WYDAWNICTWO SIGMA-NOT

MIESIĘCZNIK STOWARZYSZENIA ELEKTRYKÓW POLSKICH

Rok Jubileuszowy Zrzeszania się Polskich Inżynierów i Techników 1835-2015

Przegląd Telekomunikacyjny

Wiadomości Telekomunikacyjne

8–9
2015

ISSN 1230-3496, e-ISNN 2449-7487

Cena: 56,70 zł (w tym 5% VAT)

XXXI KRAJOWE SYMPOZJUM TELEKOMUNIKACJI I TELEINFORMATYKI KRAKÓW, 16 – 18 września 2015 r.

PRZEGLĄD TELEKOMUNIKACYJNY
+ WIADOMOŚCI TELEKOMUNIKACYJNE
Tele-Radio-Elektronika
Informatyka

TELECOMMUNICATION REVIEW
+ TELECOMMUNICATION NEWS
Tele-Radio-Electronics,
Information Technology

8–9'2015

TREŚĆ

CONTENTS

Artykuły naukowe publikowane w PTiWT uzyskują 6 punktów (zgodnie z wykazem czasopism naukowych ogłoszonym w komunikacie Ministra Nauki i Szkolnictwa Wyższego z dnia 17 grudnia 2013 r.).

Artykuły naukowe publikowane w niniejszym zeszycie są recenzowane. Zeszyt wydany w wersji elektronicznej jako pierwotnej (referencyjnej)

PRASA FACHOWA
SIGMA-NOT
www.sigma-not.pl

00-950 Warszawa
skrytka pocztowa 1004
ul. Ratuszowa 11
tel.: 022 818-09-18, 022 818-98-32
fax: 022 619-21-87

Internet: http://www.sigma-not.pl

Prenumerata
e-mail: prenumerata@sigma-not.pl

Sekretariat
e-mail: sekretariat@sigma-not.pl

Dział Reklamy i Marketingu
e-mail: reklama@sigma-not.pl

Redakcja: ul. Ratuszowa 11 (budynek Instytutu Tele- i Radiotechnicznego), VI piętro, pokój 637, tel. 22 670-08-20 (+ poczta głosowa), tel./faks: 22 619-86-99. Przyjęcia interesantów w godz. 10–14.
Adres do korespondencji: ul. Ratuszowa 11, 00-950 Warszawa 1, skrytka poczt. 1004
E-mail: przeg.tel@sigma-not.pl, przeg.tel@interia.pl Internet: www.przegladtelekomunikacyjny.pl
Czasopismo dostępne wyłącznie w prenumeracie
Artykułów niezamówionych redakcja nie zwraca.
Redakcja zastrzega sobie prawo dokonywania skrótów i poprawek w nadesłanych materiałach.
Przygotowanie: Studio DTP SIGMA-NOT, Ratuszowa 11, 00-950 Warszawa
Druk i oprawa: Drukarnia Sigma-NOT, www.sigma-not.pl

Cena zeszytu: 56,70 zł (w tym 5% VAT)

Nakład do 5000 egz.

KSTiT 2015

Niniejszy podwójny zeszyt (8-9'2015) naszego miesięcznika jest związany z **XXXI Krajowym Sympozjum Telekomunikacji i Teleinformatyki KSTiT'2015.** Odbywa się ono w Krakowie w dniach 16 – 18 września br.

Zeszyt zawiera całość materiałów konferencyjnych. Referaty plenarne oraz wybrane referaty sekcyjne są prezentowane w formie artykułów drukowanych, zaś pozostałe referaty są zamieszczone na płycie CD, stanowiącej integralną część *Przeglądu Telekomunikacyjnego i Wiadomości Telekomunikacyjnych*. **Wszystkie artykuły i referaty są recenzowane**. Obszerne informacje na temat konferencji znajdą Państwo w artykule wstępnym (II str. okładki), przygotowanym przez prof. Andrzeja Pacha – przewodniczącego Sympozjum.

Komitet Programowy – recenzenci

Krzysztof Abramski, Politechnika Wrocławska
Błażej Adamczyk, Politechnika Śląska
Marek Amanowicz, Wojskowa Akademia Techniczna
Piotr Arabas, NASK
Potr Bratoszewski, Politechnika Gdańska
Jarosław Bułat, AGH
Wojciech Burakowski, Politechnika Warszawska
Robert Chodorek, AGH
Piotr Chołda, AGH
Andrzej Chydziński, Politechnika Śląska
Janusz Cichowski, Politechnika Gdańska
Tadeusz Czachórski, Instytut Informatyki Teoretycznej i Stosowanej PAN
Andrzej Czyżewski, Politechnika Gdańska
Grzegorz Danilewicz, Politechnika Poznańska
Jacek Dańda, AGH
Andrzej Dąbrowski, Politechnika Warszawska
Andrzej Dobrogowski, Politechnika Poznańska
Jerzy Domżał, AGH
Przemysław Dymarski, Politechnika Warszawska
Piotr Gajewski, Wojskowa Akademia Techniczna
Mariusz Głąbowski, Politechnika Poznańska
Andrzej Głowacz, AGH
Ryszard Golański, AGH
Janusz Gozdecki, AGH
Tomasz Grajek, Politechnika Poznańska
Michał Grega, AGH
Adam Grzech, Politechnika Wrocławska
Piotr Hoffmann, Politechnika Gdańska
Andrzej Jajszczyk, AGH
Artur Janicki, Politechnika Warszawska
Lucjan Janowski, AGH
Mieczysław Jessa, Politechnika Poznańska
Wojciech Kabaciński, Politechnika Poznańska
Sylwester Kaczmarek, Politechnika Gdańska
Andrzej Kasprzak, Politechnika Wrocławska
Michał Kasznia, Politechnika Poznańska
Jacek Kołodziej, AGH
Jerzy Konorski, Politechnika Gdańska
Katarzyna Kosek-Szott, AGH
Józef Kotus, Politechnika Gdańska
Jerzy Kubasik, Politechnika Poznańska
Sławomir Kula, Politechnika Warszawska
Paweł Kułakowski, AGH
Artur Lasoń, AGH
Wiesław Ludwin, AGH
Adam Łuczak, Politechnika Poznańska
Sławomir Maćkowiak, Politechnika Poznańska
Wojciech Mazurczyk, Politechnika Warszawska
Wojciech Molisz, Politechnika Gdańska
Mariusz Mycek, Politechnika Warszawska
Marek Natkaniec, AGH
Marcin Niemiec, AGH
Andrzej Pach, AGH
Piotr Pacyna, AGH
Zdzisław Papir, AGH
Andrzej Paszkiewicz, Politechnika Warszawska
Michał Pióro, Politechnika Warszawska
Mirosław Popis, WAT
Grzegorz Różański, WAT
Marek Sikora, AGH
Władysław Skarbek, Politechnika Warszawska
Rafał Stankiewicz, AGH
Maciej Stasiak, Politechnika Poznańska
Maciej Szczodrak, Politechnika Gdańska
Krzysztof Szczypiorski, Politechnika Warszawska
Piotr Szotkowski, Politechnika Warszawska
Szymon Szott, AGH
Paweł Szulakiewicz, Politechnika Poznańska
Andrzej Szymański, AGH
Jacek Świderski, Instytut Energetyki Instytut Badawczy Oddział Gdańsk
Hubert Trzaska, Politechnika Wrocławska
Krzysztof Wajda, AGH
Krzysztof Walkowiak, Politechnika Wrocławska
Michał Wągrowski, AGH
Krzysztof Wesołowski, Politechnika Poznańska
Tadeusz Więckowski, Politechnika Wrocławska
Marian Wnuk, Wojskowa Akademia Techniczna
Józef Woźniak, Politechnika Gdańska
Robert Wójcik, AGH
Jacek Wszołek, AGH
Jacek Wszołek, AGH
Ryszard Zieliński, Politechnika Wrocławska
Tomasz Zieliński, AGH

