Received February 7, 2020, accepted February 25, 2020, date of publication March 23, 2020, date of current version April 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2982859

P4-MACsec: Dynamic Topology Monitoring and Data Layer Protection With MACsec in P4-Based SDN

FREDERIK HAUSER, (Student Member, IEEE), MARK SCHMIDT, (Member, IEEE), MARCO HÄBERLE, (Student Member, IEEE), AND MICHAEL MENTH, (Senior Member, IEEE)
Chair of Communication Networks, University of Tuebingen, 72076 Tübingen, Germany

Corresponding author: Frederik Hauser (frederik.hauser@uni-tuebingen.de)

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2 and the bwNET100G+ project which is funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). The authors alone are responsible for the content of this paper.

ABSTRACT We propose P4-MACsec to protect network links between P4-based SDN switches through automated deployment of MACsec, a widespread IEEE standard for securing Layer 2 infrastructures. MACsec is supported by switches and routers from many manufacturers. On these devices, it has only little performance limitations compared to VPN technologies such as IPsec. P4-MACsec suggests a data plane implementation of MACsec including AES-GCM encryption and decryption directly on P4 targets. P4-MACsec features a two-tier control plane structure where local controllers running on the P4 targets interact with a central controller. We propose a novel secure link discovery mechanism that leverages protected LLDP frames and a two-tier control plane structure for secure and efficient management of a global link map. Automated deployment of MACsec creates secure channels, generates keying material, and configures the P4 targets for each detected link between two P4 targets. It detects link changes and performs rekeying to provide a secure, configuration-free operation of MACsec. In this paper, we review the technological background of P4-MACsec and explain its architecture. To demonstrate the feasibility of P4-MACsec, we implement it on the BMv2 P4 software target, validate the prototype through experiments, and evaluate its performance through experiments considering TCP goodput and round-trip time. We publish the prototype and experiment setup under the Apache v2 license on GitHub [7].

INDEX TERMS MACsec, P4, software-defined networking, VPN.

I. INTRODUCTION

MACsec [41] is a widespread IEEE standard that protects the Layer 2 with cryptographic integrity checks or symmetric encryption. MACsec prevents man-in-the-middle attackers from inspecting, inserting, or even modifying network packets that are transmitted between two network peers. In contrast to VPN technologies such as IPsec, MACsec processing is implemented on forwarding chips of many devices without notable overhead in line rate performance [30]. Packets are protected in a point-to-point manner between MACsec peers so that control plane functions targeting higher layers, e.g., access-control lists (ACLs), can be still applied. Although mechanisms for distributed key exchange exist, MACsec deployment still requires time-consuming and complex initial setup procedures on all devices. It requires knowledge about the network topology, large efforts in switch configuration, and typically maintenance of a key server. Currently, automated deployment using a network management system with legacy switches is not feasible. Legacy network switches only support the Link Layer Discovery Protocol (LLDP) [45] that lacks timely detection of topology changes. In addition, it is vulnerable to several attacks that may result in an incorrect view of the topology. Moreover, current legacy network switches do not support an automated configuration of MACsec through a southbound protocol. Although a MIB for manipulating MACsec configuration with SNMP exists [10], only basic MACsec parameters can be modified. Additional per-switch configuration and a key exchange server are still required.

Software-Defined Networking (SDN) splits the strong binding between data and control plane. OpenFlow (OF) [48]

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

is the most widespread standard architecture and southbound protocol for SDN. It consists of SDN switches with a fixed-function data plane that are steered by a central SDN controller. To resolve inflexibilities of fixed-function data planes, P4 [37] emerged as novel domain-specific language that introduces programmability to the data plane of P4-capable packet forwarding devices such as ASICs, CPU-based targets, or field programmable gate arrays (FPGAs). Data plane behavior can be described in P4 programs that run on P4 targets so that network operators can continuously program the behavior of deployed packet processing devices. The P4Runtime [23] extends P4 targets by an API to an SDN controller. It serves roughly the same purpose as the OF southbound protocol.

In this paper, we consider MACsec to dynamically protect links between switches in SDN. We propose to use an SDN controller to continuously monitor the network topology and set up MACsec on all detected links between switches. As OF does not provide support for integrating the required MACsec functions on the data plane, we propose a concept based on P4 and call it P4-MACsec. We implement a P4 data plane with packet switching based on MAC addresses and MACsec, i.e., MACsec encryption, decryption, and integrity checks of packets. In P4-MACsec, P4 targets implement typical switching functionality. Therefore, we call them P4 switches instead of P4 targets in the following. P4 switches are managed by a two-tier control plane where each P4 switch runs a local controller that connects to a central controller. Functions of the control plane may be solely part of the local controller or part of both tiers. The control plane implements MAC address learning for packet switching, a novel mechanism for secure link discovery with encrypted LLDP packets, and automated deployment of MACsec. To demonstrate the feasibility of P4-MACsec, we provide a prototype based on the Behavioral Model version 2 (BMv2) P4 software target [6]. We perform a functional validation of P4-MACsec in a Mininet testbed through experiments and investigate on TCP goodput and round-trip time (RTT) by conducting a performance evaluation. We publish the source code of the prototype and testbed setup instructions under the Apache v2 license on GitHub [7]. In addition, we report on implementation experiences for the NetFPGA SUME [14] platform.

The rest of the paper is structured as follows. In Section II, we review technical background and related work for MACsec. Section III discusses technical background and related work on link discovery in SDN. In Section IV, we give an overview on the P4 programming model and language. Section V describes the architecture of P4-MACsec. In Section VI, we describe the prototypical implementation of P4-MACsec with Mininet that is validated in Section VII. In Section VIII, we present a performance evaluation of the Mininet prototype. In Section IX, we report on experiences in implementing P4-MACsec on the NetFPGA SUME platform. Section X concludes this work. The appendices include a list of acronyms that are used in the paper.

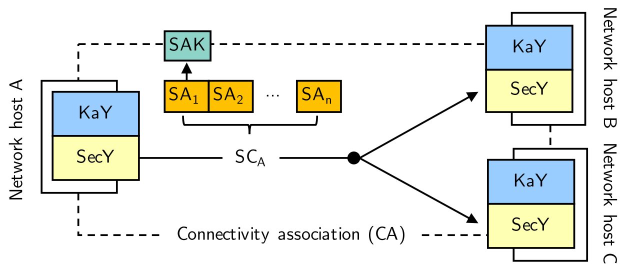

FIGURE 1. Secure communication between three stations in MACsec that are part of a CA. The unidirectional SC between the SecYs holds multiple SAs each with an SAK for encryption and decryption.

II. MACsec: FOUNDATIONS AND RELATED WORK

We give an overview of MACsec and explain how it protects the Ethernet layer. We describe mechanisms for configuration and key management and review related work on the application of MACsec in SDN.

A. OVERVIEW OF MACsec

IEEE 802.1AE [41] introduces the Media Access Control Security (MACsec) protocol. It provides point-to-point security between MACsec peers that are connected to the same LAN. Examples are links between two switches or routers, links between switches or routers and hosts, and links between hosts. MACsec ensures the integrity, confidentiality, and authenticity of Ethernet (IEEE 802) frames by applying symmetric encryption and cryptographic hash functions. In addition, it provides replay protection and a key exchange protocol to ensure perfect forward secrecy so that session keys are not affected by a compromised private key.

Figure 1 visualizes the principle and core components of MACsec. The network hosts A, B, and C are part of a LAN. Each network host has a MAC security entity (SecY) and a MAC security key agreement entity (KaY). The SecY provides secure MAC services over an insecure MAC service, i.e., it performs packet encryption, decryption, and authentication. The KaY discovers other KaYs in the LAN that participate in the same connectivity association (CA). It ensures that all network hosts are mutually authenticated and authorized. Afterwards, it creates and maintains secure channels (SCs) between the MACsec peers that are used by the SecYs to transmit and receive network packets. SCs are sender-specific, unidirectional, point-to-multi-point channels. Each SC holds multiple secure associations (SAs) that have a secure association key (SAK) used for encrypting, decrypting, and authenticating packets.

MACsec leverages cipher suites for packet encryption, decryption, and authentication. The standard defines the Advanced Encryption Standard in Galois/Counter mode (AES-GCM) with a block length of 128 bit (AES-GCM-128) as required cipher suite. If only packet authentication but no encryption is configured, MACsec applies the Galois Message Authentication Code (GMAC). Further specifications [43], [44] add GCM-AES-256, GCM-AES-XPN-128, and GCM-AES-XPN-256. Other cipher suites that meet requirements from the standard may also be applied.

