IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 5, 2020, accepted July 16, 2020, date of publication July 29, 2020, date of current version August 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012738

P4-IPsec: Site-to-Site and Host-to-Site VPN With IPsec in P4-Based SDN

FREDERIK HAUSER, (Graduate Student Member, IEEE), MARCO HÄBERLE, (Student Member, IEEE), MARK SCHMIDT, (Member, IEEE), AND MICHAEL MENTH, (Senior Member, IEEE)

Chair of Communication Networks, University of Tübingen, 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 In this work, we present P4-IPsec, a concept for IPsec in software-defined networks (SDN) using P4 programmable data planes. The prototype implementation features ESP in tunnel mode and supports different cipher suites. P4-capable switches are programmed to serve as IPsec tunnel endpoints. We also provide a client agent to configure tunnel endpoints on Linux hosts so that site-to-site and host-to-site application scenarios can be supported which are the base for virtual private networks (VPNs). While traditional VPNs require complex key exchange protocols like IKE to set up and renew tunnel endpoints, P4-IPsec benefits from an SDN controller to accomplish these tasks. One goal of this experimental work is to investigate how well P4-IPsec can be implemented on existing P4 switches. We present a prototype for the BMv2 P4 software switch, evaluate its performance, and publish its source code on GitHub [1]. We explain why we could not provide a useful implementation with the NetFPGA SUME board. For the Edgecore Wedge 100BF-32X Tofino-based switch, we presented two prototype implementations to cope with a missing crypto unit. As another contribution of this paper, we provide technological background of P4 and IPsec and give a comprehensive review of security applications in P4, IPsec in SDN, and IPsec data plane implementations. According to our knowledge, P4-IPsec is the first implementation of IPsec for P4-based SDN.

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

I. INTRODUCTION

Virtual Private Networks (VPNs) extend private networks across public networks by adding authentication and encryption to network traffic. Internet Protocol Security (IPsec) is one of the oldest, but still most widespread VPN protocols. Standardized by the IETF, it introduces protection on the Internet Protocol (IP) layer. Due to its large distribution, many implementations for network appliances and operating systems are available. Although it is criticized for its complexity, proven deployment patterns allow efficient and reliable operation.

IPsec tunnel setup requires user configuration plus keying material that is exchanged by IPsec peers via the Internet Key Exchange (IKE) protocol. The complexity grows with the number of IPsec peers, especially in highly dynamic environments such as campus or enterprise networks with many users and sites. Several works investigate how to leverage the centralized control plane of software-defined networking (SDN) to simplify IPsec operation. However, the possibilities for IPsec deployment in SDN were limited. Typical SDN switches have a fixed function data plane that does not provide support for IPsec. As a result, IPsec data plane processing needs to be moved to an additional software-based packet processing function (PPF). Besides being an additional component, this adds latency as traffic needs to be forwarded back and forth. Programmable data planes as offered by P4 are a game changer. Data plane behavior can be described in a high-level programming language. Those network programs can be executed by software or hardware devices. For IPsec, this means that instead of shifting IPsec functionality to PPFs, functions can be implemented directly on the data plane of SDN switches. In our previous work P4-MACsec [2], we introduced MACsec for P4-based SDN.

The associate editor coordinating the review of this manuscript and approving it for publication was Congduan Li.

We proposed a data plane implementation in P4 and introduced a novel concept for automated deployment and operation of MACsec.

In this paper, we present the first integration of IPsec VPN for P4-based SDN. We give an introduction on the technological background and provide an extensive survey on related work in that field. We present an IPsec data plane implementation that integrates IPsec components and processes with constructs and components under the given constraints of the P4 data plane programming language. Cryptographic operations for authentication, encryption, and decryption are implemented in P4 externs where IPsec components such as the Security Policy Database (SPD) and Security Association Database (SAD) are part of the P4 processing pipeline. P4 switches that implement the functionality of P4-IPsec can be deployed in host-to-site and site-to-site VPN scenarios. The control plane functions for IPsec operation are part of a central SDN controller that maintains IPsec tunnels without the help of distributed key exchange protocols such as IKE. As these components are steered by a centralized control plane through an authenticated and encrypted control connection, complex IKE-based key exchange protocols are substituted by controller-based tunnel setup and renewal procedures. For host-to-site operation, we introduce a client agent for Linux operating systems that runs on the roadwarrior hosts. It establishes an interface to the central SDN controller via a gRPC connection. To investigate how well P4-IPsec can be implemented on existing P4 targets, we work on three prototypes. We successfully implement a prototype for the Behavioral Model version 2 (BMv2) P4 software target and conduct a performance evaluation. We release the source code of our prototype with its testbed environment under the Apache v2 license on GitHub [1]. In addition, we report on implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch. For the latter, we present two workaround implementations and compare them in performance experiments.

P4-IPsec introduces several benefits over traditional IPsec operation. First, we improve scalability by making switches and roadwarrior hosts stateless components whose functionality is only managed by an SDN controller. Second, we improve flexibility by converting P4 targets into IPsec endpoints, i.e., IPsec tunnels can terminate close to the network hosts that should be made accessible via the VPN. This limits the size of the perimeter and improves security through better isolation. Last, we encourage open networking research and operation. Network functionality can be modified in agile development processes, source code can be audited and improved by a larger audience.

The rest of the paper is organized as follows. Section II gives an overview on IPsec, VPN, and data plane programming with P4. In Section III, we describe related work on P4-based network security applications, IPsec in SDN, and IPsec data plane implementations. Section IV presents the architecture of P4-IPsec. In Section V, we describe the prototypical implementation of P4-IPsec with Mininet and BMv2. Section VI presents the performance evaluation of that prototype. In Section VII, we report implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch. Section VIII concludes this work. The appendices include a list of the acronyms used in the paper.

II. TECHNICAL BACKGROUND

We give an introduction to VPN with IPsec and data plane programming with P4.

A. IPsec VPN

Internet Protocol Security (IPsec) is a widespread VPN protocol suite. It applies authentication and encryption on the Internet Protocol (IP) in host-to-host, site-to-site, and host-to-site communication scenarios. RFC 4301 [3] is the latest version of its specification.

1) PROTOCOLS

IPsec comprises the *Authentication Header (AH)* and *Encrypted Secured Payload (ESP)* protocol. AH [4] protects IP packets by sender authentication and packet integrity validation. It applies a hash function with a shared key (e.g., HMAC-SHA256) to calculate Integrity Check Values (ICVs) and adds packet sequence numbers to protect against replay attacks. ESP [5] protects the confidentiality of IP packets by symmetric encryption. As for AH, it also adds sender authentication, packet integrity validation, and protection against replay attacks. ESP supports symmetric ciphers such as Triple Data Encryption Standard (3DES), Blowfish, and Advanced Encryption Standard (AES). Ciphers that only apply encryption are combined with an authentication function. AES in cipher block chaining (CBC) or counter (CTR) mode are examples for such ciphers that might be combined with secure hash algorithm (SHA) for authentication. Authenticated encryption (AE) ciphers such as AES in galois/counter mode (GCM) [6] include both packet encryption and authentication. IPsec provides support for IP Payload Compression (IPComp) [7] so that the payload of IP packets can be compressed before encryption.

2) OPERATION MODES

IPsec can be deployed in either *transport* or *tunnel* operation mode. Transport mode protects IP traffic that is exchanged between two network hosts (host-to-host scenario). An AH or ESP header is inserted between the IP header and the IP payload. Tunnel mode protects IP traffic in host-to-host, host-to-site, and site-to-site communication scenarios. Figure 1 depicts how tunnel mode with ESP is applied to an IP packet. A new outer IP header with the IP addresses of the IPsec peers is created. The original IP packet is inserted between the ESP header and the ESP trailer. Encryption protects the original IP packet while authentication is applied to the complete ESP packet.

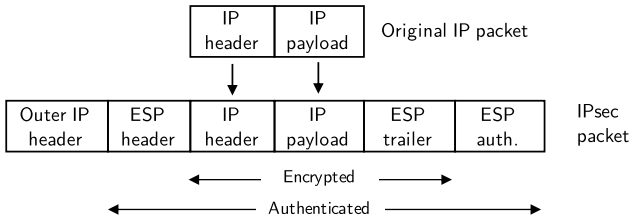


FIGURE 1. Tunnel mode with ESP. The original IP packet is inserted between the ESP header and the ESP trailer. The inner IP packet is encrypted while the complete ESP packet is authenticated.

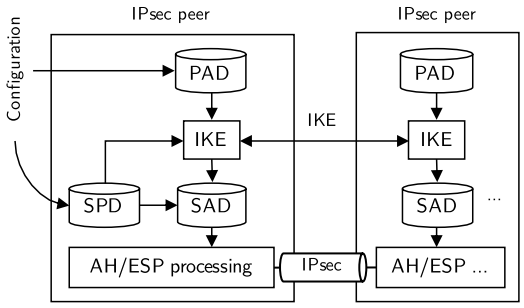


FIGURE 2. IPsec packet processing between two IPsec peers. Each peer features a SPD, SAD, PAD, and AH/ESP processing functions. The SPD and PAD are configured manually, where SAD entries are managed by the IKE daemon.

3) CORE COMPONENTS

We describe the core components of IPsec implementations that are part of hosts or gateways. The *Security Policy Database (SPD)* holds security policies that decide on traffic protection using IPsec. Entries have match keys, e.g., IP src/dst address, IP protocol, and TCP/UDP port, with an assigned action. IPsec allows three actions: DROP (discard packet), BYPASS (no protection), and PROTECT (apply IPsec protection). In case the table yields no match, the DROP action is applied. SPD entries for IPsec connections point to the protocol (AH/ESP), the operation mode (transport/tunnel), and the cipher suite. An IPsec tunnel between two peers is described by two unidirectional security associations (SAs). An IPsec SA contains all required data for AH/ESP processing, e.g., cipher keys, valid sequence numbers, and SA lifetimes for rekeying and tear down. SAs are part of the *Security Association Database (SAD)*. With the information from the SAD, packets then can be processed by *ESP/AH processing*. Although manual configuration of SAs is possible, they are typically configured between IPsec peers with the help of the Internet Key Exchange (IKE) protocol [8] that was introduced with IPsec. It authenticates both peers, sets up a secure channel for key exchange, and negotiates SAs. Today, its successor Internet Key Exchange v2 (IKEv2) [9] should be used. It is less complex and solves incompatibility issues of IKE. IKE relies on the *Peer Authentication Database (PAD)* for authentication of other IPsec peers.

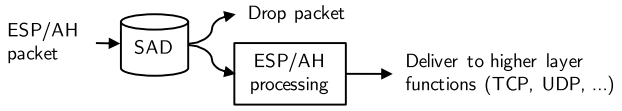


FIGURE 3. IPsec ingress processing. Arriving packets with an ESP/AH header are processed with the help of the SAD. ESP/AH processing relies on data in the SAD. In case of no match, the packet is dropped.

4) PACKET PROCESSING

IPsec differentiates between ingress and egress processing of packets. Figure 3 depicts ingress processing. Arriving packets that have an ESP/AH header are processed with the help of the SAD. If the SAD has an entry for the corresponding SA, the SA data is forwarded to the ESP/AH processing function that removes IPsec protection. Afterwards, the packet is forwarded to default network processing.

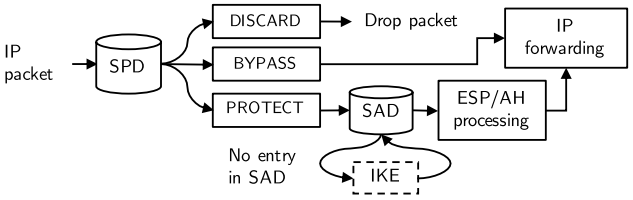


FIGURE 4. IPsec egress processing. The SPD matches packets and maps them to the actions DISCARD, BYPASS, and PROTECT. In case of BYPASS, the packet is passed to IP forwarding. In case of PROTECT, the SAD is searched for a corresponding entry for ESP/AH processing. In case of no match, the packet is dropped.

Figure 4 depicts egress processing where IP packets are matched with SPD entries as explained before. In case of PROTECT, data for ESP/AH processing is selected from the SAD. If the SAD has no matching entry, SA setup is requested from the IKE daemon. In case of BYPASS, the packet is passed to IP forwarding. In case of DISCARD, the packet is dropped.

5) DISCUSSION

Among more recent alternatives such as OpenVPN and WireGuard, IPsec is still one of the most widespread VPN technologies nowadays. IPsec implementations are part of most operating systems for computers, servers, and mobile devices. Most network hardware appliances, e.g., firewalls, routers, or security appliances, include an IPsec implementation.

