bootstrapping scheme and evaluate its performance for various in-band controlled (IBC) topologies. [40, 41] highlight the advantages of a proactive protection scheme (using fast-failover groups [44]), allowing the controller to proactively compute duplicate paths for control plane flows. On a successful failure discovery, the detecting switch automatically re-routes the incoming traffic over the assigned backup port, without needing to involve the controller in loop. In all three works, Sharma et al. assume proprietary modifications to the DHCP client hosted in switches with the goal of provisioning controller list. No multi-controller support was considered.

Schiff et al. [36] present a design of a self-organizing multi-controller control plane that relies exclusively on OpenFlow. Contrary to HSW and HHC schemes, the authors do not consider the necessity of controller state synchronization prior to switch reconfigurations (ref. Sec. 2). In [37], the authors extend their approach to include a timeout-based fault-tolerance approach where rules corresponding to failed paths *eventually* time out, thus preventing permanent switch cut-offs. Instead, we propose constant duplication of control flows incurring zero-packet-loss in case of failures. Follow-up works [6, 7] propose a timeout-free approach to ensure resilience against data plane failures, based on assumption of a controller-initiated switch discovery and OpenFlow *equal role* [23] controller association with switches. Similar to above works, we compute and iteratively expand the spanning tree so to enable loopless forwarding of control traffic, both after disabling (R)STP in HSW, and in Phase 2 and 3 of HHC.

[38] proposes atomic transactions for coordinated concurrent switch configurations by multiple controllers. The approach is orthogonal to our work but we additionally assume the requirement for distributed consensus [20, 34], as imposed by ODL [24] and ONOS [5] implementations.

Heise et al. [18] propose the usage of network calculus, i.e., rate- and burst-policed control traffic for providing upper bound guarantees for bootstrapping convergence time. They leverage fast-failover groups to implement the restoration of control flows in face of failures. Their bootstrapping concept assumes (R)STP in switches and no multi-controller support. Bentstuen et al. [4] propose an approach that relies on intent-based control flow definitions targeting ONOS [5], so to simplify the management of control flows in a *single-controller* environment. The authors also stumble upon a number of practical issues related to IBC bootstrapping and ultimately fall back to modification of the OVS's source code. To support the existing OpenFlow implementations, workarounds for these limitations are discussed in this work.

FASIC [43] minimizes congestions in single-controller IBC control plane by means of a centralized control port load monitoring. Control port is switched to a more fitting port on exceeded threshold or in case of link failures. The authors deduce that their fail-over approach results in non-negligible packet loss, related to OpenFlow's back-off interval in the case of repeated unsuccessful connection attempts. We consider this aspect in the design of the (R)STP timer. In [16], authors propose a method for re-routing IBC control flows based on observed controller load and IBC channel congestion. To this end, the authors leverage control flow *shifting* and *splitting*. Both methods are orthogonal to HSW and HHC.

10 CONCLUSIONS

This work describes the design of the first two bootstrapping schemes that autonomously bootstrap a multi-controller SDN with a resilient control plane and with automatic IP and controller list provisioning to the switches. Besides evidencing the practicability of these two approaches and quantifying the trade-offs they reveal (implementation complexity, legacy protocols needed, convergence time, flow table occupancy, network extension time), our work opens the door towards two important directions.

First, it finally enables SDN in environments where out-of-band connections are not possible, e.g., industrial networks, and hence the deployment of recent SDN-based advances for such environments [15, 19, 22, 45]. In fact, the presented schemes were demonstrated successfully in an operational industrial network with fail-safe requirements [35, 45].

Second, having the control and data plane share the same infrastructure motivates investigation of proper isolation of both traffic types and ensuring non-starvation of control traffic. This is especially relevant in industrial environments where data plane traffic often has stringent QoS requirements.

Outlook: Our evaluation currently targets industrial topologies and focuses on evaluation of impact of topology type on the achievable performance, and less so on that of its size. We leave the investigation of the applicability of our approach to large-scale topologies for future work. This said, HHC has been successfully validated with 50-switch topologies in a single broadcast domain. Finally, supporting IPv6 networks should be a straightforward extension using mechanisms already developed for IPv4 case, but is currently left as future work.

ACKNOWLEDGMENT

We thank the anonymous reviewers and our shepherd Junaid Khalid for their feedback and useful inputs on our work. We thank Sean Rohringer and Reinhard Frank for their help in the early stages of our work. This work has received funding from the European Commission's Horizon 2020 research and innovation programme under grant agreement number 780315 SEMIoTICS. This work reflects only the authors' view and the funding agency is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] 2018. IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), 1–1993.