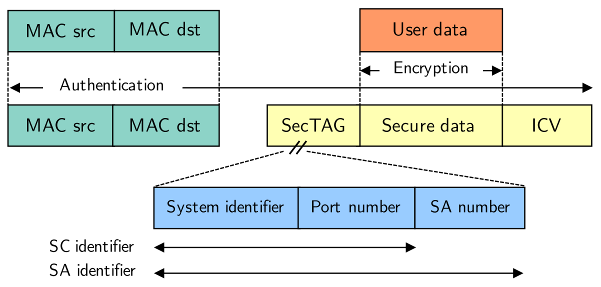

FIGURE 2. **Packet structure of MACsec applied to an Ethernet packet. The MAC source and destination addresses of the MACsec packet are adopted from the original packet. The user data is transformed into a secure data block that is followed by the ICV calculated over the whole packet (1). The SecTAG includes among other things parameters to identify the SC and SA.**

Figure 2 depicts the packet structure of MACsec. The Ethernet source and destination addresses of the MACsec packet are adopted from the original Ethernet packet. The secure data field either contains the encrypted user data of the original Ethernet packet or the user data in plain text if only MACsec packet authentication is configured. The integrity check value (ICV) field holds the result of a cryptographic hash function that is applied on the whole Ethernet packet including all header fields. The secure data and ICV are calculated by the chosen cipher suite. The security tag (SecTAG) contains MACsec information, e.g., the SC or SA identifier to indicate the corresponding SAK for packet encryption, decryption, and authentication. SecYs store multiple SAs with SAKs for each SC. When SAKs should be changed in rekeying, one SecY selects and uses a new SA for packet encryption and authentication. The SA field in the SecTAG reflects this transition so that the receiving SecY can switch to the new SA for decryption and authentication.

MACsec is supported by most access, distribution, and core switches from major manufacturers. Proprietary implementations such as the MACsec Toolkit [26] from Rambus provide control and data plane implementations that can be integrated in switches, routers, or network hosts. In addition, MACsec is part of the Linux kernel since version 4.6 [27] so that switch-to-host or host-to-host links can be protected with MACsec as well.

B. MACsec KEY EXCHANGE PROTOCOL

The 802.1AE standard does not define processes for key management or establishment of CAs and SAs between KaYs on MACsec peers. Therefore, network administrators are required to configure the CA affiliation and SAs with SAKs on every MACsec peer. IEEE 802.1X-2010 [42] introduces the MACsec Key Agreement (MKA) for automated peer discovery and exchange of SA data. With MKA, the initial CA affiliation and SA with SAK is derived from a connectivity association key (CAK). CAKs are either defined as pre-shared secret, derived from a master session key of an EAP process, or distributed by a MKA key server. Switches are required to implement additional functionalities to either exchange keying information via EAP with an AAA server or with an MKA key server. Independent of key exchange mechanisms, MACsec still needs to be initially set up on all peer devices via manual configuration.

C. COMPARISON TO VPNs

VPN technologies such as IPsec, OpenVPN, or WireGuard operate on Layer 3 or above. MACsec operates on Layer 2 and therefore provides link security for any higher-layer protocol. It applies point-to-point protection while VPNs aim at end-to-end protection. On every switch, router, or host in the network, MACsec packets are decrypted at the ingress port so that control plane functions targeting Layer 2 to 7 can be still applied. Access control lists that provide filtering based on IP addresses are an example. Then, packets are encrypted again at the egress port. MACsec is configured per Ethernet link so that administrators do not need to define additional policies for specific traffic to be encrypted. On routers and switches, MACsec is implemented on the packet forwarding chips, i.e., packet encryption and decryption is performed in line rate. In contrast, VPN technologies mostly encrypt and decrypt packets on ASICs that have limited bandwidth capacity. According to [30], IPsec traffic typically cannot exceed 40 Gb/s of bidirectional traffic while MACsec encryption and decryption scales with line rate.

D. APPLICATION OF MACsec IN SDN

Choi *et al.* [38] adopt MACsec to secure communication in vehicular networks between Linux-based electronic control units (ECUs). An SDN controller is responsible for automated setup of MACsec between ECUs to provide an end-to-end protection for network traffic. However, MACsec deployment is limited to the ECUs, i.e., MACsec deployment on SDN switches that connect the ECUs is not considered. Szyrkowiec *et al.* [55] develop an intent-based multi-layer orchestrator as application that interfaces an SDN controller. It automatically deploys protection technologies such as IPsec or MACsec on legacy switches through different southbound protocols, e.g., OpenFlow, NETCONF, or RESTCONF. However, MACsec deployment on SDN switches is not considered. Bentstuen and Flathagen [36] propose to implement MACsec for SDN but do not formulate any concrete approach. Vajaranta *et al.* [56] discuss implementation experiences and design challenges for WAN overlays using SDN and propose MACsec as viable option to implement link layer encryption. However, the presented implementation is limited to OpenVPN. Automated configuration of MACsec on SDN switches is proposed, but not part of the presented implementation. Mohamed *et al.* [49] describe a mechanism for MACsec key distribution of particular MACsec flows to switches. MACsec flows are end-to-end SCs that break up the point-to-point concept of the original standard. They are realized by configuring MACsec keys only on both end peers, but not on the peers in between. As prerequisite, all MACsec peers are expected to forward MACsec packets if no key for the received packet is found. OpenCORD [17] describes another example of MACsec deployment in SDN.

The SecY remains on the SDN switch while the KaY is transposed into a northbound application running on the ONOS SDN controller. SecYs on SDN switches are then configured via a NETCONF interface by the SDN controller. However, the authors state that MACsec support on SDN switches is still under developed so that only key agreement and configuration were implemented and tested in a Mininet simulation. Layer 2 packet encryption has not been examined.

III. LINK DISCOVERY IN SDN: FOUNDATIONS AND RELATED WORK

Automated deployment of MACsec by a SDN controller requires a topology view that is maintained through topology monitoring with link discovery between SDN switches. We give an overview of link discovery in SDN and describe the OpenFlow Discovery Protocol (OFDP). We review related work on variants of the OFDP that are optimized regarding security, efficiency, and applicability in hybrid SDN networks.

A. TOPOLOGY MONITORING AND LINK DISCOVERY IN SDN

Topology monitoring in SDN maintains a network map on the SDN controller that consists of SDN switches and links in between. In contrast to legacy networks, topology monitoring in SDN can be limited to link discovery. This is because of the mandatory connection setup routine between SDN switches and the SDN control plane whenever a SDN switch is started. Thereby, the control plane knows about the presence of all SDN switches in the network so that only links need to be detected. In OpenFlow, SDN switches establish a connection to a pre-configured SDN controller right after start. The SDN controller receives information about the SDN switch, e.g., a list of all physical ports, within the OF handshake at connection setup. With the P4Runtime API, the SDN controller may connect to P4 switches right after start.

B. OpenFlow DISCOVERY PROTOCOL (OFDP)

The OpenFlow Discovery Protocol (OFDP) was the first de-facto standard for link discovery in SDN. It is based on the Link Layer Discovery Protocol (LLDP) [45], the most widely used protocol for link detection in legacy networks. The Cisco Discovery Protocol (CDP) [2] is a proprietary alternative but less widely used. LLDP advertisements include information about the identity of a host, its capabilities, and its current status. LLDP protocol data units (PDUs) are periodically sent as payload of Ethernet frames with a multicast receiver address and the EtherType 0 × 88cc. Figure 3 depicts their structure. The PDUs may contain various type-length value (TLV) blocks, the standard defines three required TLV blocks. First, the Chassis ID TLV identifies the sending host, e.g., by its MAC address. Second, the Port ID TLV identifies the sender's port, e.g., its physical port number. Last, the Time-to-Live TLV defines the time validity of the information. Optional TLV blocks, e.g., the system's name defined by the administrator, and custom TLVs may be used as well. Network hosts

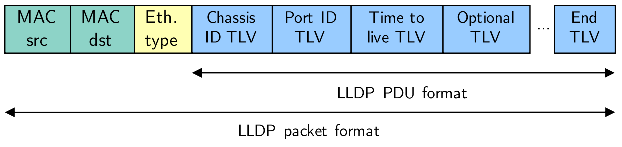

FIGURE 3. Format of LLDP packet and LLDP PDU.

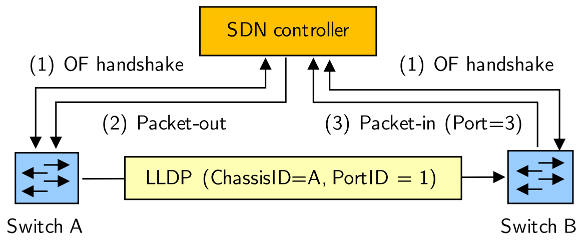

FIGURE 4. Link discovery in SDN with OFDP. The SDN controller learns about the SDN switch within the OF handshake (1). Afterwards, it sends out an LLDP packet on a particular port of an SDN switch via a packet-out message (2). Another SDN switch that receives the LLDP packet forwards it back to the SDN controller using a packet-in message (3).

that implement LLDP can receive, but not request LLDP information. Legacy switches periodically send out LLDP packets on each active port as described before. The packets are received, processed, and dropped by neighbouring LLDP agents on switches. They store the received information in the management information bases (MIBs) that can be queried via SNMP.