However, IPsec is highly criticized for its complexity. The most encompassing analysis was performed by Ferguson and Schneier [10] in 2003. The authors mainly criticize the redundancy of functionality caused by AH, ESP, and the two operation modes, the complex key exchange with IKE, and the complex configuration caused by the SPD and SAD. However, those issues can be easily solved. Transport mode and AH should be avoided. Instead, AE ciphers that combine encryption and authentication should be used in conjunction with ESP with tunnel mode. IKE should be substituted by

a less complex protocol for key exchange. In P4-IPsec, we follow those recommendations and restrict the IPsec implementation to ESP and tunnel mode with controller-based SA management without IKE.

B. DATA PLANE PROGRAMMING WITH P4

SDN introduces network programmability by shifting control plane functions to a software-based controller that determines the packet processing behavior of network devices. OpenFlow (OF) [11] is the most widely used SDN approach. It relies on data plane devices with a fixed set of functions and a southbound interface to the SDN controller. The SDN controller defines how these functions are applied to network packets. Programmable data planes extend network programmability to data plane functionality. Packet processing can be defined on an abstract layer using a dedicated programming language. Thereby, packet processing behavior is decoupled from the underlying hardware. This new principle facilitates open network research with support for agile development processes and flexible deployment options. Bifulco and Rétvári [12] give an overview on programmable data planes. Target platforms include software targets, network interface cards (NICs), NICs with a field programmable gate array (FPGA) unit, and hardware appliances with network processing units (NPUs). P4 is the most widely used data plane programming language nowadays. Initially presented as a research paper in 2014, the project is now standardized by the P4 Language Consortium under the Open Network Foundation (ONF). Its latest specification is version 16 ($P4_{16}$) [13].

1) PROCESSING PIPELINE

Figure 5 depicts a simplified view on the packet processing pipeline of P4. It consists of three core abstractions that help to express forwarding behavior.

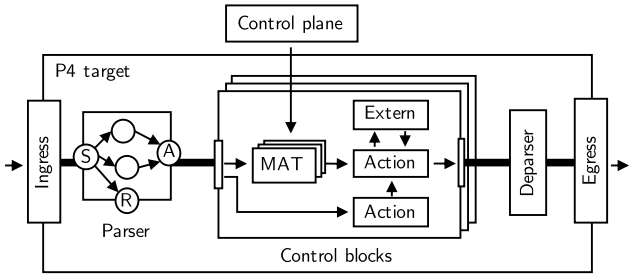


FIGURE 5. Simplified view on the P4 processing pipeline of $P4_{16}$. It comprises the parser, control blocks, and the deparser. Each control block may include MATs, actions, and externs.

a: PARSER

The parser extracts header fields of packets into internal data structures. P4 does not include predefined header types, i.e., programmers need to define packet formats and extraction behavior. Packet header formats are defined using P4 header types such as fixed- and variable-length bit strings or integers. The extraction behavior of the parser is expressed as finite state machine (FSM). Parsing is initiated in the state *start*, possible outcome states are *accept* (proceed in packet processing) and *reject* (drop the packet). Custom states that are positioned between start and end states implement the extraction of header data. The transitions between those states are formulated using conditions. For example, after successfully parsing an IP header, state transitions to TCP or UDP parsing might follow.

b: CONTROL BLOCKS

Control blocks are functions that modify packet headers and metadata. The P4 processing pipeline can include multiple control blocks that are typically separated by queues or buffers. Packet processing in control blocks is stateless: the outcome of packet processing applied to one packet can not influence packet processing applied on a subsequent packet. Actual packet processing is implemented in *actions*, code fragments within control blocks that implement read/write operations with functions provided by P4, e.g., setting header fields or adding/removing headers. Actions can be called from other actions, explicitly with the start of the control block, or implicitly by *match-action tables (MATs)*. MATs map match keys to particular actions with associated parameters. When applying a MAT to packets, the header and metadata are matched in exact, ternary, or in longest prefix manner against the keys of the MAT. If matching yields a particular row entry, the specified action is called with the associated parameters. If there is no match, a default action is applied. P4 programs only contain the declaration of MATs, their entries are maintained by a control plane via an application programming interface (API) in runtime. Some targets may provide additional functions for packet processing, e.g., particular functions such as checksum generation, or stateful components such as counters, meters, and registers. These components can be used within P4 programs as so-called *externs*. Externs have an interface with defined instantiation methods, functions, and parameters. After import and declaration, they can be used in control blocks just like any other P4 function.

c: DEPARSER

The deparser reassembles the packet header and payload and serializes it to be sent out via an egress port.

2) DEPLOYMENT MODEL

Software or hardware platforms that execute P4 programs are called P4 targets. Common software P4 targets are the BMv2 [14] software target, eBPF packet filters, and the T_4P_4S [15] software target that includes hardware interfaces via Data Plane Development Kit (DPDK) [16] and Open Data Plane (ODP) [17]. The hardware P4 targets include FPGA-based targets and NICs, NPU-based NICs, and whitebox switches featuring the Tofino application-specific integrated circuit (ASIC) from Barefoot Networks. P4 programs are implemented for a particular P4 architecture.

P4 architectures can be seen as programming models that represent the logical view of a P4 processing pipeline. They serve as an intermediate layer to decouple P4 programs from P4 targets, i.e., P4 programs that are implemented for a particular P4 architecture can be deployed to all P4 targets that implement this architecture. A front-end compiler translates P4 programs into a target-independent high-level intermediate representation (HLIR). Afterwards, the HLIR is compiled to the particular P4 target using a back-end compiler that is provided by the manufacturer.

3) CONTROL PLANE API: P4Runtime

The runtime behavior of P4 targets can be controlled by managing MATs or stateful components (e.g., counters, meters, registers, or externs) that are part of the P4 program. P4Runtime API [18] is a target- and program-independent API standardized by the P4 Language Consortium. P4Runtime uses gRPC for communication between the control plane and P4 targets and protobuf [19] data structures for packet serialization/parsing. gRPC connections can be secured with Transport Layer Security (TLS) and mutual authentication with certificates. In P4Runtime, the SDN controller establishes gRPC connections to preconfigured P4 targets. P4Runtime supports P4 object access (e.g., on MATs and externs), session management (master/slave controllers), role-based access control, and a packet-in/-out mechanism to receive and send out packets via the control plane. The PI Library is the reference implementation of the P4Runtime server that is part of P4 targets. It implements generic functionality for internal P4 objects such as MATs. This functionality can be extended by target- or architecture-specific configuration objects. *p4runtime_lib* [20] is an exemplary implementation of the P4Runtime API in Python to be used for building SDN controllers. P4Runtime API plugins are also available for common SDN controllers such as ONOS or OpenDaylight.

4) APPLICATION DOMAINS

Most research works on P4-based network applications target data center or wide area networks. In traffic management and congestion control, P4 is leveraged to implement new congestion notification mechanisms, novel traffic scheduling mechanisms, or novel mechanisms for active queue management. In routing and forwarding, special routing and forwarding mechanisms, publish-subscribe systems, or novel concepts from the area of named data networks are implemented. A large focus also lies on monitoring, where several works implement monitoring systems, sketch-based monitoring mechanisms, and in-band network telemetry (INT) systems. Besides, P4 is used in data center scenarios to implement switching, load balancing, network function virtualization (NFV), and service function chaining (SFC) mechanisms.

III. RELATED WORK

We describe related work on network security applications built with P4, IPsec in SDN, and implementation of IPsec packet processing.

A. NETWORK SECURITY APPLICATIONS WITH P4

Although network security is not the prevalent application domain of P4, some scientific work has been published in this field. We describe related work on firewalls, DDoS mitigation mechanisms, and other security applications.

1) FIREWALLS

Vörös and Kiss [21] introduce a P4-based firewall for filtering IPv4, IPv6, TCP, and UDP packets. It includes a ban list for instant drop, counters, e.g., for measuring the packet rate or unsuccessful connection attempts, and MATs for applying whitelist firewall rules. P4Guard [22] follows a similar approach. Its authors focus on simplified updated processes by deploying recompiled versions of the P4 program. Ricart-Sanchez *et al.* [23] implement a P4-based firewall for 5G networks. It includes parser definitions for filtering GPRS tunneling protocol data. CoFilter [24] introduces a hash function for efficient flow identification. It is built as P4 action and uses hashes instead of 5-tuples for flow identification to save table space. Including the function directly on the packet processing devices keeps latency low. Zaballa *et al.* [25] and Almaini *et al.* [26] introduce port knocking on P4 switches.

2) DDoS MITIGATION MECHANISMS

Paolucci *et al.* [27], [28] propose a DDoS mitigation mechanism that runs on P4 switches. A stateful mechanism detects and blocks DDoS port scan attacks with incremental TCP and UDP destination port numbers. Dimolianis *et al.* [29] also implement a DDoS attack mitigation mechanism that runs completely on P4 switches. The collected flow data is mapped to distinct time intervals where DDoS attacks are detected by analyzing the symmetry ratio of incoming and outgoing traffic. TDoSD@DP [30] implements a mitigation scheme against DDoS attacks on SIP proxies. The authors introduce a simple state machine that monitors SIP message sequences. Valid sequences of INVITE and BYE messages keep the port open. Febro *et al.* [31] implement another DDoS mitigation mechanism for SIP INVITE DDoS attacks. P4 switches keep per-port counters for INVITE or REGISTER packets that are monitored by an SDN controller to detect DDoS attacks. LAMP [32] implements cooperative mitigation of application layer DDoS attacks via in-band signaling with P4. Afek *et al.* [33] implement known mitigation mechanisms for SYN and DNS spoofing in DDoS attacks in P4. Lapolli *et al.* [34] describe a novel algorithmic approach based on the Shannon entropy to detect and stop DDoS attacks on P4 switches. Kuka *et al.* [35] introduce an FPGA-based system for DDoS attack mitigation. P4 is used to extract header data from packets and send it to an SDN

controller where DDoS attack identification is implemented. Mi and Wang [36] propose a similar approach where collected data is sent to a deep learning module that runs on a server in the network.

3) OTHER SECURITY APPLICATIONS

Lewis *et al.* [37] implement an IDS offloading mechanism in P4. A rule parser translates Snort IDS rules into MAT entries for a P4 switch. Then, IDS pipeline stages decide if packets should be forwarded, dropped, or sent to an external IDS for analysis. Poise [38] is a security-related network control system that translates high-level policies into P4 programs for network control. In P4-MACsec [2], we implement IEEE 802.1AE (MACsec) in P4 and introduce an automated deployment mechanism provisions MACsec on detected links between P4 switches. Link monitoring is implemented using a novel variant of Link Layer Discovery Protocol (LLDP). It relies on encrypted payloads and sequence numbers to protect against LLDP packet manipulation and replay attacks.

B. IPsec IN SDN

Several works investigate the application of SDN to IPsec operation. We describe and discuss operation modes, southbound interfaces, and use cases.

1) OPERATION MODES

Related work can be categorized by three different operation modes that are depicted in Figure 6.