Referaty sesji tematycznych zamieszczone na CD

Sesja 1: Architektury i protokoły komunikacyjne, cz. 1

Sesja 2: Bezpieczeństwo sieci telekomunikacyjnych i systemów teleinformatycznych, cz. 1

Sebastian Szwaczyk
Military University of Technology, Warsaw, Poland
sebastian.szwaczyk@student.wat.edu.pl

Konrad Wrona
NATO Communications and Information Agency, The Hague, Netherlands
konrad.wrona@ncia.nato.int

Sander Oudkerk
Agent Sierra Consulting Services, Amsterdam, Netherlands
sander.oudkerk@agentsierra.nl

IMPLEMENTATION OF CONTENT-BASED PROTECTION AND RELEASE IN SOFTWARE-DEFINED NETWORKS

DOI: 10.15199/59.2015.8-9.57

Abstract: **Future civilian and military communications and information systems require a new method for more dynamic and fine-grained control of access to information. A Content-based Protection and Release (CPR) model has been proposed to address the challenges introduced by future military operations. In this paper we present how the CPR concept can be applied to software-defined networks (SDN) in order to provide integrated protection of information in transit. In particular, we provide an in-depth discussion of a proof-of-concept implementation of CPR enforcement in SDNs.**

1. INTRODUCTION

An important current research and development topic in military and civilian communications and information systems (CIS) is the design of a new method for more dynamic and fine-grained control of access to information. Such a new approach must not only effectively support need-to-know and responsibility-to-share requirements in military operations, but also enable cross-layer enforcement of security policies. The enforced security policies should cover all three dimensions of security – confidentiality, integrity and availability. This is in strong contrast to the situation in a traditional military CIS environment, in which enforcement of security policies is focused mostly on confidentiality and access control mechanisms.

One of the proposals made to address the above requirements is the Content-based Protection and Release (CPR) model [1] developed by the NATO Communications and Information Agency, specifically to address the challenges related to implementation of Federated Mission Networking (FMN) [2] and future NATO operations. At the same time Protected Core Networking [3] (PCN) has been proposed as a paradigm for the implementation of a flexible and secure communication infrastructure for a federated military environment. Software-defined networks (SDN) [4] are emerging as a popular approach to network operation and management in the civilian domain. The SDN environment offers an opportunity for enforcement of much more dynamic and complex security policies at the network layer, making it very suitable for supporting both CPR and PCN. In this paper we show how the CPR concept can be applied to an SDN in order to provide integrated protection of information in transit. In particular, we provide an in-depth discussion of a proof-of-concept implementation of CPR enforcement in SDN.

2. PATH CLASS APPROACH

2.1. Introduction

In order to support CPR and PCN in an SDN environment, it is necessary to configure the packet path in the SDN based on so-called *link attributes (LA)*. Examples of possible LAs are available bandwidth, physical medium, network protection measures, etc. Our innovative approach is to derive the requirements for these LAs dynamically, based on the content properties of the transported data, by applying the CPR Policy. In this section we provide an overview of the *Path Class (PC)* approach, which allows this problem to be solved, and introduce CPR Link Requirements (CPR LR), the Enforcement Action Identifier (EAI), Possible Routes (PR) and the CPR Enforcement and Separation Service (CPRESS). Furthermore, we describe existing OpenFlow link attributes and explain how to add new attributes, which we call CPR LA. Finally, we describe how to incorporate the Enforcement Action Identifier (EAI) into the flow label of an IPv6 header.