The OFDP leverages LLDP as introduced before but delegates all functionalities to the SDN controller. It uses packet-out messages to send out created network packets over particular ports of an SDN switch and packet-in messages to receive packets from the SDN switch that match specific criteria, e.g., an LLDP EtherType. Figure 4 depicts the process of link discovery with OFDP. First, the SDN controller learns about the switch identity and its ports within the OpenFlow handshake (1). Afterwards, it creates dedicated LLDP packets for all ports of a switch that are sent out via packet-out messages (2). For incoming LLDP packets, the OpenFlow switches are configured to forward any LLDP packet as packet-in message to the SDN controller (3). The packet-in message includes the LLDP packet with the Chassis and Port ID of the sender along the identity and the ingress port of the receiving SDN switch. By repeating this process for each port on each switch, the SDN controller performs link discovery.

C. OPTIMIZED VARIANTS OF OFDP

We review related work on optimized variants of OFDP that can be subdivided into publications investigating the security of OFDP, efficiency of OFDP, and applicability of OFDP in hybrid networks.

1) SECURITY OF OFDP

Alharbi *et al.* [31], [32], Azzouni *et al.* [33], and Nguyen and Yoo [50] show that OFDP is vulnerable to spoofing attacks. Injected LLDP control messages may create fake links that redirect traffic to the host of an attacker. Azzouni *et al.* [33] demonstrate that OFDP is additionally vulnerable to controller fingerprinting, switch fingerprinting, and LLDP flooding attacks. Nguyen and Yoo [50] show that OFDP is vulnerable to replay attacks. Faked LLDP packets result in incorrect link information of the topology. As improvement, Alharbi *et al.* [31], [32] propose to add a message authentication code (MAC) and a message identifier to each packet to provide authentication, packet integrity, and that prevent replay attacks. sOFTDP [34] encrypts LLDP packets to further prevent fingerprinting attacks. Marin *et al.* [47] investigate the security of SDN topology discovery mechanisms for OpenFlow. The authors propose mandatory integrity checks of LLDP and countermeasures against out-of-band channel and host location hijacking attacks. However, the latter two are not relevant in switch-to-switch link detection as required by MACsec.

2) EFFICIENCY OF OFDP

Azzouni *et al.* [33] and Rojas *et al.* [53] show that OFDP results in too many packet-out messages as the SDN controller has to create and send out one message for every active port on each SDN switch. As an improvement, ONOS [16] applies LLDP with Port IDs set to zero. The SDN controller creates one LLDP packet for every SDN switch that is configured to output the LLDP packet on all ports. This process is repeated for all SDN switches. Adjacent SDN switches are configured to forward received LLDP packets to the SDN controller so that it learns about the unidirectional link. Pakzad *et al.* [52] propose to reduce the number of packet-out messages by rewriting LLDP packets on the SDN switch. sOFTDP [34] introduces several mechanisms to shift large parts of link discovery back to the SDN switch. It adds liveliness detection mechanisms for switch ports and memorizes topology information locally on the SDN switch that asynchronously notifies the SDN controller about specific events. Rojas *et al.* [53] propose the Tree Exploration Discovery Protocol. SDN controllers create and send out specific frames flooded in the network that explore its topology. However, all concepts that shift functionality back to the SDN switches require extensive functional changes on the fixed-function data plane of typical SDN switches.

3) APPLICABILITY OF OFDP IN HYBRID NETWORKS

Hybrid networks consist of SDN and non-SDN switches. OFDP can be only applied to detect links in networks that consist of SDN switches. Legacy switches that may connect SDN switches process and discard received LLDP packets. The Broadcast Domain Discovery Protocol (BDDP) is a non-standardized approach that is implemented by several SDN controllers [3], [15], [19]. BDDP messages adapt the LLDP packet structure but use a broadcast Ethernet address instead of a multicast Ethernet address and the custom EtherType 0 × 8999. SDN switches are programmed to forward received BDDP packets to the SDN controller, just as with LLDP. Legacy switches flood the packet via all ports because of the broadcast address. They relay BDDP packets so that links will appear as single hops no matter how many legacy devices are on the path between two SDN switches. Ochoa-Aday *et al.* [51] show that the usage of broadcast packets leads to inefficient and excessive use of network resources. The authors propose a two-phase process for topology detection. First, the SDN controller performs link discovery using OFDP as described before. Afterwards, it outputs BDDP packets on any active port for which it does not detect a direct link to another SDN switch via LLDP. This way, the SDN controller detects direct links via LLDP and indirect links via BDDP.

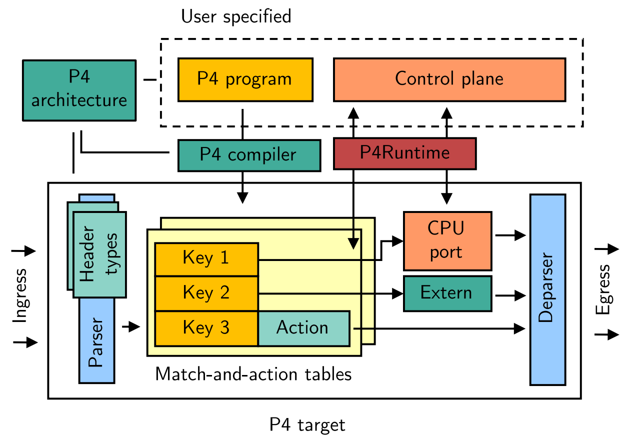

FIGURE 5. Concept of $P4_{16}$: P4 pipline and interaction with the control plane.

IV. P4: FOUNDATIONS

We give a brief overview of P4 with its core components for programming a P4 switch, describe the P4Runtime, and present examples for P4 software and hardware targets.

A. OVERVIEW

P4 is a domain-specific language for programmable data planes of packet forwarding devices. It offers high-level constructs that are optimized for specifying their forwarding behavior. P4 was first published in 2014 [37]. Today, the specification and development takes place in the non-profit P4 Language Consortium [21] with over 110 members from industry and academia. $P4_{16}$ [24] is the latest version of the language specification, the source code of all related components is available under an Apache license.

Figure 5 depicts P4's core concept and components. P4 programs describe the entire behavior of packet forwarding devices. They are formulated for a particular P4 architecture. A P4 architecture defines the P4 programming model of a packet forwarding device. P4 targets are software or hardware packet forwarding devices that implement a specific P4 architecture. Target-specific P4 compilers generate binary code from P4 programs that can be loaded on the P4 targets.

B. CORE COMPONENTS FOR PROGRAMMING A P4 SWITCH

Figure 5 shows all core components of the P4 pipeline following the specification of $P4_{16}$. We explain them in the following.

- **P4 header types** describe the format of packet headers through an ordered collection of base types. For example, an Ethernet header is described by bit vectors for the MAC source address, the MAC destination address, and the EtherType.
- **P4 parsers** are state machines that extract packet data through applying predefined sequences where data is identified and extracted based on P4 header types. For instance, the value of a parsed EtherType field of a packet determines the following extraction state which could be LLDP, MACsec, or IP.
- **P4 tables** are match-and-action structures mapping user-defined keys to particular P4 actions that may manipulate packet data.
- **P4 externs** are functions provided by a P4 target that can be used within P4 programs. P4 externs have a defined interface with a set of methods that can be used in the P4 program. An example would be a function that calculates a checksum for a given chunk of data.
- The **P4 deparser** assembles the headers back into a well-formed network packet that can be sent out via an egress port of the packet forwarding device.

C. P4Runtime

The P4Runtime framework provides an API for controlling P4 targets. Its operation is visualized in Figure 5. The P4Runtime features the manipulation of match-and-action tables through the control plane. In addition, it provides a CPU port for sending out and receiving packets similar to the packet-in and packet-out mechanism known from OpenFlow. P4Runtime leverages gRPC [9] that is based on HTTP/2 and protocol buffer [8] data structures. The connection between P4 targets and the control plane can be secured through TLS with optional client and server certificates for mutual authentication.

D. P4 SOFTWARE & HARDWARE TARGETS

The BMv2 [6] is the most widely-used P4 software target platform that features multiple P4 targets. Examples are a P4 target with the P4Runtime API (simple_switch_grpc) or a P4 target implementing the Protocol Independent Switch Architecture (PISA). The NetFPGA-SUME [57] and the Netcope NFB-200G2QL [13] are P4 targets that are based on FPGA platforms. Ethernet switches featuring the Barefoot Tofino ASIC [1] are the most widely-used P4 targets nowadays. The Barefoot Tofino ASIC implements the PISA and offers 12.8 Tb/s of packet throughput in its latest version. It is part of Ethernet and whitebox switches that also feature a general-purpose computing unit with a x86 CPU, RAM, and a SSD that runs a Linux-based operating system.