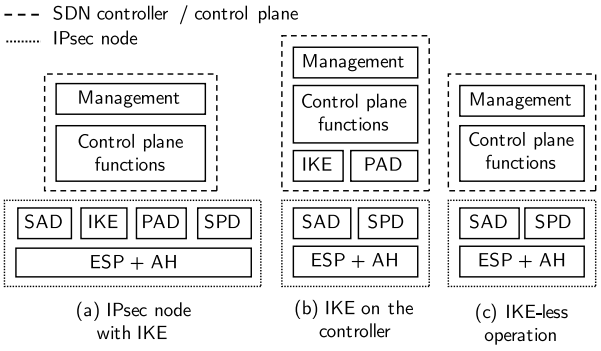


FIGURE 6. Operation modes for data plane management of IPsec. In (a), IKE is part of the IPsec node. In (b), IKE is part of the control plane. In (c), IKE is substituted by controller-based SA management.

a: IPsec NODE WITH IKE

In the first operation mode, IPsec processing nodes feature an IKE daemon, SDN assists in preconfiguration. Aragon *et al.* [39], [40] propose that an SDN controller preconfigures authentication keys in the PAD. Carrel and Weiss [41] propose that an SDN controller distributes Diffie-Hellman public values to all associated IPsec data plane nodes. Guo *et al.* [42] propose a similar approach that is compatible to older IKE daemons that only support IKEv1. Lopez-Millan *et al.* [43] propose an *IKE mode*, where the SDN controller only provides information for the configuration of the SPD, PAD, and IKE daemon. All proposals aim to reduce the message exchanges in an IKE process by preconfiguring it by an SDN controller.

b: IKE ON THE SDN CONTROLLER

In the second operation mode, the IKE daemon is part of an SDN controller. Son *et al.* [44] present an approach where the IKE daemon running on the SDN controller performs key exchange with peers and manages the SAD of the IPsec data plane nodes. This approach even supports migration schemes so that the SA can be transferred to other IPsec data plane nodes, e.g., in fail-over or load-balancing operations. Vajaranta *et al.* [45] describe a similar approach where IKE is executed as a network function that can be scaled up by creating additional instances.

c: IKE-LESS OPERATION

In the third operation mode, SAs are maintained without IKE. Lopez-Millan *et al.* [43] describe an *IKE-less* mode where the SA maintenance is delegated to an SDN controller. Here, the IPsec processing nodes only implement IPsec logic where the complete key management logic is moved to the SDN controller. As there is no IKE, no PAD is required. The authors differentiate between a proactive mode, where SPD and SAD are preconfigured by the SDN controller, and a reactive mode, where only the SPD is preconfigured by the SDN controller. Several works [39], [46]–[48] propose SA management without IKE. The SDN controller generates keying material and sets up SAs in the SAD of associated IPsec data plane nodes. Gunleifsen *et al.* [49] introduce a key management server that creates and distributes IPsec SAs for encryption virtual network functions (VNFs). In consecutive works, Gunleifsen *et al.* [50], [51] name this concept as *Software-Defined Security Associations (SD-SAs)*. Encryption VNFs only perform IPsec processing. SAs are created and distributed by an authentication center.

2) DISCUSSION ON OPERATION MODES

We briefly discuss the benefits and drawbacks of the three operation modes. The first operation mode benefits from easy migration. As legacy IPsec devices already feature an IKE daemon, they can be easily extended by an interface to profit from SDN-assisted operation of IPsec (see [43]). The second operation mode especially introduces flexibility and scalability. Separating IPsec processing and SA establishing to different entities improves scalability (see [45]). The third operation mode removes the overhead of peer-to-peer key exchange with IKE. In SDNs, IKE might be unnecessary as both IPsec peers are controlled by an SDN controller. Besides, IKE requires connectivity for IKE packet exchange between both peers which could be not given in particular scenarios (see [51]). Lopez-Millan *et al.* [43] show in an analytical evaluation that IKE-based and IKE-less operation of IPsec have approximately the same process loads in terms of messages and configuration data exchange.

3) SOUTHBOUND PROTOCOLS

On legacy network devices that feature IPsec devices, SNMP (e.g., [52]) is used for basic configuration and monitoring. Guo *et al.* [42] extend this usage in making an IKE daemon manageable by SNMP as well. Alharbi *et al.* [53] use SSH as a southbound interface to manage and monitor IPsec data plane nodes. Marin-Lopez *et al.* [46] use NETCONF with YANG configuration models. In addition to the southbound protocol, they consider east-/westbound interfaces for controller-to-controller communication via different domains. Aragon *et al.* [39] use OAuth 2.0 to deliver configuration data within authorization messages. Braadland [47] extends OpenFlow (OF) using experimenter messages for IPsec management. Sajassi *et al.* [48] leverage BGP. Li and Mao [54] use a custom southbound protocol to interface an IPsec extension module on an Open vSwitch. Li *et al.* [55] propose a custom southbound protocol with notification, configuration, and query messages that are transmitted via TCP or TLS. Lopez-Millan *et al.* [43] use NETCONF with YANG models as southbound protocol. Gunleifsen *et al.* [50], [51] rely on REST with JSON.

4) USE CASES

Use cases that benefit from controller-based operation of IPsec are SD-WAN, cloud provider networks, and dynamic VPN setup.

a: SD-WAN

Large organizations with distributed locations require network connectivity between the different sites. As dedicated links are expensive, site-to-site IPsec-VPNs over provider networks are increasingly used. However, manually setting up VPN connections between all branches is time-intensive and complex. SD-WAN [42], [53], [54] proposes IPsec data plane functionality as part of hardware appliances or software modules at the perimeter of the different sites of the organization. Then, a centralized SDN controller automatically sets up and maintains IPsec-VPN connections.

b: CLOUD PROVIDER NETWORKS

Often, internal services offered by a public or private cloud provider need to be accessed from within networks of an organization. Again, site-to-site IPsec-VPN tunnels are a cost-efficient alternative to dedicated links. Administrators define IPsec-VPN gateways via a cloud management interface. Then, the cloud orchestrator deploys IPsec-VPN gateways as VNFs on the cloud provider's infrastructure. Its runtime operation is managed by a SDN controller. In addition, controller-based operation of IPsec can be also used to dynamically connect different cloud networks by a multi-cloud orchestrator [56]. Gunleifsen *et al.* [49], [50] propose hop-by-hop protection for SFCs using IPsec and controller-based operation.

c: DYNAMIC VPN SETUP

Managing many IPsec-VPN connections to different hosts or services on a client host can be cumbersome. Dynamic VPN setup performed by a SDN controller takes over the tasks of tunnel setup and management. Van der Pol *et al.* [57] present a concept where users request VPN access to a particular network device from the SDN controller. It then automatically sets up a VPN tunnel to the remote domain. Aragon *et al.* [39] combine dynamic VPN setup with authentication and authorization to automatically deploy IPsec-VPN tunnels between IoT network devices. This introduces several advantages over traditional deployment. First, the control plane has an encompassing view on the network topology with all devices. It can monitor usage and detect outages for reliable operation. Second, the centralized control plane features northbound interfaces for management applications and southbound interfaces for controlling data plane devices. Instead of manual per-device configuration, VPNs are operated via a management layer with policy languages that allow rule validation. Last, the centralized control plane offers flexibility so that the VPN operation can be extended by other mechanisms, e.g., user authentication with 802.1X [54].

C. IMPLEMENTATION OF IPsec PACKET PROCESSING

With P4-IPsec, we present the first data plane implementation of IPsec in P4. We give an overview on IPsec data plane implementations as related work.

1) SOFTWARE IMPLEMENTATIONS WITH HARDWARE ACCELERATION

IPsec software programs represent the simplest packet processing implementations. Their I/O performance depends on the hardware, the chosen cryptographic algorithm, and the average packet size. For Linux host systems, optimization techniques such as DPDK [16], Netmap [58], and PF_RING [59] tweak network stack processing to increase packet I/O rates. Other works propose to increase IPsec packet I/O by using multiple CPU cores [60], [61] or the GPU [62]. Gallenmüller *et al.* [63] compare several mechanisms in an extensive study. Most of the described optimization mechanisms are only applicable to Linux operating systems.

IPsec packet I/O of software implementations can be improved by offloading crypto operations or IPsec operations to hardware. For the former, current CPU architectures provide hardware acceleration for common crypto operations. AES-NI [64] or ARMv8 Cryptographic Extensions [65] are examples of AES instruction sets that replace pure software implementations. System on a chip (SoC) platforms or circuit boards may contain chips for offloading cryptographic processing. Examples are the Marvell Cryptographic Engines Security Accelerator (CESA) or Intel QuickAssist [66]. Such processors can be also part of extension circuit boards that

are connected to the mainboard via PCI. FPGAs might be also used for implementing crypto operations, several vendors (e.g., [67]) supply implementations of cryptographic algorithms as program cores. For the latter, IPsec hardware accelerators are available as ASIC [68], [69], NPU [70], [71], accellerated processing unit (APU) [72], or FPGA [73], [74].

2) HARDWARE IMPLEMENTATIONS

Proprietary IPsec hardware concentrators, e.g., as sold by Cisco or Juniper, are optimized for high IPsec I/O rates and, therefore, might implement a larger degree of the overall IPsec processing operations in hardware (e.g., on ASICs). Due to their disclosed architectural details, we cannot get insight into technical details. In addition, encompassing IPsec implementations for FPGAs exist [75], [76] where only the SPD and SAD are managed by an SDN controller.

3) IMPLEMENTATIONS ON PROGRAMMABLE DATA PLANES

In 2016, a Xilinx employee reported on the P4-Development mailing list [77] that IPsec was successfully implemented in PX [78], a high-level domain specific programming language for programmable data planes. Crypto primitives are implemented via an extern mechanism similar to P4 externs. The authors report that the crypto primitives were programmed as Register Transfer Level (RTL) designs targeting FPGAs. The authors report that the principle should be the same for P4, but it was not ported so far.

IV. CONCEPT

We describe the concept of P4-IPsec. We give an overview, discuss design choices, and describe its data plane and control plane in detail.

A. OVERVIEW

Figure 7 gives an overview on the functionality of P4-IPsec.

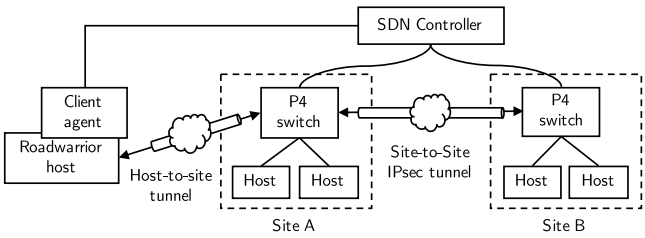


FIGURE 7. Overview on the functionality of P4-IPsec. In host-to-site operation, roadwarrior hosts run a client agent to set up an IPsec tunnel to a P4 switch via the SDN controller. In site-to-site operation, the SDN controller sets up IPsec tunnels for pairs of P4 switches.

P4-IPsec supports two IPsec tunnel operation modes: *host-to-site* and *site-to-site*. In host-to-site mode, roadwarrior hosts establish IPsec tunnels to get access to internal networks. Roadwarrior hosts run a client agent that interacts with the SDN controller for tunnel setup. In site-to-site mode, two internal networks are connected via an IPsec tunnel that is established between two P4 switches. As a core principle of P4-IPsec, every P4 switch implements the same IPsec functionality, i.e., it can act as both IPsec tunnel endpoint for roadwarrior hosts in host-to-site mode and for other P4 switches in site-to-site mode. This facilitates very flexible deployments where IPsec tunnels do not necessarily terminate at a central VPN concentrator but can be distributed to many P4 switches instead.

B. DESIGN CHOICES

P4 programs describe the packet forwarding behavior of switches or routers. Thereby, an implementation of IPsec in P4 is limited to data-plane-centric parts. Additional mechanisms such as IKE need to be part of an SDN controller implementing the control plane and interfacing the P4 switch. For P4-IPsec, our integration of IPsec in P4, we make the following design choices:

1) USE OF IKE-LESS OPERATION MODE

Referring to the results of Lopez-Millan *et al.* [43] (see Section III-B.1), we choose to implement IKE-less SA management via the SDN controller. Our proposed P4 processing pipeline comprises equivalent representations for the SAD and SPD that are both maintained by the SDN controller. Due to the lack of IKE, no PAD is required. Selecting IKE-less operation mode does not exclude an integration of IKE on the SDN controller at a later stage.

2) RESTRICTION TO ESP IN TUNNEL MODE

To keep our proposed concept as minimalistic as possible, we adopt the recommendations of Ferguson and Schneier [10] and restrict our implementation to ESP in tunnel mode.

3) IMPLEMENTATION OF CIPHER SUITES WITH EXTERNS

P4 does not provide functions for encryption, decryption, and message authentication. In contrast to related work, we decide against offloading IPsec processing to external software-based processing nodes and implement cipher suites with the help of P4 externs (see Section II-B). This should decrease the latency introduced by external processing while keeping the overall system more minimalistic. Each cipher suite is implemented by two externs; one that implements encryption functionality, and one that implements decryption functionality.

4) PROTOTYPE SIMPLIFICATIONS

We limit P4-IPsec to IPv4 and omit support for IPv6. We also omit support for IPComp. For applicability in experiments, we implement simple L3 forwarding based on longest-prefix matching (LPM). Clearly, this is not a requirement from the IPsec standard.

C. DATA PLANE OF P4-IPsec

We first give an overview on the P4 processing pipeline of P4-IPsec. For the sake of simplicity in presentation, we combine functions into function blocks and describe them later in detail.