2.2. CPR MODEL

Modern joint military missions rely on network-centric operations. It has been observed that traditional access control models such as discretionary (DAC), mandatory (MAC), and role-based (RBAC) models are not always adequate in this environment [5]. The Attribute-Based Access Control (ABAC) model (see e.g. [6]) offers a powerful and unifying extension to these well-known models. Under ABAC, requesters are permitted or denied access to a resource based on the properties, called attributes, that may be associated with users, resources, and the context. Examples of attributes are: identity, role, and military rank of users; identifier and sensitivity of resources; and, for context, time of day and threat level. Under ABAC, suitably defined attributes can represent security labels, clearances and classifications (for encoding MAC), identities and access control lists (for DAC), and roles (for RBAC). In this sense, ABAC supplements traditional access control models rather than supplanting them [6]. Policies in

ABAC can be seen as conditions on the attribute values of the entities involved in an access decision or, in other words, they are Boolean functions that map the attribute values of the user u, the resource r, and the context c to true (*permit*) when u is entitled to have access to r in context c, and false (*deny*) otherwise. The model underlying CPR policies refines ABAC in two main respects. First, in addition to the attributes of users, resources, and the context, those of terminals are considered, i.e. the capabilities of the device through which a user is trying to access a resource. Examples of terminal attributes are the hardware model, the type of encryption used to locally store data and the type of connection to the terminal (e.g. SSL). Second, the CPR (access control) policies are structured in two distinct sub-policies: a release policy, taking into account user, resource, and contextual attributes, and a protection policy, taking into account resource, terminal, and contextual attributes. This enables separation of policy management roles and reflects the current procedures used within international and governmental organizations, e.g. NATO. For example, consider the situation in which a user wants to access NATO classified information. This requires, on the one hand, connecting to a network infrastructure used for processing NATO classified information. To do this, a terminal must satisfy a number of technical requirements related to hardware and software configuration that are precisely defined in NATO technical directives and guidance documents. On the other hand, the security policy governing user access to the documents stored in the network is defined in a separate set of directives and guidance documents. A user u can access a resource r with a terminal t by checking if (i) the attributes of u and r satisfy the release policy and (ii) those of r and t satisfy the protection policy. If checks (i) and (ii) are both positive, *permit* is returned, otherwise the result is *deny*.

2.3. The concept of the Path Class approach

The PC approach is a way to configure the path in an SDN to release protected data to the requester using information about the links in the SDN. To realize the PC approach we extend the CPR model by introducing additional attributes called *CPR Link Requirements (CPR LR)*. These attributes can be explicitly included in the CPR policy or defined in the form of bridge predicates [7]. CPR LR specify requirements that links in the SDN must fulfil in order to be used to transport protected data to the requester. Examples of CPR LA are confidentiality, availability and integrity. The enforcement mechanism used to make a decision to release data or not is called CPRESS. In order to integrate CPRESS with SDN we extend it with specific functionality, explained below. We call the extended version of CPRESS the SDN CPRESS.

Based on the current user and terminal attributes and comparing these attributes to the CPR Policy, the SDN CPRESS makes a decision to release data or not to the requester. If the decision is to release, then based on the CPR LR the SDN CPRESS produces a specific PC for that data. The PC is a category of allowed LA that must characterize the links that build a path in the SDN between the requester terminal and the server. The SDN CPRESS also must map this PC to the **Enforcement Action Identifier (EAI)**. The EAI is a specific value included in the header of every packet carrying the protected data. This value is used by the controller to identify the specific PC and program specific flows on switches in the SDN. When a packet with an EAI is prepared the SDN CPRESS sends it to the first switch in the SDN. A summary of the function of the CPR SDN service is:

1. Based on CPR LR, produce a specific PC.
2. Map the PC to the EAI and put this identifier in a packet.
3. Send the packet with the EAI to the first switch in the SDN.

The process explained above is depicted in Figure 1:

1. The user using the terminal sends the request for the protected data.
2. The SDN CPRESS looks up the content properties of the requested data.
3. The SDN CPRESS contacts the Policy Administration Point (PAP) in order to obtain the CPR Policy.
4. If the SDN CPRESS decision is to release the data to the requester, the PC is produced based on the CPR LR from the CPR Policy.
5. The PC is mapped to the EAI and the SDN CPRESS puts this identifier in the packet.
6. The packet with the EAI is sent to the first switch in the SDN.

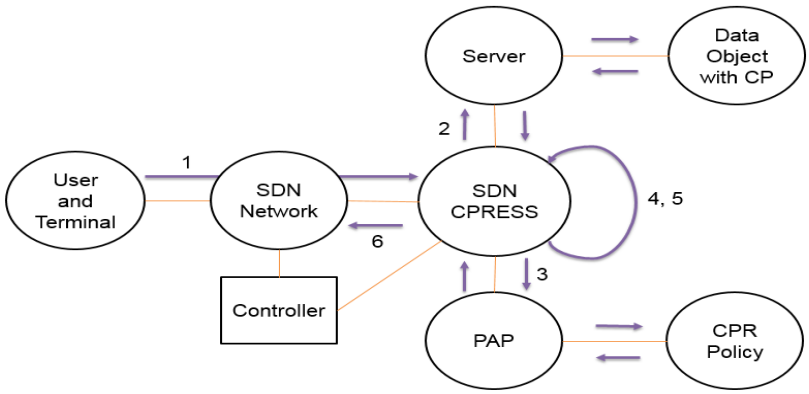

Figure 1. SDN CPRESS process

2.4. Transferring protected data through an SDN

The protected data to be transferred is contained in a packet stream. The first packet in the packet stream, which contains the EAI, is sent to the SDN by the SDN CPRESS. When the switch in the SDN receives this packet, it looks up the flow table to find the output port for the packet. If a matched flow entry does not exist, the switch sends the packet to the controller. The controller parses the packet and looks for the EAI. If the controller knows the Path Class identified by the EAI, it can produce, based on the EAI and the network topology, a set of *possible routes (PR)*. PR are routes that fulfil the CPR LR that are produced based on the EAI. Otherwise the controller communicates with the SDN CPRESS to obtain the CPR LR that are mapped to the EAI. The SDN CPRESS extracts, based on the EAI, the CPR LR from the CPR Policy and sends them to the controller. If more than one possible route exists, the controller must choose one of them but must store all possible routes in order to be able to react quickly if the chosen route fails. Furthermore, the controller programs the flows on switches using the EAI, source and destination IP addresses, and source and destination TCP ports. With these 5 attributes we can uniquely identify every connection between the requester and the server. After flows are programmed, the controller returns the packet to the network and the switches forward the packet to the requester based on the flows programmed. When a switch receives the second packet in a packet stream, it forwards this packet based on the matched flow entry, and so on for all subsequent packets. The process of transferring the protected data through an SDN is depicted in Figure 2.