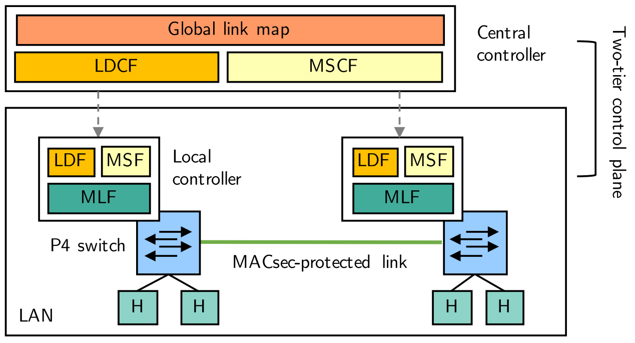

FIGURE 6. P4-MACsec in LAN with P4 switches. The local controllers consist of the MLF, LDF, and MSF. They communicate with the central controller that runs the LDCF and MSCF.

V. P4-MACsec

In this section, we describe P4-MACsec. We review its architecture and outline the three functional parts in detail: packet switching with MAC address learning, secure link discovery, and automated deployment of MACsec.

A. OVERVIEW

Figure 6 depicts the concept of P4-MACsec. It consists of a LAN with P4 switches that are steered by an SDN control plane. The switches provide Ethernet connectivity for the connected hosts and include the data plane functionality of the three functional parts. P4-MACsec features a *two-tier control plane structure*. Each P4 switch runs a local controller that connects to a central controller. The two-tier control plane offers the possibility to implement functions either on the global controller or on local controllers, or to split them and implement their parts on both the glocal controller and the local controllers. This reduces traffic in the management network, load on the central controller, and latency from forwarding packets between the SDN switches and the SDN controller. The concept of hierarchical SDN control is not novel but part of several SDN control plane architectures (e.g., [39], [40], [46]). With P4-based SDN, a two-tier control plane can be easily supported as most P4 targets feature general purpose computing capacities that can host a local controller. This is different with most OF switches which usually do not offer local computing capacities so that additional devices are needed to implement local controllers. Having the local controller on the switch eliminates dependencies on external devices and reduces the risk of failure. Bannour *et al.* [35] provide an overview of hierarchical and distributed SDN control planes and discuss their specifics. Although we see many advantages in the two-tier control plane architecture, P4-MACsec can also be implemented with only a central controller that carries out the functions of the local controllers. However, this is less elegant, causes more communication overhead, and is more prone to failure.

We design the three functional parts as follows. First, *MAC address learning for packet switching* ensures that

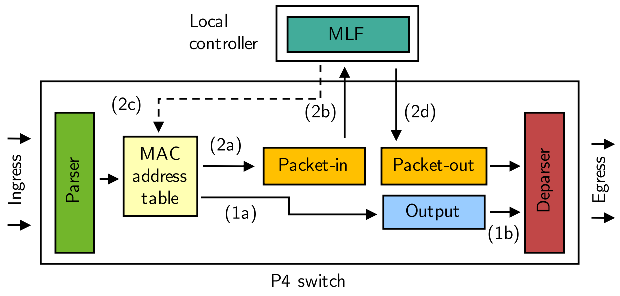

FIGURE 7. Process of forwarding Ethernet packets on the P4 processing pipeline. If the MAC table misses an entry, it triggers MAC address learning that is performed by the MLF running on the local controller.

P4 switches forward Ethernet packets to provide Layer 2 connectivity for all network hosts. We implement it as MLF on the local controller and describe its details in Section V-B. Second, *secure link discovery* detects and monitors the link topology of the network using LLDP PDUs that are protected with AES-GCM. We implement it as two-tier control plane function consisting of LDFs and the LDCF. The LDFs run on the local controllers and inform the LDCF on the central controller about local links. The LDCF composes the global link map which is the basis for automated deployment of MACsec. We describe the details of this functional part in Section V-C. Third, *automated deployment of MACsec* dynamically creates, sets up, and maintains SCs on all switch-to-switch links from the global link map. We implement it as two-tier control plane function with decentral MSFs that receive configuration from a MSCF running on the central controller. We describe its details in Section V-D.

B. ETHERNET PACKET SWITCHING WITH MAC ADDRESS LEARNING

Although MAC address learning is a typical example for a local switch function, it cannot be solely implemented on the data plane of a P4 switch. Our proposed architecture for that functional part consists of two components. First, a MLF that runs on the local controller. Second, the data plane implementation for MAC address learning with the MLF and packet switching.

Figure 7 visualizes the process with all interactions between both components. When the P4 switch receives an Ethernet packet, it first checks if the source and destination MAC address of the Ethernet packet are already part of the MAC address table. If the MAC address table has entries for both (1a), the switch forwards the packet on the port specified for the destination MAC address (1b). If the MAC table yields no match for both addresses (2a), the switch forwards the Ethernet packet to the MLF running on the local controller as packet-in message (2b). The MLF first checks if the MAC source address and the ingress port is already part of the MAC address table. If not, the MLF updates the MAC address table (2c). Afterwards, the MLF floods the Ethernet packet on all ports except the ingress port from where it received the packet through the packet-out function (2d).

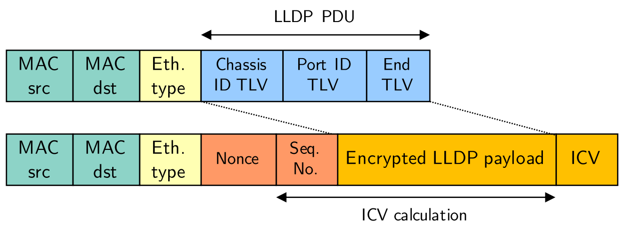

FIGURE 8. Proposed format for encrypting LLDP PDUs with AES-GCM. The EtherType from the original packet is followed by a Nonce that is used in AES-GCM encryption and decryption. A sequence number protects against replay attacks, a ICV holds a cryptographic checksum for authentication.

C. SECURE LINK DISCOVERY

The functional part of secure link discovery consists of three components. First, LDFs running on local controllers that do link detection and monitoring. Second, the LDCF running on the central controller that composes the global link map from local link information received from the LDFs. Third, the data plane implementation for receiving and sending out LLDP packets via packet-in and packet-out messages.

As novelty, we propose to create, encrypt, and decrypt LLDP PDUs on the LDF using AES-GCM with a common encryption key. We additionally introduce sequence numbers for LLDP PDUs to defend them against replay attacks. Figure 8 visualizes our proposed format of LLDP PDUs in comparison to the original format of LLDP packets. As in legacy LLDP, the MAC source address is set to the MAC address of the P4 switch. The MAC destination address and the EtherType are set to the LLDP defaults as introduced in Section III-B. LLDP PDUs consists of three TLVs. The Chassis ID TLV contains the identity of the switch as defined by the network administrator, the Port ID TLV contains the number of the physical port, the End TLV marks the end of the LLDP PDU. The common encryption key is installed and frequently updated by the LDCF on all LDFs. In addition, AES-GCM uses a 12 byte random number as nonce that is re-generated for each LLDP PDU leaving a particular port. It is part of the packet header following the EtherType. The four byte sequence number as protection against replay attacks is initialized with the LDF bootup timestamp and incremented with each packet sent out. The receiving LDF holds a sequence number counter for every physical port that is incremented with any received packet. The sequence number is part of the packet authentication of AES-GCM that is applied to the sequence number and on the LLDP PDU. Its result, the authentication tag, is stored within the ICV field following the encrypted LLDP PDU. Our approach resembles the one presented by Azzouni *et al.* [34] and protects against all attacks that were discussed in Section III-C.