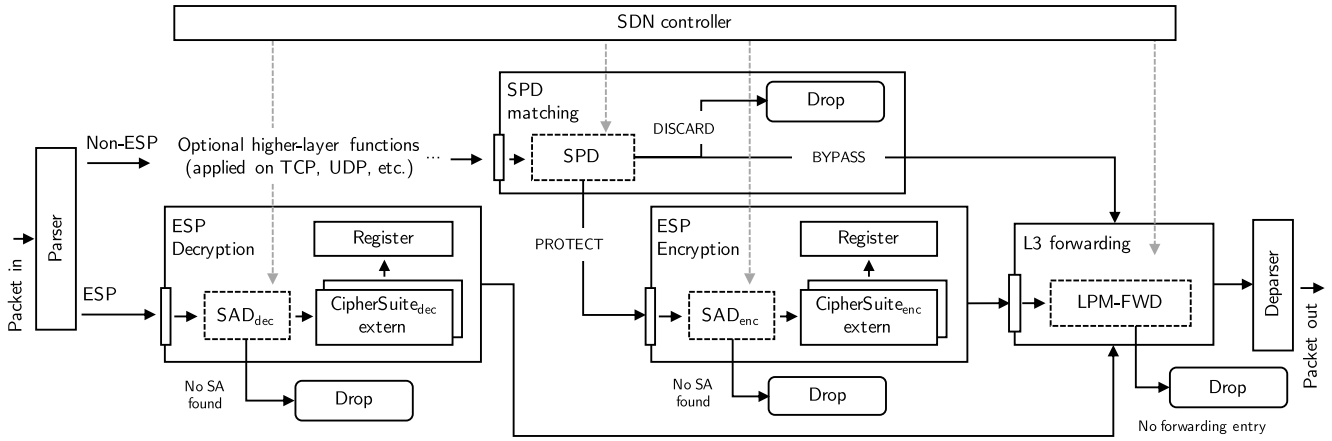


FIGURE 8. Data plane processing pipeline of P4-IPsec. For ease of understanding, related functionalities are grouped together as function block.

1) P4 PROCESSING PIPELINE

Figure 8 depicts the P4 packet processing pipeline of P4-IPsec. It consists of a parser, a deparser, and four function blocks in between. When a packet arrives via the ingress, the P4 parser first extracts the packet headers. In case of a header other than ESP, the parser forwards the packet to optional higher-layer functions that operate on protocol layers such as TCP or UDP. Afterwards, the SPD matching function block processes the packet. Following the IPsec standard, entries in the SPD determine about the action to be executed on the packet. In case of DISCARD, the packet is dropped. In case of BYPASS, the packet is passed to the L3 forwarding function block. In case of PROTECT, the packet is passed to the ESP encrypt function block. In the ESP encryption function block, encryption using SA data from the SAD_{enc} MAT is applied to the IP packet. In the L3 forwarding function block, the packet is forwarded based on the rules defined in the forwarding MAT. Going back to the parser again: if the packet has an ESP header, it is forwarded to the ESP decryption function block. It validates the packet's authenticity, decrypts the ESP message, and extracts the original IP packet that is then passed to the L3 forwarding function block. In case of missing entries in the SPD, SAD_{dec}, SAD_{enc}, or LPM-FWD MAT, the packet is dropped. As the final step, the deparser reassembles all headers and re-calculates the IPv4 checksum as some fields, e.g., the time to live (TTL), are changed. Runtime behavior of the data plane can be managed by manipulating the MATs via an SDN controller.

2) FUNCTION BLOCK: L3 FORWARDING

Figure 9 depicts the function block of *L3 forwarding*. It implements packet forwarding to the next hop via a particular output port of the P4 switch. The LPM-FWD MAT matches packets using their IPv4 destination addresses to two actions: *forward_packet* and *drop*. The forward_packet action receives the MAC address of the next hop and the output port as parameters from the MAT. Then, it sets the MAC

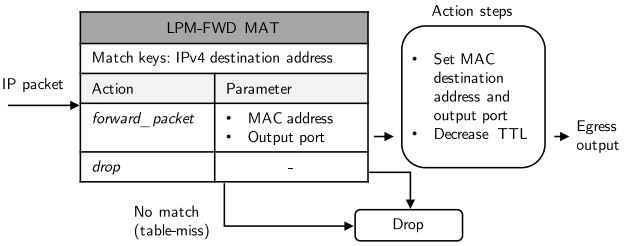


FIGURE 9. L3 forwarding function block. IP packets are processed by the LPM-FWD MAT that either applies the forward_packet or drop action. In case of no match, the drop action is applied.

destination address of the packet to the MAC address of the next hop, decreases the TTL by 1 in the IP header, and sets the output port. Afterwards, the packet is forwarded to the deparser and sent out via the egress. *drop* directly discards the packet; this action is also applied if no match in the LPM-FWD MAT is found.

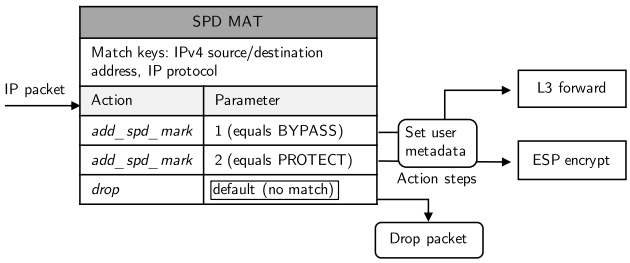


FIGURE 10. SPD matching function block. IP packets are processed by the SPD MAT. The add_spd_mark action adds the given parameter to the user metadata of the packet. It decides if the packet is protected by IPsec (PROTECT) or forwarded without protection (BYPASS) in later stages.

3) FUNCTION BLOCK: SPD MATCHING

Figure 10 depicts the function block of *SPD matching*. We introduce a security policy (SP) MAT that resembles the SPD from the IPsec standard (see Section II-A). It matches

given packets with SPD rules and adds a mark to the user metadata of each packet that is used in further processing within the P4 processing pipeline. We implement IPv4 source and destination address and IP protocol as exemplary match keys. Due to P4's flexibility in defining packet parsing, more match keys, e.g., TCP/UDP ports or even application-layer ports, could be added easily. Actions are either *add_spd_mark* or *drop*. The *add_spd_mark* action adds "spd_mark = 1" for BYPASS or "spd_mark = 2" for PROTECT to the user metadata field of the packet. *drop* directly discards the packet; this action is also applied if no match is found.

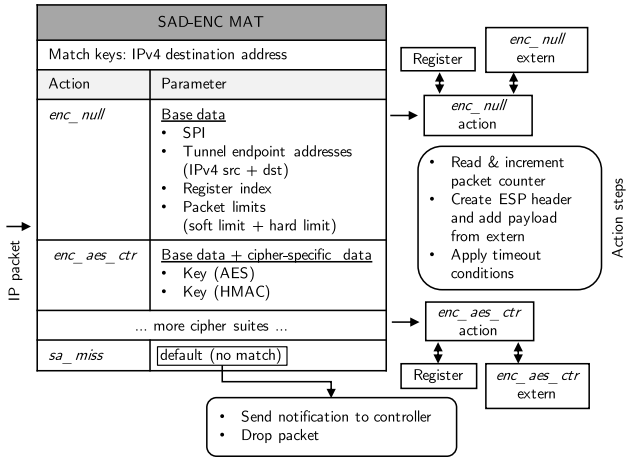


FIGURE 11. ESP encryption function block. IP packets are processed by the SAD-ENC MAT. It holds entries for each SA with the corresponding data that is required for applying the associated cipher suite externs for encryption.

4) FUNCTION BLOCK: ESP ENCRYPTION

Figure 11 depicts the function block of *ESP encryption*. We introduce a SAD-ENC MAT that resembles the SAD from the IPsec standard (see Section II-A). Each entry in the MAT represents a particular SA that is identified by the IPv4 destination address, i.e., packets are matched based on their IPv4 destination address.

We implement cipher suites as actions that rely on externs and registers. Representing a variety of cipher suites, we implement the *AES-CTR* and *NULL* cipher suites as examples. The *NULL* cipher suite is intended for testing purposes only. It uses the identity function instead of encrypting data and skips calculating an integrity check value. Cipher suite actions receive two types of parameters: base data that is required by all cipher suites and cipher-specific data.

We first describe base data. The *Security Parameter Index (SPI)* is part of the ESP header. It identifies the SA. The *tunnel endpoint addresses* (IPv4 source/destination address) identify the source and destination of the IPsec tunnel. Both are part of the new outer IPv4 header that encapsulates the ESP frame. The *register index* points to a particular index that holds the packet counter for the particular SA used by the cipher suite extern. *Packet limits* declare timeout conditions in terms of packet count thresholds for SAs. If a soft limit is reached, rekeying is triggered. If a hard limit is reached, packets that belong to that SA are dropped.

The NULL cipher suite is an example that only requires this set of base data. Typical cipher suites that implement particular encryption and authentication mechanisms require additional parameters such as keys, initialization vectors (IVs), or even additional constructs to keep cipher state, e.g., registers. AES-CTR, as an example for such a cipher suite, requires a key for AES and a key for keyed-hash message authentication code (HMAC).

The functionality within the cipher suite action is as follows. First, the packet counter for the particular SA is read from the register and incremented. Second, an ESP header is created with the SPI and sequence number of the packet. For the creation of the ESP packet, the action passes the original IP packet, the newly created ESP header, and the required keys of the cipher suite to the corresponding extern. The cipher suite extern performs encryption/authentication and responds with the ESP packet. Fourth, the new outer IP packet is created with the tunnel endpoint addresses. It encapsulates the newly created ESP packet. Last, timeout conditions are checked. The user metadata structure includes flags for *soft_limit_reached* and *hard_limit_reached* that are set in case of matching conditions. For soft_limit_reached, the packet is forwarded to the SDN controller to trigger tunnel renewal. In case of hard_limit_reached, the packet is dropped.

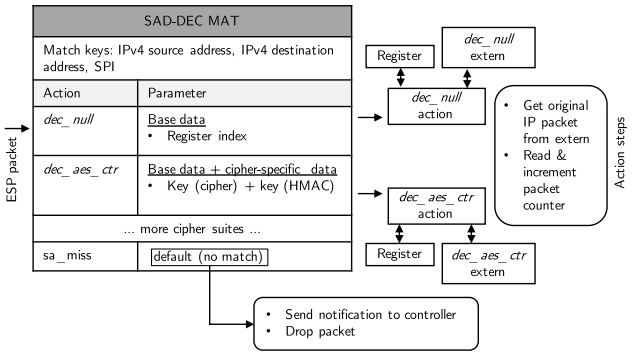


FIGURE 12. ESP decryption function block. ESP packets are processed by the SAD-DEC MAT. It holds entries for each SA with the corresponding data that is required for applying the associated cipher suite externs for decryption.

5) FUNCTION BLOCK: ESP DECRYPTION

Figure 12 depicts the function block of *ESP decryption*. We introduce the SAD-DEC MAT that resembles the decryption SAD from the IPsec standard (see Section II-A). Each entry in the MAT represents a particular SA that is identified by the outer IPv4 source and destination address (tunnel endpoints), and the SPI. As in the function block of ESP Encryption, cipher suites are implemented as actions that rely on externs and registers with a different set of action parameters.

The functionality within the cipher suite action is as follows. First, the packet counter for the particular SA is read

from the register and incremented. Second, the original IP packet is extracted from the ESP packet. Therefore, the action passes the ESP packet and the required keys of the cipher suite to the corresponding extern. The cipher suite extern performs decryption/authentication and responds with the original IP packet. Last, timeout conditions are checked as described in ESP encryption.

D. CONTROL PLANE OPERATION OF P4-IPsec

We first give an overview on the control plane operation of P4-IPsec. We describe how configuration data is generated on the SDN controller and how it is set up in both host-to-site and site-to-site operation mode.

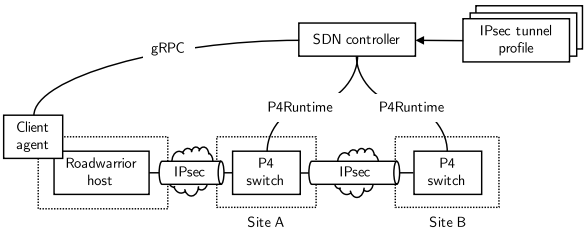


FIGURE 13. Control plane operation in P4-IPsec. The client agent on the roadwarrior host holds a control channel via gRPC to the SDN controller. P4 switches are connected via P4Runtime. The SDN controller includes IPsec tunnel profiles with configuration data for tunnel setup and management.