2.5. Sharing CRP LR between SDNs

A PCN environment may be composed of multiple SDNs (e.g. Protected Core Segments (PCN)). In such a case, the CPR LR that are produced in the first network have to be shared with other networks included in the packet's path. When the controller in another network receives the packet with an EAI that it cannot map to the CPR LR, it has to ask the controller in the network where the EAI was created. Based on the CPR LR received from the controller in the originating network, other controllers can program flow on the switches.

2.6. EAI in a packet with protected data

The OpenFlow specification [8] lists mandatory fields that switches **must** support in the matching process. Every field describes the connection between hosts. If any change is made in any of these fields the connection between the hosts will be lost, so none of these fields can be used for any other purpose, including encoding EAI. However, OpenFlow also specifies optional fields that switches **should** support. Below is a short discussion of fields that are potentially useful for implementing the PC approach.

OFPXMT_OFB_VLAN_PCP = 7, VLAN priority: VLAN priority is used to prioritize different classes of traffic (voice, video, data, etc.). This field has 3 bits. If VLAN networks are used and QoS is not important, it is possible to use this field for the eight values of EAI.

OFPXMT_OFB_IP_DSCP = 8, IP DSCP and **OFPXMT_OFB_IP_ECN = 9, IP ECN:** These fields are used to support QoS in a network that uses the IPv4 protocol in the network layer. Since these fields are often not used, we can make use of this one byte for 256 values of EAI. Of course this means that we would lose the ability to use QoS.

OFPXMT_OFB_MPLS_TC = 35, MPLS TC: TC is a 3-bit field for implementing QoS. It is possible to use these bits for the EAI but of course we would then lose the ability to implement QoS.

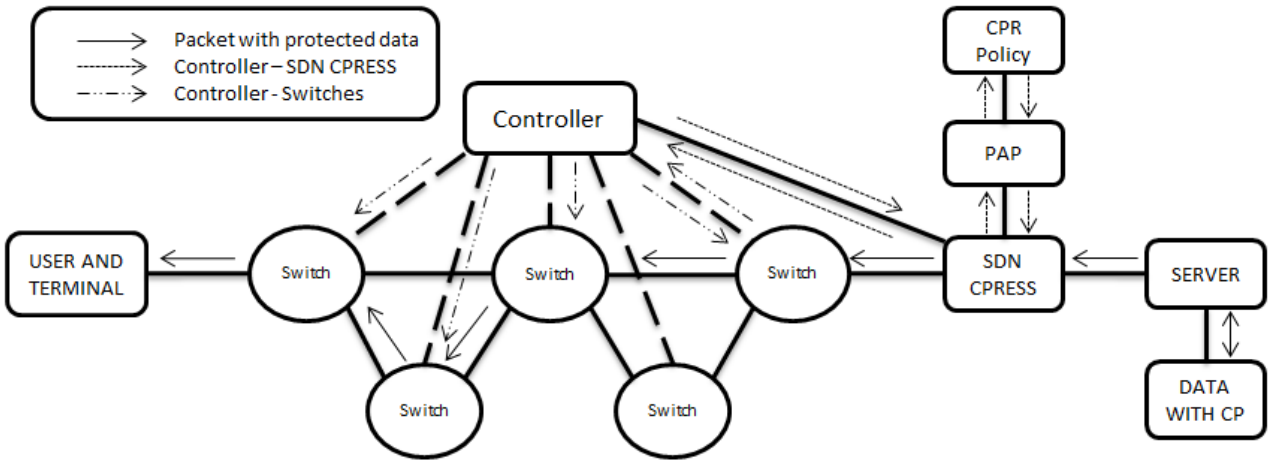

Figure 2. Packet transmission from server through an SDN to end user

2.7. Link Attributes

In the OpenFlow 1.3.4 specification each port of each switch is described by the following structure:

```
/* Description of a port */
struct ofp_port {
uint32_t port_no;
uint8_t pad[4];
uint8_t hw_addr[OFP_ETH_ALEN];
uint8_t pad2[2];
char name[OFP_MAX_PORT_NAME_LEN];
uint32_t config;
uint32_t state;
uint32_t curr; /* Current features. */
uint32_t advertised; /* Features being advertised by the port. */
uint32_t supported; /* Features supported by the port. */
uint32_t peer; /* Features advertised by peer. */
uint32_t curr_speed; /* Current port bitrate in kbps. */
uint32_t max_speed; /* Max port bitrate in kbps */
};
```

The fields most relevant to the PC approach are *curr*, *advertised*, *supported* and *peer*, because these fields describe the link mode (speed and duplexity), link type (copper or fibre), and link features (auto-negotiation and pause). These four variables store the LA for links connected to a specific switch port. To distinguish CPR link attributes from the attributes described by the OpenFlow specification we call these attributes **OF Link Attributes (OF LA)**. OF LA are listed in Section 2.8.

The port status is described using three flags:

OFPPS_LINK_DOWN = 1 << 0
OFPPS_BLOCKED = 1 << 1
OFPPS_LIVE = 1 << 2.

By sending the message from the controller to the switch we can configure the port:

OFPPC_PORT_DOWN = 1 << 0
OFPPC_NO_RECV = 1 << 2
OFPPC_NO_FWD = 1 << 5
OFPPC_NO_PACKET_IN = 1 << 6.

If one of these flags or OF LA is changed, the switch sends a PORT_STATUS message to the controller, giving one of three possible reasons:

OFPPR_ADD = 0
OFPPR_DELETE = 1
OFPPR_MODIFY = 2.

The controller can use the OFPMP_PORT_ DESCRIPTION request to obtain the description of all the standard ports of the OpenFlow switch.