[2] Astrit Ademaj, Thomas Enzinger, and Marius Stanica. 2018. TSN System Requirements v0.2. IEC/IEEE. http://www.ieee802.org/1/files/public/docs2018/60802-stanica-tsn-system-requirements-0518-v02.pdf

[3] S. Banerjee and K. Kannan. 2014. Tag-In-Tag: Efficient flow table management in SDN switches. In *10th International Conference on Network and Service Management (CNSM) and Workshop*. 109–117.

[4] Ole Ingar Bentstuen and Joakim Flathagen. 2018. On Bootstrapping In-Band Control Channels in Software Defined Networks. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1–6.

[5] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, et al. 2014. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 1–6.

[6] Marco Canini, Iosif Salem, Liron Schiff, Elad M Schiller, and Stefan Schmid. 2017. A self-organizing distributed and in-band SDN control plane. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2656–2657.

[7] Marco Canini, Iosif Salem, Liron Schiff, Elad Michael Schiller, and Stefan Schmid. 2018. Renaissance: A self-stabilizing distributed SDN control plane. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 233–243.

[8] Dick Caro. 2009. *Automation Network Selection: A Reference Manual, 2Nd Edition* (2nd ed.). International Society of Automation, USA.

[9] Cisco Systems, Inc. 2017. Spanning Tree Protocol Problems and Related Design Considerations. https://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/10556-16.html. (2017).

[10] Tom St Denis and Simon Johnson. 2007. Chapter 6 - Message - Authentication Code Algorithms. In *Cryptography for Developers*, Tom St Denis and Simon Johnson (Eds.). Syngress, Burlington, 251 – 296.

[11] Josef Dorr. 2018. IEC/IEEE P60802 JWG TSN Industrial Profile: Use Cases Status Update 2018-05-14. IEC/IEEE. https://1.ieee802.org/tsn/iec-ieee-60802/

[12] Michael Eischer and Tobias Distler. 2017. Scalable Byzantine Fault Tolerance on Heterogeneous Servers. In *Dependable Computing Conference (EDCC), 2017 13th European*. IEEE.

[13] K. Elmeleegy, A. L. Cox, and T. S. Eugene Ng. 2009. Understanding and Mitigating the Effects of Count to Infinity in Ethernet Networks. *IEEE/ACM Transactions on Networking* 17, 1 (2009), 186–199.

[14] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News* 33, 2 (2002).

[15] Jochen W. Guck, Amaury Van Bemten, and Wolfgang Kellerer. 2017. DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments. *IEEE Transactions on Network and Service Management* 14, 4 (2017), 1003–1017.

[16] B. Görkemli, S. Tatlıcıoğlu, A. M. Tekalp, S. Civanlar, and E. Lokman. 2018. Dynamic Control Plane for SDN at Scale. *IEEE Journal on Selected Areas in Communications* 36, 12 (2018), 2688–2701.

[17] Peter Heise. 2018. *Real-time guarantees, dependability and self-configuration in future avionic networks*. Ph.D. Dissertation.

[18] Peter Heise, Fabien Geyer, and Roman Obermaisser. 2017. Self-configuring deterministic network with in-band configuration channel. In *Software Defined Systems (SDS), 2017 Fourth International Conference on*. IEEE, 162–167.

[19] Dominik Henneke, Lukasz Wisniewski, and Jürgen Jasperneite. 2016. Analysis of realizing a future industrial network by means of Software-Defined Networking (SDN). In *2016 IEEE World Conference on Factory Communication Systems (WFCS)*. IEEE, 1–4.

[20] Heidi Howard, Malte Schwarzkopf, Anil Madhavapeddy, and Jon Crowcroft. 2015. Raft refloated: do we have consensus? *ACM SIGOPS Operating Systems Review* 49, 1 (2015), 12–21.

[21] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. 2012. Logically centralized?: State distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*. ACM.

[22] Dong Li, Ming-Tuo Zhou, Peng Zeng, Ming Yang, Yan Zhang, and Haibin Yu. 2016. Green and reliable software-defined industrial networks. *IEEE Communications Magazine* 54, 10 (2016), 30–37.

[23] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.

[24] J. Medved, R. Varga, A. Tkacik, and K. Gray. 2014. OpenDaylight: Towards a Model-Driven SDN Controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. 1–6.

[25] Abubakar Siddique Muqaddas, Andrea Bianco, Paolo Giaccone, and Guido Maier. 2016. Inter-controller traffic in ONOS clusters for SDN networks. In *2016 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[26] Abubakar Siddique Muqaddas, Paolo Giaccone, Andrea Bianco, and Guido Maier. 2017. Inter-controller traffic to support consistency in ONOS clusters. *IEEE Transactions on Network and Service Management* 14, 4 (2017), 1018–1031.