1) OVERVIEW

Figure 13 depicts the control plane interaction in the two operation modes of P4-IPsec, host-to-site and site-to-site mode. In both operation modes, IPsec tunnels are set up by the SDN controller on the basis of IPsec tunnel profiles. Those can be manually defined by an administrator or generated by another software component, e.g., a network operation platform. In host-to-site operation mode, the SDN controller interacts with the client agent via a gRPC tunnel and with the P4 switch via P4Runtime. In site-to-site operation mode, the SDN controller interacts with both P4 switches via P4Runtime. On roadwarrior hosts, configuration data is converted into *ip xfrm* commands that set up the tunnel. For P4 switches, the SDN controller directly writes to MATs and receives notifications, e.g., if an SA needs to be renewed, via P4Runtime. For the sake of simplicity, we restrict our implementation to proactive IPsec tunnel setup. In site-to-site mode, IPsec tunnels are set up and kept alive for all configured P4 switches. For host-to-site mode, the client agent presents a selection of available tunnels. The user then can select one or multiple IPsec tunnel profiles to be set up by the SDN controller.

This mechanism can be extended or substituted by more sophisticated approaches such as on-demand VPN setup. Predefined conditions (e.g., a request for a network resource in an internal network) may trigger IPsec tunnel setup via the SDN controller.

2) IPsec TUNNEL PROFILES

An IPsec tunnel between two peers consists of two unidirectional SAs, each identified by a unique SPI. Due to their direction, the first peer of an SA is called "left" where the second peer of the SA is called "right". We denote the SA from the left to the right peer as SPI_i and the SA from the right to the left peer as SPI_j, respectively. Each SA requires two MAT entries: one for encrypting ESP packets in the SAD-ENC MAT and one for decrypting ESP packets in the SAD-DEC MAT.

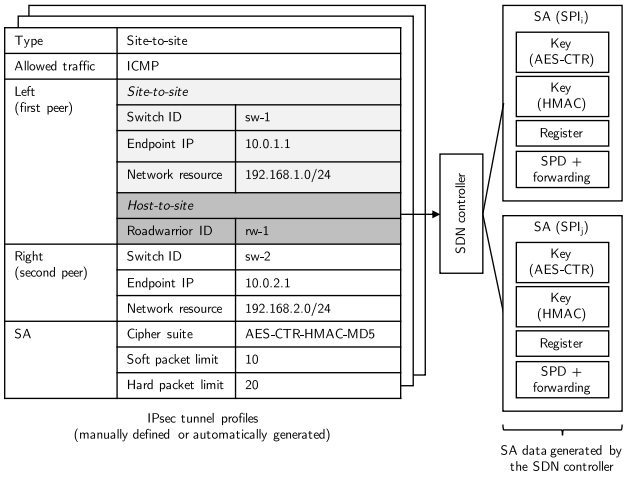


FIGURE 14. IPsec tunnel profiles with the associated SA data generated by the SDN controller. The configuration data in the IPsec tunnel profiles depends on the operation mode (host-to-site or site-to-site). SA data depends on the cipher suite that is defined in the IPsec tunnel profile.

Figure 14 depicts how the SDN controller generates configuration data for the roadwarrior hosts or P4 switches. IPsec tunnel profiles are the basis for any IPsec tunnel. As basic information about the tunnel, they include information about the type of IPsec tunnel (host-to-site or site-to-site) and the allowed traffic that is set to PROTECTED via SPD rules. The left peer (first) can be a P4 switch (in site-to-site operation mode) or a roadwarrior host (in host-to-site operation mode). In case of site-to-site operation mode, this field holds the switch ID (unique identifier of the P4 switch), endpoint IP (public IP address of the P4 switch), and network resource (internal network behind the P4 switch). In case of host-to-site operation mode, this field only holds the roadwarrior ID. The right peer (second) is always a P4 switch. Therefore, it holds the same data as in the left peer field in site-to-site operation mode as described before. The SA field holds the cipher suite and soft/hard packet limits. On the basis of an IPsec tunnel profile, the SDN controller generates configuration data for both SAs. In case of the AES-CTR-HMAC-MD5 cipher suite, SA data includes keys for AES-CTR and HMAC, register indexes, and configuration data for the SPD and forwarding function block. In case of the NULL cipher suite, keying material is not needed.

In our prototype, IPsec tunnel profiles are manually defined by an administrator. In practice, they can be generated by a software component, e.g., a network operation platform, on the basis of user/device profiles, groups, network resources, and permission models.

3) CONTROLLER CONNECTION

We describe the management connections in site-to-site and host-to-site operation mode.

a: SITE-TO-SITE OPERATION MODE (P4Runtime)

Site-to-site operation mode relies on P4Runtime for managing the P4 switches. Explained in Section II-B.3, the control plane connection to the P4 switches is established by the SDN controller. Therefore, it holds a list of connection data (name, address, port identity) of all assigned P4 switches.

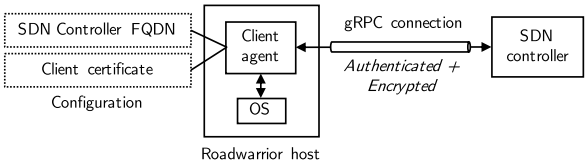


FIGURE 15. Control plane connection between the client agent and SDN controller. The client agent depends on the FQDN of the SDN controller and a client certificate to establish a gRPC connection to the SDN controller.

b: HOST-TO-SITE OPERATION MODE (gRPC)

Figure 15 depicts the connection between the client agent running on the roadwarrior host and the SDN controller. The required configuration data for the start of the client agent are the FQDN of the SDN controller and the client certificate. At start, the client agent establishes a gRPC tunnel to the SDN controller. The gRPC tunnel is protected with TLS, i.e., the client agent and SDN controller perform a mutual authentication using certificates and establish an encrypted connection. Certificates can be created and deployed to all roadwarrior hosts running the client agent and the SDN controller with a public key infrastructure (PKI). Roadwarrior host access can be removed by simply revoking the associated client certificate. In addition, gRPC provides support for optional multifactor authentication with token-based authentication via the Google Authenticator service. After connection setup, the client agent and SDN controller exchange configuration and signaling data via the gRPC tunnel. The client agent implements interfaces to interact with the roadwarrior host's operating system for configuration and signaling. The management connection to the P4 switch as a remote peer is established with P4Runtime as described before.

Host-to-site operation mode requires that the SDN controller is dual-homed. It has an interface to a management network where it holds P4Runtime connections to the P4 switches and another interface that makes it accessible via the Internet for client agents running on roadwarrior hosts. On the latter interface, it has an IP address that is publically reachable via the Internet. Although mutual certificate-based authentication protects against malicious P4-IPsec agents, this public interface should be protected as every publically available web service, e.g., with a firewall.

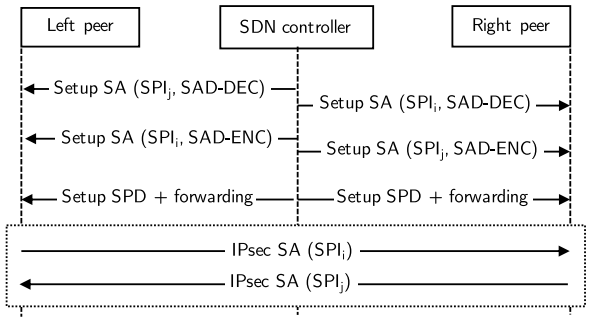


FIGURE 16. Setup procedure for a bidirectional IPsec tunnel on two peers. The SDN controller sets up decryption and encryption for both SAs followed by SPD and forwarding entries.

4) TUNNEL MANAGEMENT OPERATIONS

We describe the elementary management operations of tunnel setup, tunnel renewal, and tunnel deletion that apply to both operation modes.

a: TUNNEL SETUP

IPsec tunnel setup is a three-step process. First, the SDN controller sets up both SAs in the SAD-DEC MATs of both peers. Second, the SDN controller sets up both SAs in the SAD-ENC MATs of both peers. Setting up SA entries for decryption first ensures that no ESP packets get lost if one peer immediately starts to send ESP packets. Last, the SDN controller sets up the SPD and installs forwarding rules if required.

b: TUNNEL RENEWAL

IPsec SAs have a limited lifetime, i.e., keying material needs to be renewed on a regular basis. Both client agent and P4 switch notify the SDN controller if an SA needs to be renewed. For the client agent, this notification is triggered by the kernel implementation of IPsec that sends expiration messages in the case that soft and hard timeout limits are reached. For the P4 switches, this is implemented using packet counters in registers that are checked with each packet processing within an ESP encryption or decryption function block. We adopt the principle presented by Lopez-Millan *et al.* [43] that implements tunnel renewal without risking packet loss. When the SDN controller received the SA expire notification, it generates a new SA that is identified by a new SPI. Then, tunnel renewal follows the principle of tunnel setup as described before. First, the new SA is installed in the SAD-DEC MAT. Second, the existing SA in the SAD-ENC MAT is replaced by the new SA via a modify operation. Last, as it can be ensured that no packets are encrypted using the previous SA, its entry can be removed from the SAD-DEC MAT.

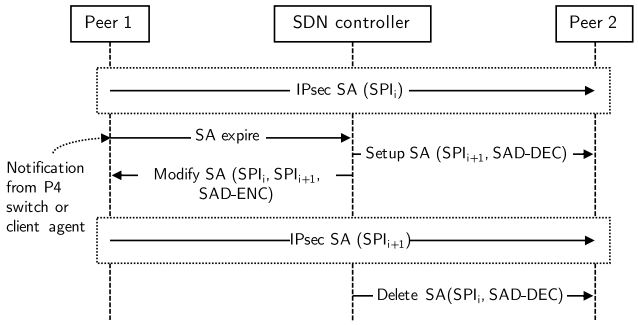


FIGURE 17. Renewal procedure for a unidirectional SA within an IPsec tunnel on two peers. After receiving an SA expire message, the SDN controller installs a new decryption SA on the remote peer. Afterwards, it replaces the expired encryption SA with new SA data. As a cleanup step, the old decryption SA is removed.

c: TUNNEL DELETION

If an IPsec tunnel should be deleted, the SDN controller removes the associated entries in the SPD, SAD-ENC, and SAD-DEC MAT. This is triggered on the SDN controller, e.g., when an IPsec profile is removed.

V. PROTOTYPICAL IMPLEMENTATION

We describe our software-based prototype of P4-IPsec. We present the testbed environment and outline the three parts of the implementation in detail. We publish the source code for both parts under the Apache v2 license on GitHub [1].

A. TESTBED ENVIRONMENT

Our prototypical implementation of P4-IPsec includes a softwarized testbed environment. We use Mininet to create network topologies that consist of BMv2 P4 switches and network hosts. We build it with Vagrant [79], a tool that simplifies the creation and management of virtual environments. All resources and setup steps are part of a configuration file. Executing *vagrant up* in a console within the repository folder automatically sets up and launches the testbed environment. It includes a virtual machine running Ubuntu 16.04 with all dependencies: libyang, sysrepo, mininet, protobuf, gRPC, PI/P4Runtime, BMv2, and P4C. The versions of all components can be found in the setup scripts.

B. DATA PLANE IMPLEMENTATION

We implement the P4 data plane implementation for the BMv2 P4 software target. We extend its *simple_switch* architecture by externs programmed in C++ for the AES-CTR-HMAC-MD5 and NULL IPsec cipher suites. Each cipher suite is implemented by two externs, one for encryption and one for decryption. For AES-CTR-HMAC-MD5, we use OpenSSL to apply AES-CTR for encryption/decryption and HMAC-MD5 for packet authentication. We implement the P4 processing pipeline as $P4_{16}$ program. It relies on the cipher suite externs and uses registers to store packet counters for the SAs. We run the P4 program on our extended *simple_switch* P4 target. We encapsulate our modified simple_switch P4 target within the *simple_switch_grpc* P4 target so that P4Runtime API can be used for interaction with the SDN controller.

C. CLIENT AGENT

We implement the client agent as Python 3.6 command line tool for Linux hosts. We integrate a gRPC client using the gRPC library [80] as an interface to the SDN controller. For IPsec tunnel setup, the client agent translates configuration data from the SDN controller into particular *XFRM* commands from the *iproute2* tool to configure IPsec on the roadwarrior host. In addition, it sets up IP routes for routing IP traffic via the IPsec tunnel. The received and applied configuration data is cached so that proper teardown configuration can be applied in case of tunnel shutdown. We implement rekeying with the help of *Netlink* [81]. The client agent monitors Netlink messages by listening on the corresponding Netlink socket and binding to the XFRMNLGRP_EXPIRE address so that XFRM Expire messages can be received. When receiving an XFRM Expire message, it extracts parameters such as SPI and IP addresses of the tunnel endpoints and notifies the SDN controller for tunnel renewal.