2.8. Including CPR Link Attributes

It is impossible to add CPR LA without changing OpenFlow messages and the switch implementation. The OF LA available in the OFPMP_PORT_DESCRIPTION reply are:

OFPPF_10MB_HD = 1 << 0
OFPPF_10MB_FD = 1 << 1
OFPPF_100MB_HD = 1 << 2
OFPPF_100MB_FD = 1 << 3
OFPPF_1GB_HD = 1 << 4
OFPPF_1GB_FD = 1 << 5
OFPPF_10GB_FD = 1 << 6
OFPPF_40GB_FD = 1 << 7
OFPPF_100GB_FD = 1 << 8
OFPPF_1TB_FD = 1 << 9
OFPPF_OTHER = 1 << 10
OFPPF_COPPER = 1 << 11
OFPPF_FIBER = 1 << 12
OFPPF_AUTONEG = 1 << 13
OFPPF_PAUSE = 1 << 14
OFPPF_PAUSE_ASYM = 1 << 15.

The packet with these attributes is depicted in Figure 3.

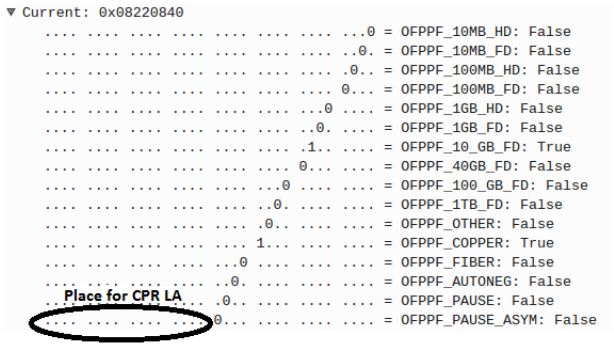

Figure 3 Open Flow Link Attributes

Figure 3 shows that there are another two currently unused bytes (circled, tagged "Place for CPR LA"). We can declare new link attributes in these two bytes.

3. IMPLEMENTATION OF PATH CLASS APPROACH DEMONSTRATOR

3.1. Overview

This section describes how individual components of the demonstrator interact with each other. The following components are used to demonstrate the PC approach (depicted in Figure 4):

- A Python script, which describes the SDN topology.
- A Mininet instance with changed source code to invoke the Java program.
- A Java program, which creates the configuration file.
- A configuration file.
- An Open vSwitch with the modified source code for reading the ports' configurations from the configuration file.
- A controller with the CPRDemonstrator module.

The Python script describes the network topology. The easiest way to describe the network topology is to use the high-level application program interface (API) delivered with the Mininet. We assume that the user writing the Python script uses the Mininet API because it allows another program to create the configuration file. (The purpose and structure of the configuration file are explained in the next section.)

The standard version of the Mininet parses the Python script and based on that, it starts and connects switches. Starting switches means that the Mininet

invokes commands that create virtual interfaces in the operating system and configures these interfaces. Also, the Mininet invokes the program that oversees the operation of the switches. The modified version of the Mininet, before starting to parse the Python script, invokes the Java program, which creates the configuration file. When the configuration file is created the Mininet parses the Python script and starts switches.

The Java program, which creates the configuration file, receives the path for the Python script from the Mininet. Based on this script the Java program creates the configuration file. This file is used by the switches to obtain link attributes for each port. Based on this file, the *curr* field of the structure presented in section 2.8 is populated.

If the configuration file is created successfully, the Mininet starts and connects switches based on the Python script. When each switch is started it reads the configuration for each port from the configuration file. When all ports are configured the switch tries to connect to the controller.

The Floodlight controller in OpenFlow version 1.3 listens for connections from the switches on port 6653. If both sides support OpenFlow v. 1.3, the connection is established and communication between the controller and the switch is available. The CPRDemonstator module is responsible for:

- Storing the topology of the network.
- Programming the flows on the switches based on the EAI (CPR LR) and LA.
- Reacting to events in the network (e.g. when a port goes down).
- Communicating with another CPRDemonstrator module to share the information about CPR LR.

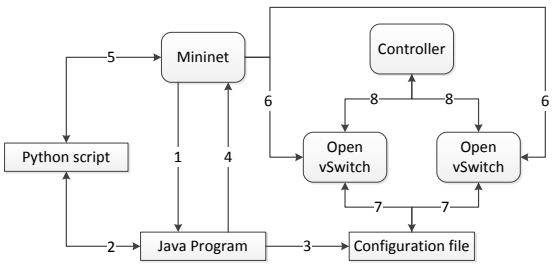

Figure 4 Starting and configuring switches

Figure 4 illustrates the process of starting the switches and configuring them from the configuration file. We assume that the modified version of the Mininet is started and that the Python script exists. The process is as follows:

1. If the modified version of the Mininet was started, it invokes the Java program.
2. The Java program reads the Python script.
3. Based on the Python script the Java program creates the configuration file for the switches' ports.
4. The Java program returns the exit code to the Mininet.
5. If the exit code indicates that the configuration file was created successfully the Mininet starts parsing the Python script.
6. Based on the Python script the Mininet creates and starts switches.
7. While switches are starting, when ports are opening, the configuration for each port is read from the configuration file.
8. After configuration the switches are ready to communicate with the controller.

3.2. Transporting CPR Link Attributes

As mentioned above, in the OFPMP_PORT_DESCRIPTION reply there are two currently unused bytes. Figure 3 shows the location for the CPR LA. Our initial implementation supports three CPR LA: confidentiality, integrity and availability. Each CPR LA can take one of the following values: NONE, LOW, MEDIUM, or HIGH. Therefore only 2 bits are necessary to store each attribute and in the initial implementation 2 bytes are divided into 3 fields in the following way (see also Figure 6):

- 4 bits for availability
- 6 bits for integrity
- 6 bits for confidentiality

Only the 2 least significant bits of each field are used in the initial demonstrator – the other bits can be used for encoding more values for each attribute in the future. When the controller sends the OFPMP_PORT_DESCRIPTION request to the switch, the switch builds each port description based on the fields in the *ofputil_phy_port* structure. Figure 5 depicts the OFPMP_PORT_DESCRIPTION reply that contains the CPR LA.