Figure 9 visualizes the process of secure link discovery with all interactions among the three components. At startup, the LDF initiates a connection to the LDCF through a preconfigured IP address or FQDN (1). The LDCF installs the common key that is used for encrypting and decrypting LLDP packets with AES-GCM and instructs the LDF to start secure link discovery (2). The LDF generates and

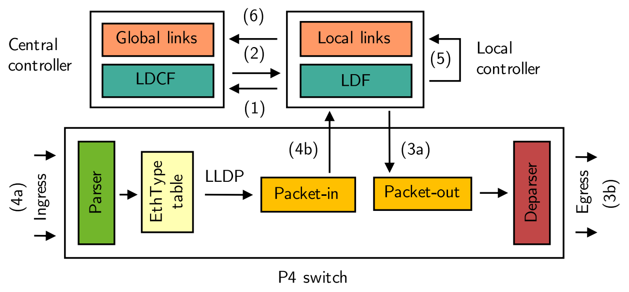

FIGURE 9. **Process of secure link discovery using the local LDF, the LDCF, and the processing pipeline on the P4 switch.**

transmits encrypted LLDP packets for all active ports via packet-out messages to the P4 switch (3a) which then outputs the received packets on the specified ports (3b). As all other P4 switches received the same instruction to start link discovery, the P4 switch now receives encrypted LLDP packets from other P4 switches (4a) that it sends as packet-in messages to the LDF (4b). It performs decryption and extracts the Chassis and Port ID of the distant switch from the LLDP PDU along the physical port number of the packet-in message to update the map of local links (5). Finally, all changes of the local link map are sent to the global link map on the central controller (6). The global link map consists of bidirectional links in the form of two tuples that indicate the identity of the P4 switch and the physical port of the link as $(\text{Switch}_A, \text{Port}_A) \rightarrow (\text{Switch}_B, \text{Port}_B)$. The link topology can change at any time, e.g., when links between switches are added or when cables break. Therefore, link discovery is executed whenever the LDF running on the local controller receives status messages from its assigned switches, e.g., due to a port-down notification when a cable breaks. Link discovery is additionally performed after a fixed time interval of 30 seconds so that security can be sustained even if status messages from the P4 switches get lost.

In comparison to other approaches presented in Section III-C, our proposal does not require modifications of the SDN switches. All required mechanisms are implemented as control plane functions that rely on packet-in and packet-out mechanisms as offered by the CPU port in P4 or the packet-in and packet-out messages of OpenFlow. A two-tier control plane as used by P4-MACsec is not mandatory, the LDCF and LDF could be also combined into one application to be run on a central controller.

D. AUTOMATED DEPLOYMENT OF MACsec

Automated deployment of MACsec consists of three parts. First, a MSCF that creates two unidirectional SCs for each link of the global link map. Second, a MSF that runs on the local controller. It receives configuration data from the MSCF and sets up the SCs on the P4 switch. Third, the data plane implementation of MACsec that consists of the P4 processing pipeline and implementations of the MACsec validate and protect functions that can be used as P4 externs.

Figure 10 visualizes the automated deployment of MACsec with all interactions between the three components. We later describe the details of the MACsec validate and protect function. The P4 processing pipeline is an extension of the Ethernet packet forwarding pipeline that was introduced in Section V-B. At start, the MSCF creates and maintains MACsec SCs based on the global link map (a). It passes SC configuration data to the MSF (b) which updates various P4 tables in the processing pipeline (c). For the P4 processing pipeline, it sets a MACsec flag to entries of the MAC address table if a SC for that particular link exists.

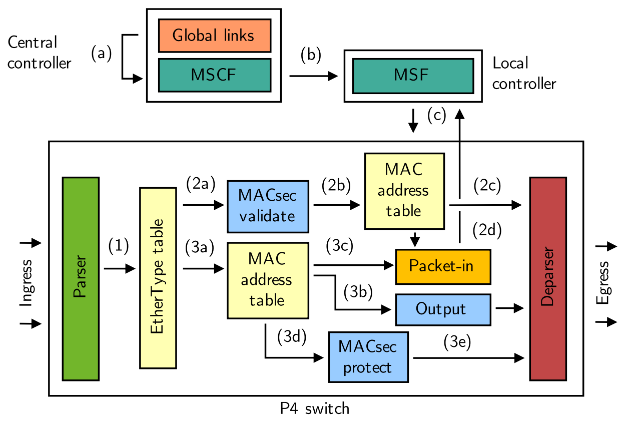

FIGURE 10. **Process of automated MACsec deployment with the MSF, MSCF, and the processing pipeline of the P4 switch.**

Then, the data plane processing for packets works as follows. First, ingress packets are matched in an EtherType table (1). Ethernet packets matching the MACsec EtherType are forwarded to the MACsec validate function (2a). It validates its authenticity, optionally decrypts the secure data, and returns an Ethernet packet. Afterwards, the processing pipeline continues with Ethernet packet forwarding, i.e., it consults the MAC address table (2b), outputs the packet in case of a match for both source and destination MAC address (2c), or sends the packet as packet-in message to the MLF on the local controller otherwise (2d). Ethernet packets matching other EtherTypes are forwarded to the MAC address table (3a). If the MAC address table holds no flag for an SC for the particular destination MAC address, the packets are either sent out (3b) or passed to the MLF (3c) as explained in Section V-B. If the MAC table yields a match for source and destination MAC addresses of the packet and an SC flag, it forwards the packet to the MACsec protect function (3d). The MACsec protect function responds with a MACsec packet that can be sent out (3e) via the egress of the processing pipeline.

Figure 11 visualizes the MACsec protect and validate function. As SCs are unidirectional, the P4 switch has an ingress MACsec SC (IG-SC) table and an egress MACsec SC (EG-SC) table that maps secure channel identifiers (SCIs) to SAs. SAKs are part of the SA table which holds SAs for both ingress and egress SCs. The MACsec protect function (1) either encrypts or authenticates Ethernet payloads. It is applied to Ethernet packets if the MAC address table in the Ethernet packet forwarding pipeline has a flag set

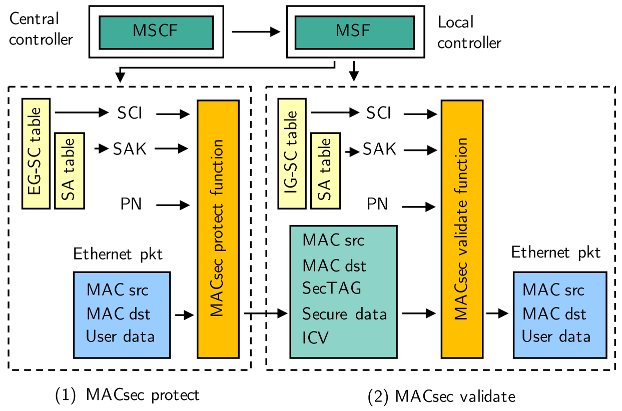

FIGURE 11. MACsec protect (1) and MACsec validate (2) functions implemented as P4 externs within the P4 processing pipeline.

for the particular physical port. Within the protect function, an EG-SC table maps the egress port number as SCI to security association identifiers (SAIs). The SA table holds the SAKs to be used for the protect function. The MACsec protect function receives the SAK and SCI from the SA table, the packet number from a packet counter as part of the switch, and the Ethernet packet. The AES-GCM cipher is initialized with a concatenation of the SCI and packet number as initialization vector. Afterwards, the EtherType and the payload are concatenated and encrypted. The MACsec protect function creates a new Ethernet packet with the MAC source and MAC destination address of the Ethernet packet, the MACsec EtherType, the SecTAG, the secure data, and the ICV. The MACsec validate function (2) works in a similar manner. Again, an IG-SC table maps SCIs to SAIs. The MACsec validate function receives the SAK and SCI from the IG-SC and SA table, the packet number from a packet counter, and the MACsec packet. It then checks the integrity and optionally decrypts the packet. It returns an Ethernet packet with the original Ethernet header and payload to the processing pipeline where forwarding and MAC address learning are performed as described before.

The MSCF creates and maintains MACsec SCs whenever the global link map changes. SCs exist until the corresponding link is deleted, e.g., in case of a link failure, the corresponding SC with all its SAs is deleted. If a new link is detected, the MSCF creates a new MACsec SC with SAs. In addition, MACsec SCs and SAs are renewed on a regular basis. Administrators define timeouts for encryption keys, the MSCF generates and installs new SAs on the P4 switches after the defined time interval through the MSF. Whenever SCs are created, changed, or deleted, configuration data and SAs are passed to the MSF that programs the P4 switch through writing in the EG-SC, IG-SC, and SA table.

VI. PROTOTYPICAL IMPLEMENTATION WITH MININET

In the following, we describe a prototypical implementation of P4-MACsec. We review the Mininet testbed environment and describe the three components of P4-MACsec in detail.

A. TESTBED ENVIRONMENT

We use the Mininet [12] network emulator to build the testbed environment for the prototypical implementation. We use the BMv2 P4 software target [6] for implementing the P4 switch and run the local controllers and the central controller as Python applications. For testing purposes, we additionally run Mininet network hosts that are connected to the P4 switches. All testbed components are executed within a KVM/QEMU VM that runs Ubuntu 16.04 with 4 CPU cores and 4 GB RAM. The hypervisor host features an Intel Core i5 8250U CPU, 16 GB RAM, and an SSD. It runs Manjaro Linux in Version 18.1.5.

We publish the source code with instructions for the testbed under the Apache v2 license on GitHub [7]. As of 2020-02, the maintainers of BMv2 discuss about integrating our implementation as an example for a crypto extern in the BMv2 source code repository [5].