D. SDN CONTROLLER

We implement the SDN controller as a command line tool in Python 2.7. We use the p4runtime_lib [20] to integrate the interface to the P4 switch and the gRPC library to integrate the interface to the client agent. The SDN controller features a simple command line interface (CLI) for development and testing purposes that displays information about all active IPsec tunnels. P4Runtime and p4runtime_lib facilitate easy implementation of individual SDN controllers for prototypes. Nevertheless, those functions could be also integrated into existing SDN controllers such as ONOS or OpenDaylight.

VI. PERFORMANCE EVALUATION WITH THE SOFTWARE SWITCH BMv2

We describe the testbed environment and report the experiment results performed with the P4-IPsec software prototype introduced in Section V.

A. METHODOLOGY

We conduct the performance experiments in the testbed environment presented in Section V-A. The testbed runs on a Lenovo Thinkpad T480s (Intel i5-8250U CPU, 16 GB RAM) with Manjaro Linux. The Vagrant file in the repository includes the version numbers of all software components from the testbed environment.

Figure 18 depicts the experiment setup. S_1 and S_2 are BMv2 P4 switches, H_1 and H_2 are Linux hosts that are attached to them. S_1 and S_2 are connected via two unidirectional virtual links. We do not configure any additional delay or bandwidth limitations on these links. The traffic between H_1 and H_2 is forwarded by S_1 and S_2. We set up IPsec tunnels with different cipher suites and conduct TCP

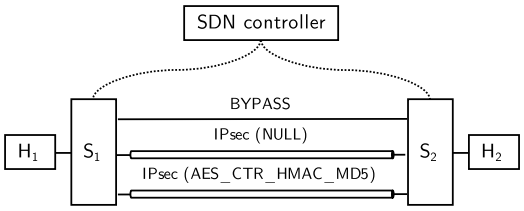


FIGURE 18. Experiment setup for evaluation of the P4-IPsec prototype on the BMv2 switches S_1 and S_2.

goodput measurements between H_1 and H_2 using iperf in version 3.7.

The results in the following represent measured average values with confidence intervals for a significance of $\alpha = 5\%$. Thus, the true averages lie within the displayed ranges with a probability of $1 - \alpha$.

B. DATA PLANE EVALUATION

We investigate how P4-IPsec's data plane implementation affects the overall throughput on the BMv2 P4 software target.

1) EXPERIMENT DESCRIPTION

We analyze TCP goodput for three different configurations. In the first scenario, we install BYPASS rules in the SPD so that traffic is only forwarded and not handled by IPsec. In the second scenario, we establish an IPsec tunnel between S_1 and S_2 with the NULL cipher suite. In the third scenario, we establish an IPsec tunnel between S_1 and S_2 with the AES_CTR_HMAC_MD5 cipher suite. Each experiment comprises 20 runs, each with a duration of 60 s and an MTU set to 1450 B. We configure soft and hard packet limits for rekeying to 50000 and 51000 resulting in an average of six rekeyings per run, three for each of the two SAs of the IPsec tunnel.

2) RESULTS & DISCUSSION

Figure 19(a) depicts the results. For forwarding without IPsec (BYPASS), TCP goodput is 48.90 Mbit/s. For IPsec forwarding with the NULL cipher suite, TCP goodput is 47.25 Mbit/s. For IPsec forwarding with the AES_CTR_HMAC_MD5 cipher suite, TCP goodput is 47.21 Mbit/s. Hence, the experimental results show similar TCP goodput rates between 47.21 Mbit/s and 48.90 Mbit/s for all three scenarios. The drop in performance is caused by IPsec processing, while the differences between both IPsec cipher suites are negligible. As all three results are still similar, we allocate the moderate overall TCP goodput to the runtime performance of the BMv2 P4 target. The low throughput of BMv2 is due to the fact that its use is intended for testing and not for production purposes. Thus, the results show that the overhead of our IPsec implementation on BMv2 only slightly reduces the TCP goodput and the impact of encryption/decryption operations is negligible on this platform.

C. CONTROL PLANE EVALUATION

We investigate how P4-IPsec's controller-based operation of IPsec affects the time needed for IPsec tunnel setup and renewal.

1) EXPERIMENT DESCRIPTION

In common IPsec deployment, SAs are set up between two IPsec peers using IKE message exchange. In P4-IPsec, an SDN controller sets up and renews IPsec tunnels, which may take longer due to controller operation and table updates on P4 switches.

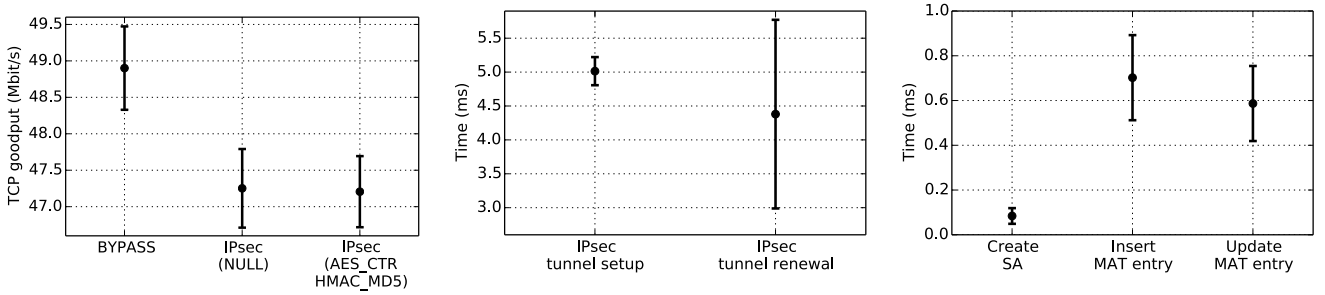
For *IPsec tunnel setup*, the SDN controller generates two unidirectional SAs, installs them on both P4 switches, and modifies the SPD MATs of both peers so that traffic is protected using IPsec. In our experiment, we measure the time for IPsec tunnel setup. It starts when the southbound connections between the SDN controller and the two P4 switches are established and ends with the last confirmation of the MAT modifications on the two P4 switches. For *IPsec tunnel renewal*, the SDN controller generates one unidirectional SA and installs it on both P4 switches. Tunnel renewal is triggered by a P4 switch if the packet counter of a SA reaches the soft packet limit. The measurement is started on the SDN controller when it receives the soft timeout notification from one P4 switch and it is stopped when the SDN controller has received all confirmations of the P4 switches about all MAT modifications. The details of both operations including sequence diagrams can be found in Section IV-D.4. We recorded measurement data for IPsec tunnel setup and tunnel renewal within the experiment on TCP goodput for the AES_CTR_HMAC_MD5 cipher suite as described before.

In the testbed environment, latency on the management link between the SDN controller and P4 switches is very low as all components run on the same host and as we have not configured any extra delay on the links. In real-world deployments, link latencies are significant, but they impact both IKE message exchange and controller-based operation of IPsec. By keeping the link latencies minimal, we derive an upper bound on potentially additional latency due to controller operation and MAT modification on P4 switches.

2) RESULTS & DISCUSSION

Figure 19(b) depicts the measured averages for IPsec tunnel setup and renewal times. Tunnel setup takes 5.02 ms while the time for tunnel renewal is 4.38 ms. These results show that the control plane overhead is low.

To further investigate both operations, we also analyze the durations of three major components: creating SA data, inserting new MAT entries, and updating existing MAT entries. Figure 19(c) depicts their average times. Creating keying material for a unidirectional SA takes 0.084 ms. Installing a new MAT entry via a write operation takes 0.702 ms. Updating an existing MAT entry takes 0.587 ms. Thus, the effort for key generation is almost negligible compared to MAT modifications.



(a) TCP goodput measured by iperf3 on the receiving host. Three variants are considered: without IPsec forwarding (BYPASS), with IPsec forwarding but without encryption, and with IPsec forwarding using the AES_CTR_HMAC_MD5 cipher suite.

(b) IPsec tunnel setup and renewal times. Times are measured on the SDN controller from initiation to completion of these processes.

(c) Times for creation of a single SA, insertion of a single MAT entry, and update of a single MAT entry. Times are measured on the SDN controller from initiation to completion of these processes.

FIGURE 19. Measurement results for experiments with P4-IPsec using BMv2 software switches in a virtual environment with almost zero link delays. Average values are shown with confidence intervals for a significance of $\alpha = 5\%$.

VII. IMPLEMENTATION ON HARDWARE P4 TARGETS

In the following, we describe implementation experiences for the NetFPGA SUME board and Edgecore Wedge 100BF-32X P4 switch.

A. NetFPGA SUME

We give an overview of the platform and describe implementation experiences.

1) OVERVIEW ON PLATFORM & DEVELOPMENT

NetFPGA SUME is an open source hardware development board for prototyping network applications. Its main part is a Xilinx Virtex-7 690T FPGA with 4 SFP+ ports that acts as programmable data plane. It supports throughput rates up to 100 GBit/s. The NetFPGA SUME board can be programmed via the Software Defined Specification Environment for Networking (SDNet) [78], a proprietary predecessor of P4 from Xilinx. Support for P4 programmability was introduced with the P4-NetFPGA tool [82]. First, a P4-to-SDNet compiler translates $P4_{16}$ programs into SDNet. Then, the SDNet compiler generates hardware description language (HDL) blocks in Verilog that can be validated in generic and platform-specific FPGA simulations. Finally, the HDL representation is synthesized into a hardware design to program the FPGA. In addition to the P4 program, custom functions can be implemented in a HDL and included in the hardware design. Programmers may implement custom HDL blocks or integrate semiconductor intellectual property cores that can be used as P4 externs in the P4 program. The P4-NetFPGA tool only supports the SimpleSumeSwitch architecture, i.e., P4 programs defined for more sophisticated architectures such as Portable Switch Architecture (PSA) need to be transformed to this architecture.

2) IMPLEMENTATION EXPERIENCES

We report on implementation experiences about porting our software-based implementation P4-IPsec for the NetFPGA SUME board. First, P4-NetFPGA is currently limited to *packet header manipulation*. P4-IPsec requires modifications of packet payloads, i.e., we were required to parse packet payloads as an additional header field. As P4-NetFPGA does not support parsing variable-length header fields, the implementation is limited to packets with a fixed length. Second, P4-NetFPGA lacks a packet streaming function for *data exchange between the P4 pipeline and P4 externs*. Instead, data between the P4 pipeline and externs is currently exchanged via blocks of bits. As this data transfer needs to be executed within one clock cycle of the FPGA, the data size is limited. We observed that this limit is about 10 kbit for one function call. This limits the maximum packet size to be processed through a P4 extern to about 600 B. During the synthesis, the Vivado suite optimizes the hardware implementation through several algorithms. In various experiments, we observed a practical upper bound of about 140 B for packets. Either the hardware implementation did use more resources than offered by the FPGA, or data transfer and calculation within the P4 extern exceeded one clock cycle. A packet streaming function was announced in 2018, but is still not available. Last, we encountered several *more general problems* with P4-SDNet and the NetFPGA SUME board. Probably due to a bug, we were not able to access the values of an LPM table for IP routing with our SDN controller. We solved that problem by using exact matching tables instead, an approach that is not acceptable for a production implementation. In addition, we experienced several stability problems. No matches in MATs were found when data was written to hardware registers. Finally, we missed many important details in the documentation. With hope for improved support, we managed to implement a very limited prototype. It only allows to apply the NULL cipher on fixed-length packets that do not exceed a total length of 140 B.

Scholz *et al.* [83] report on implementation experiences of cryptographic hashing functions in P4 data planes. The NetFPGA SUME board is also one of the platforms examined

where the authors present results that correspond to our results. As a workaround, the authors propose to move the externs subsequent to the synthesized P4 program. However, the workaround can be applied only if the P4 program does not rely on the output of the extern. This makes it inapplicable to P4-IPsec. Besides, implementing this workaround requires extensive knowledge about HDLs and FPGA programming.

B. EDGECORE WEDGE WITH TOFINO

We give an overview of the platform and describe why a direct adoption of P4-IPsec is not feasible. We present two workaround implementations and evaluate their performance in experiments.