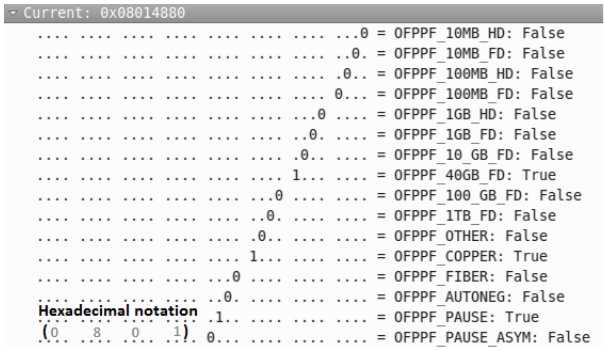

Figure 5 OFPMP_PORT_DESCRIPTION reply with CPR LA

In Figure 5 the grey numbers in the last row describe the values of bits in hexadecimal notation. Figure 6 depicts the values in binary notation with divisions of bits for each CPR LA.

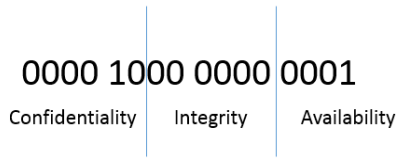

Figure 6 Binary values of the CPR LA

As explained earlier, currently only the two least significant bits are used to store the value of each CPR LA, so in the above examples these values are:

- Confidentiality = 10
- Integrity = 00
- Availability = 01.

The following mapping is used between the binary values and the CPR LA values:

- 00 = NONE
- 01 = LOW
- 10 = MEDIUM
- 11 = HIGH.

In the example depicted in Figure 6, the encoded values of the CPR LA are:

- Confidentiality = MEDIUM = 000010
- Integrity = NONE = 000000
- Availability = LOW = 0001.

The CPRDemonstrator module in the controller can interpret these values of the CPR LA in order to configure appropriate communication paths, as described in the following section.

3.3. The process of sharing protected data

This section describes the process of receiving a request, creating an EAI and sending a response to the requester. The process is depicted in Figure 7. Throughout the entire process we assume that user and terminal always fulfil the requirements to receive the protected data.

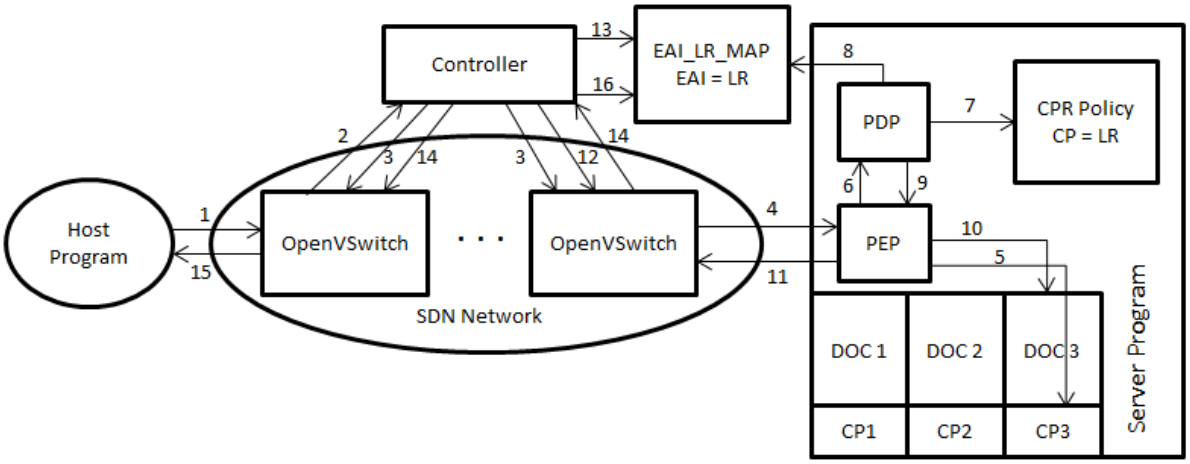

Figure *7 Send request, generate EAI and send response processes*

The process is as follows:

1. The HOST program sends a User Datagram Protocol (UDP) request for a document stored at the server.
2. The switch sends a PACKET_IN message to the controller.
3. The controller checks the EAI. The EAI is equal to 0 so the controller calculates the shortest path and programs the flows on the switches. The controller sends a PACKET_OUT message to the switch that sent the PACKET_IN message.
4. The switches, based on the programmed flows, forward the request to the server.
5. The Policy Enforcement Point (PEP) part of the server program receives the request, parses it and looks for the content properties of the document requested.
6. The PEP sends these content properties to the Policy Decision Point (PDP) part.
7. The PDP looks at the CPR Policy and retrieves the requirements for the links in the SDN.
8. The PDP creates the EAI and stores it in the EAI_LR_MAP file.
9. The PDP returns the EAI to the PEP.
10. The PEP creates a new packet with the value of the EAI in the Type of Service (TOS) field of the IPv4 header and the data from the requested document in the data field in the UDP header.
11. The PEP sends the packet to the SDN.
12. The switch sends a PACKET_IN message to the controller.
13. The controller looks for the EAI. The EAI has a value different from 0 so based on the EAI the controller reads the CPR LR from the EAI_LR_MAP file. Based on these requirements the controller calculates the possible paths between the server and the host.
14. The controller programs the shortest of the possible paths on the switches.
15. The switches, based on the programmed flows, forward the response to the host.
16. After the end of transmission the controller deletes the EAI from the EAI_LR_MAP file so that the value can be used again for another EAI.

4. TESTS

4.1. Network topology

The test network topology described in the Python script is depicted in Figure 8. The LA described in the configuration file assume that all links in the network have:

- Speed: 10Gb/s
- Medium: Copper
- Pause: False
- Auto-negotiation: False
- Pause Asymmetric: False.

The links between the host and the switch have these CPR LA:

- Confidentiality: HIGH (H)
- Integrity: HIGH (H)
- Availability: HIGH (H).

CPR LA for links between switches are depicted in Figure 8. For example the link between s2 and s3 has these CPR LA:

- Confidentiality: HIGH (H)
- Integrity: LOW (L)
- Availability: MEDIUM (M).

The addressing scheme used is:

- 10.0.0.1 to 10.0.0.8 for hosts h1 to h8
- 10.0.0.9 for the server.