B. P4 SWITCH

We extend the simple_switch_grpc [29] P4 target of the BMv2 P4 software target [6] to later run our P4 program that describes the data plane functions. Figure 12 depicts its parts. First, we implement the MACsec protect and validate functions as P4 externs within the simple_switch_grpc P4 target. We program the extensions in C++ and use the EVP interface of OpenSSL [20] to apply AES-GCM for encryption, decryption, and packet authentication. Both functions can be used as P4 externs within the P4 processing pipeline. When accessing the functions from the P4 processing pipeline, packet header data are exchanged using P4 attributes where packet payload data can be accessed directly. Second, we implement an interface to the local controller. It leverages the P4Runtime API via gRPC and allows the local controller to modify entries of the P4 tables in the processing pipeline. In addition, it holds the CPU port interface that provides packet-in and packet-out messages as known from OpenFlow. Packets sent from the P4 pipeline to the CPU port are forwarded to the local controller, packets received from the CPU port are injected into the P4 processing pipeline. The data plane functions of P4-MACsec described as P4 processing pipeline in Section V are implemented as $P4_{16}$ program using P4 constructs as introduced in Section IV. The P4 program then is executed on the modified simple_switch_grpc P4 target.

C. LOCAL CONTROLLER

We implement the local controller as Python 2.7 application. Figure 12 depicts its parts. We use the gRPC library [4] to program the interfaces to the associated P4 switch and to the central controller. We use the Scapy library [28] to create and parse LLDP packets and the cryptography library [25] for applying AES-GCM to encrypt and decrypt LLDP packets. For development and testing purposes, we include a simple CLI in the local controller. It provides functions to write/read table entries and to display status changes on the P4 switch.

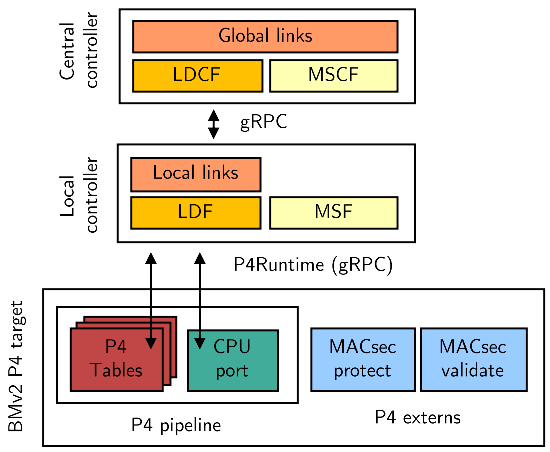

FIGURE 12. Structure of the prototypical implementation of MACsec. It consists of a BMv2 P4 software switch that implements the data plane functions of P4-MACsec and the two-tier control plane with the functions as introduced in Section V.

D. CENTRAL CONTROLLER

We implement the central controller similar to the local controller as Python 2.7 application. Figure 12 depicts its parts. It also leverages the gRPC library [4] to build a gRPC interface to the local controller. It also features a simple CLI for development and testing purposes which displays information about the current topology and active MACsec SCs. The control plane functions of P4-MACsec can also be integrated in ONOS [15] or other controller frameworks. However, our objectives was a lightweight prototype using a slim and easy-to-understand controller implementation which directly leverages the P4Runtime API and gRPC library for communication. Thereby, we avoided dependencies on other controller frameworks which increase error space and implementation complexity.

VII. FUNCTIONAL VALIDATION

We describe experiments for functional validation, execute them on the testbed from Section VI, and report their results.

A. EXPERIMENT I: COMPLIANCE TO THE MACsec STANDARD

We first perform an experiment to examine the compliance of P4-MACsec to the IEEE 802.1AE standard. Therefore, we create a virtualized testbed that consists of a KVM virtual machine running Ubuntu Server in Version 18.04.1 LTS and our implementation of P4-MACsec on the BMv2 P4 software target as described in Section VI. The P4 switch connects another Ubuntu Server 18.04.1 LTS KVM/QEMU virtual machine that represents a network host behind a MACsec-enabled switch. We configure a static MACsec connection between the P4 switch and the Linux host to check whether the MACsec implementation for BMv2 is compatible with the Linux implementation of MACsec. On the Linux host, we configure the static MACsec connection using the iproute2 tools. For MACsec setup on the P4 switch, we use a

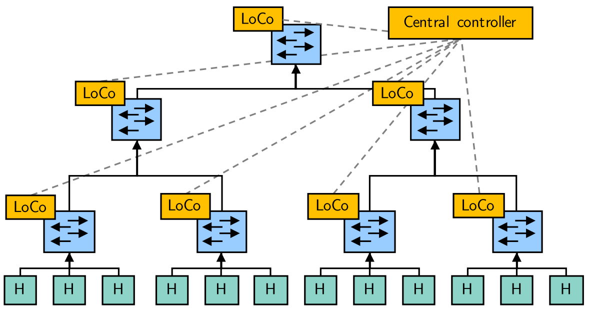

FIGURE 13. Virtual testbed environment that consists of network hosts, access switches, distribution switches, and a core switch. Each P4 switch is steered by a local controller (LoCo) that connects to the central controller.

simple Python script that adds the corresponding entries in the EG-SC, IG-SC, and SA tables of the P4 processing pipeline. We successfully validate that the Linux host communicates with the P4 switch via MACsec in different communication scenarios, e.g., ICMP or streaming random data via TCP connections with netcat. This does not validate a full compliance to all parts of the MACsec standard but demonstrates that the P4 switch can communicate via MACsec with legacy devices.

B. EXPERIMENT II: FULL P4-MACsec SCENARIO

We now investigate the complete set of functionality of P4-MACsec. Therefore, we create the topology depicted in Figure 13. It follows the model of hierarchical network switches that consists of core, aggregation, and access switches. A set of 12 network hosts is split into four groups, each attached to an access switch. The four access switches are connected to two aggregation switches that are connected by a single core switch. The testbed network is a single Layer 2 domain, i.e., network packets are forwarded based on their MAC address. After starting the Mininet testbed, we verify the following aspects. First, we examine that topology monitoring works correctly. In initial link discovery, we verify that the detected topology matches the actual network topology. Afterwards, we sporadically remove and re-add links between switches and supervise the process of link monitoring on the central controller via a CLI. Second, we examine that automated deployment of MACsec and rekeying works correctly. We investigate MACsec setup after changes in link monitoring through supervising the EG-SC, IG-SC, and SA tables on all P4-MACsec switches. Last, we examine packet switching and correct setup of MACsec protection. Therefore, we use ICMP and netcat to create network traffic between various pairs of network hosts in the experiment scenario. We investigate packet traces on links between switches and verify that all packets are protected by MACsec.

VIII. PERFORMANCE EVALUATION

We describe experiments for performance evaluation, execute them on the testbed from Section VI, and analyze their results.

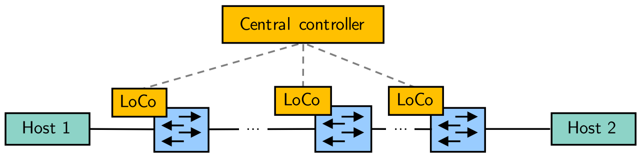

FIGURE 14. Evaluation testbed that consists of two network hosts that are attached to two P4 switches. In between, we vary from 0 to 6 additional P4 switches to form variable-length P4 switch chains for the evaluation experiments.

A. EVALUATION SETUP

Figure 14 depicts the evaluation setup. It consists of two network hosts that are attached to two P4 switches with 0 to 6 P4 switches in between. We perform performance evaluation experiments to investigate the goodput and RTT. We vary the number of P4 switches between the two network hosts and measure goodput and RTT for 1 to 8 hops. For each evaluation experiment, we consider three scenarios. In the first scenario, MACsec is disabled, i.e., the P4 switches between Host 1 and Host 2 only perform MAC address learning and Layer 2 forwarding. In the second scenario, we enable MACsec so that all packets between Host 1 and Host 2 are protected with AES-GCM implemented as P4 extern. In the third scenario, we enable MACsec but skip AES-GCM encryption and decryption in the P4 extern so that only plaintext payloads are sent within the MACsec packets. Although encapsulating plaintext payloads is not part of the MACsec standard, we still use this scenario to measure the effect of AES-GCM encryption/decryption by comparing it to network packet exchange with a P4 extern.