1) OVERVIEW ON PLATFORM & DEVELOPMENT

The Edgecore Wedge 100BF-32X [84], features 32 QSFP28 network ports with throughput rates up to 100 Gbit/s. The QSFP28 ports interface the Tofino switching ASIC from Barefoot Networks which is fully programmable with P4. The Tofino ASIC connects to a CPU module via PCIe. It features an Intel Pentium D1517 processor (1.6 GHz, 4 cores), 8 GB RAM, and a 32 GB SSD. For programming and managing the Tofino ASIC, the CPU module runs the Barefoot P4 Software Development Environment on top of a Linux-based operating system. It loads and manages P4 programs during execution, provides management APIs (e.g., P4Runtime), and exposes an interface for network packet exchange between the P4 processing pipeline and the CPU module. Due to its optimization for high-speed packet processing with bandwidths up to multiple Tbit/s in data centers or core networks, user-defined P4 externs that may contain computation-intense functions are not supported.

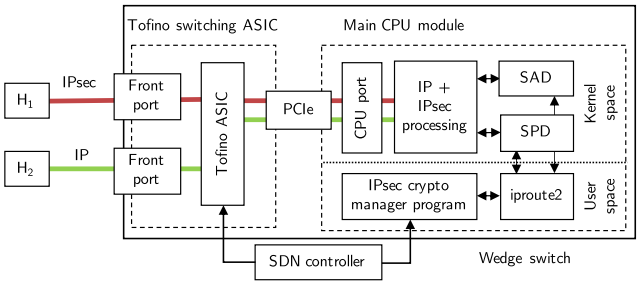


FIGURE 20. First workaround implementation. We relocate IPsec processing to the main CPU module that interfaces the Tofino switching ASIC via a PCIe CPU port.

2) WORKAROUND IMPLEMENTATIONS

In our first workaround implementation, we *relocate the P4 externs of P4-IPsec to the main CPU module*. Figure 20 depicts the concept. We replace all P4 extern function calls in the P4 processing pipeline by packet transfers via the CPU port to the main CPU module. On the main CPU module, we use the IPsec kernel functions of the Linux operating system for IPsec processing. We implement a simple IPsec crypto manager program that translates P4-IPsec configuration from the SDN controller into IPsec configuration for the Linux host. We implement the IPsec crypto manager in Python 3. It relies on iproute2 commands to manage the SPD and SAD of the Linux host.

We briefly evaluate this first workaround implementation with experiments on latency and TCP goodput. As depicted in Figure 20, we attach two physical hosts running Ubuntu 16.04 LTS via 100 Gbit/s links to the front ports of the Wedge switch. We enable IPsec on the link between H_1 and the switch while the link between the switch and H_2 remains unprotected. First, we measure the latency that is introduced by IPsec processing and packet exchange with the CPU module. We send 100 ICMP echo requests from H_1 to H_2 and measure an average round-trip time of about 1.5 ms. Second, we investigate the maximum TCP goodput. We generate TCP transmissions with iperf3 in three experiments, each performed with five runs and a duration of 30 s. For getting a reference, we measure the maximum TCP goodput between the P4 processing pipeline and the main CPU module. Therefore, we assign an IP address to the virtual network interface of the CPU port on the main CPU module and run iperf3 measurements between H_1 and the main CPU module. We measure an average TCP goodput of about 3.3 Gbit/s. We consider this as upper bound for the main CPU module. Now, we measure TCP goodput between H_1 and H_2. When using the NULL cipher suite, we measure an average TCP goodput of about 2 Gbit/s. For the AES-GCM-256 cipher suite, the average TCP goodput drops to about 1.4 Gbit/s. We repeat the experiment for 16 concurrent IPsec tunnels and calculate the average of 10 runs with a duration of 300 s. The maximum TCP goodput remains at 2 Gbit/s for IPsec with the NULL cipher suite and 1.4 Gbit/s for IPsec with the AES-GCM-256 cipher suite. We attribute the large differences in TCP goodput to the rather slow CPU with a base frequency of 1.6 GHz. Still, we consider this a very reasonable performance that might be sufficient for scenarios where only few shared network resources should be accessed sporadically by roadwarrior hosts.

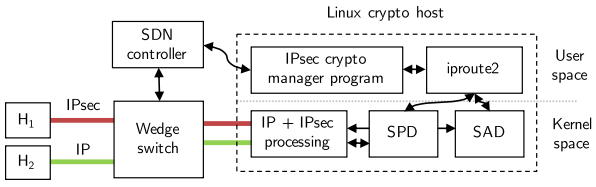


FIGURE 21. Second workaround implementation. We forward IPsec-related flows to a Linux crypto host.

In our second workaround implementation, we *forward IPsec-related flows to a crypto host*. This approach is used by several past works on integrating IPsec with fixed function SDN data planes (e.g., [45]). As depicted in Figure 21, we set up a Linux crypto host for offloading IPsec processing. We deploy the IPsec crypto manager program from the previous workaround implementation as an interface between the crypto host and SDN controller. We deploy a simple P4 program on the Wedge switch that forwards IPsec flows based

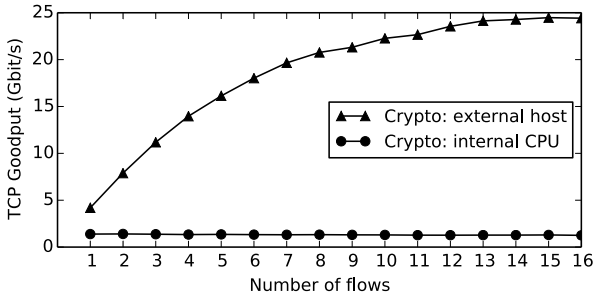


FIGURE 22. Average TCP goodput for both workaround implementations and 1-16 IPsec tunnels with the AES-GCM-256 cipher suite.

on a MAT. The SDN controller writes/edits the forwarding MAT on the Wedge switch and sends configuration messages to the IPsec crypto manager program.

For a simple evaluation, we set up a crypto host with an Intel Xeon Gold 6134 CPU (8 cores, 16 threads), 128 GB RAM, and a 240 GB SSD, running Ubuntu 18.04 LTS. We perform the same experiments as for the first workaround implementation. The round-trip time is about 2 ms which is slightly larger than in the previous approach. Figure 22 compares TCP goodput results for IPsec tunnels with the AES-GCM-256 cipher suite of both workaround implementations. For a single IPsec tunnel with the AES-GCM-256 cipher suite, we measure an average TCP goodput of about 4 Gbit/s. It can be increased by running multiple connections over the same crypto host. For 16 parallel IPsec tunnels, we measure an overall average TCP goodput of about 24 Gbit/s. This effect can be attributed to receive-side scaling (RSS) of the network interface card, which can leverage multiple cores, but only one per IPsec tunnel. In case of multiple IPsec tunnels, the overall TCP goodput can be increased through RSS by leveraging the processing power of more than a single core. Crypto capacity can be scaled up by increasing the number of crypto hosts connected to the switch. TCP goodput on each crypto host can be further improved by optimization techniques as presented in Section III-C.

Chen [85] presents an implementation of AES encryption for Tofino-based P4 switches. It uses a novel Scrambled Lookup Table technique that allows throughput rates of up to 10.92 Gbit/s for AES-128. However, the current concept is limited to blockwise encryption of packets with a maximum payload size of 16 bytes so that it is in its current form not a suitable base for IPsec support. If subsequent versions of this work introduce block chaining, integrating P4-IPsec could be an interesting follow-up work.

VIII. CONCLUSION

In this work, we proposed the first implementation of IPsec in P4. The proposed data plane implementation features ESP in tunnel mode and provides support for multiple cipher suites with the help of P4 externs. P4-IPsec supports automated operation of IPsec in host-to-site and site-to-site scenarios. IPsec tunnels are set up and managed by an SDN controller based on predefined tunnel profiles. For interaction with remote hosts in host-to-site scenario, we introduce a client agent for Linux hosts. We introduced the fundamentals of IPsec and data plane programming with P4, gave an extensive review on related work, and presented the architecture of P4-IPsec.

P4 programmable data planes open up the possibility of implementing IPsec on SDN-capable data plans for the first time. However, the implementation on P4 switches is still challenging. For the BMv2 software switch, the implementation was straightforward, but moderate data rates make its practical application difficult. However, the controller-supported signaling was not a bottleneck. Due to the platform limitations of the NetFPGA SUME board, we were not able to build a working prototype. With the Tofino-based Wedge switch, we were successful. Even though it does not support P4 externs, we presented two workaround implementations that leverage the main CPU module for crypto functions or an external crypto host, respectively.

We have shown that security use cases can benefit from P4, but crypto functions are still missing on P4 hardware switches. Therefore, we advocate for P4 hardware targets that either include P4 externs for those operations or offer powerful interfaces so that developers can run individual functions on the CPU module of such switches. Such features have the potential to massively foster the deployment of P4 targets in practice and stimulate further network research.

LIST OF ACRONYMS

SDN	software-defined networking
ONF	Open Network Foundation
OF	OpenFlow
ODP	Open Data Plane
NFV	network function virtualization
VNF	virtual network function
SFC	service function chaining
BMv2	Behavioral Model version 2
MAT	match-action table
VPN	Virtual Private Network
IP	Internet Protocol
IPsec	Internet Protocol Security
ESP	Encrypted Secured Payload
AH	Authentication Header
PPF	packet processing function
SP	security policy
SPD	Security Policy Database
SA	security association
SAD	Security Association Database
PAD	Peer Authentication Database
SPI	Security Parameter Index
IKE	Internet Key Exchange
IKEv2	Internet Key Exchange v2
IPComp	IP Payload Compression
AE	authenticated encryption
ICV	Integrity Check Value
IV	initialization vector

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
CBC	cipher block chaining
CTR	counter
GCM	galois/counter mode
HMAC	keyed-hash message authentication code
SHA	secure hash algorithm
TLS	Transport Layer Security
PKI	public key infrastructure
SoC	system on a chip
FPGA	field programmable gate array
NPU	network processing units
ASIC	application-specific integrated circuit
NIC	network interface card
NPU	network processing unit
APU	accellerated processing unit
DPDK	Data Plane Development Kit
SDNet	Software Defined Specification Environment for Networking
HDL	hardware description language
HLIR	high-level intermediate representation
RTL	Register Transfer Level
PSA	Portable Switch Architecture
TTL	time to live
INT	in-band network telemetry
LPM	longest-prefix matching
CLI	command line interface
FSM	finite state machine
API	application programming interface
LLDP	Link Layer Discovery Protocol
MACsec	Media Access Control Security
RSS	receive-side scaling

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments. The authors alone are responsible for the content of this paper.