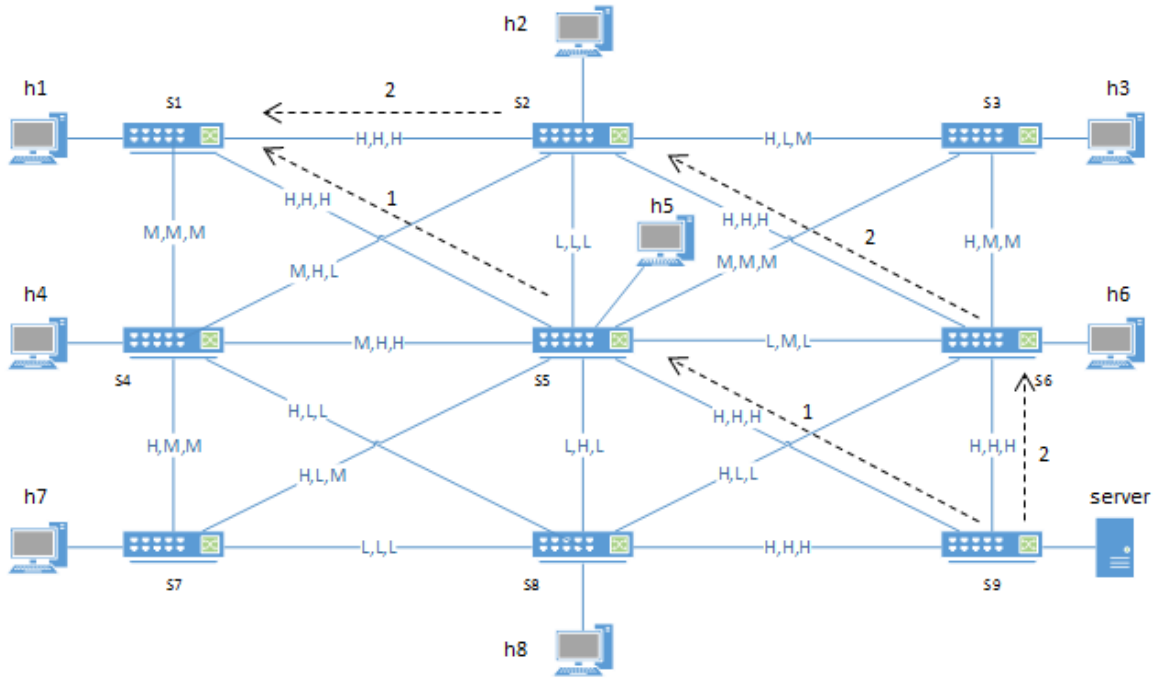

Figure 8 Test network topology

4.2. Demonstrating the PC Approach

The first step in demonstrating the Path Class approach is to start the host and server programs on nodes emulated by the Mininet. The server program must be started on the same node as the one that is emulating s9. Once started, the server program listens for requests on port 50000 of the UDP. Next, the host program can be used to generate a request for a document.

Logs from the controller are shown in Figure 9. The first part describes the request. The controller received a PACKET_IN message from switch number 1. This is the switch connected to h1. Shown next is the h1 IP address as a source and the server IP address as a destination. When the controller receives the PACKET_IN message it gets the EAI value from the request. The next line shows that the value of the EAI is equal to 0. If the EAI is 0, the controller used the Dijkstra algorithm [9] to obtain the shortest path. The next line shows the calculated and programmed path encoded using switch numbers.

The next part of the log shows the controller's reaction to the server response. First, one can see that the controller received the PACKET_IN message from switch number 9. This is the switch connected to the server. Next one sees the server IP address as a source and the h1 IP address as a destination. This time the EAI value in the message is equal to 1. This is the same value that is seen in the server terminal. In the next step the controller recovers the requirements for the EAI value. The listed LR are the same as the ones listed in the server terminal. Based on the source and destination IP addresses as well as the network topology, the controller

calculates all paths. From all paths the controller chooses only those paths that fulfil requirements. For the network topology depicted in Figure 8, one can see that only two paths from server to h1 fulfil the LR of H,H,H. The paths are s9-s5-s1 and s9-s6-s2-s1. From this set of possible paths, the controller choses the shortest one, which is programmed on the switches.

The controller programs flows on the switches with idle time equal to 10 seconds. After this time has lapsed the flows on the switches are removed and the switches send the FLOW_REMOVED message to the controller. The controller receives this message, knows that the transmission is complete and deletes the EAI value form the EAI_LR_MAP file. Thus the same EAI value can be used later for identifying other requirements.

```
PACKET_IN message received from switch: 00:00:00:00:00:00:00:01
Source IP address : 10.0.0.1. Destination IP address: 10.0.0.9.
EAI = 0
It is request. Shortest path will be programmed.
Programmed path: [1, 5, 9]

PACKET_IN message received from switch: 00:00:00:00:00:00:00:09
Source IP address : 10.0.0.9. Destination IP address: 10.0.0.1.
EAI = 1
Requirements for EAI = 1:
Bandwidth: 10GB_FD
Medium: COPPER
Autonegotiation: FALSE
Pause: FALSE
Pause Asymetric: FALSE
Confidentiality: HIGH
Integrity: HIGH
Availability: LOW
All paths between switch number: 9 and switch number: 1.
{0=[9, 8, 7, 4, 1], 1=[9, 8, 7, 4, 5, 1], 2=[9, 8, 7, 4, 5, 6, 3, 2, 1],
Paths which fulfill requirements:
{0=[9, 6, 2, 1], 1=[9, 5, 1]}
Shortest path which fulfill requirements (programmed route):
[9, 5, 1]

Flows for EAI = 1 removed. We can use this EAI for another requirements.
```