B. TCP GOODPUT

We first investigate the TCP goodput in P4-MACsec in our setting with software switches. To that end, we measure TCP transmissions between Host 1 and Host 2 with iperf3 [11]. Host 1 runs an iperf server, Host 2 runs an iperf client. We perform three runs, each with a duration of 30 seconds. Figure 15 depicts the results calculated as average over the three runs. The observed TCP goodput decreases with the number of hops, which is a result of increased round-trip time due and in particular of added queuing delay. However, we witness that forwarding without MACsec achieves significantly larger throughput than forwarding with MACsec. Obviously, there is additional packet processing delay with MACsec. However, this is only to a minor degree due to encryption, which can be seen by the fact that MACsec without encryption decreases the TCP goodput in a similar manner. Therefore, we conclude that the mere usage of externs on BMv2 increase the packet processing delay and reduces the TCP goodput.

C. ROUND-TRIP TIME (RTT)

In the second experiment, we investigate the RTT between the two network hosts that are connected by 1 to 8 P4 switches in between. We use the ping tool on Host 1 to send 1000 consecutive ICMP echo requests to Host 2. We set an idle period of

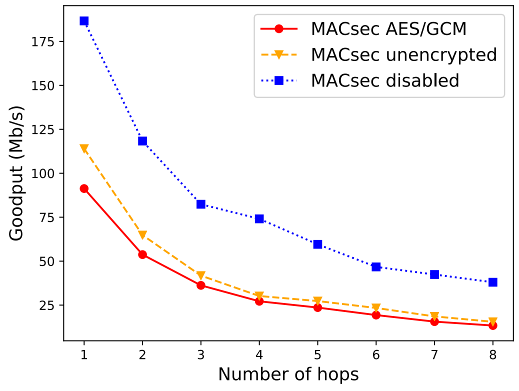

FIGURE 15. TCP goodput evaluation with 1 to 8 hops represented by P4 switches between two network hosts with iperf3. We consider three scenarios: disabled MACsec, enabled MACsec with encryption and decryption using AES-GCM, and enabled MACsec without encryption and decryption.

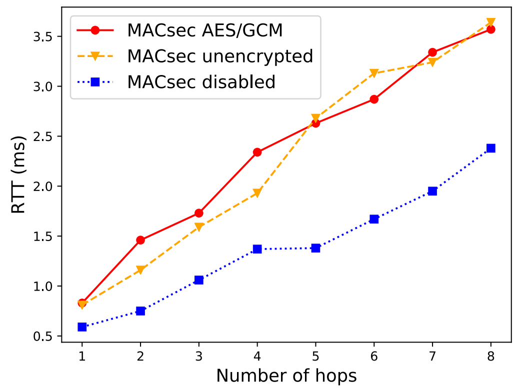

FIGURE 16. RTT evaluation with 1 to 8 hops represented by the P4 switches between two network hosts with pings. We consider three scenarios: disabled MACsec, enabled MACsec with encryption and decryption using AES-GCM, and enabled MACsec without encryption and decryption.

0.01 seconds between two ICMP packets and perform three runs of the experiment. Figure 16 depicts the RTTs calculated as average over the three runs. The evaluation results are similar to those of the experiment for TCP goodput. Enabled MACsec causes an increase of the RTT. Again, applying or omitting AES-GCM in the P4 extern does not cause large differences in the RTT. As in the experiment for TCP goodput, the interaction between the P4 pipeline and the MACsec protect and MACsec validate P4 externs in BMv2 seems to cause the negative effects on the RTT.

D. MACsec SETUP TIME

As described in Section V, the two-tier control plane automatically configures and enables MACsec on all assigned P4 switches. As described in Section II-B, MACsec deployment requires manual configuration effort regardless of static key setup on all switches or MKA. With P4-MACsec, that configuration effort completely disappears. The process of link discovery, MACsec setup, and MACsec rekeying are performed within nearly not measurable time. The two-tier control plane processes all events sequentially, i.e., delays might visibly increase only with a very large number of controlled P4 switches.

IX. IMPLEMENTATION ON NetFPGA SUME

In the following, we briefly describe the NetFPGA SUME [14] platform and outline our experiences in implementing P4-MACsec for that platform.

A. NetFPGA SUME PLATFORM

The NetFPGA SUME board is an open-source platform for rapid prototyping of network applications with support for bandwidths up to 100 Gb/s. It features a Virtex-7 690T FPGA, four SFP+ network transceivers, and a PCI Express interface to the host system [57]. The P4-NetFPGA project [22] transforms the NetFPGA SUME board into a hardware P4 target. P4 programs are transformed into SDNet descriptions by the P4-SDNet compiler that creates HDL modules which run as part of the reference architecture of the NetFPGA SUME board.

B. IMPLEMENTATION OF P4-MACsec

We modify the P4 processing pipeline of our software prototype to cope with limitations of the NetFPGA SUME architecture, e.g., a missing look-ahead function in packet parsing or the limitation to a single control block instead of multiple control blocks in the P4 processing pipeline. We implemented AES-GCM based on a publicly available Verilog module from OpenCores [18]. However, we were not able to create a fully working P4-MACsec switch due to two severe limitations. First, the NetFPGA SUME platform does not provide functions to parse or access variable-length payloads of network packets. Therefore, payloads of network packets need to be parsed as headers, which limits the implementation to fixed-length packets. Last, exchange of packet data between the P4 processing pipeline and the P4 external function is limited. Currently, data that is transferred from the P4 processing pipeline to a P4 external function needs to be transmitted within one clock cycle of the FPGA. Due to timing limitations, it is only possible to transmit very small amounts of data. The developers from the P4-NetFPGA project confirmed that its current version does not provide support for processing complete network packets within P4 externs. We were able to increase data to be exchangeable to 128 bytes by reducing the base clock frequency of the NetFPGA. However, this is still far away from applicability to real-world problems. A packet streaming function was announced, but is not available so far. Summing up, both limitations did not allow us to build a prototype that is suitable for real-world scenarios with variable-length packets exceeding a total length of 128 bytes.

Scholz *et al.* [54] pursued implementation of cryptographic hash functions on P4 data planes. NetFPGA SUME is part of their examined platforms. The authors confirm our experiences and propose a workaround where cryptographic hash functions are relocated to the platform's egress path behind the synthesized P4 program. However, if packet processing within the P4 programm relies on the output of the cryptographic hash functions, other workarounds need to be considered. All proposed workaround modifications require in-depth knowledge about FPGA programming. From our point of view, this conflicts with P4's original idea of platform-independent and abstract network programming.

X. CONCLUSION

In this work we proposed P4-MACsec, a concept to automatically protect links between switches with MACsec in P4-based SDN. Our concept features a P4 data plane implementation for MACsec including encryption and decryption using AES-GCM. P4 switches are steered by a novel two-tier control plane that consists of local controllers running on all P4 switches that connect to a central controller. We presented a novel mechanism for link discovery using encrypted LLDP packets and automated deployment of MACsec link protection. P4-MACsec completely eliminates previous configuration efforts for MACsec. We presented the architecture of P4-MACsec and demonstrated its feasibility in a prototypical implementation for the BMv2 P4 software target. We used that prototype to experimentally validate P4-MACsec in a virtualized testbed built with Mininet and performed evaluation experiments. We also reported on unsuccessful efforts to implement P4-MACsec on the NetFPGA SUME platform. Our work is an example for P4 switches supporting security features like authentication, encryption, and identity checks. Other applications may be traffic protection on different layers, e.g., Layer 3 VPNs. Therefore, P4 switches should offer native functional blocks for encryption and decryption and overhead-free interfaces to P4 externs.

LIST OF ACRONYMS

MACsec	Media Access Control Security
MKA	MACsec Key Agreement
SDN	Software-Defined Networking
OF	OpenFlow
BMv2	Behavioral Model version 2
FPGA	field programmable gate array
AES-GCM	Advanced Encryption Standard in Galois/Counter mode
LLDP	Link Layer Discovery Protocol
SecY	MAC security entity
KaY	MAC security key agreement entity
CA	connectivity association
SC	secure channel
SA	secure association
SAK	secure association key
GMAC	Galois Message Authentication Code
ICV	integrity check value
SecTAG	security tag
CAK	connectivity association key
ACL	access-control list
ECU	electronic control unit
OFDP	OpenFlow Discovery Protocol
CDP	Cisco Discovery Protocol

PDU	protocol data unit
TLV	type-length value
MIB	management information base
MAC	message authentication code
BDDP	Broadcast Domain Discovery Protocol
SCI	secure channel identifier
SAI	security association identifier
RTT	round-trip time
MLF	MAC address learning function
LDF	link discovery function
LDCF	link discovery controller function
MSF	MACsec function
MSCF	MACsec controller function
IG-SC	ingress MACsec SC
EG-SC	egress MACsec SC
PISA	Protocol Independent Switch Architecture

ACKNOWLEDGMENT

The authors would like to thank Joshua Hartmann for fruitful discussions and programming contributions.