REFERENCES

[1] *P4-IPsec Repository on GitHub*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/uni-tue-kn/p4-ipsec
[2] F. Hauser, M. Schmidt, M. Haberle, and M. Menth, "P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN," *IEEE Access*, vol. 8, pp. 58845–58858, 2020.
[3] K. Seo and S. Kent, *Security Architecture for the Internet Protocol*, document RFC 4301, Dec. 2005.
[4] S. Kent, *IP Authentication Header*, document RFC 4302, Dec. 2005.
[5] S. Kent, *IP Encapsulating Security Payload (ESP)*, document RFC 4303, Dec. 2005.
[6] J. Viega and D. McGrew, *The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)*, document RFC 4106, Jun. 2005.
[7] A. Shacham, B. Monsour, R. Pereira, and M. Thomas, *IP Payload Compression Protocol (IPComp)*, document RFC 3173, Sep. 2001.
[8] D. Carrel and D. Harkins, *The Internet Key Exchange (IKE)*, document RFC 2409, Nov. 1998.
[9] C. Kaufman, E. Paul Hoffman, Y. Nir, P. Eronen, and T. Kivinen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, document RFC 7296, Oct. 2014.
[10] N. Ferguson and B. Schneier, *A Cryptographic Evaluation of IPsec*. Santa Clara, CA, USA: Counterpane Internet Security, 2000.
[11] 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, 2008.
[12] R. Bifulco and G. Retvari, "A survey on the programmable data plane: Abstractions, architectures, and open problems," in *Proc. IEEE 19th Int. Conf. High Perform. Switching Routing (HPSR)*, Jun. 2018, pp. 1–7.
[13] *P4_16 Language Specification, Version 1.2.1*. Accessed: Jul. 26, 2020. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.1.html
[14] *BMv2*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/p4lang/behavioral-model
[15] *T4P4S*. Accessed: Jul. 26, 2020. [Online]. Available: http://p4.elte.hu/
[16] *DPDK*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.dpdk.org/
[17] *ODP*. Accessed: Jul. 26, 2020. [Online]. Available: https://opendataplane.org/
[18] *P4 Runtime Specification, Version 1.2.0*. Accessed: Jul. 26, 2020. [Online]. Available: https://p4.org/p4runtime/spec/v1.2.0/P4Runtime-Spec.html
[19] *Google Protocol Buffers*. Accessed: Jul. 26, 2020. [Online]. Available: https://developers.google.com/protocol-buffers/
[20] *P4 Language Tutorials*. Accessed: Jul. 26, 2020. [Online]. Available: https://githubcomlangtutorialstreemasterutils runtimelib
[21] A. Kiss, "Security middleware programming using P4," in *Security Middleware Programming Using*, T. Tryfonas, Ed. Cham, Switzerland: Springer, 2016, pp. 277–287.
[22] R. Datta, S. Choi, A. Chowdhary, and Y. Park, "P4Guard: Designing p4 based firewall," in *Proc. MILCOM - IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 1–6.
[23] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, and Q. Wang, "Hardware-accelerated firewall for 5G mobile networks," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 446–447.
[24] J. Cao, J. Bi, Y. Zhou, and C. Zhang, "CoFilter: A high-performance switch-assisted stateful packet filter," in *Proc. SIGCOMM*, 2018, pp. 9–11.
[25] E. O. Zaballa, D. Franco, Z. Zhou, and S. Michael Berger, "P4Knocking: Offloading host-based firewall functionalities to the network," in *Proc. Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, 2020, pp. 7–12.
[26] A. Almaini, A. Al-Dubai, I. Romdhani, and M. Schramm, "Delegation of authentication to the data plane in software-defined networks," in *Proc. IEEE Int. Conferences Ubiquitous Comput. Commun. (IUCC) Data Sci. Comput. Intell. (DSCI) Smart Comput., Netw. Services (SmartCNS)*, Oct. 2019, pp. 58–65.
[27] F. Paolucci, F. Cugini, and P. Castoldi, "P4-based multi-layer traffic engineering encompassing cyber security," in *Proc. Opt. Fiber Commun. Conf.*, 2018, pp. 1–3.
[28] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 Edge node enabling stateful traffic engineering and cyber security," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. 84–95, Oct. 2019.
[29] M. Dimolianis, A. Pavlidis, and V. Maglaris, "A multi-feature DDoS detection schema on p4 network hardware," in *Proc. 23rd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2020, pp. 1–6.
[30] A. Febro, H. Xiao, and J. Spring, "Telephony denial of service defense at data plane (TDoSDDP)," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, 2018, pp. 1–6.
[31] A. Febro, H. Xiao, and J. Spring, "Distributed SIP DDoS defense with p4," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2019, pp. 1–8.
[32] G. Grigoryan and Y. Liu, "LAMP: Prompt layer 7 attack mitigation with programmable data planes," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–4.
[33] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.
[34] C. Lapolli. Ã., J. A. A. Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, Oct. 2019, pp. 19–27.
[35] M. Kuka, K. Vojanec, J. Kucera, and P. Benacek, "Accelerated DDoS attacks mitigation using programmable data plane," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–3.
[36] Y. Mi and A. Wang, "ML-pushback: Machine learning based pushback defense against DDoS," in *Proc. 15th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2019, p. 80.

[37] B. Lewis, M. Broadbent, and N. Race, "P4ID: P4 enhanced intrusion detection," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2019, pp. 1–4.
[38] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable in-network security for context-aware BYOD policies," in *Proc. Secur. Symp.*, 2020, pp. 1–7.
[39] S. Aragon, M. Tiloca, M. Maass, M. Hollick, and S. Raza, "ACE of spades in the IoT security game: A flexible IPsec security profile for access control," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–9.
[40] S. Aragon, M. Tiloca, and S. Raza, *IPsec profile of ACE*. Fremont, CA, USA: Internet Engineering Task Force, 2017,
[41] D. Carrel and B. Weis, *IPsec Key Exchange using a Controller*. Fremont, CA, USA: Internet Engineering Task Force, 2019.
[42] X. Guo, K. Yang, A. Galis, X. Cheng, B. Yang, and D. Liu, "A policy-based network management system for IP VPN," in *Proc. Int. Conf. Commun. Technol. Process.*, 2003, pp. 1630–1633.
[43] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework," *Comput. Standards Interface*, vol. 66, Oct. 2019, Art. no. 103357.
[44] J. Son, Y. Xiong, K. Tan, P. Wang, Z. Gan, and S. Moon, "Protego: Cloud-scale multitenant IPsec gateway," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 473–485.
[45] M. Vajaranta, J. Kannisto, and J. Harju, "IPsec and IKE as functions in SDN controlled network," in *Proc. Int. Conf. Netw. Syst. Secur. (NSS)*, 2017, pp. 521–530.
[46] R. Lopez, G. Lopez-Millan, and F. Pereniguez-Garcia, *Software-Defined Networking (SDN)-based IPsec Flow Protection*. Fremont, CA, USA: Internet Engineering Task Force, 2019.
[47] A. S. Braadland, "Key Management for data plane encryption in SDN using wireGuard," M.S. thesis, Dept. Inf. Secur. Commun. Technol., Norwegian Univ. Sci. Technol., Norway, China, 2017.
[48] A. Sajassi, A. Banerjee, S. Thoria, D. Carrel, B. Weis, and J. Drake, *Secure EVPN*. Fremont, CA, USA: Internet Engineering Task Force, 2019.
[49] H. Gunleifsen, V. Gkioulos, and T. Kemmerich, "A tiered control plane model for service function chaining isolation," *Future Internet*, vol. 10, no. 6, p. 46, Jun. 2018.
[50] H. Gunleifsen, T. Kemmerich, and V. Gkioulos, "A Proof-of-Concept demonstration of isolated and encrypted service function chains," *Future Internet*, vol. 11, no. 9, p. 183, Aug. 2019.
[51] H. Gunleifsen, T. Kemmerich, and V. Gkioulos, "Dynamic setup of IPsec VPNs in service function chaining," *Comput. Netw.*, vol. 160, pp. 77–91, Sep. 2019.
[52] *Cisco: IPsec Management Configuration Guide*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_conn_imgmt/config%uration/xe-16/sec-ipsec-management-xe-16-book/sec-ipsec-snmp-supp.html
[53] A. Alharbi, A. Bahnasse, M. Talea, H. A. Oulahyane, and F. E. Louhab, *Smart SDN Policy Management Based VPN Multipoint*. Cham, Switzerland: Springer, 2019, pp. 250–263.
[54] Y. Li and J. Mao, "SDN-based access authentication and automatic configuration for IPSec," in *Proc. 4th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Dec. 2015, pp. 996–999.
[55] W. Li, F. Lin, and G. Sun, "SDIG: Toward software-defined IPsec gateway," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–8.
[56] M. Mechtri, I. Houidi, W. Louati, and D. Zeghlache, "SDN for inter cloud networking," in *Proc. IEEE SDN for Future Netw. Services (SDN4FNS)*, Nov. 2013, pp. 1–7.
[57] R. van der Pol, B. Gijsen, P. Zuraniewski, D. F. C. Romáo, and M. Kaat, "Assessment of SDN technology for an easy-to-use VPN service," *Future Gener. Comput. Syst.*, vol. 56, pp. 295–302, Mar. 2016.
[58] L. Rizzo, "Netmap: A Novel Framework for Fast Packet I/O," in *Proc. Annu. Tech. Conf.*, 2012, pp. 101–112.
[59] *N. Pf_ring*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.ntop.org/products/packet-capture/pf_ring/
[60] W. Li, S. Hu, G. Sun, and Y. Li, "Adaptive load balancing on multi-core IPsec gateway," in *Proc. Int. Conf. Algorithms Archit. Parallel Process. (ICAPP)*, 2018, pp. 1–7.
[61] G. Xie, H. Jiang, and K. Salamatian, "Load balancing by ruleset partition for parallel IDS on multi-core processors," in *Proc. Int. Conf. Comput., Commun. Netw. (ICCCN)*, 2013, pp. 1–7.
[62] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2010, pp. 195–206.
[63] S. Gallenmuller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *Proc. ACM/IEEE Symp. Architectures for Netw. Commun. Syst. (ANCS)*, May 2015, pp. 29–38.
[64] *Intel AES-NI*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.intel.de/content/www/de/de/architecture-and-technology/adva%nced-encryption-standard-aes/data-protection-aes-general-technology.html
[65] *Architecture Reference Manual*, Arm Ltd, Cambridge, U.K., 2017.
[66] J. DiGiglio and D. Ricci, *High Performance, Open Standard, Virtualization with NFV and SDN*. Alameda, CA, USA: Wind River, 2013.
[67] *Algotronix AES IP-Cores*. Accessed: Jul. 26, 2020. [Online]. Available: http://www.algotronix-store.com/AES_IP_Cores_s/20.htm
[68] C.-S. Ha, J. Hyoung Lee, D. Soo Leem, M.-S. Park, and B.-Y. Choi, "ASIC design of IPSec hardware accelerator for network security," in *Proc. IEEE Asia–Pacific Conf. Adv. Syst. Integr. Circuits*, 2004, pp. 168–171.
[69] A. Hodjat, P. Schaumont, and I. Verbauwhede, "Architectural design features of a programmable high throughput AES coprocessor," in *Proc. Int. Conf. Inf. Technol.*, 2004, pp. 1–7.
[70] Y. Liu, D. Xu, W. Song, and Z. Mu, "Design and implementation of high performance IPSec applications with multi-core processors," in *Proc. Int. Seminar Future Inf. Technol. Manage. Eng.*, Nov. 2008, pp. 595–598.
[71] J. Meng, X. Chen, Z. Chen, C. Lin, B. Mu, and L. Ruan, "Towards High-Performance IPsec on Cavium OCTEON Platform," in *Int. Conf. Trusted Syst. (INTRUST)*, 2011, pp. 37–46.
[72] J. Park, W. Jung, G. Jo, I. Lee, and J. Lee, "PIPSEA: A practical IPsec gateway on embedded APUs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1255–1267.
[73] M. Rao, J. Coleman, and T. Newe, "An FPGA based reconfigurable IPSec ESP core suitable for IoT applications," in *Proc. 10th Int. Conf. Sens. Technol. (ICST)*, Nov. 2016, pp. 1–5.
[74] M. Vajaranta, V. Viitamaki, A. Oinonen, T. D. Hamalainen, A. Kulmala, and J. Markunmaki, "Feasibility of FPGA accelerated IPsec on cloud," in *Proc. 21st Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2018, pp. 569–572.
[75] M. Korona, K. Skowron, M. Trzepinski, and M. Rawski, "FPGA implementation of IPsec protocol suite for multigigabit networks," in *Proc. Int. Conf. Syst., Signals Image Process. (IWSSIP)*, May 2017, pp. 1–5.
[76] B. Driessen, E. B. Kavun, O. Mischke, and C. Paar, "IPSecco: A lightweight and reconfigurable IPSec core," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs (ReConFig)*, 2012, pp. 1–7.
[77] *[P4-dev] VPNs*. Accessed: Jul. 26, 2020. [Online]. Available: https://lists.org.pipermail.lists.org.html
[78] *Xilinx SDNet PX Programming Language User Guide*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.xilinx.com.support.documentation.sw.manuals.xilinx.ug
[79] *Vagrant*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.vagrantup.com/
[80] *GRPC*. Accessed: Jul. 26, 2020. [Online]. Available: https://grpc.io/
[81] *Netlink*. Accessed: Jul. 26, 2020. [Online]. Available: http://man7.org/linux/man-pages/man7/netlink.7.html
[82] *P4-NetFPGA Wiki*. Accessed: Jul. 26, 2020. [Online]. Available: https://github.com/NetFPGA/P4-NetFPGA-public/
[83] D. Scholz, A. Oeldemann, F. Geyer, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in P4 data planes," in *ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Sep. 2019, pp. 1–6.
[84] *EdgeCore Wedge 100BF-32X*. Accessed: Jul. 26, 2020. [Online]. Available: https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=3%35
[85] X. Chen, "Implementing AES encryption on programmable switches via scrambled lookup tables," in *Proc. Workshop Secure Program. Netw. Infrastruct.*, Aug. 2020, pp. 1–7.

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

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

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

MICHAEL MENTH (Senior Member, IEEE) is a Professor at the Department of Computer Science, University of Tübingen, and the Chair Holder of Communication Networks. His special interests include performance analysis and optimization of communication networks, resilience and routing issues, resource and congestion management, software-defined networking and Internet protocols, industrial networking, and the Internet of Things.