Figure 9 Creating routes for request and response

4.3. Reacting to a PORT_DOWN event

If during transmission one of the currently utilized links goes down, the controller deletes the flows that are associated with the paths that are using the broken link. When the switch receives the next packet in the stream it sends the PACKET_IN message to the controller and the controller re-calculates the possible paths for the data in this packet. Example logs from the controller console are shown in Figure 10.

```
PORT_DOWN in path: [1, 5, 9]. Flows on switches removed.

PORT_DOWN in path: [9, 5, 1]. Flows on switches removed.
Topology updated.

PACKET_IN message received from switch: 00:00:00:00:00:00:00:09
Source IP address : 10.0.0.9. Destination IP address: 10.0.0.1.
EAI = 1
Requirements for EAI = 1:
Bandwidth: 10GB_FD
Medium: COPPER
Autonegotiation: FALSE
Pause: FALSE
Pause Asymetric: FALSE
Confidentiality: HIGH
Integrity: HIGH
Availability: LOW
All paths between switch number: 9 and switch number: 1.
{0=[9, 8, 7, 4, 1], 1=[9, 8, 7, 4, 5, 1], 2=[9, 8, 7, 4, 5, 6, 3, 2, 1],
Paths which fulfill requirements:
{0=[9, 6, 2, 1]}
Shortest path which fulfill requirements (programmed route):
[9, 6, 2, 1]
Topology updated.

Flows for EAI = 1 removed. We can use this EAI for another requirements.
```

Figure 10 Reacting to a PORT_DOWN event

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented how the Content-based Protection and Release (CPR) concept can be applied to an SDN in order to provide integrated protection of information in transit. In particular, we have provided an in-depth discussion of a proof-of-concept implementation of CPR enforcement in SDN. We have shown that it is feasible to effectively enforce CPR policies related to confidentiality, integrity and availability protection in the SDN environment. Including CPR support in SDN is in line with the NATO doctrine of in-depth security and can significantly improve security posture and mitigate possible risks in a federated mission networking environment.

The purpose of the demonstrator is to show how the CPR model can be applied to an OpenFlow SDN environment using the Path Class approach; it is not intended to provide a complete implementation, suitable for operational deployment. Therefore, our current implementation of the demonstrator is based on some simplifying assumptions:

- The user is always authorized to receive protected data.
- The functionality of the SDN CPRESS, PEP and PDP is implemented in one server program.
- Communication between the controller and the SDN CPRESS is performed via files, not via network sockets.
- Every SDN makes use of the same set of LA.
- The switches are configured via configuration files.
- The channel between switch and controller is not secured.

Possible future work on extending the demonstrator could start with addressing or removing the above assumptions. The most important extension would be to implement full support for enforcement of CPR policies. The current demonstrator is based on the assumption that attributes of the user and his terminal always fulfil the applicable CPR Policy and that CPRESS must enforce the relevant protection policy only in regard to the network path.

The SDN CPRESS, PEP and PDP are implemented in one server program. Although the developed software can be easily split into these 3 separate components, the integrated approach was taken in order to avoid potential problems with communication and thus to simplify the development tasks.

For demonstration purposes we do not implement an interface for communication between the controller and the SDN CPRESS; they communicate via a file, thus implying that the controller and the SDN CPRESS have to be on the same machine. Extending the demonstrator by implementing network communication between these elements would allow the controller and the SDN CPRESS to also be run in a real, and not only an emulated, SDN environment.

Currently communication between controllers is limited to exchanging the EAI value mapped to the LR, based on the assumption that every PCS uses the same set of LA. In a real system every PCS can have its own

LA (and/or LR), which are mapped to another set of content properties (CP). It is recommended that a mechanism be implemented that would allow controllers to negotiate LR if different LA are used in each network.

OpenFlow specifications specify only a few LA. For the PC approach these LA are not sufficient. Section 2.8 describes how we add new CPR LA to OpenFlow. The idea of extending the *current* field in the OFPMP_PORT_DESCRIPTION reply is promising, because current OF specification allows that to be done without any other modification to the protocol. However, this introduces the challenge of how switches can recognize the values for CPR LA. In our current solution switches are configured from the configuration file, so the administrator must specify the configuration for each port in the network. It is recommended that algorithms that allow switches to recognize CPR LR be investigated and implemented.

In order to obtain better performance of the controller, it is recommended that a more robust approach to reacting to events in the network be implemented. In the current demonstrator, when the controller detects that a path goes down it must repeat all calculations based on the next PACKET_IN message. For the sake of performance, it would be advantageous to calculate paths once and keep them updated, so that alternate path information is available to be instantly programmed if the current path goes down.

OpenFlow specification marks the use of the TLS protocol as optional. Therefore, both the Floodlight controller and the Open vSwitch do not support this protocol by default. Currently, the controller and switches communicate using the not-secure TCP and are vulnerable to several potential attacks [10]. Therefore, any operational implementation of our solution needs to implement communication using the TLS protocol.

REFERENCES

[1] K. Wrona and S. Oudkerk, "Content-based Protection and Release Architecture for Future NATO Networks," in Proc. of the IEEE Military Communications Conference MILCOM, San Diego, CA, USA, 2013.

[2] A. Domingo and H. Wietgrefe, "On the federation of information in coalition operations: Building single information domains out of multiple security domains," in Proc. of the IEEE Military Communications Conference MILCOM, San Diego, CA, USA, 2013.

[3] G. Hallingstad and S. Oudkerk, "Protected core networking: An architectural approach to secure and flexible communications," IEEE Communications Magazine, vol. 46, no. 11, 2008.

[4] D. Kreutz, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE, vol. 103, no. 1, 2015.

[5] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, and P. Samarati. "Access control policies and languages in open environments," in Secure Data Management in Decentralized Systems. Springer, 2007.

[6] X. Jin, R. Krishnan, and R. Sandhu. "A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC," in Proc. of the DBSec, number 7371 in LNCS, pages 41-55, 2012.

[7] A. Armando, S. Oudkerk, S. Ranise, and K. Wrona, "Content-based Protection and Release for Access Control in NATO Operations," in Proc. of the 6th International Symposium on Foundations & Practice of Security (FPS). La Rochelle, France: Springer, 2013.

[8] Open Networking Foundation, "OpenFlow Switch Specification", version 1.3.4, 2014.

[9] Dijkstra, E. W., "A note on two problems in connexion with graphs," Numerische Mathematik 1: 269–271, 1959.

[10] G. Pickett, "Abusing Software Defined Networks," in Proc. of the Black Hat Europe. Amsterdam, Netherlands, 2014.