REFERENCES

[1] *Barefoot Networks: Tofino 2*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino-2/

[2] *Cisco Discovery Protocol Configuration Guide*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cdp/configuration/15-mt/cdp-15-mt-book/nm-cdp-discover.html

[3] *Floodlight: OpenSource SDN Controller*, Accessed: Feb. 2, 2020. [Online]. Available: http://www.projectfloodlight.org/floodlight/

[4] *GitHub: GRPC Python*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/grpc/grpc/tree/master/src/python/grpcio

[5] *GitHub: Open Pull Requests of p4lang/Behavioral-Model: Add Crypto Extern to Behavioral-Model*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model/pull/834

[6] *GitHub: P4lang/Behavioral-Model (BMv2)*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model

[7] *GitHub: Uni-Tue-kn/p4-Macsec*. Accessed: Mar. 29, 2020. [Online]. Available: https://github.com/uni-tue-kn/p4-macsec

[8] *Google Protocol Buffers*. Accessed: Feb. 2, 2020. [Online]. Available: https://developers.google.com/protocol-buffers/

[9] *Grpc*. Accessed: Feb. 2, 2020. [Online]. Available: https://grpc.io/

[10] *IEEE8021-SECY-MIB: Definitions of Managed Objects Supporting IEEE 802.1AE MACsec*. Accessed: Apr. 14, 2019. [Online]. Available: http://www.ieee802.org/1/files/public/MIBs/IEEE8021-SECY-MIB-200601100000Z.txt.

[11] *Iperf*. Accessed: Feb. 2, 2020. [Online]. Available: https://iperf.fr/

[12] *Mininet*. Accessed: Feb. 2, 2020. [Online]. Available: http://mininet.org/

[13] *Netcope P4*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.netcope.com/en/products/netcopep4

[14] *NetFPGA*. Accessed: Feb. 2, 2020. [Online]. Available: https://netfpga.org

[15] *ONOS*. Accessed: Feb. 2, 2020. [Online]. Available: https://onosproject.org/

[16] *ONOS Wiki: Network Discovery*. Accessed: Feb. 2, 2020. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Network+Discovery

[17] *OpenCORD Wiki: MAC Security*. Accessed: Feb. 2, 2020. [Online]. Available: https://wiki.opencord.org/display/CORD/MAC+Security

[18] *Open Cores: GCM-AES*. Accessed: Feb. 2, 2020. [Online]. Available: https://opencores.org/projects/gcm-aes

[19] *Open Daylight*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.opendaylight.org/

[20] *OpenSSL*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.openssl.org/

[21] *P4 Language Consortium*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/

[22] *P4->NetFPGA Wiki*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/NetFPGA/P4-NetFPGA-public

[23] *P4 Runtime Spec. 1.0.0*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/p4runtime/spec/v1.0.0/P4Runtime-Spec.html

[24] *P4_16 Language Specification, Version 1.2.0*. Accessed: Feb. 2, 2020. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.0.html

[25] *Pyca/Cryptography Documenta*. Accessed: Feb. 2, 2020. [Online]. Available: https://cryptography.io/en/latest/

[26] *Rambus: MACsec Toolkit for Ethernet Security*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.rambus.com/security/software-protocols/secure-communication-toolkits/macsec-tk/

[27] *RHD Blog: MACsec: A Different Solution to Encrypt Network Traffic*. Accessed: Feb. 2, 2020. [Online]. Available: https://developers.redhat.com/blog/2016/10/14/macsec-a-different-solution-to-encrypt-network-traffic/,

[28] *Scapy*. Accessed: Feb. 2, 2020. [Online]. Available: https://scapy.net/

[29] *Simple Switch Grpc*. Accessed: Feb. 2, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model/tree/master/targets/simple_switch_grpc

[30] *Cisco: Innovations in Ethernet Encryption (802.1AE—MACsec)*. Accessed: Feb. 2, 2016. [Online]. Available: https://www.cisco.com/c/dam/en/us/td/docs/solutions/Enterprise/Security/MACsec/WP-High-Speed-WAN-Encrypt-MACsec.pdf

[31] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of topology discovery in software defined networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 502–505.

[32] T. Alharbi, M. Portmann, and F. Pakzad, "The (In)Security of topology discovery in openflow-based software defined network," *Int. J. Netw. Secur. Its Appl.*, vol. 10, no. 3, pp. 01–16, May 2018.

[33] A. Azzouni, N. T. Mai Trang, R. Boutaba, and G. Pujolle, "Limitations of openflow topology discovery protocol," in *Proc. 16th Annu. Medit. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, Jun. 2017, pp. 1–3.

[34] A. Azzouni, R. Boutaba, N. Thi Mai Trang, and G. Pujolle, "SOFTDP: Secure and efficient topology discovery protocol for SDN," 2017, *arXiv:1705.04527*. [Online]. Available: http://arxiv.org/abs/1705.04527

[35] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.

[36] O. I. Bentstuen and J. Flathagen, "On bootstrapping in-band control channels in software defined networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[37] P. Bosshart, "P4: Programming Protocol-independent Packet Processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95. Jul. 2014.

[38] J.-H. Choi, S.-G. Min, and Y.-H. Han, "MACsec extension over software-defined networks for in-vehicle secure communication," in *Proc. 10th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2018, pp. 180–185.

[39] Y. Fu, "Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks," in *IEEE Int. Conf. Netw. Protocols (ICNP)*, 2014, pp. 569–576.

[40] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st workshop Hot topics Softw. defined Netw.*, 2012, pp. 19–24.

[41] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security*, Standard 802.1AE-2006, 2006.

[42] *IEEE Standard for Local and Metropolitan Area Networks—Port-Based Network Access Control*, Standard 802.1X-2010, 2010.

[43] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security Amendment 1: Galois Counter Mode—Advanced Encryption Standard—256 (GCM-AES-256) Cipher Suite*, Standard 802.1AEbn-2011, 2011.

[44] *IEEE Standard for Local and Metropolitan Area Networks—Media Access Control (MAC) Security Amendment 2: Extended Packet Numbering*, Standard 802.1AEbw-2013, 2013.

[45] *IEEE Standard for Local and Metropolitan Area Networks—Station and Media Access Control Connectivity Discovery, Corrigendum 2: Technical and Editorial Corrections*, Standard 802.1AB-2009, 2015.

[46] T. Kohler, F. Durr, and K. Rothermel, "ZeroSDN: A highly flexible and modular architecture for full-range distribution of event-based network control," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 4, pp. 1207–1221, Dec. 2018.

[47] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into SDN topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1101–1114.

[48] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69, Mar. 2008.

[49] P. S. Mohamed, "Network controller provisioned MAC sec keys," U.S. Patent 14 763 484, Aug. 27, 2019.

[50] T.-H. Nguyen and M. Yoo, "Analysis of link discovery service attacks in SDN controller," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 259–261.

[51] L. O. Aday. (2015). *Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks*. [Online]. Available: https://upcommons.upc.edu/handle/2117/77672.

[52] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *Proc. 8th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2014, pp. 1–8.

[53] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "TEDP: An enhanced topology discovery service for software-defined networking," *IEEE Commun. Lett.*, vol. 22, no. 8, pp. 1540–1543, Aug. 2018.

[54] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmuller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in p4 data planes," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–6.

[55] T. Szyrkowiec, M. Santuari, M. Chamania, D. Siracusa, A. Autenrieth, V. Lopez, J. Cho, and W. Kellerer, "Automatic intent-based secure service creation through a multilayer SDN network orchestration," *J. Opt. Commun. Netw.*, vol. 10, no. 4, p. 289, Apr. 2018.

[56] M. Vajaranta, J. Kannisto, and J. Harju, "Implementation experiences and design challenges for resilient SDN based secure WAN overlays," in *Proc. Asia Joint Conf. Inf. Secur. (AsiaJCIS)*, Aug. 2016, pp. 17–23.

[57] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

FREDERIK HAUSER (Student Member, IEEE) received the master's degree in computer science from the University of Tuebingen, Germany, in 2016, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, network function virtualization, and network security.

MARK SCHMIDT (Member, IEEE) received the Diploma degree in computer science from the University of Tuebingen, Germany, in 2010, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, OpenFlow, high-speed networks, and virtualization.

MARCO HÄBERLE (Student Member, IEEE) received the master's degree in computer science from the University of Tuebingen, Germany, in 2018, where he is currently pursuing the Ph.D. degree with the Chair of Communication Networks. Since then, he has been a Researcher with the Chair of Communication Networks, University of Tuebingen. His main research interests include software-defined networking, P4, network security, and automated network management.

MICHAEL MENTH (Senior Member, IEEE) is currently a Professor with the Department of Computer Science, University of Tuebingen, and the Chair holder of Communication Networks. He has published more than 150 articles in the field of computer networking. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, software-defined networking and the Internet protocols, industrial networking, and the Internet of Things.