[27] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*. 305–319.

[28] P60802 Project: TSN Profile for Industrial Automation (TSN-IA). 2018. Use Cases IEC/IEEE 60802 v1.3. IEC/IEEE. https://1.ieee802.org/tsn/iec-ieee-60802/

[29] Aurojit Panda, Wenting Zheng, Xiaohe Hu, Arvind Krishnamurthy, and Scott Shenker. 2017. SCL: Simplifying Distributed SDN Control Planes. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 329–345.

[30] Shifei Qian, Feng Luo, and Jinpeng Xu. 2017. An Analysis of Frame Replication and Elimination for Time-Sensitive Networking. In *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*. ACM, 166–170.

[31] M. Rifai, N. Huin, C. Caillouet, F. Giroire, D. Lopez-Pacheco, J. Moulierac, and G. Urvoy-Keller. 2015. Too Many SDN Rules? Compress Them with MINNIE. In *2015 IEEE Global Communications Conference (GLOBECOM)*. 1–7.

[32] Ermin Sakic, Nemanja Đerić, and Wolfgang Kellerer. 2018. MORPH: An adaptive framework for efficient and Byzantine fault-tolerant SDN control plane. *IEEE Journal on Selected Areas in Communications* 36, 10 (2018), 2158–2174.

[33] Ermin Sakic and Wolfgang Kellerer. 2018. Impact of Adaptive Consistency on Distributed SDN Applications: An Empirical Study. *IEEE Journal on Selected Areas in Communications* 36, 12 (2018), 2702–2715.

[34] Ermin Sakic and Wolfgang Kellerer. 2018. Response time and availability study of RAFT consensus in distributed SDN control plane. *IEEE Transactions on Network and Service Management* 15, 1 (2018), 304–318.

[35] Ermin Sakic, Vivek Kulkarni, Vasileios Theodorou, Anton Matsiuk, Simon Kuenzer, Nikolaos E Petroulakis, and Konstantinos Fysarakis. 2018. VirtuWind: An SDN and NFV-based architecture for softwarized industrial networks. In *International Conference on Measurement, Modelling and Evaluation of Computing Systems*. Springer, 251–261.

[36] Liron Schiff, Stefan Schmid, and Marco Canini. 2015. Medieval: Towards A Self-Stabilizing, Plug & Play, In-Band SDN Control Network. In *ACM Sigcomm*

Symposium on SDN Research (SOSR).

[37] Liron Schiff, Stefan Schmid, and Marco Canini. 2016. Ground control to major faults: Towards a fault tolerant and adaptive SDN control network. In *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*. IEEE, 90–96.

[38] Liron Schiff, Stefan Schmid, and Petr Kuznetsov. 2016. In-band synchronization for distributed SDN control planes. *ACM SIGCOMM Computer Communication Review* 46, 1 (2016).

[39] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. 2013. Automatic bootstrapping of OpenFlow networks. In *Local & Metropolitan Area Networks (LANMAN), 2013 19th IEEE Workshop on*. IEEE, 1–6.

[40] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. 2013. A demonstration of automatic bootstrapping of resilient OpenFlow networks. In *13th IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 1066–1067.

[41] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. 2013. Fast failure recovery for in-band OpenFlow networks. In *Design of Reliable Communication Networks (DRCN) 2013 9th International Conference on the*. IEEE, 52–59.

[42] Specification, OpenFlow Switch. 2015. Version 1.5.1, Standard, Open Networking Foundation. (2015).

[43] Yu-Lun Su, I-Chih Wang, Yao-Tsung Hsu, and Charles H-P Wen. 2017. FASIC: A Fast-Recovery, Adaptively Spanning In-Band Control Plane in Software-Defined Network. In *GLOBECOM 2017 IEEE Global Communications Conference*. IEEE, 1–6.

[44] Niels LM Van Adrichem, Benjamin J Van Asten, Fernando A Kuipers, et al. 2014. Fast Recovery in Software-Defined Networks. *EWSDN* 14 (2014), 61–66.

[45] Petra Vizarreta, Amaury Van Bemten, Ermin Sakic, Khawar Abbasi, Nikolaos E Petroulakis, Wolfgang Kellerer, and Carmen Mas Machuca. 2019. Incentives for a Softwarization of Wind Park Communication Networks. *IEEE Communications Magazine* 57, 5 (2019), 138–144.

[46] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. 2017. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE industrial electronics magazine* 11, 1 (2017), 17–27.

[47] Yang Zhang, Eman Ramadan, Hesham Mekky, and Zhi-Li Zhang. 2017. When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network. In *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 1–7.